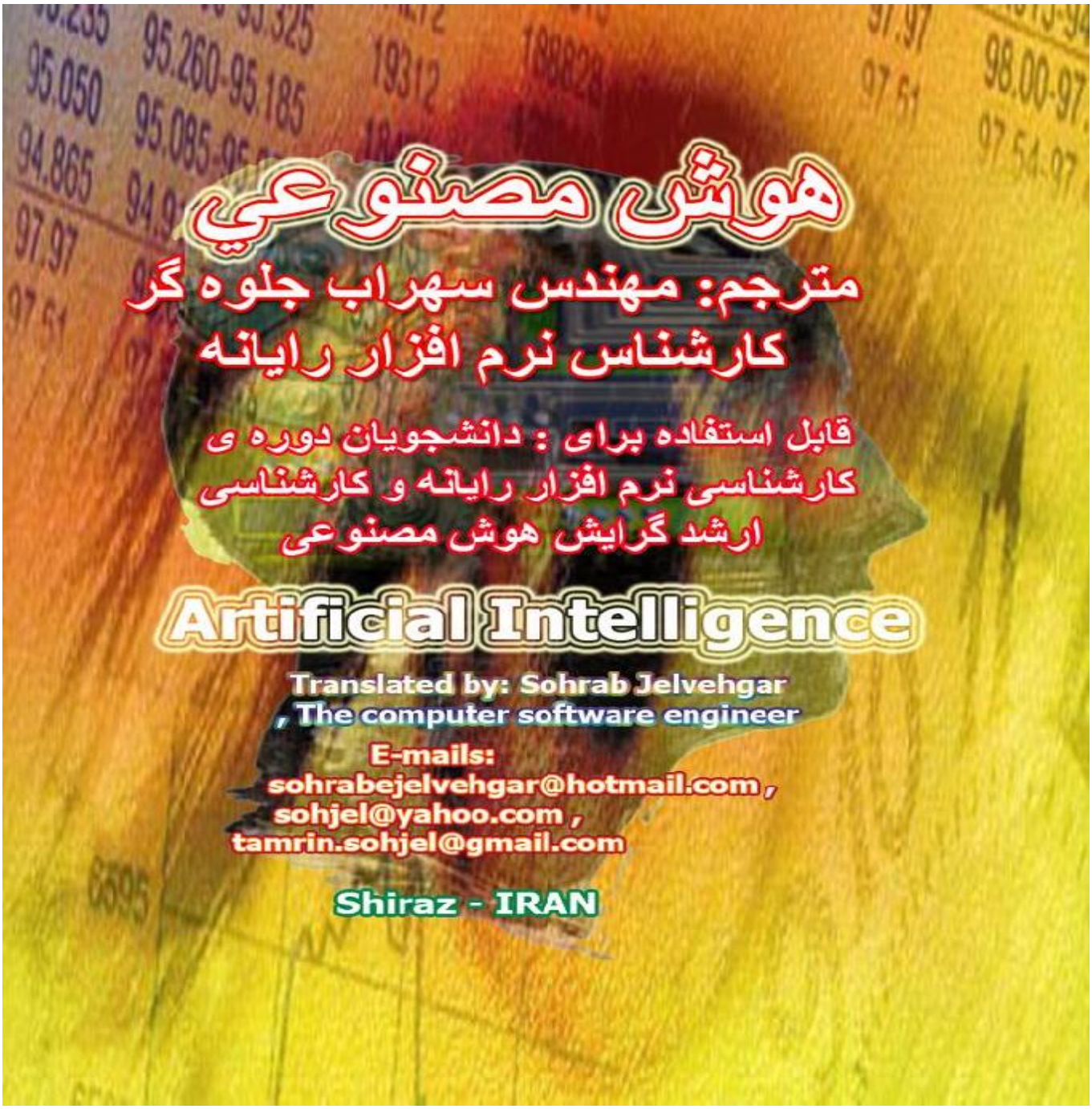


حمایت مالی از مترجم

لطفا در صورت تمایل وجهی را به میزان دلخواه به حساب اینجانب که در زیر آمده است پرداخت نمایید :

۸۱۵۳۹۴/۰ (به ترتیب از چپ به راست : هشتصد و پانزده هزار و سیصد و نود و چهار ممیّز صفر) بانک ملّت شعبه ی میدان پیام شیراز ، کد شعبه : ۳۹۳۵۴ (به حروف : سی و نه هزار و سیصد و پنجاه و چهار) واقع در شیراز ، خیابان شریف آباد ، میدان پیام ، به نام سهراب جلوه گر

این مجموعه در بیست و نه فصل می باشد و در زیر برخی از منابع مورد استفاده در تهیه ی این مجموعه را ملاحظه می نمایید .

فهرست برخی از منابع

دانشگاه آزاد بوزن ایتالیا ، به نشانی اینترنتی http://www.inf.unibz.it

دانشگاه کوبلنز آلمان ، به نشانی اینترنتی http://www.uni-koblenz.de

دانشکده ی علوم کامپیوتر دانشگاه دولتی تگزاس ایالات متحده ی آمریکا ، به آدرس اینترنتی www.cs.txstate.edu

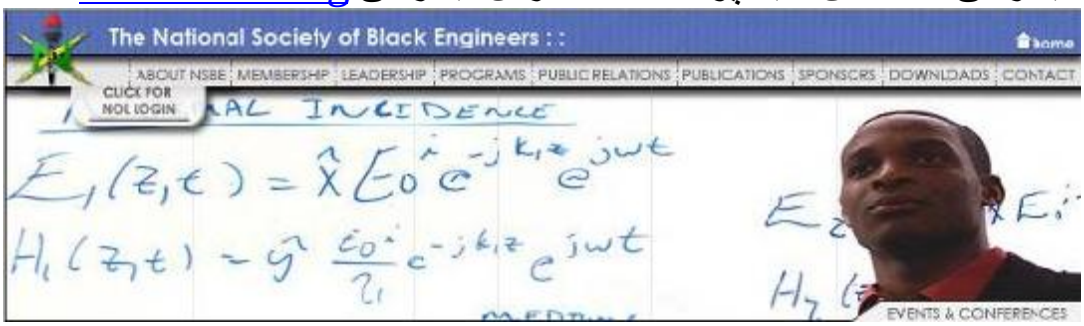
مدرسه ی علوم کامپیوتر و مهندسی ، به نشانی اینترنتی www.cse.unsw.edu.au

دانشگاه سانفرانسیسکو آمریکا ، به نشانی اینترنتی www.cs.usfca.edu

دانشکده ی علوم کامپیوتر و اطلاعات نروژ ، به آدرس اینترنتی
www.idi.ntnu.no

جامعه ی هوش محاسباتی ، به نشانی اینترنتی
http://ebrains.la.asu.edu/~jennie/tutorial/index.htm

پایگاه اینترنتی مهندسان سیاهپوست ، به آدرس اینترنتی www.nsbe.org



دانشگاه ملی تایوان ، به آدرس اینترنتی www.csie.ntu.edu.tw

http://www-formal.stanford.edu (مطالب جان مک کارتی ، استاد دانشگاه استنفورد)

دانشگاه کالیفرنیا ، با آدرس اینترنتی http://www.cs.berkeley.edu (مطالب استوارت راسل و پیتر نورویگ)

دانشکده ی علوم کامپیوتر دانشگاه کورک ایرلند ، با آدرس اینترنتی http://www.cs.ucc.ie

چرا این مجموعه به صورت کتاب چاپ نشد ؟!

متاسفانه چند سال تلاش اینجانب برای چاپ این مجموعه از طریق دانشگاه آزاد اسلامی راه به جایی نبرد ، خود نیز استطاعت مالی برای چاپ این مجموعه را نداشتم ، در ضمن انتشاراتی ها هم حاضر به سرمایه گذاری برای چاپ این مجموعه نشدند . خلاصه به هر دری زدم که این مجموعه را چاپ کنم بسته بود . سرانجام ناچار به انتشار اینترنتی این مجموعه شدم . امید است که این مجموعه مورد استفاده ی دانشجویان و علاقه مندان قرار گیرد و این جانب را از نظرات خود بهره مند نمایند .

با تشکر ، مهندس سهراب جلوه گر کارشناس نرم افزار رایانه
sohrabejelvehgar@hotmail.com
sohjel@yahoo.com
tamrin.sohjel@gmail.com

تابستان ۱۳۸۷

فهرست

فصل اول

هوش مصنوعی

مروری بر هوش مصنوعی [1]

پرسش های بنیادین

سوال : هوش مصنوعی چیست ؟

جواب : هوش مصنوعی دانش و مهندسی ساخت ماشین های هوشمند و مخصوصا برنامه های کامپیوتری هوشمند می باشد . هوش مصنوعی ، وابسته به کار کامپیوترهای مورد استفاده برای فهم هوش انسانی می باشد ، ولی هوش مصنوعی لازم نیست که خودش را به متدهایی که به صورت زیستی [2] قابل مشاهده اند محدود نماید .

سوال : بله ، ولی هوش چیست ؟

جواب : هوش ، بخش محاسباتی توانایی به دست آوردن اهداف در جهان می باشد . انواع و درجه های هوش در افراد و حیوانات و برخی از ماشین ها متنوع است .

سوال : آیا یک تعریف جامع از هوش که به ارتباط آن با هوش بشری وابسته نباشد وجود ندارد ؟

جواب : هنوز نه . مساله این است که ما هنوز نمی دانیم که در حالت کلی چه انواعی از روال های محاسباتی را برای فراخوانی هوش می خواهیم استفاده نماییم . ما برخی از مکانیزم های هوش را می دانیم و نه بیش تر .

سوال : آیا هوش یک چیز منفرد است تا این که کسی بتواند پرشش بله یا خیر "آیا این ماشین هوشمند است یا نه ؟" را بپرسد ؟

جواب : نه ، هوش شامل طرزکارها می باشد و هوش مصنوعی به چگونگی ساخت کامپیوترها و نه چیزهایی بیش تر پی برده است . اگر انجام یک کار فقط نیازمند مکانیزم هایی که امروزه به سادگی قابل فهم هستند می باشد و برنامه های کامپیوتری کارهای خیلی موثر را در این موارد می توانند انجام دهند این جور برنامه ها باید " تاحدودی هوشمند " نامیده شوند .

سوال : آیا هوش مصنوعی در مورد شبیه سازی هوش بشری نمی باشد ؟

جواب : بعضی اوقات ، اما نه همیشه یا حتی معمولا . از یک طرف ما می توانیم چیزهایی را در مورد چگونگی ساخت ماشین های حل مسایل با مشاهده ی دیگر افراد و یا فقط با مشاهده ی رفتارهای خودمان یاد بگیریم . از طرف دیگر ، بیش تر کارها در هوش مصنوعی شامل مطالعه ی مسایل جهان که برای هوش ارایه می شود می باشد و بیش تر از مطالعه ی افراد و یا حیوانات می باشد . محققان هوش مصنوعی برای استفاده از متدهایی (رفتارهایی) که در افراد مشاهده نمی شوند یا شامل محاسبه کننده هایی که به مراتب بیش تر از آنچه که افراد می توانند انجام دهند ، آزاد هستند .

سوال : IQ چیست ؟ آیا برنامه های کامپیوتری دارای IQ می باشند ؟

جواب : نه . IQ براساس نرخی است که هوش در بچه ها توسعه پیدا می نماید . IQ نسبت سنی است که در آن یک بچه به طور مطلوب یک حساب معین را نسبت به سنی که در آن است می تواند انجام دهد . مقیاس برای بزرگسالان به روش مناسب دیگری توسعه داده می شود . IQ به خوبی

[1] Artificial Intelligence
[2] biologically

با اندازه های متفاوت موفقیت یا عدم موفقیت در زندگی ارتباط برقرار می نماید ، اما ساختن کامپیوترهایی که بتوانند امتیاز بالا را در تست های IQ انجام بدهند به طور کمی به سودمندی آن ها بستگی دارد . به عنوان مثال ، توانایی یک بچه برای تکرار به صورت معکوس اعداد متوالی به توانایی های عقلانی وی وابسته می باشد . به هر حال " توالی رقم (ترتیب عدد) " برای حتی کامپیوترهای به شدت محدود ، کم می باشد . به هر حال برخی از مسایل موجود در تست های IQ ایجاد چالش می نمایند .

سوال : در مورد سایر مقایسه های میان انسان و هوش کامپیوتر چه می دانید ؟

جواب : آرتور آر جنسن [1] ، یک محقق خبره (ماهر) در هوش انسانی " یک فرضیه ی ابتکاری " که همه ی انسان ها دارای مکانیزم های عقلانی شبیه هستند و تفاوت ها در هوش وابسته به مکانیزم های کمی و وضعیت های فیزیولوژیکی هستند را پیشنهاد می کند " . من آن ها را به صورت سرعت [2] ، حافظه ی کوتاه مدت [3] ، توانایی به صورت صحیح و حافظه های بازیافتنی طولانی مدت [4] می بینم (تقسیم بندی می کنم) . به هر حال نظر جنسن در مورد هوش انسانی درست است ولی در موقعیت فعلی این نظر در مورد هوش مصنوعی معکوس می باشد . برنامه های کامپیوتری دارای سرعت و حافظه ی زیادی می باشند اما توانایی آن ها برابر است با مکانیزم های عقلانی که طراحان برنامه به خوبی برای قرار دادن در برنامه ها می فهمند . برخی توانایی هایی که بچه ها به صورت نرمال تا نوجوانی آن ها را توسعه نمی دهند و برخی از توانایی هایی که تا دو سالگی توانایی آن را ندارند شاید در آن باشد . ماهیت آن در مورد واقعیتی که علوم شناختی هنوز موفقیتی در تایین آن به صورت دقیق برای بیان این که توانایی انسان چگونه است ، به مراتب پیچیده تر می باشد . به احتمال یقین سازماندهی مکانیزم های عقلانی برای هوش مصنوعی می تواند از آن چه که در مورد افراد می باشد متفاوت باشد . هرگاه که افراد برخی از کارها را بهتر از کامپیوترها انجام می دهند و یا کامپیوتر ها تعداد زیادی محاسبه را برای انجام کار به خوبی انسان ها انجام می دهند ، این نشان می دهد که طراحان برنامه فهم مکانیزم های عقلانی لازم برای انجام کار را به طور مناسب نداشته اند .

سوال : تحقیقات هوش مصنوعی از چه زمانی شروع شد ؟

جواب : بعد از جنگ جهانی دوم تعدادی از افراد به صورت مستقل کار بر روی ماشین های هوشمند را شروع کردند . ریاضیدان انگلیسی ، آلن تورینگ [5] شاید اولین آن ها باشد . وی یک سخنرانی را در مورد هوش مصنوعی در سال ۱۹۴۷ میلادی ارایه نمود . همچنین وی شاید اولین کسی باشد که گفت برنامه نویسی کامپیوترها برای هوش مصنوعی نسبت به ساخت ماشین ها بهتر می باشد . تا اواخر سال ۱۹۵۰ ، تعداد زیادی محقق در مورد هوش مصنوعی وجود داشت و بیش تر آن ها اساس کار خود را برمبنای برنامه نویسی کامپیوترها گذاشته بودند .

سوال : آیا هدف هوش مصنوعی این است که فکر بشری را در کامپیوتر قرار دهد ؟

جواب : برخی از محققان گفتند که آن ها دارای چنین هدفی می باشند ، اما شاید آن ها می خواهند که از تعبیر مجازی استفاده نمایند . فکر بشری دارای غرابت زیادی می باشد و من فکر نمی کنم که هر کسی به طور کامل بتواند همه ی آن ها را تقلید نماید .

سوال : آیا اهداف هوش مصنوعی در سطح هوش انسانی می باشد ؟

جواب : بله . نهایت تلاش این است که برنامه های کامپیوتری که می توانند مسایل را حل کنند و به اهداف دسترسی پیدا کنند را در جهان ، به خوبی انسان ها بسازند .

سوال : هوش مصنوعی تا چه حدی به سطح هوش بشری رسیده است ؟

---

[1] Arthur R. Jensen
[2] speed
[3] short term memory
[4] retrievable long term memory
[5] Alan Turing

جواب : تعداد کمی از افراد فکر می کنند که سطح هوش بشری با نوشتن برنامه های طولانی که هم اکنون در حال نوشتن و سرهم بندی پایگاه های دانش [1] واقعیات ، با استفاده از زبان هایی که اکنون برای بیان دانش استفاده می شوند ، قابل دسترسی می باشند . به هر حال ، بیش تر محققان هوش مصنوعی تصور می کنند که ایده های بنیادین جدید مورد نیازند ، و بنابراین قابل پیش بینی نیست که سطح هوش انسانی قابل دسترسی باشد .

سوال : آیا کامپیوترها نوع درستی از ماشین ها برای پیاده سازی هوش می باشند ؟

جواب : کامپیوترها برای شبیه سازی هر نوع از ماشین ها می توانند برنامه ریزی شوند . بسیاری از محققان ماشین های غیر کامپیوتری را طراحی کردند ، امیدوار بودند که آن ها به طرق مختلف از برنامه های کامپیوتری بهتر باشند . به هر حال ، آن محققان معمولا ماشین های اختراع شده اشان را در یک کامپیوتر شبیه سازی می کنند و شک می کنند که ماشین جدید ارزش ساختن را داشته باشد . زیرا بیلیون ها دلار برای ساخت کامپیوترهای سریع و سریع تر باید پرداخته شود . نوع دیگری از ماشین باید خیلی سریع باشند تا بتوانند یک برنامه را در یک کامپیوتر شبیه سازی کننده ی ماشین اجرا نمایند .

سوال : آیا کامپیوتر ها به اندازه ی کافی برای هوش سریع می باشند ؟

جواب : برخی از افراد فکر می کنند که کامپیوترهای به مراتب سریع تر برای ایده های جدید لازم هستند . نظر شخصی من این است که کامپیوترهای سی سال پیش هم در صورتی که ما بدانیم آن ها را چگونه برنامه ریزی نماییم به اندازه ی کافی سریع می باشند . البته به طور مجزا از بلند پروازی های محققان هوش مصنوعی ، کامپیوتر ها روند سریع تر شدن را دنبال می کنند .

ماشین های موازی چیستند ؟ **جواب** : ماشین های با چند پردازشگر [2] به مراتب سریع تر از ماشین های با یک پردازشگر می باشند . موازات [3] خودش هیچ سودی ندارد و ماشین های موازی تا حدودی برای برنامه نامناسب می باشند . در موقعی که سرعت بالا مورد نیاز می باشد ، باید این نامناسب بودن را بپذیریم .

سوال : فرزند ماشيني [4] که می تواند با خواندن و آموزش از راه آزمایش بهبود داشته باشد چیست ؟

جواب : این فکر در بسیاری از مواقع پیشنهاد شده است و شروع آن در سال ۱۹٤۰ بوده است . سرانجام برای استفاده ساخته شد . اما برنامه های هوش مصنوعی هنوز به آن حد نرسیده اند که قادر باشند به اندازه ای که یک بچه از تجربه ی محیط یاد می گیرد را یاد بگیرند .

سوال : آیا یک سیستم هوش مصنوعی می تواند خودش را به سطح بالاتر و بالاتر هوش ، با فکر کردن در مورد هوش مصنوعی ارتقا دهد ؟

جواب : بله ، من این طور فکر می کنم . اما ، ما هم اکنون در مرحله ای از هوش مصنوعی نمی باشیم که این پردازه بتواند شروع شود .

سوال : شطرنج [5] چیست ؟

جواب : برنامه های بازي شطرنج در حال حاضر در سطح استادی اجرا می شوند ، اما آن ها این کار را با مکانیزم های محدود شده در مقایسه با آن چه که به وسیله ی انسان ها ی اجراکننده ی شطرنج انجام می شود به جای انجام تعداد زیادی محاسبات برای فهم ، انجام می دهند . ما این مکانیزم ها را بهتر می فهمیم و می توانیم برنامه های شطرنج در سطح انسانی را بسازیم که محاسبه های کم تری را نسبت به برنامه های فعلی انجام می دهند . متاسفانه رقابت و اهداف تجاری ساخت کامپیوترها از استفاده از شطرنج به عنوان یک دامنه ی علمی جلوگیری کرده است .

سوال : Go چیست ؟

جواب : بازی چینی و ژاپنی Go بازی ای روی مقوا (کارتی) است که در آن بازی کننده ها

[1] knowledge base
[2] processor
[3] parallelism
[4] child machine
[5] chess

به صورت دایره ای حرکت می کنند . Go ضعف فهم مکانیزم های عقلانی ما را در بازی کردن نمایان می سازد . برنامه های Go علی رغم تلاش بسیار زیا د آن ها ، برای بازیگران خیلی بد می باشند (نه به اندازه ای که برای شطرنج وجود داشت) . به نظر می رسد که مساله این باشد که یک وضعیت در بازی Go باید به مجموعه ای از زیر وضعیت ها که اول به صورت مجزا به دنبال یک تحلیل از اثرات متقابل آن هایی که باید تحلیل شوند ، تبدیل شوند . افراد این مطلب را در شطرنج نیز استفاده می نمایند ، اما برنامه های شطرنج به وضعیت ها به صورت یک کل رسیدگی می کنند . برنامه های شطرنج این کمبود مکانیزم عقلانی را با انجام هزاران یا چند میلیون محاسبه در لحظه در مورد Deep Blue (نام ابررایانه ای ساخت شرکت IBM می باشد که در مقابل قهرمان شطرنج دنیا ، گری کاسپاروف [1] بازی کرد [2]) جبران می کنند . دیر یا زود ، تحقیقات هوش مصنوعی براین ضعف چیره خواهند شد .

سوال : آیا برخی از افراد نمی گویند که هوش مصنوعی ایده ی بدی می باشد ؟

جواب : فیلسوفی به نام جان سرل [3] می گوید که ایده ی ماشینی غیر بیولوژیکی که می تواند هوشمند شود ، ایده ای نقض کننده می باشد . وی استدلال اطاق چینی [4] را پیشنهاد می کند [5] . فیلسوفی به نام هیوبرت دریفس [6] می گوید که هوش مصنوعی غیرممکن می باشد . دانشمند علوم کامپیوتری به نام جزف ویزنباوم [7] می گوید که ایده ی هوش مصنوعی ضد بشری و غیراخلاقی می باشد . برخی دیگر از افراد می گویند که چون که هوش مصنوعی تا کنون به اهداف خود نرسیده است پس غیر ممکن می باشد . سایر افرا د از این که می بینند کمپانی های سرمایه گذاری کننده ورشکست می شوند مایوس می شوند .

استدلال اطاق چینی : یک گمان آزمایشی طراحی شده توسط جان سرل [8] برای کم ارزش کردن ادعاهای مطرح شده توسط هوش مصنوعی قوی [9] و همچنین کارکردگرایی [10] می باشد . [11]

هوش مصنوعی قوی : دیدی که می گوید مغز بشر یک ابزار محاسباتی می باشد و کامپیوترها به طور کلی قادرند که فکر کنند [12] . به عنوان تعریفی دیگر ، هوش مصنوعی قوی یک شکل فرضی از هوش مصنوعی است که می تواند به درستی استدلال نماید و مسایل را حل کند ؛ هوش مصنوعی قوی می تواند درک نماید یا خودآگاه [13] باشد اما ممکن است پردازش های فکری شبیه بشر را داشته باشد یا نداشته باشد . [14]

سوال : آیا تیوری تجزیه پذیری [15] و پیچیدگی محاسباتی [16] کلیدهایی برای هوش مصنوعی نمی باشند ؟ [توجه کنید که افراد عامی و مبتدیان در علم کامپیوتر نمی توانند پاسخگوی این سوالات باشند ، این ها کاملا شاخه های منطقی ریاضی و علم کامپیوتر می باشند و جواب این سوال باید تا حدودی تکنیکی باشد .]

جواب : نه . این تیوری ها وابسته می باشند ولی مسایل اساسی هوش مصنوعی را جوابگو نمی باشند . در سال ۱۹۳۰ منطقدانان ریاضیات و مخصوصا کورت گودل [17] و آلن تورینگ ثابت کردند که الگوریتم هایی برای ضمانت این که همه ی مسایل را در دامنه های مهم ریاضی بتوانند حل کنند وجود ندارد . جمله ای از منطق مرتبه ی اول [18] یک مثال می باشد و

---

[1] Garry Kasparov
[2] مترجم
[3] John Searle
[4] Chinese room argument
[5] http://www-formal.stanford.edu/jmc/chinese.html
[6] Hubert Dreyfus
[7] Joseph Weizenbaum
[8] John Searle (1980)
[9] Strong AI
[10] functionalism
[11] en.wikipedia.org/wiki/Chinese_room_argument
[12] www.ucd.ie/philosop/documents/2.%20definitions%20of%20some%20key%20terms.htm
[13] self-aware
[14] en.wikipedia.org/wiki/Strong_AI
[15] computability theory
[16] computational complexity
[17] Kurt Godel
[18] First Order Logoc(FOL)

یک معادله ی چند جمله ای با چند متغیر راه حل صحیح دیگری می باشد . انسان ها همه ی مسایل را همیشه در این زمینه ها حل کرده اند و آن را به صورت یک استدلال مطرح کرده اند (معمولا با برخی اضافات) که کامپیوترها ذاتا قادر به انجام کارهایی که افراد می توانند انجام دهند نمی باشند ، راجر پنروز [1] این مطلب را ادعا می کند . به هر حال ، افراد حل مسایل دلخواه را در این موارد یا هر کدام از آن ها نمی توانند ضمانت کنند . در دهه ی ۱۹۶۰ دانشمندان علوم کامپیوتر و مخصوصا استیو کوک [2] و ریچارد کارپ [3] تیوری مسایل با دامنه ی **NP – کامل** [4] را توسعه دادند . مسایل در این دامنه ها قابل حل می باشند ، اما به نظر می رسد که زمان برای حل این مسایل با افزایش اندازه به صورت نمایی افزایش یابد . جملات حساب گزاره ای یک مثال پایه ای برای مسایل با دامنه ی NP – کامل می باشند . انسان ها اغلب مسایل را در مورد مسایل NP – کامل در زمان هایی به مراتب کوتاه تر از آن چه که توسط الگوریتم های عمومی ضمانت می شوند حل می کنند ، اما در حالت کلی نمی توانند این مسایل را به سرعت حل کنند . چه چیزی برای هوش مصنوعی مهم است که الگوریتم هایی به اندازه ی افراد توانمند ، برای حل مسایل باید داشته باشد . شناسایی برای این که کدام زیر دامنه ها برای الگوریتم ها وجود دارد مهم است ، اما تعداد زیادی از حل کنندگان مسایل هوش مصنوعی به آسانی با زیر دامنه ها سازگار نمی باشند . تیوری پیچیدگی کلاس های عمومی ، مسایل پیچیدگی محاسباتی نامیده می شود . تا کنون این تیوری با هوش مصنوعی به اندازه ای که انتظار می رفته است همکاری (تعامل) نداشته است . موفقیت در مسایل حل شده توسط بشر و توسط برنامه های هوش مصنوعی به نظر می رسد که وابسته به خصوصیات مسایل و متدهای حل مساله ای که نه توسط محققان و نه توسط اجتماع هوش مصنوعی به دقت قابل تعریف باشند وابسته باشد . تیوری پیچیدگی الگوریتمی که به نحوی توسط سولومونوف [5] ، کولموگوروف [6] و چایتین [7] (به طور مجزا از یکدیگر) توسعه داده شد نیز وابسته می باشند . پیچیدگی شی ء سمبولیک را به وسیله ی کوتاه ترین برنامه ای که آن را تولید می کند تعریف کرد . اثبات این که برنامه ی کاندید ، کوتاه ترین یا نزدیک به کوتاه ترین است یک مساله ی غیر قابل حل می باشد ، حتی زمانی که شما نمی توانید ثابت کنید که برنامه کوتاه ترین می باشد باید اشیاء ارایه شده توسط برنامه های کوچک که آن ها تولید می کنند روشن شود .

مسایل NP- کامل - در تیوری پیچیدگی ، مسایل NP- کامل ، سخت ترین مسایل در NP هستند ، در صورتی که شما بتوانید یک راه برای حل یک مساله ی NP- کامل به سرعت پیدا نمایید ، آن گاه شما می توانید این الگوریتم را برای حل تمام مسایل NP به سرعت استفاده نمایید .

شاخه های هوش مصنوعی

سوال : شاخه های هوش مصنوعی چیستند ؟

جواب : در زیر یک لیست از شاخه های هوش مصنوعی آمده است ، اما به یقین برخی از شاخه ها وجود ندارند ، زیرا هنوز کسی آن ها را نشناخته است . برخی از این ها ممکن است به صورت موضوعات یا افکاری بیش از شاخه های کامل ملاحظه شوند .

هوش مصنوعی منطقی [8] - این که یک برنامه در حالت کلی ، واقعیات وضعیت معینی که در آن باید عمل کند را می داند و اهدافی که توسط عبارات بعضا با زبان منطقی ریاضی ارایه می شوند را می داند . برنامه در مورد عملی که باید انجام دهد با استنتاج عملکردهای معین که برای به دست آوردن اهداف ، مناسب هستند تصمیم گیری می نماید . اولین مقاله ی پیشنهاد شده در این مورد

[1] Roger Penrose
[2] Steve Cook
[3] Richard Karp
[4] NP-complete
[5] Solomonoff
[6] Kolmogorov
[7] Chaitin
[8] logical AI

McC59 بود . McC89 جدیدتر می باشد . McC96b برخی از افکار در برگیرنده ی هوش مصنوعی منطقی را لیست می کند . Sha97 هم یک متن مهم است .

جستجو [1] - برنامه های هوش مصنوعی معمولا تعداد زیادی از احتمالات را بررسی می کنند ، به عنوان مثال حرکت های درون یک بازی شطرنج یا استنتاج های یک قضیه ی اثبات کننده ی برنامه .

شناخت الگو [2] - در زمانی که یک برنامه ، ملاحظات برخی از نوع ها را می سازد ، اغلب برای مقایسه چیزی که با یک الگو می بیند ، برنامه ریزی شده است . برای مثال ، یک برنامه ی بینایی ممکن است یک الگو ی چشم و یک الگوی بینی را در یک منظره برای پیداکردن یک صورت تطابق دهد . الگوهای پیچیده تر ، مانند یک زبان متنی طبیعی ، در یک وضعیت شطرنج ، یا در تاریخچه ی برخی از رویدادهایی که مطالعه می شوند وجود دارند . این الگوهای پیچیده تر به متدهای کاملا مختلفی نسبت به الگوهای ساده ای که بیش تر مطالعه شده اند نیاز دارند .

نمایش [3] - واقعیاتی در مورد دنیایی که باید به طریقی نمایش داده شوند می باشند . معمولا از زبان های منطقی ریاضی استفاده می شود .

استنتاج [4] - از برخی از واقعیات ، دیگر واقعیات می تواند استنتاج شود . استنتاج منطقی ریاضی برای برخی از اهداف ، کافی می باشد ، اما متدهای استنتاج جدید غیر یکنواخت از سال ۱۹۷۰ به منطق اضافه شده اند . ساده ترین نوع استدلال غیر یکنواخت [5] ، استدلال پیش فرض است که در آن یک نتیجه گیری به صورت پیش فرض استنتاج می شود ، اما نتیجه گیری می تواند در صورتی که مدرک (دلیل) عوض شود ، عوض بشود . برای مثال ، زمانی که ما از یک پرنده حرف می زنیم ، ما نتیجه می گیریم که می تواند پرواز کند ، اما این استنتاج زمانی که ما در مورد پنگوین حرف می زنیم می تواند عوض شود . احتمال این وجود دارد که یک نتیجه گیری که خصوصیات غیر یکنواخت را از استدلال تولید می کند عوض شود . معمولا استدلال منطقی در مجموعه ای از استنتاج ها که می توانند از یک مجموعه مقدمات گرفته شوند ، یک تابع افزایشی یکنواخت از مقدمات می باشد .

دانش عقل سلیم و استدلال [6] - محیطی است که در آن هوش مصنوعی از سطح انسانی خیلی دور می باشد ، علی رغم این واقعیت که از سال ۱۹۵۰ روی آن تحقیق شده است . به عنوان مثال ، در توسعه ی سیستم های استدلال غیر یکنواخت و تیوری های عملکرد ، هنوز ایده های بیش تری مورد نیاز می باشد . سیستم Cyc شامل مجموعه ای بزرگ از واقعیات عقل سلیم می باشد .

یادگیری از تجربه [7] - برنامه ها این کار را انجام می دهند . براساس پیوندگرایی [8] و شبکه های عصبی [9] به هوش مصنوعی نزدیک می شوند و در هوش مصنوعی تخصصی می شوند . در یادگیری از تجربه ، همچنین آموزش قوانین بیان شده در منطق وجود دارد . Mit97 یک متن آموزشی جامع در مورد آموزش ماشینی [10] می باشد . برنامه ها فقط می توانند واقعیات یا رفتاری را یاد بگیرند که بتوانند آن ها را نمایش دهند و متاسفانه سیستم های آموزشی تقریبا همه براساس توانایی های خیلی محدود شده برای ارایه اطلاعات می باشند .

برنامه ریزی [11] - برنامه ریزی برنامه ها با واقعیت های عمومی در مورد جهان (مخصوصا واقعیاتی در مورد اثرات عملکردها) شروع می شوند ، واقعیات ، در مورد وضعیت های مشخص و یک عبارت در مورد هدف می باشند . از آن ها یک استراتژی برای رسیدن به هدف به دست می آید . در بسیاری از موارد

[1] search
[2] pattern recognition
[3] representation
[4] inference
[5] non-monotonic
[6] common sense knowledge and reasoning
[7] learning from experience
[8] connectionism
[9] neural nets
[10] machine learning
[11] planning

استراتژی فقط یک ترتیب از عملکردها می باشد .

شناخت شناسی [1] - یک مطالعه بر روی انواع دانش که برای حل مسایل در جهان لازم می شوند می باشد .

هستی شناسی [2] - هستی شناسی ، مطالعه ی انواع چیزهایی است که موجودند . در برنامه ها و عبارات به انواع مختلفی از اشیا توجه می شود و ما این انواع و خصوصیات اساسی آن ها را مطالعه می کنیم . تاکید بر هستی شناسی از سال ۱۹۹۰ شروع شد .

اکتشافات [3] - یک اکتشاف ، راهی برای پی بردن به چیزی یا یک ایده ی جاسازی شده در یک برنامه می باشد . این واژه به صورت متنوع در هوش مصنوعی به کار می رود . از توابع مکاشفه ای [4] در برخی از مواقع در جستجو برای اندازه گیری این که برای یک گره در یک درخت جستجو برای رسیدن به اهداف چه مسافتی طی می شود ، استفاده می شود . مستندات اکتشاف [5] دو گره را در یک درخت جستجو برای دیدن این که کدام بهتر از دیگری است مقایسه می کند ، عقیده ی من این است که تشکیل دادن یک مقدمه ی پیشرفته برای رسیدن به هدف ، شاید مفیدتر باشد .

برنامه نویسی ژنتیک [6] - برنامه نویسی ژنتیک یک تکنیک برای به وجود آوردن برنامه هایی برای حل یک مساله با جفت برنامه های تصادفی Lisp و انتخاب مناسب ترین تولیدات از میلیون ها تولید می باشد . برنامه نویسی ژنتیک توسط گروه جان کوزا [7] توسعه داده شد .

کاربردهای هوش مصنوعی

سوال : کاربردهای هوش مصنوعی چیست ؟

جواب : برخی از کاربردها در این جا آمده است .

تیوری بازی ها [8] - با پرداخت چند دلار شما می توانید ماشین هایی را بخرید که می توانند به صورت حرفه ای بازی شطرنج را پیاده سازی کنند . در آن ها برخی از موارد هوش مصنوعی وجود دارد ، اما آن ها به خوبی افراد بازی می کنند و برخلاف افراد هستند که بیش تر با محاسبات جبری بی فکر بازی می کنند و به صدها هزار از وضعیت ها نگاه می کنند .

سخن شناسی [9] - در دهه ی ۱۹۹۰ ، سخن شناسی کامپیوتری به سطحی کاربردی برای اهدافی محدود رسید . بنابراین خطوط هوایی ایالات متحده صفحه کلید درختی را که برای اطلاعات پرواز استفاده می شد را با سیستمی که از سخن شناسی استفاده می کرد ، برای شماره های پرواز و نام شهرها جایگزین کردند ؛ این روش کاملا مناسب بود .

فهم زبان طبیعی [10] - فقط گرفتن کلمات متوالی در یک کامپیوتر کافی نمی باشد . فقط تجزیه ی جملات هم کافی نمی باشد . کامپیوتر باید بفهمد که متن در چه موردی می باشد و این مورد به زودی برای دامنه های خیلی محدود امکان پذیر می باشد .

هدف کامپیوتر [11] - جهان از اشیایی سه بعدی تشکیل شده است اما ورودی ها برای چشم بشر و دوربین های تلویزیون کامپیوتری دو بعدی می باشند . برخی از برنامه ها فقط در حالت دوبعدی می توانند کار کنند ، اما دید کامپیوتری کامل ، اطلاعات سه بعدی ناتمام که فقط مجموعه ای از دیدهای دو بعدی نیستند را لازم دارد . در حال حاضر فقط راه های محدودی برای نمایش اطلاعات سه بعدی به صورت مستقیم وجود دارد و این راه ها از قرار معلوم به خوبی آن چه انسان ها استفاده می کنند نمی باشد .

---

[1] epistemology
[2] ontology
[3] heuristics
[4] heuristic functions
[5] heuristic predicates
[6] genetic programming
[7] John Koza

[8] game playing
[9] speech recognition
[10] understanding natural language
[11] computer vision

سیستم های خبره [1] - " مهندسی دانش [2] " در یک زمینه ی معین با خبرگان گفتگو می کند و تلاش می کند که به دانش آن ها در یک برنامه ی کامپیوتری برای اجرای برخی از کارها جسم بدهد . چگونگی انجام این کارها به صورت خوب ، وابسته به این است که مکانیسم های عقلی لازم برای کار در وضعیت فعلی هوش مصنوعی وجود داشته باشند . زمان این کار زیاد نمی باشد و نتایجی مایوس کننده وجود دارد . یکی از اولین سیستم های خبره در سال ۱۹۷٤ به نام MYCIN بود ، که اثرات باکتری های موجود در خون و معالجات آن را تشخیص می داد . این دستگاه این کار را بهتر از دانشجویان پزشکی یا دکترها انجام می داد . یعنی هستی شناسی آن شامل باکتری ، نشان ها (علامت ها) و معالجات بود و کارها را به موقع انجام می داد . از زمانی که خبرگان با مهندسان همکاری کردند چیزهایی را در مورد بیماران ، دکترها ، سلامتی ، بهبود و ... دانستند و واضح است که دانش مهندسان تحت تاثیر آن چه که خبرگان به آن ها می گفتند در یک چارچوب کاری معین قرار داشت . در وضعیت فعلی هوش مصنوعی ، این مطلب باید درست باشد . مزیت سیستم های خبره ی امروزی وابسته به کاربران آن ها می باشد که دارای عقل سلیم می باشند . به عنوان تعریفی دیگر از سیستم های خبره ، یک سیستم خبره یک دسته از برنامه های کامپیوتری توسعه داده شده توسط محققان هوش مصنوعی در دهه ی ۱۹۷۰ و به کار گرفته شده به صورت تجاری در سراسر دهه ی ۱۹۸۰ است . در واقع ، برنامه هایی ساخته شده از قوانینی که اطلاعات (که معمولا توسط کاربر سیستم به کار گرفته می شوند) را تحلیل می کنند در مورد یک دسته ی مشخص از مسایل هستند ، به خوبی مسایل را تحلیل می نمایند و وابسته به طراحی شان ، یک زمینه ی توصیه شده از عملکرد کاربر برای به کارگیری تصحیحات می باشند . [3]

طبقه بندی اکتشافی [4] - یکی از بیش ترین انواع عملی (امکان پذیر) سیستم ها ی خبره ی ارایه شده توسط هوش مصنوعی برای قراردادن برخی از اطلاعات در یک مجموعه ی طبقه بندی شده ی معین با استفاده از چند منبع معین می باشد . مثالی در این مورد نصیحت کردن کسی برای دریافت کارت اعتباری پیشنهاد شده ، ثبت پرداخت او ، همچنین چیزی که او می خرد و ثبت موسسه ای که او از آن خرید می نماید ، می باشد .

پرسش های بیش تر

سوال : تحقیق هوش مصنوعی چگونه انجام می شود ؟

جواب : تحقیق هوش مصنوعی دارای دو زمینه ی تیوری [5] و آزمایشی [6] می باشد . زمینه ی آزمایشی دارای هر دو صورت اساسی و کاربردی می باشد . دو رشته ی اساسی در تحقیق وجود دارد ، یکی بیولوژیکی است و براساس این است که انسان ها هوشمند هستند و هوش مصنوعی باید انسان ها ، روان شناسی یا فیزیولوژی آن ها را تقلید کند . رشته ی دیگر ، حادثه ای [7] می باشد و براساس مطالعه و فرمول بندی واقعیات حواس مشترک در مورد جهان و مسایلی که در جهان برای دستیابی به اهداف استفاده می شود ، می باشد . اثرات متقابل این دو ، تا حدی به هم نزدیک می باشد و هر دو سرانجام باید به موفقیت دستیابند . این یک مسابقه می باشد .

سوال : ارتباط میان هوش مصنوعی و فلسفه چیست ؟

جواب : هوش مصنوعی دارای ارتباطات زیادی با فلسفه می باشد ، مخصوصا با فلسفه ی تحلیلی مدرن [8] ارتباط زیادی دارد . هر دو فکر و عقل سلیم را مطالعه می کنند .

---

1 expert systems
2 knowledge engineer
3 en.wikipedia.org/wiki/Expert_systems

4 heuristic classification
5 theoretical
6 experimental
7 phenomenal
8 modern analytic philosophy

سوال : قبل از یادگیری هوش مصنوعی و یا در موقع مطالعه ی آن ، چه چیزهایی را باید مطالعه کنیم ؟

جواب : باید ریاضیات و مخصوصا منطق ریاضیات [1] را مطالعه کنیم . برای نزدیکی زیستی به هوش مصنوعی ، روان شناسی و فیزیولوژی سیستم های عصبی را مطالعه می کنیم . تعدادی زبان برنامه نویسی و در کم ترین حالت ، سی [2] ، لیسپ [3] و پرولوگ [4] را باید بلد باشیم . همچنین فکر خوبی است که یکی از زبان های پایه ای ماشین را یاد بگیریم . کارها بیش تر با زبان هایی که مد هستند انجام می شوند . در اواخر دهه ی ۹۰ این زبان ها شامل C++ و جاوا [5] بود .

سوال : چه سازمان ها و موسساتی در زمینه ی هوش مصنوعی فعالیت می کنند ؟

جواب : انجمن آمریکایی هوش مصنوعی [6] ، کمیته ی هماهنگی اروپا برای هوش مصنوعی [7] و جامعه ی هوش مصنوعی و شبیه سازی رفتار [8] جوامع علمی علاقه مند به هوش مصنوعی می باشند . شرکت ماشین آلات محاسبه کننده [9] یا SIGART یک گروه مهم در زمینه ی هوش مصنوعی می باشد . کنفرانس بین المللی هوش مصنوعی [10] . تعاملات الکترونیک با هوش مصنوعی [11] و هوش مصنوعی [12] و روزنامه ی تحقیقات هوش مصنوعی [13] و تعاملات IEEE با تحلیل الگو و ماشین های هوشمند [14] چهار روزنامه ی مهم منتشرکننده ی مقالات هوش مصنوعی می باشند . در حال حاضر چیز دیگری که مناسب درج در این قسمت باشد را پیدا نکرده ایم .

[1] mathematical logic
[2] C
[3] Lisp
[4] Prolog
[5] Java
[6] The American Association for Artificial Intelligence (AAAI) با آدرس اینترنتی http://www.aaai.org
[7] European Coordinating Committee for Artificial Intelligence (ECCAI) به آدرس اینترنتی http://eccai.org
[8] Society for Artificial Intelligence and Simulation of Behavior (AISB) با آدرس اینترنتی http://www.cogs.susx.ac.uk/aisb
[9] Association for Computing Machinery (ACM) به آدرس اینترنتی http://www.acm.org/sigart
[10] The International Joint Conference on AI (IJCAI) به آدرس اینترنتی http://www.ijcai.org
[11] *Electronic Transactions on Artificial Intelligence* با آدرس اینترنتی http://www.ida.liu.se/ext/etai
[12] به آدرس اینترنتی http://www.elsevier.nl/locate/artint
[13] *Journal of Artificial Intelligence Research* به آدرس اینترنتی http://www.jair.org
[14] IEEE Transactions on Pattern Analysis and Machine Intelligence به آدرس اینترنتی http://computer.org/tpami

هوش مصنوعی

ریوس مطالب
هوش مصنوعی چیست ؟
تاریخچه ی مختصر
چگونگی فن

چرا هوش مصنوعی را مطالعه می کنیم ؟

- کنجکاوی علمی
  - تلاش برای درک موجودیت هایی که هوش را ارایه می نمایند
- چالش های مهندسی
  - ساخت سیستم هایی که هوش را ارایه می کنند
- برخی از کارها که به نظر می رسد برای اجرا می توان از هوش کمک گرفت
  - مثل بازی شطرنج
- پیشرفت در عملکرد کامپیوتر و روش های محاسبه ، حل مسایل پیچیده را توسط کامپیوتر ممکن می سازد
- شاید انسان ها از کارهای ملال آور و خطرناک خلاص شوند
  - مثل خنثی کردن مین یا تمیز کردن استخر شنا

هوش مصنوعی چیست؟

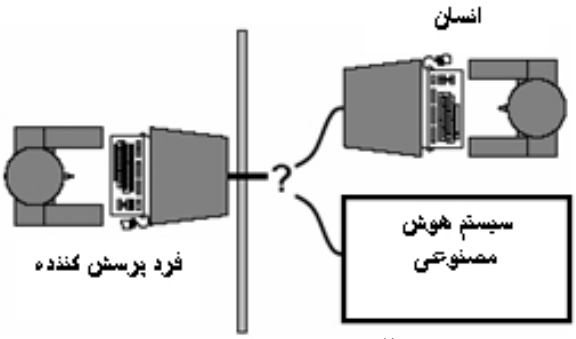
سیستم هایی که همانند انسان ها فکر می کنند.	**سیستم هایی که معقولانه (rationally) فکر می کنند.**
سیستم هایی که شبیه انسان ها عمل می کنند.	**سیستم هایی که معقولانه عمل می کنند.**

عمل کردن مثل انسان : آزمایش تورینگ [1] - آلن تورینگ در سال ۱۹۵۰ در مقاله ی ماشین آلات محاسبه و هوش در مورد شرایطی برای ماشین هوشمند بحث کرده است . او گفته است که اگر ماشین بتواند کاملا وانمود کند که مانند بشر می باشد آن گاه شما می توانید مطمین باشید که هوشمند است . این آزمایش توانست برخی از افراد و نه همه را راضی نماید . فرد پرسش کننده می تواند با ماشین کار کند و یک فرد (در طرف مقابل) با ماشین تایپ کند (برای جلوگیری از

[1] Turing test

احتیاج ماشین به تقلید ظاهر یا صدای شخص) و فرد باید پرسش کننده را دارای این عقیده نماید که فرد می باشد و ماشین باید تلاش کند که فرد پرسش کننده را گمراه نماید [و خود را به جای فرد جا بزند] . آزمایش تورینگ یک جانبه (غیرمنصفانه) می باشد . یک ماشین که آزمایش را می گذراند باید کاملا هوشمند باشد ، ولی یک ماشین ، برای تقلید ، بدون دانستن به اندازه ی کافی در مورد افراد می تواند هوشمند باشد . کتاب مغز کودکان [1] دانیل دنت [2] دارای یک بحث عالی در مورد آزمایش تورینگ و جوانب مختلف آزمایش تورینگ که به کار گرفته می شوند می باشد . برخی از افراد به سادگی تصور می کنند که یک برنامه ی نسبتا گنگ هوشمند می باشد . [3]

اجزای اصلی پیشنهاد شده برای هوش مصنوعی عبارتند از : دانش ، استدلال ، زبان و آموزش . اشکالات آزمایش تورینگ این بود که : آزمایش تورینگ تجدید پذیر [4] ، سودمند [5] یا جوابگو [6] برای تحلیل ریاضی نیست.



فکرکردن مانند انسان : علم شناخت [7]

- برای تولید تیوری های چگونگی عملکرد مغز بشر تلاش می کند
- از مدل های کامپیوتری هوش مصنوعی و روش های تجربی روانشناسی استفاده می نماید
- بیش تر روش های هوش مصنوعی به طور مستقیم براساس مدل های شناختی نمی باشند
  - اغلب دشوار است که به صورت برنامه های کامپیوتری ترجمه شوند
  - مسایل عملکرد
- علم شناخت اساسا از هوش مصنوعی مجزا می باشد

در دهه ی ۱۹۶۰ انقلاب شناخت به وجود آمد : علم شناخت دارای اشتراک با هوش مصنوعی می باشد . دانشمندان علم شناخت طبیعت هوش را با یک دید روانی مطالعه می کنند و بیش تر مدل های کامپیوتری که به توضیح رخدادهای درون مغز ما در مدت حل مساله ، به یادآوردن ، درک و دیگر فعالیت های روانشناسی کمک می کنند را می سازند . یک اشتراک اساسی میان هوش مصنوعی و علم شناخت برای روانشناسی این بوده است که در مدل پردازش

---

[1] Brainchildren
[2] Daniel Dennett
[3] http://www-formal.stanford.edu/jmc/whatisai/node1.html
[4] reproducible
[5] constructive
[6] amenable
[7] cognitive science

اطلاعات فکری انسان ، تشبیه " ***مغز به صورت کامپیوتر*** " به صورت لفظ به لفظ کاملا انجام می شود . [1]

هربرت سیمون (۱۹۱۶-۲۰۰۱) در مورد علم شناخت می گوید : " هوش مصنوعی می تواند دو هدف داشته باشد . یکی استفاده از توانایی کامپیوترها برای تکمیل فکر بشری ، همان طوری که ما از موتورها برای تکمیل توانایی بشر یا اسب استفاده می کنیم ، علم روبوتیک و سیستم های خبره شاخه های اصلی این هدف اول می باشند . هدف دیگر ، استفاده از هوش مصنوعی کامپیوتری برای فهمیدن چگونگی فکر بشر است . اگر شما برنامه ها را با چیزی که می توانند انجام دهند تست نکنید و بدون فهمیدن چگونگی انجام آن کارها تست کنید ، آن گاه آن شمایید که واقعا علم شناخت را انجام می دهید ؛ شما در حال استفاده از هوش مصنوعی برای فهمیدن عملکرد بشری هستید . " [2]

تیوری های علمی فعالیت های داخلی فکر موارد زیر را لازم دارد :

- چه سطحی از تجرد [3] وجود دارد ؟ " دانش " یا " جریان ها [4] "
- تایید اعتبار [5] چگونه است ؟ مستلزم : ۱- پیش بینی و امتحان کردن موضوعات رفتار انسانی (بالا به پایین) یا ، ۲- شناخت مستقیم داده های عصب شناختی [6] (پایین به بالا) می باشد . هر دو (علم شناخت و علم عصب شناسی [7] در حال حاضر از هوش مصنوعی مجزا هستند. هر دو دارای این ویژگی مشترک با هوش مصنوعی هستند: **تیوری های در دسترس ، هیچ چیزی را شبیه هوش انسان تولید نمی کنند.** بنابراین ، هر سه زمینه در یک جهت اساسی مشترک هستند!

فکر منطقی (معقولانه) : قوانین فکر - قواعد اصولی [8] بیش از تشریحی حاکم می باشند ، چند مدرسه ی یونانی شکل های مختلف منطقی را توسعه دادند: نمادها [9] و قوانین استنتاج [10] برای تفکرات ، برای فکر مکانیزاسیون ممکن است به کار روند یا ممکن است به کار نروند.

مسایل :

- همه ی رفتارهای هوش با بررسی های منطقی انجام نمی شوند.
- محدودیت منابع : تفاوتی میان حل یک مساله در کل و حل آن در عمل تحت محدودیت منابع گوناگون نظیر زمان ، محاسبه و دقت وجود دارد .

عمل کردن به صورت منطقی (معقولانه)

رفتار منطقی [11] : کار درست را انجام می دهد. **کار درست** ، کاری است که برای بیش ترین دستیابی به هدف ، پیش بینی می شود و اطلاعات قابل دسترسی را ارایه می کند. حتما لازم نیست فکرکردن را شامل شود ، اما فکر کردن باید در انجام کار معقولانه وجود داشته باشد.

مزایا : ۱- کلی تر است . ۲- هدفش به خوبی تعریف شده است .

ارسطو [1] می گوید : هر هنر ، هر تحقیق و غیره ، هر عمل و عکس العمل (پیگرد) برای انجام صحیح ، احتیاج به فکر دارد.

---

[1] http://www.aaai.org/AITopics/html/cogsci.html
[2] همان منبع قبلی
[3] abstraction
[4] circuits
[5] validate
[6] neurological
[7] cognitive neuroscience
[8] normative
[9] notation
[10] rules of derivation
[11] rational behavior

عامل های منطقی [2] - یک عامل [3] ، موجودیتی [4] است که مشاهده می شود و عمل می کند. این رشته درباره ی طراحی مولفه های منطقی است. به صورت خود به خود ، یک مولفه ، تابعی برای درک تاریخچه برای عملیات می باشد. $f : p^* \rightarrow A$. برای هر کلاس ارایه شده از محیط ها و کارها ، ما به دنبال عامل (یا کلاس عامل ها) با بهترین کارکرد می گردیم . توجه کنید که ، محدودیت های محاسباتی ، عملکرد کاملا منطقی را غیر قابل دسترس می کند ، در نتیجه ، بهترین برنامه را برای ارایه به منابع ماشینی طراحی می کند.

سابقه ی AI

- **فلسفه [5] :** منطق ، متدهای استدلال ذهنی به صورت توابعی از سیستم های فیزیکی شامل آموزش ، زبان و عقلانیت می باشد .
- **ریاضیات [6] :** نمایش رسمی و اثبات الگوریتم ها ، محاسبه ، تصمیم پذیری [7] (تصمیم ناپذیری [8]) و احتمال [9] .
- **روان شناسی [10] :** انطباق [11] ، پدیده های ادراکی و تکنیک های آزمایشی کنترل حرکت (psychophysics : علم روابط میان روان و تن - و غیره) .
- **علم اقتصاد [12] :** تیوری رسمی تصمیمات عقلی
- **زبان شناسی [13] :** ارایه ی دانسته ها و ارایه ی گرامر.
- **علم اعصاب [14] :** لایه فیزیکی قابل تغییر برای فعالیت عقلی
- **تیوری کنترل [15] :** سیستم های تعادلی ، طرح عامل های مطلوب ساده می باشند .

تاریخچه ی هوش مصنوعی - در سال ۱۹۴۳ ، مک کلوچ و پیتز [16] : مدل مداری بولین مغز | در سال ۱۹۵۰ ، ماشین آلات محاسبه گر و هوش تورینگ | سال های ۱۹۵۲ تا ۱۹۶۹ نظر و دست به دست کردن | سال های دهه ی ۱۹۵۰ ، برنامه های هوش مصنوعی اولیه ، شامل برنامه ی چک کننده ی ساموئل [17] ، نظریه ی منطقی نوئل و سیمون [18] ، ماشین هندسی گلرنتر [19] | سال ۱۹۵۶ نشست دارتموند [20] : " هوش مصنوعی " پذیرفته شد | الگوریتم کامل رابینسون [21] برای استدلا ل منطقی | در سال های۱۹۶۶ تا ۱۹۷۴ ، پیچیدگی محاسباتی پیدا

---

[1] Aristotle
[2] rational agents
[3] agent
[4] entity
[5] philosophy
[6] mathematics
[7] decidability
[8] undecidability
[9] probability
[10] psychology
[11] adaptation
[12] economics
[13] Linguistics
[14] Neuroscience
[15] control theory
[16] McCulloch وPitts
[17] samuel
[18] Newell & Simon
[19] Gelernter
[20] Dartmouth meeting
[21] Robinson

می کند و پژوهش در مورد شبکه ی عصبی [1] تقریبا ناپدید می شود | در سال های ۱۹۶۹ تا ۱۹۷۹ توسعه ی اولیه ی سیستم های برمبنای دانش [2] | در سال های ۱۹۸۰ – ۸۸ توسعه ی عظیم سیستم های خبره ی صنعتی | در سال های ۱۹۹۸ – ۹۳ سیستم های خبره ی صنعتی ورشکست می کند : " زمستان هوش مصنوعی " | در سال های ۱۹۸۵ تا ۱۹۹۵ شبکه های عصبی شهرت می یابند | سال ۱۹۸۸ تجدید حیات احتمال ، افزایش اساسی در عمق تکنیک "Nouvelle AI" : ALife,GAs و محاسبه کننده ی انعطاف پذیر | سال ۱۹۹۵- عامل ها و عامل ها و هر کجا ..... | سال ۲۰۰۳ به سطح انسانی هوش مصنوعی پرداخته می شود .

زندگی مصنوعی [3] - دارای دورنماهای بسیار خوبی در زیست شناسی و علم کامپیوتر می باشد . این شاخه از علم در مورد کشت مصنوعی ، رفتارهای شبیه زندگی نظیر رویش ، سازگاری ، تولید مثل ، اجتماعی شدن ، آموزش و حتی مرگ تحقیق می نماید . محدود به دستورالعمل ابتدایی سوپ نمی باشد ، دانشمندان زندگی مصنوعی می توانند از ترکیبی از اجزا و برنامه های کامپیوتری که در زمان ، صرفه جویی می کنند و دیگر فن آوری های شگفت انگیز ، مانند آن هایی که برای تولید رفتارها و تجسمات شکل شناسی جدید [4] جستجو می نمایند استفاده نمایند . منابع لیست شده در زیر شما را با چیزهای شگفت انگیز نظیر شبکه های عصبی کامپیوتری که شبیه زندگی واقعی هستند آشنا می کند ؛ ویروس های کامپیوتری که شبیه ویروس های زنده عمل می نمایند ؛ الگوریتم های ژنتیکی و تکاملی که استراتژی های زندگی را تحت کنترل در می آورند و از آن ها برای حل مساله استفاده می نمایند . [5] **کریس جی. لنگتون** [6] در مورد زندگی مصنوعی می گوید : زندگی مصنوعی نامی ارایه شده برای یک نظم جدید می باشد که زندگی " طبیعی " را با تلاش برای از نو خلق کردن پدیده های بیولوژیکی جدید با استفاده از ابزارهای " مصنوعی " مطالعه می کند ، ارایه شده است . زندگی مصنوعی دیدگاه تحلیلی زیست شناسی قدیمی را با برخوردی مصنوعی تکمیل می کند که در آن بیش تر پدیده های زیستی را با کنار گذاشتن ارگانیزم های زندگی برای دیدن چگونگی کارکرد آن ها مطالعه می کند و برای قراردادن سیستم ها در کنار هم (که شبیه جانداران زنده عمل می کنند) تلاش می کند . [7]

وضعیت دانش

– **در حال حاضر کدامیک از موارد زیر می تواند انجام شود؟** اجرای یک بازی تنیس جدولی . (می شود) | رانندگی با ایمنی در یک جاده ی مارپیچی کوهستانی . (می شود) | خرید مایحتاج هفتگی از فروشگاه های وب (می شود) | اجرای یک بازی پل (می شود) | ترجمه کردن بلافاصله ی زبان گفتاری انگلیسی به زبان گفتاری سوئدی (می شود) | مکالمه ی موفق با دیگر افراد برای مدت یک ساعت (نمی شود) | انجام دادن یک عمل جراحی پیچیده (نمی شود) .

بازی پل ، بازی ای کارتی و چهار نفره می باشد . مانند سایر بازی های کارتی ، یکی از خصوصیات اولیه ی این بازی این است که بازی ای با اطلاعات ناقص می باشد که در آن بازی کننده های مختلف دارای اطلاعات مختلفی در مورد وضعیت واقعی بازی می باشند . این یکی از

---

[1] neural network
[2] knowledge-based systems
[3] AI یا ALife یا Artificial Life
[4] neomorphic
[5] http://www.aaai.org/AITopics/html/alife.html
[6] Chris G. Langton
[7] همان منبع قبلی

خصوصیات این بازی می باشد که باعث می شود ماشین ها بازی کننده های شایسته ای برای بازی پل نباشند .[1]

منابع :

http://aima.eecs.berkeley.edu/slides-pdf/chapter01.pdf
http://www.uni-koblenz.de/~beckert/Lehre/KI-fuer-IM/02Introduction.pdf
http://www-formal.stanford.edu/jmc/whatisai/node1.html
http://www-formal.stanford.edu/jmc/whatisai/node2.html
http://www-formal.stanford.edu/jmc/whatisai/node3.html
http://www-formal.stanford.edu/jmc/whatisai/node4.html
http://www-formal.stanford.edu/jmc/whatisai/node5.html
http://www-formal.stanford.edu/jmc/whatisai/node6.html

[1] http://www.cirl.uoregon.edu/research/bridge.html

فصل دوم

عامل های هوشمند

عامل های هوشمند [1]
فصل دوم

ریوس مطالب

- ☒ **عامل ها و محیط ها**
- ☒ **عقلانیت** [2]
- ☒ **PEAS (ارزیابی عملکرد** [3] **، محیط** [4] **، عمل کننده ها** [5] **و حسگرها** [6] **)**
- ☒ **انواع محیط**
- ☒ **انواع مولفه**

عامل ها

* یک عامل می تواند هر چیزی که
  - در یک **محیط** عمل می کند
  - محیط خود را با استفاده از **حسگرها** دریافت می کند
  - روی محیط خود با استفاده از **عمل کننده ها** عمل می نماید
  - پردازش را در طرف اهدافش بیشینه می کند

باشد .

تعریف های دیگری در مورد عامل :

- **R & N :** یک عامل می تواند هر چیزی باشد که می تواند به صورت دریافت کننده ی محیط خود با استفاده از حسگرها و عمل کننده روی محیط توسط محرک ها ، دیده شود .
- **وولریج** [7] **:** یک عامل ، یک سیستم کامپیوتری است که در یک محیط قرار داده می شود و قادر است به صورت خودمختار عمل نماید .

---

[1] intelligent agents
[2] rationality
[3] Performance measure یا معیار کارایی
[4] Environment
[5] Actuators
[6] Sensors
[7] Woolridge

* ابزار ادراکی برای تحلیل سیستم ها :
- روبوت ها [1] ، سافتبوت ها [2] ، چراغ های ترافیک ، ترموستات ها [3]
* ما به عامل های هوشمند علاقه مندیم زیرا
- اهدافی که هوش لازم دارد را پیگیری می نمایند

مثال هایی از عامل ها

- عامل انسانی :
  - چشم ها ، گوش ها ، پوست ، حس چشایی [4] و ... برای حسگرها
  - دست ها ، انگشتان ، پاها ، دهان [5] و ... برای عمل کننده ها
- روبوت
  - دوربین ، مادون قرمز [6] ، ضربه گیر یا سپر [7] و ... برای حسگرها
  - گیرنده ها (ابزارهایی که چیزی را می گیرند) [8] ، تایرها ، نورها ، بلند گوها و ... برای عمل کننده ها
- عامل نرم افزاری (سافتبوت)
  - توابع به صورت حسگرها
    - اطلاعات آماده شده به صورت ورودی برای توابع به صورت رشته های بیتی یا سمبل های کد شده
  - توابع به صورت عمل کننده ها
    - نتایج خروجی را اجرا می کنند

عامل یا برنامه

- موضوعات ما تا کنون به نظر می رسد که با یک درجه خوبی برای عامل های نرم افزاری و برنامه های منظم به کار گرفته شوند .
- **خودمختاری [9] در عامل ها**
  - عامل ها وظایف خود را تا حد زیادی به صورت مستقل انجام می دهند
  - عامل ها وابسته به کاربران یا برنامه های دیگر برای " راهنمایی " برنامه ریزی می شوند
  - مبنای سیستم های خودمختار براساس عملکردشان در تجربه اشان و دانششان می باشد
  - عامل ها به دانش اولیه با توانایی یادگیری نیازمندند
  - عامل ها دارای انعطاف برای کارهای به مراتب پیچیده تر می باشند

---

[1] robots
[2] softbots
[3] thermostats
[4] taste buds
[5] mouth
[6] infrared
[7] bumper
[8] grippers
[9] autonomy

مطلب دیگری در مورد خودمختاری

- خودمختاری ، چگونگی (کیفیتی) است که اغلب به عامل ها نسبت داده می شود .
- یک عامل خودمختار قادر است که به ادراک هایش استناد نماید و سپس برای تصمیم گیری ، تجربه نماید
- بیش تر عامل ها دارای خودمختاری کامل نمی باشند
- چالش : طراحی یک عامل که می تواند درباره ی خودمختاری خودش استدلال نماید و می داند که چه هنگام درخواست کمک نماید .

برنامه نویسی عامل گرا

- عامل ها می توانند به صورت گام منطقی بعدی بعد از اشیا تصور شوند .
  - "اشیا آ ن را برای آزادی انجام می دهند و عامل ها آن را برای پول انجام می دهند ".
- اشیا ، دریافت کننده های عملکردها هستند و عامل ها ، عمل کننده ها هستند .
- تصور عامل به صورت یک برچسب هدف نسبت برای یک توصیف ذهنی ، کم تر مفید می باشد .
- عامل (واسطه) برای ما به عنوان برنامه نویسان دارای یک چکیدگی مفید می باشد
  - به ما اجازه می دهد که در مورد یک برنامه ، در سطحی بالاتر فکر کنیم .
- در صورتی که چیزی برای فهمیدن ، پیش گویی یا توضیح رفتارش به ما کمک کند ، به صورت یک عامل رفتار خواهد کرد .
  - برای مثال ، ترموستات ها به صورت عامل رفتار می نمایند

مفیدی تشبیه عامل

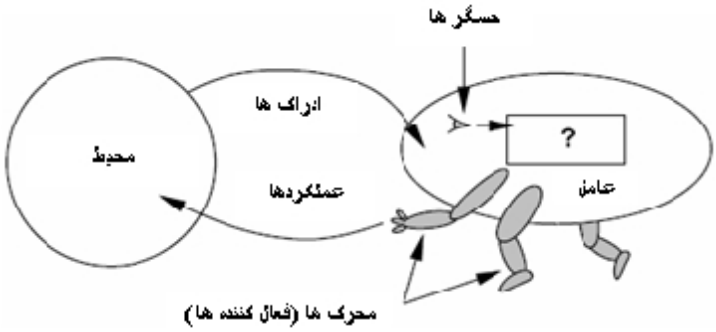
- چگونه با این دردسرها رفتار نماییم ؟ ما قبلا فهمیدیم که چگونه یک برنامه را بنویسیم .
- عامل ها میل دارند به صورت فعالیت هایي داراي فعالیت پیش نیاز یا پي آمد نباشند [1] .
  - مشخص کردن این که در آینده ، در همه ی موارد چه کاری باید انجام دهند ، مشکل است .
- این مفید است که در مورد عامل ها با فرض این که هوشمند هستند ، صحبت نماییم
- "روبوت می خواهد منبع تغذیه را پیدا نماید "
- "خادم [2] ، تصور می کند که مخدوم [3] راه اندازی مجدد شده است ."
- انتساب وضعیت های ذهنی به برنامه ها ، حالت ارادی [4] نام دارد .

عامل ها و محیط ها – عامل ها شامل انسان ها ، ربوت ها ، ترموستات ها (تنظیم کننده های حرارت) و ... می باشند ؛ مولفان دارای عامل ها هستند ، ورزشکاران حرفه ای دارای عامل ها هستند ، ستاره های سینما دارای عامل ها هستند و شما نیز دارای عامل ها می باشید . زیرا عامل ، کسی یا چیزی دارای تخصص می باشد که عهده دار انجام کاری برای شما می شود . برنامه های کامپیوتری که به شما کمک می کنند که دانسته های محاسباتی خود را افزایش دهید "

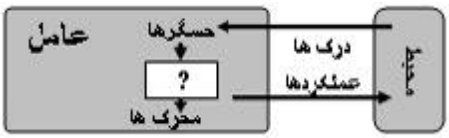
---

[1] open-ended
[2] server
[3] client
[4] intentional stance

عامل ها " می باشند . تحقیقات دانشمندان هوش مصنوعی در حال افزایش توانایی ، استقلال و ... عامل ها می باشد . [1] تابع عامل از تاریخچه های ادراکی ، به عملیات نگاشت می کند : $f : p^* \rightarrow A$. برنامه ی عامل ، در معماری فیزیکی برای تولید f اجرا می شود.



یا



- **ادراک ها (دریافت ها) :** اطلاعات فرستاده شده به حسگرهای یک عامل (نور ، صدا ، امواج الکترو مغناطیسی ، علایم (سیگنال ها)
- **حسگرها :** مکانیزم های یک عامل برای جمع آوری اطلاعات در مورد محیط شان (چشم ها ، گوش ها ، سلول های فتوالکتریک و ...)
- **محرک ها :** روش عامل برای تحت تاثیر قرار دادن محیطش (تایرها ، بازوها ، رادیوها ، نورها و ...)
- **عملکردها :** تغییرات واقعی بر محیط (حرکت ، غلطیدن)

- یک جدول راهی ساده برای مشخص کردن یک نگاشت از ادراکات به عملیات می باشد
  - جدول ها می توانند خیلی بزرگ شوند
  - تمام کار توسط طراح انجام می شود
  - بدون خودمختاری ، تمام عملیات از قبل قابل پیش بینی می باشند
  - آموزش می تواند مدت زمان زیادی ببرد
- نگاشت به صورت مجازی [2] توسط یک برنامه تعریف می شود که می تواند به صورت های زیر باشد :
  - برمبنای قانون [3]
  - شبکه های عصبی
  - الگوریتم

ساختار عامل های هوشمند

- **عامل = معماری + برنامه**

---

[1] http://www.aaai.org/AITopics/html/agents.html
[2] implicitly
[3] rule based

- **برنامه ی عامل :** پیاده سازی نگاشت ادراک – عملکرد عامل می باشد
- **معماری :** ابزاری که به وسیله ی آن می توان برنامه ی عامل را اجرا نمود (به عنوان مثال کامپیوتر همه منظوره ، ابزار تخصصی ، روبوت و ...)

برنامه ی عامل ها

- ما می توانیم رفتار عامل مان را با یک تابع F توصیف نماییم :
- عملکرد = (تاریخچه ی ادراکی ، ادراک فعلی) F
- یک رشته ی ادراکی را به یک عملکرد ، نگاشت می نماید .
- در واقع ، به کار گیری این تابع ، کار یک برنامه ی عامل می باشد .
  - این همان چیزی است که ما بیش تر وقت خود را روی آن می گذاریم .

مثال : دنیای جاروبرقی

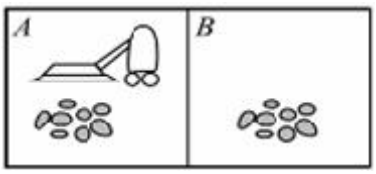
- بیایید با یک مثال خیلی ساده شروع کنیم .
- دو فضای A و B وجود دارند . هر فضا یا می تواند تمیز باشد یا کثیف .
- این ، محیط عامل می باشد
- حسگرها : حسگر کثیفی و تایین محل .
- محرک ها : جارو برقی و چرخ ها .
- ادراک ها : تمیز و پاک
- عملکردها : حرک به چپ ، راست ، مکش و هیچ کار .
- در این مثال ساده ، ما می توانیم همه ی رشته های ادراکی ممکن و عملکردهای وابسته را لیست نماییم
- این ، یک عامل برمبنای جدول [1] نام دارد
- **سوال :** چگونه برای هر رشته ی ادراکی ، بهترین عملکرد را انتخاب نماییم ؟
- ما به یک ارایه ی فشرده تری برای این جدول نیازمندیم .

عقلانیت [2]

- به طورکلی ، عقلانیت به معنی "انجام چیز درست می باشد "
- دقت بیش تری لازم می باشد – "چیز درست " ، چیست ؟
- ما به یک تعریف از موفقیت نیازمندیم .
- با معیار کارایی (ارزیابی عملکرد) شروع کنید
  - این ، وضعیت جهانی می باشد که ما می خواهیم عامل به آن دسترسی پیدا نماید .
  - "هر دو فضا تمیز هستند ." (و شاید ملاک های بیش تری مورد نیاز باشد ، نظیر کمینه نمودن زمان ، توان مصرف شده ، یا تعداد عملیات انجام شده)
  - ما ممکن است که یک اندازه گیری نرده ای (اسکالر) یا یک وضعیت بولین را ترجیح دهیم .
- توجه نمایید که این یک خصوصیت یک نتیجه است

[1] table-based
[2] rationality

- یک عملکرد عقلانی ، عملکردی است که معیار کارایی یک عامل ، ادراک ها و عملکردهای آن را بیشینه می نماید
- R & N : عامل های عقلانی : برای هر رشته ی ادراکی ، باید عملکردی که انتظار می رود معیار کارایی و ملاک ارایه شده توسط رشته ی ادراکی و هر چه در دانش عامل وجود دارد را بیشینه می کند ، انتخاب نماید .
- ما نیازی نداریم که عامل قادر باشد که آینده را پیش بینی نماید یا رویدادهای بد را پیش گویی نماید .
- جمع آوری اطلاعات هم ممکن است یک عملکرد عقلانی باشد .
  - عبور بدون توجه از خیابان ، نامعقولانه می باشد .
- عامل های عقلانی باید دارای توانایی یادگیری (آموزش) [1] باشند (به جز در موارد خیلی ساده و محیط هایی که به خوبی قابل فهم می باشند)
  - آموزش به معنی بهبوددادن عملکرد یک عامل است .
  - این ممکن است به معنی کاهش عدم قطعیت یا ارایه ی ملاحظاتی به حساب یا گزارش باشد .



عامل های یک جارو برقی

- تابع Reflex-Vacuum-Agent([location,status]) یک عملکرد را برمی گرداند
- در صورتی که وضعیت [2] برابر با کثیف [3] باشد مکش را برمی گرداند |
- در غیر این صورت ، در

عملکرد	رشته ی ادراکی
Right	[A,Clean]
Suck	[A,Dirty]
Left	[B,Clean]
Suck	[B,Dirty]
Right	[A,Clean],[A,Clean]
Suck	[A,Clean],[A,Dirty]
٨	٨

صورتی که محل [4] برابر با A باشد ، راست را برگردان

- در غیر این صورت ، در صورتی که محل برابر با B بود ، چپ را برگردان

تابع درست چیست؟ آیا می تواند در یک برنامه ی عامل کوچک به کار گرفته شود ؟

---

[1] learning
[2] status
[3] Dirty
[4] location

عامل عقلانی [1] - یک عامل هوشمند هر کدام از اعمالی که مقدار معیارکارایی ارایه شده توسط رشته ی ادراکی مورد نظر را به بیش ترین مقدار افزایش می دهد را انتخاب می نماید . عامل هوشمند ، به همه چیز وقوف [2] ندارد . دریافت ادراکی ممکن است همه ی اطلاعات را به وجود نیاورد. عامل هوشمند ، نهان بین [3]هم نمی باشد . عملکرد به وجود آمده ممکن است به صورتی که انتظار داشتیم نباشد . از این رو ، عامل هوشمند ، موفقیت آمیز نمی باشد . عامل هوشمند دارای ابتکار [4] ، یادگیری [5] و خودمختاری می باشد .

PEAS - برای طراحی یک عامل هوشمند ، ما باید محیط عملیاتی [6] را مشخص کنیم . مثلا درطراحی یک تاکسی خودکار ، موارد زیر را باید مشخص نماییم : معیارکارایی ؟؟ | محیط ؟؟ | عمل کننده ها (محرک ها) ؟؟ | حسگرها ؟؟

مثال ، طراحی یک تاکسی خودکار ، معیارکارایی ؟؟ ایمنی [7] ، هدف [8] ، مزایا [9] ، رعایت قانون [10] ، آسایش و راحتی [11] و ..... | محیط ؟؟ خیابان های ایالات متحده / آزاد راه های ایالات متحده ، ترافیک [12] ، عابران پیاده [13] ، هوا [14] و ..... | عمل کننده ها ؟؟ فرمان [15] ، شتاب دهنده [16] ، ترمز [17] ، بوق [18] ، بلندگو یا نمایش دهنده [19] و ..... | حسگرها ؟؟ ویدئو [20] ، شتاب سنج [21] ، درجه ها [22] ، حس گرهای موتور [23] ، صفحه کلید [24] ، GPS و .....

مثال : عامل های خرید اینترنتی [25] ، معیارکارایی ؟؟ | محیط ؟؟ عمل کننده ها ؟؟ | حسگرها ؟؟ را باید مشخص نماییم . | معیارکارایی ؟؟ قیمت [26] ، کیفیت [27] ، تناسب [28] ، کارایی [29] | محیط ؟؟ سایت های فعلی و آینده ی www ، فروشنده ها [30] ، حمل کننده ها [31] | عمل کننده ها ؟؟

---

[1] rational agent
[2] omniscient
[3] clairvoyant
[4] exploration
[5] learning
[6] task environment
[7] safety
[8] destination
[9] profit
[10] legality
[11] comfort
[12] traffic
[13] pedestrains
[14] weather
[15] steering
[16] accelerator
[17] brake
[18] horn
[19] speaker/display
[20] video
[21] accelerometer
[22] gauges
[23] engine sensors
[24] keyboard
[25] shopping
[26] price
[27] quality
[28] appropiateness
[29] efficiency
[30] vendors
[31] shippers

نمایش دادن به کاربر [1] ، دنبال کردن URL ، پرکردن فرم | حسگرها ؟؟ صفحات HTML (متنی ، گرافیکی ، اسکریپتی)

محیط ها

- یک معیار برای وجود یک عامل ، وجود آن در یک محیط می باشد .
- لازم نیست که به صورت فیزیکی باشد – ممکن است به صورت یک محیط نرم افزاری باشد .
- نگرش R & N به محیط عملیاتی
- تشکیل شده از :
  - معیار کارایی
  - محیط
  - محرک های در دسترس برای عامل
  - محرک هایی که برای عامل قابل دسترسی می باشند

ویژگی محیط ها

- قابل مشاهده بودن [2]
- قطعی / اتفاقی
- دوره ای (اپیزودیک) و ترتیبی
- ثابت (استاتیک) و پویا (دینامیک)
- گسسته و پیوسته
- تک عاملی و چند عاملی

قابل مشاهده بودن

- محیط ، کاملا قابل مشاهده است ، در صورتی که حسگرهای عامل همیشه اطلاعات کاملی در مورد اجزای مربوط محیط بدهند
- آیا چیزی وجود دارد که عامل ، نیاز به دانستن آن داشته باشد در شرایطی که نمی تواند آن را حس نماید ؟
- بازی شطرنج ، کاملا قابل مشاهده می باشد .
- جارو برقی : به صورت جزیی قابل مشاهده می باشد (در صورتی که در فضای مجاور ، کثیفی وجود داشته باشد نمی تواند آن را ببیند)

قطعی / اتفاقی

- ما می توانیم فکر کنیم که جهان دارای انتقال میان وضعیت ها می باشد .
- وضعیت جاری * عملکردهای عامل ← وضعیت جدید
- در صورتی که انتقال وضعیت ، منحصر به فرد باشد ، جهان به صورت قطعی می باشد .
- آیا یک عملکرد همیشه نتیجه ی یکسانی را تولید می کند ؟
- بازی شطرنج ، بازی ای قطعی می باشد
- دنیای جاروبرقی ، قطعی است
- رانندگی یک اتومبیل ، تصادفی می باشد

---

[1] display to user
[2] observability

- نکته : ما اجتناب می کنیم که در این جا ، سوالات را با مکانیک کوانتومی (ذره ای) موشکافی نماییم – ممکن است که جهان ، قطعی باشد ، اما با توجه به پیچیدگی اش به نظر تصادفی (اتفاقی) برسد .
- جهان برای عامل چگونه به نظر می رسد .

دوره ای و ترتیبی

- دوره ای : هر عملکرد به صورت مستقل می باشد .
  - عامل ، ادراک می کند ، تصمیم می گیرد و عمل می نماید و دوباره [از نو] شروع می نماید .
  - تصمیم گیری بعدی ، وابسته به وضعیت های قبلی نمی باشد .
  - عامل فیلترکننده ی اسپم ، دوره ای می باشد .
- در صورتی که عامل باید یک سری از عملیات را برای رسیدن به یک عمل یا رسیدن به یک هدف انجام دهد ، محیط ترتیبی می باشد .
  - به تصمیم گیری های آینده باید توجه شود .
  - رانندگی یک اتومبیل ، ترتیبی می باشد .

ثابت و پویا (دینامیک)

- یک محیط ثابت ، مادامی که عامل در حال تصمیم گیری در مورد یک عملکرد می باشد ، ثابت باقی می ماند .
- عامل برای رسیدن به یک تصمیم تحت محدودیت یا فشار زمانی نمی باشد
  - فیلترکردن اسپم ، ثابت می باشد .
  - بازی شطرنج ، ثابت می باشد .
- در یک محیط پویا ، در زمانی که عامل در حال تصمیم گیری در مورد این می باشد که چه کاری را انجام بدهد ، محیط تغییر پیدا می کند .
- عامل در این مورد باید " به اندازه ی کافی با سرعت " عمل نماید
  - رانندگی یک اتومبیل ، پویا می باشد
- **نیم پویا :** محیط تغییر نمی کند ، اما معیارکارایی در طول زمان تغییر می کند .
  - انجام یک آزمایش در زمان محدود
  - بازی شطرنج با یک ساعت .
- هنوز برای عمل کردن به سرعت ، ما زیر فشار می باشیم .

گسسته و پیوسته

- ما می توانیم در مورد گسسته و پیوسته در مورد ادراک ها ، عملکردهای عامل یا وضعیت های ممکن محیط ، صحبت کنیم .
- در صورتی که مقادیر هر کدام از موارد گفته شده به صورت مجموعه ای گسسته باشد ، آن گاه محیط گسسته می باشد .
  - گسسته ، شبیه محدود نمی باشد .
  - یک محیط فیلترکننده ی اسپم ، گسسته می باشد ، حتی در میان تعداد (قابل شمارش) نامحدود از پیام های الکترونیکی .
- یک محیط پیوسته دارای متغیرهای تغییر کننده به صورت پیوسته است .
  - زاویه های فرمان در یک محیط رانندگی اتومبیل .

- حسگر خوان های با مقادیر واقعی (ما می توانیم در مورد ادراک ها در این مورد موشکافی نماییم ؛ در این جا نکته این است که آیا یک تغییر قابل تشخیص میان دو مقدار وجود دارد یا نه)
- زمان ، عنصری است که ما همواره با آن سروکار داریم .

تک عاملی یا چندعاملی

- تک عاملی ، عامل ما در حال عمل برروی خودش می باشد .
  - جهان ممکن است هنوز به صورت اتفاقی باشد
  - فیلترکردن اسپم ، به صورت تک عاملی می باشد .
- چندعاملی : عملکردها / اهداف / روش های عامل های دیگر باید به حساب آیند .
  - بازی شطرنج
- نتایج
  - گرچه یک جهان ممکن است دارای عامل های دیگری باشد ، اما ما ممکن است به خاطر پیچیدگی استنتاج ، با آن به صورت تک عاملی و اتفاقی رفتار نماییم .
    - به عنوان مثال یک عامل کنترل کننده ی علایم ترافیکی .

برخی مثال ها

- بازی شطرنج ، ماشینی که در آن پول می ریزند و کالایی را دریافت می کنند [1]
- روبوتی که در هارنی [2] به من قهوه می دهد
- مدار مریخ [3]
- عامل مکالمه ای
- عامل تشخیص طبی

مثال :

	تاکسی	فروشگاه اینترنتی	تخته نرد	منفرد
قابل مشاهده ؟؟	**نه**	**نه**	**بله**	**بله**
نتیجه ی فرایند ؟؟	**نه**	**تاحدودی**	**نه**	**بله**
دوره ای ؟؟	**نه**	**نه**	**نه**	**نه**
ایستا ؟؟	**نه**	**نیمی**	**نیمی**	**بله**
مجزا ؟؟	**نه**	**بله**	**بله**	**بله**
تک مولفه ای ؟؟	**نه**	**بله (به جز حراج ها)**	**نه**	**بله**

نوع محیط تا حد زیادی طرح عامل را تایین می کند . البته دنیای واقعی اندکی قابل مشاهده ، دوره ای ، ترتیبی ، پویا ، دنباله دار و چند عاملی می باشد.

ارزیابی عملکرد (معیار کارایی)

[1] slot-machine
[2] Harney
[3] Mars orbiter

عامل جاروبرقی

- تعدادی از کاشی ها در یک زمان معین تمیز می شوند
  - براساس گزارش عامل یا تایید اعتبار یک مرجع با صلاحیت
  - به هزینه ی عامل و اثرات جانبی توجه ننمایید
- انرژی ، اغتشاش [1] ، گم شدن قطعات مفید ، اثاثیه ی خراب ، خراش کف
  - ممکن است منجر به کارهای ناخواسته شود
    - عامل تمیزی مجدد کاشی ها را تمیز می نماید ، فقط بخشی از اتاق را پوشش می دهد

عامل های نرم افزاری

- معمولا به صورت سافتبوت ها شناخته می شوند
- در محیط های مصنوعی که کامپیوترها و شبکه ها یک شالوده [2] را به وجود می آورند می باشند
- می توانند خیلی پیچیده با نیازمندی های زیاد در عامل باشند
  - وب جهانی [3] و محدودیت های زمان واقعی [4]
- محیط های طبیعی و مصنوعی ممکن است با هم ادغام شده باشند
  - تعامل کاربر
  - حسگرها و عمل کننده ها در دنیای واقعی
    - دوربین ، دما ، بازوها ، تایرها و ...

عامل های متحرک [5]

- برنامه ها می توانند از یک ماشین به ماشین دیگر بروند
- در یک محیط اجرایی مستقل از پایگاه [6] اجرا شوند
- به محیط اجرایی عامل نیازمندند
- تغییر پذیری لازم یا مناسب وضعیت برای عامل نمی باشد
- فایده های عملی اما غیرتابعی :
  - هزینه ی کاهش داده شده ی ارتباطی
  - محاسبه ی ناهماهنگ [7] (وقتی که شما متصل نمی باشید)
- کاربردها :
  - بازیابی اطلاعات توزیع شده
  - مسیریابی شبکه ی مخابراتی

عامل های اطلاعاتی [8]

- رشد انفجاری اطلاعات را مدیریت می نمایند
- اطلاعات چند منبع توزیع شده را به کار می گیرند یا مقایسه [9] می نمایند

---

[1] noise
[2] infrastructure
[3] World Wide Web
[4] real - time
[5] mobile agents
[6] independent-platform
[7] asynchronous
[8] information agents
[9] collate

- عامل های اطلاعاتی می توانند ثابت یا متحرک باشند
- اطلاعات درون وب یا یک سند واقعی (مثلا یک سند کاغذی)
  - برای صفحات تفسیر کننده ی وب شناخته می شوند
  - منبع داده ای در داده های بی ساختارند
  - پاسخگویی به سوال با استفاده از دانش قوی متدهای آماری انجام می شود

برنامه های محیط

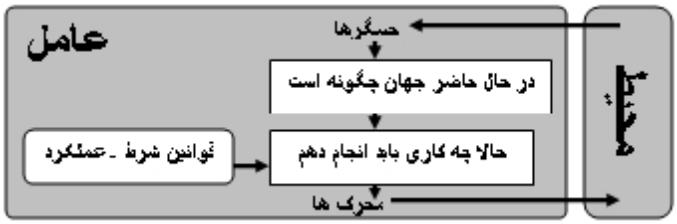
- شبیه سازی کننده های محیط برای آزمایش های توسط عامل ها
  - یک ادراک را به یک عامل ارایه می نمایند
  - یک عملکرد را دریافت می نمایند
  - محیط را به روز می نمایند
- اغلب به کلاس های محیط برای عملیات وابسته یا انواع عامل ها تقسیم می شوند
- خیلی از اوقات مکانیزم هایی را برای اندازه گیری عملکرد عامل ها ارایه می نمایند

انواع عامل ها

چهار نوع اصلی وجود دارد : عامل های بازتابی ساده [1] ـ عامل های بازتابی با حالت یا وضعیت یا عامل هایی که وضعیت جهان را حفظ می نمایند [2] ـ مولفه های براساس هدف یا هدف گرا [3] ـ مولفه های سودمند [4] . همه ی این ها می توانند در مولفه های آموزشی به کار گرفته شوند.

عامل های بازتابی ساده

- دارای جدول جستجوی ساده می باشند و ادراک ها را به عملیات نگاشت می نمایند (خیلی بزرگ و پرهزینه می باشند)
- تعدادی از وضعیت ها می توانند توسط قانون های شرط – عملکرد خلاصه شوند



- پیاده سازی این نوع عامل ها آسان می باشد ولی دارای کاربرد کمی می باشند .

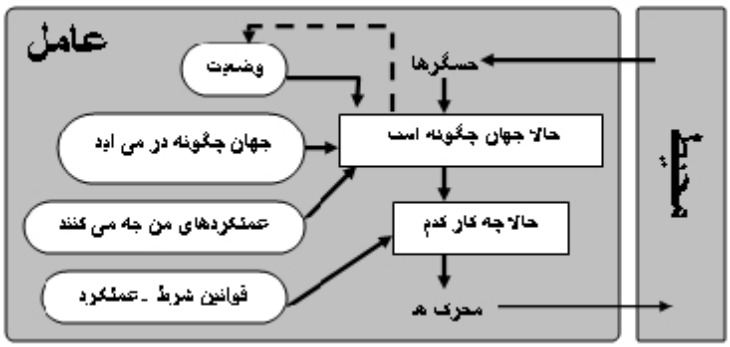
مثال تابع Reflex-Vacuum-Agent([location,status]) یک عملکرد را برمی گرداند ، در صورتی که وضعیت برابر کثیف (Dirty) بود ، مکش (Suck) را برمی گرداند ، در غیر این صورت ، اگر محل (location) برابر A بود ، راست (Right) را برگردان ، در غیر این صورت ، اگر محل (location) برابر B بود ، چپ (Left) را برگردان .

---

[1] simple reflex agents
[2] reflex agents with state
[3] goal-based agents
[4] utility-based agents

عامل هایی که وضعیت جهان را حفظ می کنند (عامل های بازتابی براساس مدل)

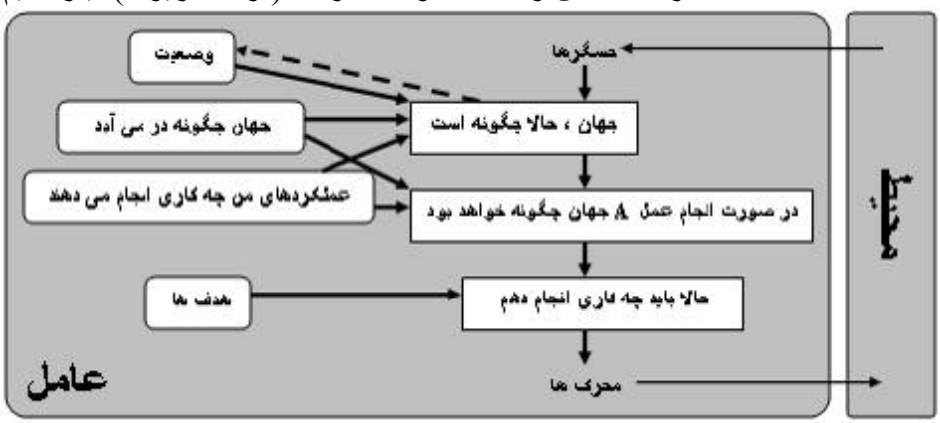
- اطلاعات عامل به تنهایی در مورد مشاهده پذیری جزیی کافی نمی باشند
- لازم است که جریان تغییرات جهان را نگهداری نماییم
  - تکامل : به طور مستقل از عامل یا سبب شده توسط عملکرد عامل می باشند
  - دانش درمورد چگونگی اثرات جهان – **مدل** جهان



مثال تابع Reflex-Vacuum-Agent([location,status]) یک عملکرد را برمی گرداند
متغیرهای static : last_A,last_B,numbers دارای مقدار اولیه ی ∞
در صورتی که وضعیت (status) = کثیف (Dirty) باشد ...

عامل های هدف گرا

- وضعیت و عملکردها نمی گویند که کجا برویم
- به **اهداف** برای ساختن رشته ای از عملکردها (برنامه ریزی) نیازمندیم

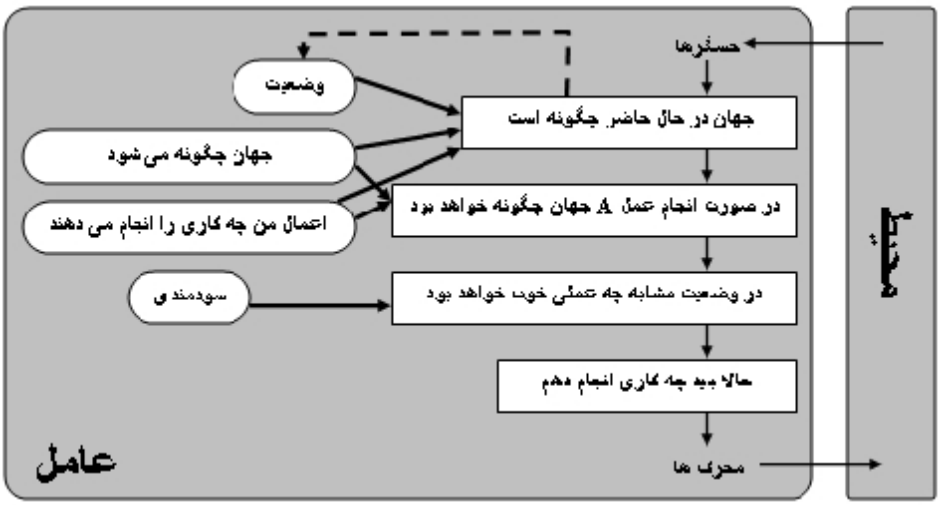


- هدف گرایی : از قوانین یکسان برای اهداف مختلف استفاده می نماید
- بازتاب : به مجموعه ای کامل از قوانین برای هر هدف نیاز خواهیم داشت

مطالبی در مورد عامل های هدف گرا : دانستن وضعیت فعلی محیط ، همیشه کافی نمی باشد . عملکرد درست ممکن است همچنین به این که عامل در تلاش برای رسیدن به چه چیزی است باشد . عملکردهایی را که برای رسیدن به هدف ها کمک می کنند را انتخاب نمایید . جستجو و برنامه ریزی برای حل این مساله مورد استفاده قرار می گیرند . استدلال هدف گرا برای محیط های ترتیبی خیلی مفید می باشد . مثل ، بازی شطرنج ، رانندگی تاکسی ، خلبانی سفینه ی فضایی . عملکرد درست برای یک رشته ی ادراکی ارایه شده وابسته به دانش عامل و وضعیت جاری آن و چیزی که در حال حاضر در تلاش برای رسیدن به آن است می باشد .

عامل های سودمند

- هدف ها ممکن است در محیط های با پیچیدگی بالا ، کافی نباشند .
- چند رشته از عملکردها برای دسترسی به برخی اهداف لازمند (پردازش دودویی)
- نیازمند انتخاب میان عملکردها و رشته ها می باشیم
- سودمندی : وضعیت را به عددی حقیقی نگاشت می نماید
  - درجه ی ارضا را بیان می کند و سبک و سنگینی میان متناقض را مشخص می نماید
- سودمندی برای مقایسه ی شرایط مطلوب وابسته به رشته های عملکرد ، استفاده شوند .
- می تواند تخمینی از هزینه ، زمان ، یا مقدار وابسته به نتایج مختلف باشد .
- سودمندی ها در محیط هایی که به صورت جزیی قابل مشاهده اند یا محیط های اتفاقی ، خیلی مفید می باشند .



- سودمندی ها برخی اوقات یک چیز بحث انگیز در هوش مصنوعی می باشند .
- فرض : نتایج می توانند به صورت خطی در یک مقیاس یکسان ، مرتب شده باشند .
  - مثال : عامل رانندگی تاکسی
  - طراح باید نتایج را به طور صریح ارزیابی نماید (به صورت کمی و کیفی)
- سودمندی در دامنه های با احتمال ، خیلی مفید می باشد
  - تجارت برخط ، اکتشاف در محیط های دارای عدم قطعیت ، قماربازی [1] .

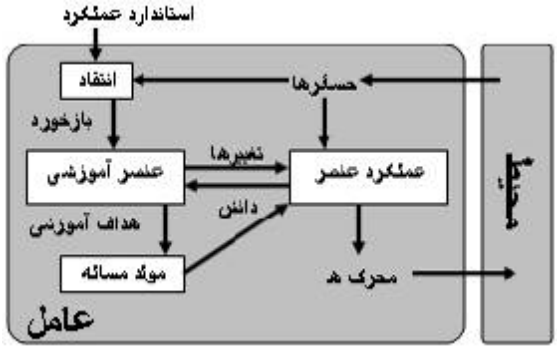
عامل های آموزشی

- عنصر آموزشی [2] : بهبودهایی را به وجود می آورد
- عنصر کارایی [3] : انتخاب عملکردهای خارجی
- انتقاد [4] : جمع آوری بازخورد [5] در مورد چگونگی عمل عامل
- تولید کننده ی مساله [6] : عملکرد (آزمایش) های پیشنهادی (اکتشافی)

---

[1] gambling
[2] learning element
[3] performance element
[4] critic
[5] feedback
[6] problem generator

- اغلب ، یک عامل ممکن است نیاز به بروزرسانی برنامه ی عامل خود داشته باشد .
- برنامه نویسان ممکن است فهم کاملی از محیط نداشته باشند .
- محیط ، ممکن است در طول زمان ، تغییر نماید .
- برنامه نویسی با دست ممکن است خسته کننده باشد .
- یک عامل آموزشی ، عاملی است که عملکرد خود را با توجه به مجموعه ای از عملکردها در طول زمان ، بهبود می بخشد .
- آموزش یا انطباق در محیط های پیچیده ، ضروری می باشد .
- یک عامل آموزشی هم به *عنصر عملکرد یا کارایی* [1] و هم به *عنصر آموزشی* [2] نیازمند می باشد
  - عنصر کارایی : عملکرد یا عملکردهای جاری را انتخاب می نماید
  - عنصر آموزشی : درستی عنصر کارایی را ارزیابی می نماید .



- آموزش ، می تواند به صورت برخط یا برون از خط باشد .
- آموزش ، ممکن است *غیرفعال* یا *فعال* باشد .
- آموزش ممکن است نظارت شده یا نظارت نشده باشد .
- *انتساب اعتبار* [3] در زمانی که آموزش در محیط های ترتیبی می باشد ، مساله ای بزرگ می باشد .
- اغلب ، آموزش در هوش مصنوعی به صورت یک موضوع مجزا ، عمل می کند ؛ ما برای جمع کردن آن با دیگر موضوعات ، تلاش می کنیم .

خلاصه

عامل ها **، روی *محیط ها* به وسیله ی *محرک ها* و *حس گرها* اثر می گذارند. تابع عامل چگونکی عملکرد عامل را در همه ی شرایط تشریح می کند. ارزیابی عملکرد (معیارکارایی) ترتیب محیط را ارزیابی می کند. یک عامل هوشمند بی عیب ، عملکرد مورد انتظار را بیشینه می کند . برنامه های عامل برخی از توابع عامل را اجرا می کنند. توضیحات *PEAS* محیط های کاری را تعریف می کند.**

محیط ها در چند بعد طبقه بندی می شوند: *قابل مشاهده ؟ ـ قطعی ؟ ـ دوره ای ؟ ـ پویا ؟ ـ گسسته ؟ ـ تک مولفه ای ؟*

---

[1] performance element
[2] learning element
[3] credit assignment

چند معماری موجود عامل های پایه عبارتند از: *عامل های بازتابی ساده– عامل هایی که وضعیت جهان را حفظ می نمایند– عامل های هدف گرا - عامل های سودمند*

منابع :

http://aima.eecs.berkeley.edu/slides-pdf/chapter02.pdf
http://www.inf.unibz.it
http://cs.usfca.edu

فصل سوم

حل مساله و جستجو

حل مساله [1] و جستجو [2]
فصل سوم

خلاصه ی ریوس مطالب

- **عامل های حل کننده ی مساله**
- **انواع مساله**
- **فرمول بندی [3] مساله**
- **مثال هایی از مسایل**
- **الگوریتم های پایه ای جستجو**

یک عامل هدف گرا می تواند توجه کند به این که چه کاری را می تواند انجام دهد و عملکردهایی که به نتیجه می رسند را انتخاب نماید . برنامه ی عامل از درک ها و هدف ، به صورت ورودی استفاده می کند . ما به یک نوع عامل هدف گرا به نام عامل حل مساله توجه می کنیم .

جستجو به عنوان روش حل مساله

- بسیاری از مسایل می توانند به صورت رسیدن به یک حالت هدف [4] ، از یک نقطه ی شروع دیده شوند
- فضای حالت [5] ، مساله و راه حل های ممکن آن را به یک صورت رسمی تری تعریف می نماید
  - معمولا مسایل باید به صورت تجزیه شده [6] باشند
- عملکردهای عامل ، وضعیت را تغییر می دهد
  - فضای حالت را برای یک راه حل ملاقات می نماید
- اطلاعات ، در مورد مساله ی معین یا با دامنه ی عمومی می توانند برای بهبود جستجو استفاده شوند
  - تجربه از موارد قبلی مساله
  - روش های بیان شده به صورت اکتشافات

---

[1] problem solving
[2] search
[3] formulation
[4] goal state
[5] state space
[6] abstracted

- نوع های ساده تر مساله
- محدودیت های وضعیت های معین مساله

انگیزه [1]

- روش های جستجو ، روش هایی مهم برای رهیافت های بسیار برای حل مساله می باشند
- استفاده از جستجو ، به یک فرمول بندی مجزای مساله و مراحل ممکن برای به وجود آوردن راه حل ها نیازمند می باشد
- الگوریتم های جستجو ، پایه ای برای بهینه سازی [2] و روش های برنامه ریزی می باشند

عامل های حل کننده ی مساله

یک عامل حل کننده ی مساله ، برای پیداکردن یک رشته از عملکردها که به هدف می رسند تلاش می کند . به عنوان مثال ، چه رشته از حرکت ها یک مکعب رابیک [3] را حل می کنند ؟ . چگونه من از دانشگاه فلوریدای جنوبی [4] با اتومبیل به سمت لیورمور رانندگی نمایم ؟ . چگونه من می توانم اجزای روی یک تراشه را بچینم ؟ . کدام رشته از عملیات یک روبوت را در یک فضا حرکت می دهد ؟ /

- فرمول بندی مساله
  - وضعیت های ممکن وابسته به دنیا برای حل مساله کدامند
  - چه اطلاعاتی برای عامل در دسترس می باشند
  - رفتن عامل از یک وضعیت به وضعیت دیگر چگونه می تواند باشد
- فرمول بندی هدف
  - وضعیت هدف چیست
  - ویژگی های مهم وضعیت هدف چه می باشند
  - چگونه عامل می فهمد که به هدف رسیده است
  - آیا چند وضعیت پایانی ممکن وجود دارد
    - آیا آن ها با هم برابرند یا برخی بهترند

جستجو

- پردازه ی رسیدگی کردن به صورت ترتیبی به عملکردها برای پیداکردن رشته ای از عملیات که ما را از شروع به هدف می رسانند ، جستجو نام دارد .
- یک الگوریتم جستجو ، یک رشته از عملیات که برای عامل اجرا می شوند را برمی گرداند .

---

[1] motivation

[2] optimization

[3] Rubik's cube ، یک پازل به شکل یک مکعب پلاستیکی که با مربع هایی با چند رنگ پوشانده شده است ، هر بازیگر تلاش می کند مربع ها را تغییر دهد تا این که همه ی مربع های هر وجه دارای یک رنگ بشوند [انتشارات دانشگاه آکسفورد ، سال ۲۰۰۴ میلادی ، گرفته شده از لغتنامه ی Babylon]

[4] University of South Florida = USF

- جستجو معمولا به صورت "برون خط " انجام می شود .
- نکته : فرض بر این است که محیط به صورت استاتیک (ثابت) می باشد .
- همچنین فرض شده که محیط ، گسسته می باشد .
- محیط (معمولا) به صورت قطعی می باشد .

برخی از مسایل جستجوی کلاسیک

- مسایل سرگرمی : برای مطالعه به صورت مثال ها یا برای مقایسه ی الگوریتم ها ، مفید می باشد
  - پازل ۸ – تایی
  - دنیای جارو
  - مکعب رابیک
  - N – وزیر
- مسایل دنیای واقعی : معمولا کار زیادی می برند ، اما جواب معمولا جالب می باشد
  - مسیر یابی
  - مسافرت شخص دوره گرد
  - طرح VLSI
  - جستجو در اینترنت

وضعیت

- ما معمولا در مورد وضعیتی که یک عامل در آن می باشد صحبت می کنیم .
- به مقادیر متغیرهای مربوط توصیف کننده ی محیط و عامل ، اشاره می کند .
  - در دنیای جارو : (۰و۰) ، تمیز می باشد
  - در مساله ی رومانی : در t=0 در بخارست هستیم
  - در مکعب رابیک : نظم فعلی مکعب (چگونگی قرار گرفتن مربع های روی مکعب در حال حاضر)

فرمول بندی مساله

- شناختن نوع مساله
  - عامل چه دانشی در مورد وضعیت جهان و نتیجه ی عملکرد خودش دارد
  - آیا اجرای عمل نیازمند به روز کردن اطلاعات دارد ؟
  - تعریف وضعیت های جهان
- توضیح رسمی برای عملکرد عامل
  - توضیح هدف
  - عملگرها : عملکردهای عامل

انتخاب وضعیت ها و عملکردها

مسایل برای حل باید **مجزا** باشند

- شرح وضعیت های قابل تمیز در مدت فرایند حل مساله
  - وابسته به عملکرد و دامنه
- عملکردها (عملگرها) عامل را از وضعیتی به وضعیت دیگر می برند
  - وابسته به وضعیت ها ، توانایی عامل و خصوصیات محیط
- انتخاب وضعیت ها و عملگرهای مناسب
  - می تواند باعث تفاوت میان این که یک مساله می تواند حل شود یا نمی تواند حل شود بشود

انواع مساله

- مسایل تک حالته [1] : در صورتی که مساله قطعی و کاملا قابل مشاهده باشد ، در نتیجه ، مساله ، تک حالته خواهد بود ، عمل کننده (عامل) دقیقا می داند در کدام حالت قرار خواهد گرفت ؛ راه حل به صورت ترتیبی است.
  - جهان در دسترس می باشد و اثر عملکردها را می دانیم
  - عامل ، وضعیتی که بعد از یک رشته عملیات در آن خواهد بود را می داند
- مسایل چندحالته [2] : در صورتی که مساله غیرقابل مشاهده باشد ، در نتیجه مساله ، تطبیقی (ترکیبی [3]) خواهد بود . عامل ، ممکن است تصمیمی در مورد کجا قرار گرفتن نداشته باشد ؛ راه حل (در صورت وجود) ترکیبی می باشد.
  - جهان فقط تا حدودی در دسترس می باشد
  - عامل به چند وضعیت ممکن توجه دارد
- مسایل احتمالی [4] : در صورتی که مساله غیرقطعی [5] و یا اندکی قابل مشاهده [6] باشد ، در نتیجه ، مساله ی احتمال می باشد .
  - در طول اجرا به مشاهده و حس کردن نیازمند می باشد
  - به درخت عملیات نیازمندیم
- مسایل اکتشافی [7] : در صورتی که مساله دارای فضای حالت ناشناخته باشد ، در نتیجه ، مساله ، اکتشافی می باشد .
  - عامل نتایج عملکردش را نمی داند
  - آزمایش های عامل برای کشف وضعیت های جهان و اثرات عملکردها می باشد

مسایل خوب تعریف شده

- وضعیت اولیه
  - نقطه ی شروع که عامل از آن شروع می کند ، مثلا در مورد مساله ی کشور رومانی ، وضعیت اولیه ، شهر آراد می باشد

---

[1] single-state problems
[2] multiple-state problems
[3] conformant
[4] contingency problems
[5] nondeterministic
[6] partially observable
[7] exploration problems

- فضای حالت
  - مجموعه ای از همه ی وضعیت هایی که از وضعیت اولیه توسط رشته ای از عملکردها قابل دسترسی می باشند
  - مسیر ، رشته ای از عملکردها که راهنمایی کننده از یک وضعیت به وضعیت دیگر می باشد
- عملکردها (عملگرها و توابع جانشین [1]) چه عملکردهایی را عامل می تواند داشته باشد ؟
  - توضیح مجموعه ای از عملکردهای ممکن می باشد ، مثلا در مورد جارو ، چپ ، راست ، بالا ، پایین ، مکش و هیچ کار می تواند باشد
- آزمایش هدف
  - تشخیص می دهد که آیا یک وضعیت ارایه شده ، یک وضعیت هدف می باشد یا نه
- راه حل
  - مسیری از وضعیت اولیه و یک وضعیت هدف

تابع جانشین

- تابع جانشین برای یک وضعیت داده شده ، یک مجموعه از جفت های عملکرد / وضعیت جدید را برمی گرداند .
  - این به ما می گوید که برای یک وضعیت داده شده ، چه کارهایی را می توانیم انجام دهیم و در چه جاهایی مقدم هستند .
- در یک جهان قطعی ، هر عملکرد با یک وضعیت منفرد ، جفت شده شده است .
  - مثلا در دنیای جارو برقی :

$$(In(0,0))\rightarrow$$
$$('Left',In(0,0)),('Right',In(0,0)),('Suck',In(0,0),'Clean')$$

  - در مساله ی رومانی :

$$In(Arad) \rightarrow$$
$$((Go(Timisoara),In(Timisoara),(Go(Sibiu),In(Sibiu)),(Go(Zerind),In(Zerind))$$

  - در جهان های اتفاقی ، یک عملکرد ، شاید با تعدادی وضعیت ها ، جفت شده باشد .

آزمون هدف :

- آزمون هدف : تشخیص می دهد که آیا یک وضعیت داده شده ، یک وضعیت هدف است .
  - شاید یک وضعیت هدف منحصر به فرد وجود داشته باشد یا تعدادی .
  - دنیای جاروبرقی : هر فضای تمیز .
  - در بازی شطرنج – مات

[1] successor functions

- در مساله ی رومانی : در بخارست باشیم

هزینه ی مسیر

- هزینه ی مسیر ، هزینه ی یک عامل برای رفتن از وضعیت اولیه به وضعیتی که به تازگی بررسی شده است می باشد .
- اغلب ، مجموع هزینه برای هر عملکرد می باشد .
  - این هزینه ی مرحله [1] نام دارد
- فرض ما بر این است که هزینه های مراحل ، غیرمنفی هستند .
  - اگر هزینه های مسیر منفی شوند ، چه اتفاقی می افتد ؟

فضای حالت

- ترکیب حالت های مساله و توابع جانشین (راه های رسیدن به وضعیت ها) منجر به مفهوم فضای حالت می شود .
- این گرافی است که همه ی وضعیت های ممکن جهان و انتقال های میان آن ها را ارایه می نماید .
- ما اغلب در مورد اندازه ی این فضاها به صورت یک معیار سختی مساله صحبت می کنیم .
  - پازل ۸- تایی : $\frac{9!}{2} = 181{,}000$ حالت (آسان)
  - پازل ۱۵ – تایی : تقریبا ۱,۳ تریلیون [2] حالت (تقریبا آسان)
  - پازل ۲۴- تایی : تقریبا 10^{25} وضعیت (دشوار)
  - مسافرت شخص دوره گرد : $!۲۰ = 2.43\times10^{18}$ وضعیت (سخت)

عملیات حل مساله

- هزینه ی مسیر
  - هزینه ی عامل را برای اجرای عملیات در یک مسیر تایین می نماید
  - مجموع هزینه های عملیات فردی در یک مسیر
- هزینه ی جستجو
  - زمان و حافظه ی لازم برای محاسبه ی یک راه حل
- مجموع هزینه = هزینه ی مسیر + هزینه ی جستجو
- برای مسایل چند حالته ما از **مجموعه ای از وضعیت ها** استفاده می نماییم
  - هزینه ها و دیگر تعریف ها تغییر نمی کنند

شکل محدود شده ی کلی ، دارای الگوریتم زیر می باشد :
تابع Simple-Problem-Solving-Agent(percept) یک عملکرد را بر می گرداند
متغیر static :
seq ، که یک ترتیب عملکرد است و در ابتدا دارای مقدار تهی می باشد

---

[1] step cost
[2] در انگلیسی برابر است با عدد ۱ که ۱۸ صفر در جلوی آن قرار داده شود . (گرفته شده از Babylon)

state ، توصیف وضعیت فعلی جهان
goal ، یک هدف که به صورت اولیه دارای مقدار NULL می باشد
problem ، یک فرمول بندی مساله
state←Update-State(state,percept)
در صورتی که seq ، خالی است کارهای زیر را انجام بده
goal←Formulate-Goal(state)
problem ← Formula-Problem(state,goal)
seq ← Search(problem)
(پایان شرط)
action ← Recommendation(seq,state)
seq ← Remainder(seq,state)
عمل (action) را برگردان

نکته : این ، راه حلی **برون خطی** [1] می باشد ؛ حل به صورت " چشم بسته " اجرا می شود. راه حل های **برخط** [2] شامل عمل کننده ی بدون دانش تمام می شوند.

استراتژی های جستجو - یک استراتژی با چیدن گره ها به صورت مرتب تعریف می شود. استراتژی ها به وسیله ی موارد زیر ارزیابی می شوند : تمامیت [3] – آیا همیشه ، اگر راه حل موجود باشد آن را پیدا می کند ؟ | پیچیدگی زمانی – تعداد گره های به وجود آمده / توسعه داده شده | پیچیدگی فضا – بیشینه ی تعداد گره های موجود در حافظه | بهینگی - آیا همیشه ، کم هزینه ترین راه را پیدا می کند ؟ | زمان و پیچیدگی فضا توسط پارامترهای زیر ارزیابی می شوند : b – ماکزیمم عوامل منشعب شده از درخت جستجو . d – عمق کم هزینه ترین راه حل. m – ماکزیمم عمق فضای حالت (ممکن است بی نهایت باشد) .

روش های ناآگاهانه [4] - این روش ها فقط از داده های قابل دسترس از تعریف مساله استفاده می کنند.

پیچیدگی

- چرا در مورد پیچیدگی الگوریتم ها نگران هستیم ؟
  - زیرا یک مساله ممکن است در حالت کلی قابل حل باشد اما در حالت عملی خیلی طولانی باشد
- چگونه می توان پیچیدگی الگوریتم ها را ارزیابی نمود ؟
  - با استفاده از تحلیل مجانبی [5] ، تخمین زمان (یا تعداد عملیات) لازم برای حل یک نمونه از اندازه ی n یک مساله در زمانی که n به سمت بی نهایت می رود

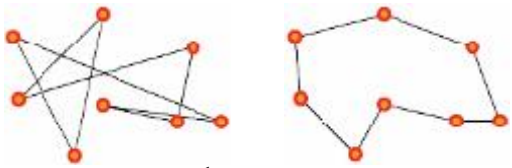
مثال : مساله مسافرت فروشنده ی دوره گرد [6]

- n شهر توسط جاده به هم متصل می باشند

---

[1] offline
[2] online
[3] completeness
[4] uninformed search
[5] asymptotic
[6] Traveling Salesperson Problem (TSP)

- مساله : پیدا کردن یک مسیر که از تمام شهرها عبور کند و حتی الامکان کوتاه ترین مسیر باشد



- مساله ی سخت : فقط الگوریتم های شناخته شده دارای پیچیدگی نمایی هستند
  - تعداد عملیات به صورت نمایی رشد می کند

چرا پیچیدگی نمایی " مشکل " است ؟

این به آن معنی است که تعداد عملیات لازم برای محاسبه ی راه حل دقیق مساله به صورت نمایی با اندازه ی مساله رشد می کند (در زیر تعداد شهر ها آورده شده است) .

- $\exp(1)=2.72$
- $\exp(10)=2.20\times10^{4}$
- $\exp(100)=2.69\times10^{43}$
- $\exp(500)=1.40\times10^{217}$
- $\exp(250{,}000)=10^{108{,}573}$
- در حالی که در کامپیوتر سریع = 10^{12} عملیات در ثانیه را ما داریم

بنابراین ...

در حالت عمومی ، مسایل با پیچیدگی نمایی در همه ی موارد قابل حل نمی باشند اما در اندازه ها ی کوچک نمایی قابل حل می باشند !

پیچیدگی

- مسایل با زمان چندجمله ای [1] (P) : ما می توانیم الگوریتم هایی را پیدا کنیم که این مسایل را در یک زمان که به صورت چندجمله ای با اندازه ی ورودی رشد می کنند حل کنیم .
  - برای مثال : مرتب کردن n عدد به صورت صعودی : الگوریتم های ضعیف این کار را با پیچیدگی n^2 انجام می دهند ، الگوریتم های بهتر این کار را در $n \log(n)$ انجام می دهند .
  - ما نمی توانیم وضعیت مرتبه ی چند جمله ای را مشخص کنیم ، شاید خیلی بزرگ باشد !
- آیا الگوریتم هایی که به بیش از زمان چندجمله ای نیاز داشته باشند هم وجود دارند ؟
  - برای برخی از مسایل ، ما هیچ الگوریتم چند جمله ای نداریم
  - میان این ها مسایل غیرقطعی چندجمله ای [2] وجود دارند

---

[1] polynomial-time
[2] nondeterministic-polynomial-time(NP)

- برای مثال : مساله ی مسافرت فروشنده ی دوره گرد

نکته ای در مورد مسایل NP-hard

- یک مساله به صورت چندجمله ای غیرقطعی است اگر
  - یک راه حل برای آن مساله پیدا شده باشد (گمان)
  - الگوریتمی وجود داشته باشد که بتواند در زمان چند جمله ای باشد چه گمان درست باشد و چه نباشد
- سختی [1] : هیچ الگوریتم چندجمله ای که بتواند مساله را حل کند وجود نداشته باشد
  - برای نمایش آسان نمی باشد و معمولا قابل کاهش به صورت مسایل شناخته شده نمی باشد
- در عمل ، الگوریتم ها زمان بیش تر از چند جمله ای برای حل مساله دارند (با فرض این که $P \neq NP$)

پیچیدگی : تحلیل مجانبی

- اندازه ی ورودی مساله : n
- تعداد عملیات برای حل مساله : f(n)
- یک تابع ارایه شده g(n)
- تعریف f(n) ، در صورتی که $f(n) \leq k\, g(n)$ باشد ، O(g(n)) می باشد
  - برای برخی از مقادیر k
  - برای هر $n > n_0$
- مثال : $O(n^2)$ بدتر از O(n) می باشد اما ...
  - برای $n<110$ داریم $n^2 + 1 < 100n + 1000$

یادآوری : مسایل خوب تعریف شده

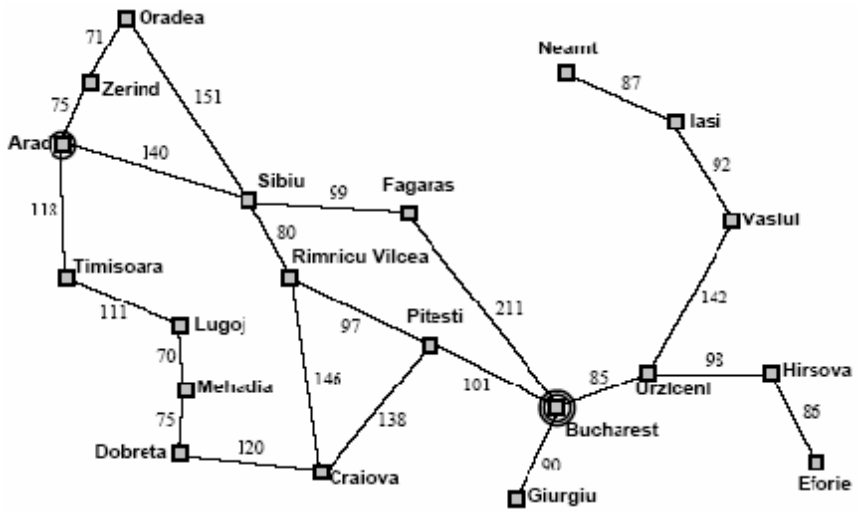
- وضعیت اولیه
  - نقطه ی شروعی که عامل از آن آغاز می نماید
- فضای حالت
  - مجموعه ی همه ی وضعیت های قابل دسترس از وضعیت اولیه توسط هر رشته از عملیات
  - مسیر رشته ای از عملیات مرتبط از یک وضعیت به وضعیت دیگر می باشد
- عملیات (عملگرها ، توابع جانشین)
  - تشریح مجموعه ای از عملیات ممکن
- آزمون هدف
  - این مطلب را که آیا وضعیت ارایه شده یک وضعیت هدف است را تشخیص می دهد
- راه حل

[1] hardness

- مسیری از یک وضعیت اولیه به یک وضعیت هدف

یادآوری : جستجو برای راه حل ها

- پیمایش فضای جستجو
  - از وضعیت اولیه به وضعیت نهایی (هدف)
  - رشته ای مجاز از عملیات به صورت تعریف شده توسط توابع جانشین (عملگرها)
- روال عمومی
  - امتحان کردن برای وضعیت هدف
  - وضعیت جاری را توسعه دهید
    - تشخیص مجموعه وضعیت های قابل دسترسی
    - عدم موفقیت را در صورتی که مجموعه خالی بود برگردان
  - از مجموعه ی وضعیت های قابل دسترس یکی را انتخاب کن
  - به وضعیت انتخاب شده برو
- یک درخت جستجو ساخته می شود
  - گره ها به صورت وضعیت های بیش تر که ملاقات می شوند اضافه می شوند

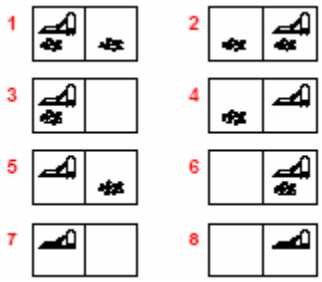


مثال : رومانی [1] - در یک تعطیلی در کشور رومانی ؛ در حال حاضر در شهر آراد [2] هستیم. هواپیما فردا آراد را به مقصد شهر بخارست [3] ترک می کند.

هدف فرمول بندی [4] **:** در شهر بخارست باشیم.

فرمول بندی مساله [5] **:** حالت ها : شهرهای مختلف و عملکردها : حرکت بین دو شهر

پیداکردن راه حل : ترتیب شهرها:Arad و Sibiu , Fagaras , Bucharest



مثال : دنیای جاروبرقی -

وضعیت های عامل جاروبرقی :

- تمام ترکیب های دریافت ها (درک ها) ، زیرا محیط کاملا قابل مشاهده می باشد .
- وضعیت های ۷ و ۸ وضعیت های هدف می باشند (کثیف نمی باشند)
- برای هر عمل هزینه برابر ۱ می باشد .

حالت تکی ، در شماره ی پنج شروع می شود . راه حل ؟؟ [Right,Suck] . ترکیب ، شروع در {1,2,3,4,5,6,7,8} راست به {2,4,6,8} می رود . راه حل ؟؟ [Right,Suck,Left,Suck] احتمال ، در شماره ی پنج شروع می شود . قانون مرفی [6] : مکیدن (عمل مکش جاروبرقی) می تواند یک فرش تمیز را کثیف کند .حس کننده ی محلی [7] : کثیف و فقط تایین محل . راه حل ؟؟ [اگر کثیف بود مکش کن ، راست]

فرمول بندی مساله ی تک حالته

یک مساله توسط چهار عنصر زیر تعریف می شود :

---

[1] Romania
[2] Arad
[3] Bucharest
[4] Formulate goal
[5] Formulate problem
[6] Murphy's Law
[7] local sensing

حالت اولیه - در شهر Arad هستیم .

تابع جانشین ، S(x) = مجموعه ای از جفت حالات عملکرد ، به عنوان مثال ؛ S(Arad)={<Arad → Zerind,Zerind>,…}

آزمایش هدف ، می تواند ، صریح [1] باشد مثلا ، " در بخارست " =x یا غیرصریح [2] باشد ، مثلا ، NoDirect(x) باشد .

هزینه ی مسیر - مجموعه ای از فاصله ها ، تعدادی از عملیات اجرا شده و ..... . c(x,a,y) یک هزینه ی مرحله [3] است ، طبق پیش فرض باید بزرگ تر یا مساوی صفر باشد یا c(x,a,y)>=0 . یک راه حل ، ترکیبی از عملیات است . سیر آن ، از حالت اولیه به یک حالت هدف است.

انتخاب یک فضای حالت (وضعیت) - دنیای واقعی خیلی پیچیده است ، در نتیجه فضای حالت باید از راه حل مساله جدا شود . حالت (مجرد) = مجموعه ای از حالت های واقعی است . عملکرد (مجزا) = مخلوطی پیچیده از عملکردهای واقعی ، مثلا ، "Arad → Zerind" . مجموعه ای پیچیده از مسیر های ممکن ، انحراف ها ، توقف ها و ..... را ارایه می کند. برای تحقق پذیری ، هر حالت واقعی در شهر آراد باید برای بعضی حالت های واقعی در شهر زرینند [4] ضمانت شود. راه حل (مجزا) = مجموعه ای از مسیرهای واقعی که راه حل هایی در دنیای واقعی هستند . هر عمل مجزا باید " آسان تر " از مساله ی اصلی باشد.

جستجو در فضای حالت

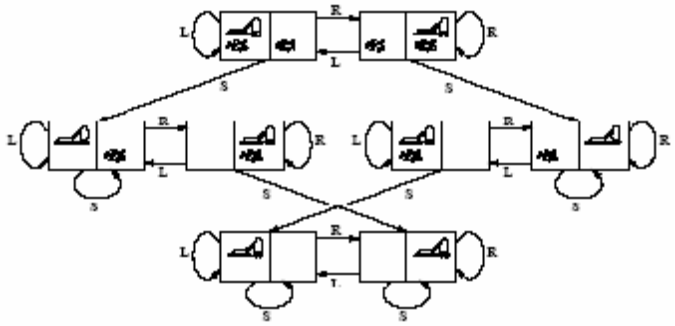
- بیش تر مسایل جستجو خیلی بزرگ تر از آن هستند که در حافظه جا گیرند
- ما یک درخت جستجو را با شروع از وضعیت اولیه و به کارگیری مکرر تابع جانشین ، تولید می نماییم .
- ایده ی اصلی : از یک وضعیت ، توجه کنید که چه چیزهایی می تواند انجام شود . سپس توجه کنید که از هریک از وضعیت های آن ها چه چیزهایی می توانند انجام بگیرند
- برخی از سوالات :
  - آیا پیداکردن راه حل بهینه ، ضمانت شده است ؟
  - تا کی ، جستجو انجام خواهد شد ؟
  - به چه مقدار از فضا نیاز خواهیم داشت ؟

مثال : گراف فضای حالت جاروبرقی

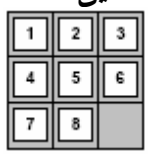
وضعیت ها ؟؟ کثیفی صحیح و جای روبوت (بدون توجه به میزان کثیفی و .....)
عملکردها ؟؟ چپ (Left) ، راست (Right) ، مکش (Suck) ، بدون عملکرد (NoOp)
آزمون هدف ؟؟ بدون کثیفی
هزینه ی مسیر ؟؟ در هر عملکرد یک بار (صفر برای بدون عملکرد (NoOp))

---

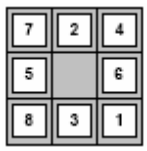
[1] explicit
[2] implicit
[3] step cost
[4] zerind



مثال : پازل هشت تایی



وضعیت هدف

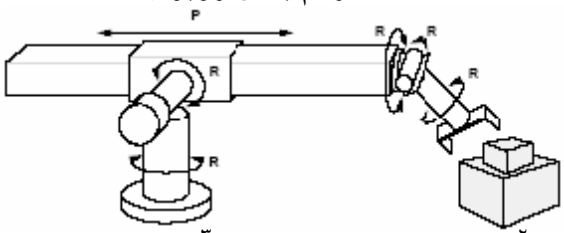


وضعیت شروع

حالت ها ؟؟ کاشی ها (بدون توجه به وضعیت های میانی)
عملکردها ؟؟ حرکت کردن در کاشی های خالی چپ ، راست ، بالا ، پایین
آزمایش هدف ؟؟ = وضعیت هدف (داده شده)
هزینه ی مسیر ؟؟ در هر حرکت یک بار

نکته : راه حل معمولی پازل n – تایی NP-hard می باشد. مسایل NP-hard مسایلی هستند که در یک زمان چند جمله ای نمی توانند حل شوند . [1]

مثال : سرهم بندی روبوتیک



حالت ها ؟؟ مختصات [2] مقدار دهی صحیح شده ی [3] تکه های متصل شده ی روبوت برای سرهم بندی
عملکردها ؟؟ حرکات [4] پیوسته ی [5] اتصالات [6] روبوت
آزمایش هدف ؟؟ کامل کردن سرهم بندی **بدون این که ربوت را شامل شود!**
ارزیابی مسیر ؟؟ زمان اجرا کردن

تعریف : **لیستی از گره ها که تولید شده اند اما هنوز توسعه داده نشده اند را fringe می نامیم .**

---

[1] lorrie.cranor.org/pubs/diss/node1.html
[2] coordinates
[3] real-valued
[4] motions
[5] continuous
[6] joints

الگوریتم های جستجوی درختی - ایده ی اصلی : برون خطی می باشد ، اکتشاف شبیه سازی شده از حالت ها به وسیله ی تولیدکننده های جانشین از حالت های تازه کشف شده .

الگوریتم :

تابع TREE-SEARCH(problem , sterategy) یک راه حل یا عدم موفقیت را برمی گرداند .

در ابتدا درخت جستجو از حالت اولیه ی مساله استفاده می کند .

در حلقه

اگر کاندید یا داوطلبی برای توسعه نبود عدم موفقیت را برگردان

یک گره [1] برگ [2] را با توجه به استراتژی یا روش انتخاب کن

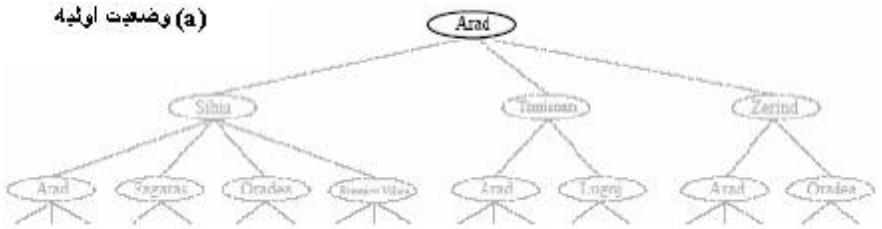
اگر محتوای گره یک حالت هدف بود ، یک راه حل متناظر را برگردان

در غیراین صورت گره را توسعه بده و گره های منتج را به درخت جستجو اضافه کن

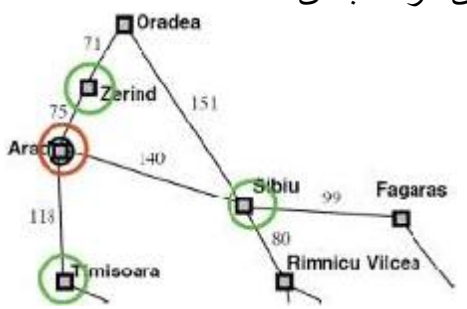
پایان حلقه

مثال جستجوی درختی

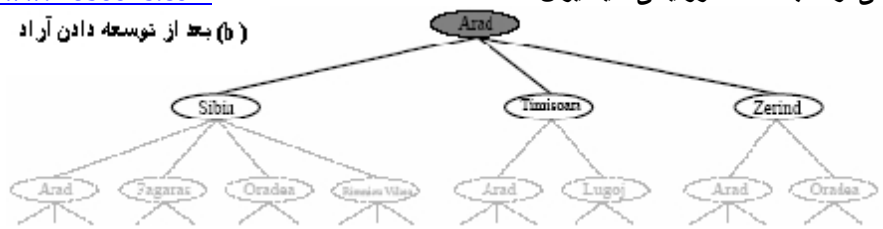
- **درخت جستجو**
  - گراف جستجو : یک وضعیت می تواند از چند مسیر قابل دسترسی باشد
  - ریشه یک گره جستجو متناظر با وضعیت اولیه (آراد) می باشد



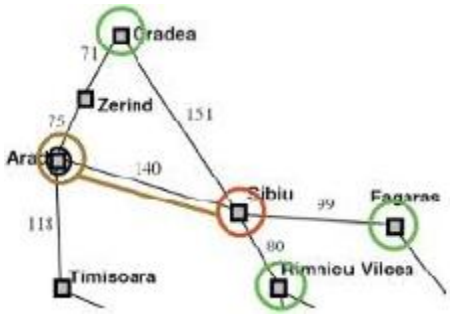
- شهرهای متصل شده به وضعیت اولیه :
  - سیبیو [3]
  - تیمیسوارا [4]
  - زرینند
- کاندیدهای حرکت بعدی

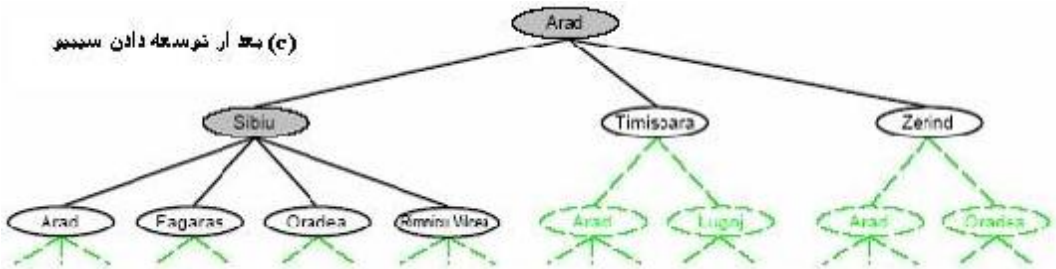


[1] node
[2] leaf
[3] Sibiu
[4] Timisoara



- وضعیت فعلی
- کاندیداهای حرکت بعدی :
  - آراد
  - فاگاراس [1]
  - اُرادیا [2]
  - ریمنیکو ویلک [3]





درخت ها

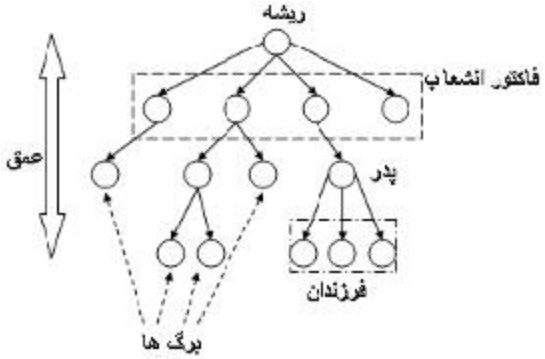
- یک درخت ، یک گراف [4] بدون حلقه ی [5] مستقیم [6] می باشد
  - گراف مستقیم : مجموعه ای از گره های متصل شده توسط لبه ها (یال ها) [7]
    - مسیر : رشته ای از لبه ها (که امکان دارد تکرار هم شده باشند)
  - متصل شده : یک مسیر متصل کننده ی هر جفت از گره ها وجود دارد

---

[1] Fagaras
[2] Oradea
[3] Rimnicu Vilc
[4] graph
[5] acyclic
[6] direct
[7] edges

- بدون حلقه : حداکثر یک مسیر متصل کننده ی هر جفت از گره ها وجود دارد
- درخت ها به صورت گسترده ای در علم کامپیوتر استفاده می شوند (رفتار الگوریتمی خوبی دارند)
- درخت ها دارای اصل محلیت می باشند
- درخت ها دارای قدرت زیاد برای تعریف الگوریتم های بازگشتی می باشند

در زیر تصویری از یک درخت را مشاهده می نمایید



- گره ها حداکثر دارای یک پدر هستند
- ریشه ، گره ای متمایز است که دارای پدری نمی باشد
- عمق می تواند نامحدود باشد (برای خاتمه ی الگوریتم مهم می باشد)

الگوریتم جستجوی درختی

- فضای جستجو را با تولید جانشین های وضعیت های به وجود آمده به وجود می آورد
- توسعه ی وضعیت : تولید یک درخت جستجو

الگوریتم

تابع General-Search(problem,strategy) یک راه حل یا عدم موفقیت را به وجود می آورد

درخت جستجو را با استفاده از حالت اولیه ی مساله مقداردهی اولیه می نماید

در حلقه کارهای زیر را انجام بده

در صورتی که داوطلبی برای توسعه وجود ندارد ، عدم موفقیت را برگردان

یک گره برگ را برای توسعه با توجه به روش انتخاب نما

در صورتی که گره شامل یک وضعیت هدف می باشد راه حل متناظر را برگردان

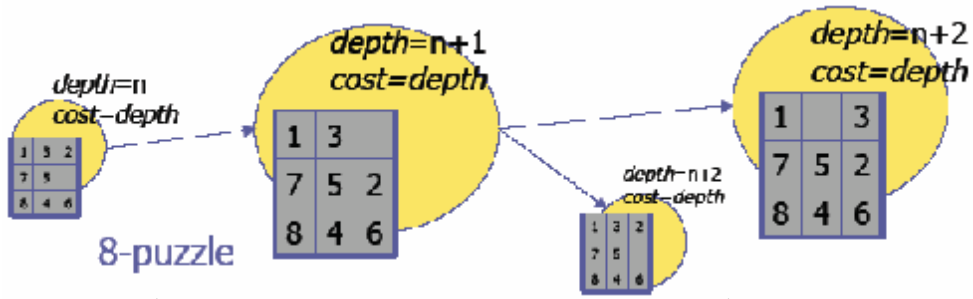
در غیر این صورت گره را توسعه بده و گره های نتیجه شده را به درخت جستجو اضافه نما

پایان الگوریتم

روش : روش جستجو توسط نظمی که براساس آن گره ها توسعه داده می شوند مشخص می شود .

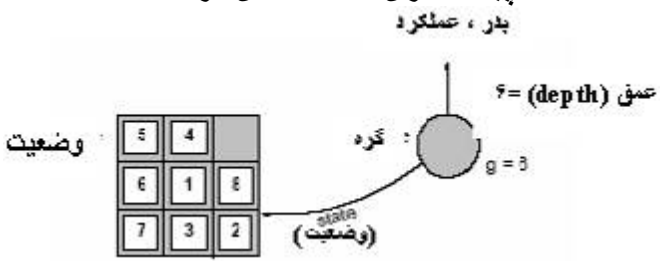
وضعیت ها و گره های درخت

- وضعیت : ارایه ی یک پیکربندی فیزیکی
- گره : ساختمان داده ای در یک درخت جستجو است
  - پدر ، فرزندان ، عمق ، هزینه ی مسیر
  - وضعیت
- وضعیت ها پدران ، فرزندان ، عمق ، یا هزینه ی مسیر ندارند !



- توسعه گره های جدید را با استفاده از وضعیت + عملگر مساله برای ساختن وضعیت های بعدی می سازد

پیاده سازی : حالت های گره ها



یک حالت ، یک پیکربندی فیزیکی (نمایشی از یک پیکربندی فیزیکی) است . یک گره ، ساختمان داده ای است که جزء اصلی یک درخت جستجو ، شامل والد [1] ، فرزندان [2] ، عمق [3] و ارزیابی مسیر g(x) است. حالت ها یا وضعیت ها ، والد ، فرزند ، عمق یا ارزیابی مسیر ندارند.

تابع EXPAND گره های جدید را می سازد و فیلدهای مختلف پر می کند و از SuccessorFn مساله برای ساختن حالت های مورد نظر استفاده می کند.

پیاده سازی : جستجوی درختی عمومی

الگوریتم

تابع Tree-Search(problem,fringe) یک راه حل و یا اشکال را برمی گرداند .

fringe ← Insert(Make-Node(Initial-State[problem]),fringe)

کارهای زیر را انجام بده (loop do)

---

[1] parent
[2] children
[3] depth

در صورتی که fringe خالی است ، عدم موفقیت را برگردان
node ← Remove-Front(fringe)
در صورتی که Goal-Test(problem,State(node)) ، گره را برگردان
fringe ← InsertAll(Expand(node,problem),fringe)

تابع Expand(node,problem) مجموعه ای از گره ها را برمی گرداند
یک مجموعه ی خالی ← successors
برای هر عملکرد و نتیجه در Successor-Fn(problem,State[node]) کارهای زیر را انجام بده
یک گره ی (Node) جدید ← s
Parent-Node[s] ← node ; Action[s] ← action ; State[s] ← result
Path-Cost[s] ← Path-Cost[node]+Step-Cost(node,action,s)
Depth[s] ← Depth[node]+1
S را به successors اضافه کن
Successors را برگردان

پیاده سازی الگوریتم های جستجو

تابع General-Search(problem,Queuing-Fn) یک راه حل یا عدم موفقیت را برمی گرداند
fringe ← make-queue(make-node(initial-state[problem]))
در حلقه کارهای زیر را انجام بده
اگر fringe خالی می باشد عدم موفقیت را برگردان
node ← Remove-Front(fringe)
در صورتی که Goal-Test[problem] به کار گرفته شده برای state(node) موفق شد node را برگردان
← Queuing-Fn(fringe,Expand(node,Operators[problem]))
fringe

Queuing-Fn(queue,elements) یک تابع صف بندی است که مجموعه ای از عناصر را به صف وارد می نماید و نظم توسعه ی عناصر را تشخیص می دهد . تنوعات تابع صف بندی ، تنوعات الگوریتم جستجو را به وجود می آورد .

روش های جستجو

- یک روش با انتخاب ترتیبی از توسعه ی گره تعریف می شود
- ارزیابی روش ها
  - تمامیت : اگر راه حلی وجود داشته باشد همیشه روش ، آن را پیدا می نماید ؟
  - پیچیدگی زمانی : تعداد گره های تولید شده / توسعه داده شده
  - پیچیدگی فضایی : بیشینه ی تعداد گره های موجود در حافظه

- بهینگی : آیا همیشه روش ، کم هزینه ترین راه حل را پیدا می کند ؟

- پیچیدگی زمانی و فضایی با موارد زیر ارزیابی می شوند
  - بیشینه ی فاکتور انشعاب درخت جستجو
  - عمق کم هزینه ترین راه حل
  - بیشینه ی عمق درخت جستجو (شاید بی نهایت باشد)

جستجوی نا آگاهانه

- ساده ترین الگوریتم های جستجو ، آن هایی هستند که هیچ اطلاعات اضافی که در شرح مساله وجود دارد را استفاده نمی نمایند .
- ما این روش ها را روش های جستجوی نا آگاهانه می نامیم .
  - برخی اوقات ، این روش ها ، روش ضعیف نامیده می شوند .
- در صورتی که ما دارای اطلاعات اضافی در مورد چگونگی یک وضعیت غیر هدف باشیم ، در این صورت ما می توانیم جستجوی *مکاشفه ای* را انجام دهیم .

روش های جستجوی ناآگاهانه- چنان که گفته شد ، فقط از اطلاعات قابل دسترسی از تعریف مساله استفاده می نمایند . شامل جستجوی اول سطح [1] | جستجوی با هزینه ی یکسان [2] | جستجوی اول عمق [3] | جستجوی با عمق محدود شده [4] | جستجوی عمیق شونده ی تکراری [5] می باشند .

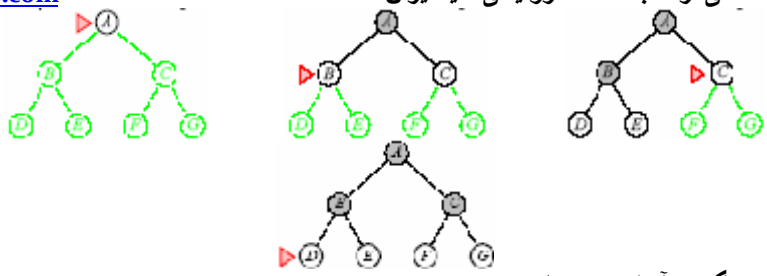
جستجوی اول سطح

- با توسعه دادن یک گره کار می کند ، سپس هر یک از فرزندانش را توسعه می دهد ، سپس ، هریک از فرزندان آن ها را توسعه می دهد و ... .
- همه ی گره های در عمق n ، قبل از یک گره در عمق n+1 ملاقات می شوند .
- ما می توانیم جستجوی اول سطح را با استفاده از یک صف ، پیاده سازی نماییم .

به عبارت دیگر ، جستجوی اول سطح ، کم عمق ترین گره توسعه داده نشده را توسعه می دهد. **پیاده سازی :** ریشه [6] یک صف FIFO [7] می باشد . جانشینان جدید به انتها می روند.

---

[1] Breadth-first
[2] Uniform-cost
[3] Depth-first
[4] Depth-limited
[5] Iterative deeping
[6] fringe
[7] First-In-First-Out



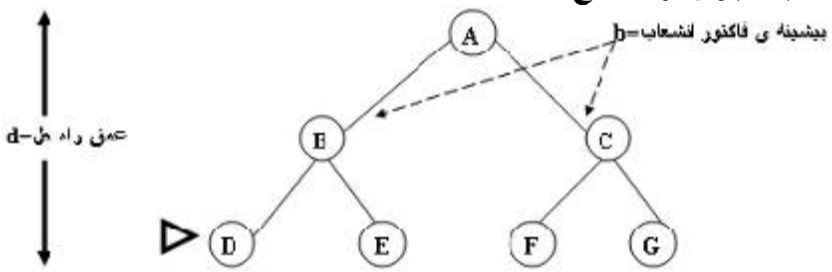
مثال دیگر : آراد به بخارست

- آراد را از صف بردارید
- سیبیو ، تیمیسوارا و زرینند را به صف ، اضافه نمایید
- سیبیو را از صف بردارید و تست نمایید
- ارادیا ، فاگاراس ، ریمنیکو ویلسیا را به صف ، اضافه نمایید
- تیمیسوارا را از صف بردارید و تست نمایید
- لوگاج را به صف ، اضافه نمایید
- ...

برخی نکات ریز

- چگونه ما از ملاقات مجدد آراد جلوگیری نماییم ؟
  - با استفاده از لیست closed : یک لیست از وضعیت های توسعه داده شده را نگهداری نماییم .
  - آیا ما به closed-list در این جا نیاز داریم ؟ راه حل ما یک مسیر می باشد ، نه یک شهر .
- چگونه ما از وارد کردن دوباره ی ارادیا ، اجتناب نماییم ؟
  - open-list : یک لیست از وضعیت هایی که تولید شده اند اما ، توسعه داده نشده اند .
- چرا ما آزمون هدف را در زمانی که فرزندان را تولید کردیم ، به کار نبردیم ؟
  - در واقع ، هیچ تفاوتی ندارد . گره ها ملاقات می شوند و به همان روش یا راه ، تست می شوند .

خصوصیات جستجوی اول سطح



کامل ؟؟ آیا جستجوی اول سطح برای پیداکردن یک راه حل ، ضمانت شده است ؟ بله (در صورتی که b محدود باشد)

توضیح : تصور نمایید که راه حل در عمق n می باشد . از آنجایی که همه ی گره ها یا در عمق n یا در بالای n ، قبل از هر چیزی در عمق n+1 ملاقات می شوند ، یک راه حل ، پیدا خواهد شد .

زمان ؟؟ جستجوی اول سطح ، به زمان اجرای $O(b^{d+1})$ نیاز دارد .

- b ، فاکتور انشعاب : میانگین تعداد فرزندان می باشد .
- جستجوی اول سطح $1+b+b^2+b^3+...+b^d+b(b^d-1)=O(b^{d+1})$ گره را ملاقات خواهد کرد (به d و چگونگی آن در فرمول توجه نمایید)

فضا ؟؟ $O(b^{d+1})$ (هر گره را در حافظه نگهداری می کند)

توضیح : جستجوی اول سطح ، باید همه ی درخت جستجو را در حافظه نگهداری نماید (زیرا ما می خواهیم رشته ی عملکردها را برای رسیدن به هدف ، بدانیم) .

بهینگی ؟؟ (اگر چند راه حل وجود داشته باشد ، آیا جستجوی اول سطح ، بهترین راه حل را پیدا می کند ؟ بله

توضیح :

- جستجوی اول سطح ، کم عمق ترین راه حل در درخت جستجو را پیدا می نماید . در صورتی که هزینه ی مراحل ، یکسان باشند ، این روش بهینه خواهد بود ، در غیر این صورت ، لزوما بهینه نخواهد بود .
- آراد ← سیبیو ← فاگاراس ← بخارست در ابتدا پیدا خواهند شد (فاصله = ۴۵۰)
- آراد ← سیبیو ← ریمنیکو ویلسیا ← پیتستی ← بخارست ، کوتاه تر می باشد . (فاصله = ۴۱۸)

فضا ، مساله ای بزرگ است (چون که باید گره ها در حافظه نگهداری شوند) ؛ فضای زیادی اشغال می کند . می تواند گره ها را با سرعت ۱۰۰ مگابایت در ثانیه [1] تولید کند بنابراین می تواند در بیست و چهار (۲۴) ساعت هشت هزار و ششصد و چهل گیگا بایت (۸۶۴۰) تولید نماید. در کل ، فضای مورد نیاز جستجوی اول سطح ، یک مساله ی بزرگ تر از زمان مورد نیاز می باشد .

جستجوی با هزینه ی یکسان

- به یاد بیاورید که جستجوی اول سطح در زمانی که هزینه ی مراحل ، غیر یکسان باشد ، غیربهینه می باشد .
- ما می توانیم این اشکال را رفع نماییم : اول کوتاه ترین مسیرها را توسعه می دهیم .
- یک هزینه ی مسیر را به گره های توسعه داده شده ، اضافه نمایید .
- از یک صف اولویت برای منظم کردن آن ها به صورت افزایشی بر اساس هزینه ی مسیر ، استفاده نمایید .
- برای پیداکردن کوتاه ترین مسیر ، ضمانت شده است .
- در صورتی که هزینه ها ، یکسان باشند ، این با جستجوی اول سطح ، برابر می باشد .

[1] 100MB/sec

پس ، جستجوی با هزینه ی یکسان ، کم هزینه ترین گره توسعه داده نشده را توسعه می دهد. پیاده سازی : ریشه = صفی با ارزیابی مسیر است که کم ترین در اول صف قرار دارد . **در صورتی که گام ها همه با هم برابر باشند با جستجوی اول سطح برابر است.**

کامل ؟؟ بله ، اگر $\text{cost} \geq \epsilon$.

زمان ؟؟ تعداد گره ها در صورتی که g کوچک تر یا مساوی بهای راه حل مطلوب باشد و $O(b^{\lceil C^*/\epsilon \rceil})$ در جایی که C^* راه حل مطلوب باشد خوب است .

فضا ؟؟ تعداد گره ها در صورتی که مقدار g کوچک تر یا مساوی هزینه (بها) ی راه حل مطلوب باشد و $O(b^{\lceil C^*/\epsilon \rceil})$

بهینگی ؟؟ بله – گره ها را با افزایش مقدار g(n) توسعه می دهد. عمیق ترین گره ی توسعه داده نشده را توسعه می دهد.

پیاده سازی : ریشه = صفی LIFO است و جانشین ها را در جلو قرار می دهد.

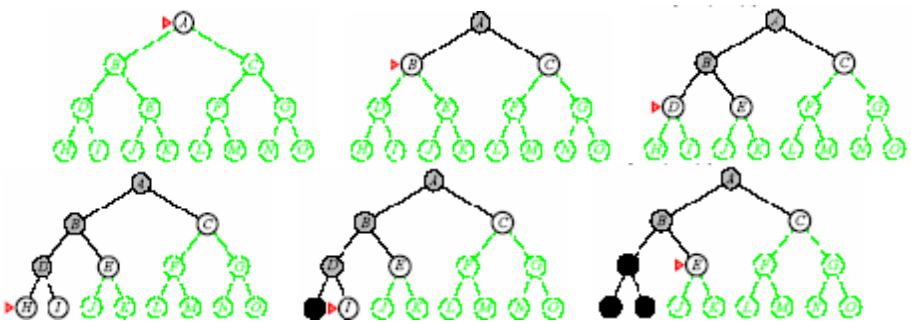
جستجوی اول عمق

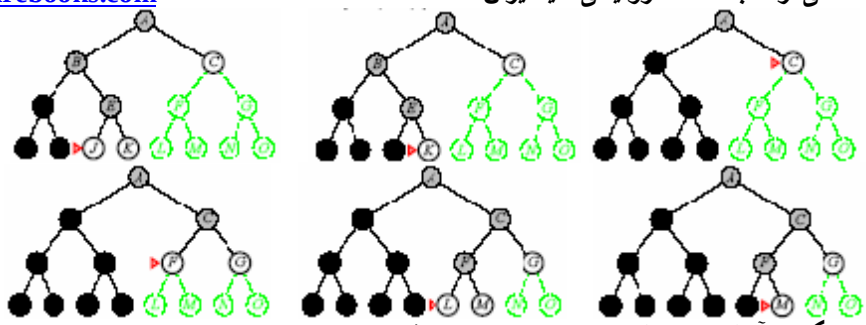
- دارای روش متفاوتی نسبت به جستجوی اول سطح می باشد .
  - همیشه ، عمیق ترین گره را توسعه می دهد .
- یک فرزند را توسعه می دهد ، سپس ، سمت چپ ترین فرزند آن را توسعه می دهد و به همین ترتیب .
- با استفاده از یک پشته هم می توانیم روش جستجوی اول عمق را پیاده سازی نماییم .

شبه برنامه ی این روش را در زیر مشاهده می نمایید :

```
push(initialState)
do
node = pop()
if goalTest(node)
        return node
else
        children = successor-fn(node)
        for child in children
                push(child)
```

پس ، جستجوی اول عمق ، عمیق ترین گره ی توسعه داده نشده را توسعه می دهد. پیاده سازی دیگر : ریشه = صفی LIFO است و جانشین ها را در جلو قرار می دهد.





مثال دیگر : آراد به بخارست (به صورت پشته)

- آراد را از پشته بردارید (pop)
- سیبیو ، تیمیسوارا و زرینند را به پشته اضافه نمایید (push)
- سیبیو را از پشته بردارید و تست نمایید
- ارادیا ، فاگاراس و ریمنیکو ویلسیا را به پشته اضافه نمایید
- ارادیا را ازپشته بردارید و تست نمایید
- فاگاراس را از پشته بردارید و تست نمایید
- بخارست را به پشته اضافه نمایید
- ...

خصوصیات جستجوی اول عمق - کامل بودن ؟؟ نه : در مواردی که عمق نامحدود است و در مورد حلقه ها درست کار نمی کند . بازنگری ، برای جلوگیری کردن از حالت های تکرار شده در طول مسیر منجر به کامل شدن در حالت های محدود می شود . به عبارت دیگر ، ما می توانیم در یک مسیر طولانی ، به صورت نامحدود ، سرگردان باشیم بدون این که به یک راه حل برسیم .

زمان ؟؟ $O(b^m)$. در صورتی که m به مراتب بزرگ تر از d باشد ، بسیار بد است . ولی اگر راه حل ها پیچیده باشند ، این روش به مراتب سریع تر از جستجوی اول سطح می باشد. توجه کنید که m ، بیشینه ی عمق درخت می باشد . همچنین در برخی از موارد ، m ممکن است نامحدود باشد .

فضا ؟؟ O(bm) ، که فضایی خطی می باشد .

- ما فقط نیاز به نگهداشتن شاخه ای که به تازگی جستجو شده است داریم .
- این حسن بزرگ جستجوی اول عمق می باشد .

بهینگی ؟؟ نه . ما ممکن است یک راه حل را در عمق n ، تحت یک فرزند بدون دیدن یک راه حل کوتاه تر ، تحت فرزند دیگر پیدا نماییم . (در مثال قبل که با استفاده از پشته آن را حل کردیم ، اگر اول ریمنیکو ویلسیا را از پشته برمی داشتیم چه اتفاقی می افتاد ؟)

اول ، یک سوال

- چرا ما به الگوریتم هایی که یک جستجوی جامع را انجام می دهند توجه می کنیم ؟ آیا چیزی سریع تر وجود ندارد ؟
- بسیاری از مسایلی که ما به آن ها توجه می کنیم ، NP – کامل می باشند .
  - هیچ الگوریتم با زمان چندجمله ای شناخته شده ای وجود ندارد
  - بدتر از آن ، تعدادی هم غیرقابل تخمین می باشند .

جستجوی اول سطح

- با استفاده از یک صف ، پیاده سازی شده است .

- یک انتقال با مرتبه ی سطحی درخت جستجو را انجام می دهد .
- تمام گره های در سطح n ، قبل از هر گره در سطح n+1 ارزیابی می شوند .
- مزایا :
  - کامل می باشد ، در زمانی که هزینه ی مرحله به صورت یکسان می باشد ، راه حل بهینه را پیدا خواهد نمود
- ضعف ها :
  - به فضایی به صورت نمایی نیازمندیم .

جستجوی اول عمق

- می تواند با استفاده از پشته ، پیاده سازی شود .
- ابتدا یک گره توسعه داده می شود ، سپس فرزندانش توسعه داده می شوند و ...
- در هر لحظه ، فقط یک شاخه از درخت جستجو در حافظه نگهداری می شود .
- مزایا :
  - به فضایی به صورت خطی نیازمندیم .
- معایب :
  - نا کامل و غیربهینه می باشد

جلوگیری از جستجوی نامحدود

- چند روش برای جلوگیری از جستجوی نامحدود اول عمق وجود دارد .
- closed-list
  - ممکن است که همیشه به ما کمک نکند .
  - در این صورت ما باید تعداد زیادی گره را به صورت نمایی در حافظه نگهداری نماییم .
- جستجوی با عمق محدود شده
- جستجوی عمیق کننده ی تکراری جستجوی اول عمق

جستجوی با عمق محدود شده - با محدود نکردن عمق l با جستجوی اول عمق برابر است . گره های در عمق l هیچ جانشینی ندارند.

- جستجوی با عمق محدود شده ، همان جستجوی اول عمق است که در آن عمق جستجو محدود شده است (تا یک عمقی در درخت پایین می رویم و بیش تر از آن پایین نمی رویم)
- عملیات جستجو در این عمق ، متوقف می شود .
- در این جا مشکل دیگری به وجود می آید
  - اگر راه حل در عمقی بیش تر از l باشد ، چطور ؟
  - چگونه ما یک l خوب را به دست آوریم ؟
- در مساله ی رومانی ، ما می دانیم که بیست (۲۰) شهر وجود دارد ، بنابراین l=19 ، یک انتخاب خوب می باشد .
- در مورد پازل ۸ – تایی چه طور ؟

شبه کد پیاده سازی با استفاده از پشته :

```
push(initialState)
```

```
do
node = pop()
if goalTest(node)
        return node
else
        if depth(node) < limit
                children = successor-fn(node)
for child in children
                        push(child)
```

پیاده سازی بازگشتی :

تـابع function Depth-Limited-Search(problem,limit) مقـادیر soln ، fail و یـا cutoff را برمی گرداند

Recursive-DLS(Make-Node(Initial-State[problem]),problem,limit)

تابع function Recursive-DLS(node,problem,limit) مقادیر soln ، fail و یا cutoff را برمی گرداند

cutoff-occurred? ← false

در صورتی که Goal-Test(problem,State[node]) صحیح بود گره (node) را برگردان . (در صورتی که مفدار تابع درست بود node را برگردان ;)

در غیر این صورت ، اگر Depth[node]=limit بود ، مقدار cutoff را برگردان .

در غیر این صورت ، برای هر successor (جانشین) در Expand(node,problem) کارهای زیر را انجام بده :

result ← Recursive-DLS(successor,problem,limit)

اگر result=cutoff بود سپس مقدار true را در cutoff-occurred بریز .

در غیر این صورت ، اگر result < > failure بود result را برگردان (< > علامت نامساوی می باشد) .

در صورتی که cutoff-occurred صحیح بود cutoff را برگردان و در غیر این صورت failure را برگردان.

پایان الگوریتم

جستجوی عمیق کننده ی تکراری

- بر مبنای ایده ی جستجوی با عمق محدود شده ، توسعه می یابد .
- جستجوی با عمق محدود شده را ابتدا با عمق l=1 ، سپس با عمق l=2 و ... انجام دهید .
- سرانجام ، l=d می شود ،
  - این به این معنی می باشد که جستجوی عمیق کننده ی تکراری ، کامل می باشد .
- اشکال : برخی از گره ها تولید شده اند و چند بار ، توسعه داده شده اند .
- در سطح یک : تعداد b گره ، تعداد d مرتبه تولید می شوند .
- در سطح دو : تعداد b^2 گره ، d-1 بار تولید می شوند .
- ...
- در نتیجه در سطح d ، تعداد b^d یک بار تولید می شوند .

- محموع زمان اجرا برابر ($O(b^d)$ می باشد . که اندکی کم تر از تعداد گره های تولید شده در جستجوی اول سطح می باشد .
- هنوز به حافظه ای به صورت خطی نیازداریم .

پس ، عمق محدودشده را با افزایش محدودیت ها به کار می گیرد .

تـابع Iterative-Deepening-Search(problem) function یـک راه حـل را برمـی گرداند .

ورودی ها probles که یک مساله است ، می باشد

برای depth مساوی با صفر تا بی نهایت (∞) کارهای زیر را انجام بده

result ← Depth-Limited-Search(problem,depth)

در صورتی که result< >cutoff بود result را برگردان ;

پایان حلقه

پایان الگوریتم

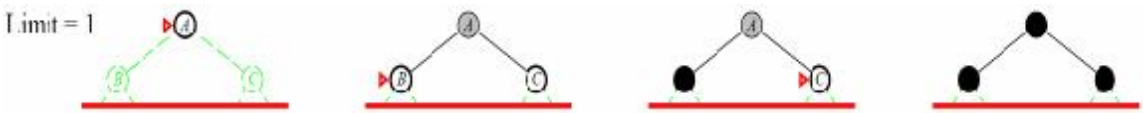
شبه کدی به صورت دیگر برای جستجوی عمیق کننده ی تکراری

```
d = 0
while (1)
result = depth-limited-search(d)
if result == goal
return result
else
d = d + 1
```

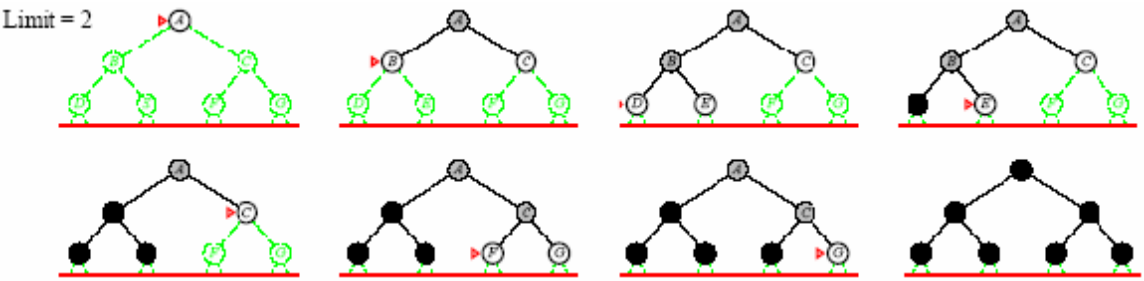
مثال :

جستجوی عمیق شونده ی تکراری با l=0

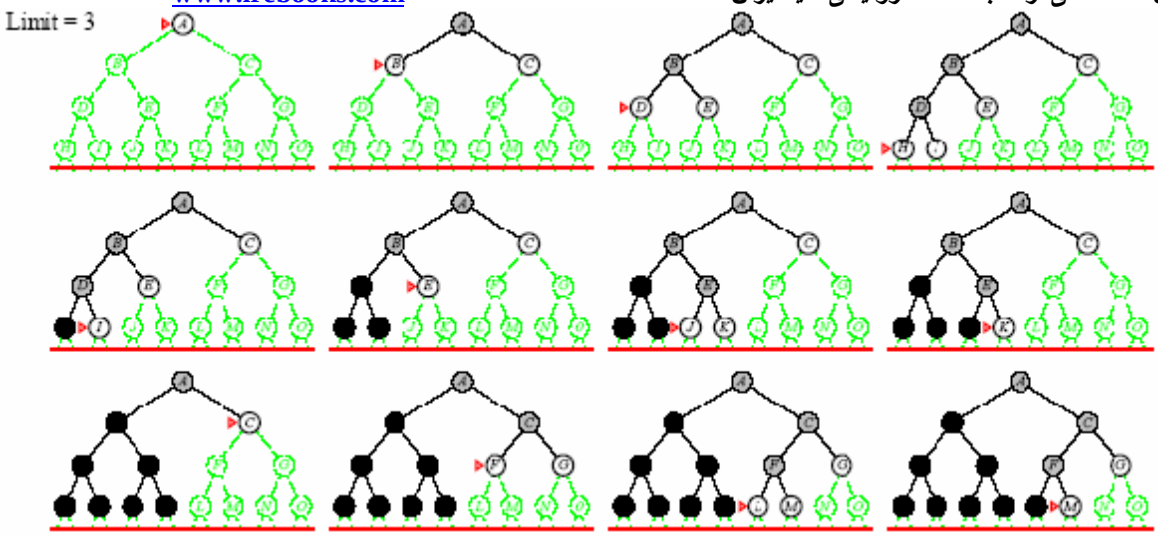
جستجوی عمیق شونده ی تکراری با l=1



جستجوی عمیق شونده ی تکراری با l=2



جستجوی عمیق شونده ی تکراری با l=3



- جستجوی اول سطح و اول عمق را با هم ترکیب می کند
  - لایه ها را شبیه جستجوی اول سطح به وجود می آورد
  - اما در هر اجرا یک اول عمق را انجام می دهد (صرفه جویی در فضا)
- وضعیت ها چند بار توسعه داده می شوند

در واقع ، جستجوی عمیق کننده ی تکراری ، شبیه جستجوی اول سطح می باشد که در آن همه ی گره های در عمق n قبل از هر گره در عمق n+1 بررسی می شوند . نظیر جستجوی اول سطح ، ما می توانیم بهینگی را در جهان های با هزینه ی غیریکسان با توسعه دادن با توجه به هزینه ی مسیر ، ترجیحا با استفاده از عمق ، به دست آوریم . این ، *جستجوی دراز کننده ی طول* [1] نام دارد . همه ی مسیرها را هزینه ی کم تر از p جستجو می کند . p را با δ افزایش می دهد . در جهان های پیوسته ، δ چه باید باشد ؟

جریان معکوس [2]

- در زمانی که جستجوی اول عمق و همتایانش به وضعیت عدم موفقیت می رسند ، چه اتفاقی می افتد ؟
- به بالای والد می روند و برادر بعدی را امتحان می کنند .
- فرض : جدیدترین عملکرد انتخاب شده ، عملکردی است که باعث عدم موفقیت شده است .
- این ، جریان معکوس به ترتیب وقوع [3] نام دارد – جدیدترین چیزی را که انجام داده اید ، بی اثر نمایید .
- این می تواند یک مساله باشد – عدم موفقیت ، شاید یک نتیجه از یک تصمیم گیری قبلی باشد .
  - مثال : ۴ – وزیر
- محدودیت ها می توانند به شما برای محدود کردن اندازه ی فضای جستجو کمک نمایند .

---

[1] iterative lengthening search
[2] backtracking
[3] chronological backtracking

- جریان *معکوس هوشمند* [1] برای تحلیل دلیلی برای عدم موفقیت و غیرفعال کردن جستجو برای آن نقطه تلاش می نماید .
  - می تواند جدیدترین متغیر ناسازگار را غیرفعال نماید (جریان معکوس)
  - همچنین می تواند بررسی مستقیم [2] را انجام دهد – آیا یک انتساب ممکن مقدارها برای متغیرهایی در این نقطه وجود دارد ؟
  -

خصوصیات جستجوی عمیق شونده ی تکراری

کامل بودن ؟؟بله

زمان ؟؟ $(d+1)b^0 + db^1 + (d-1)b^2 + ... + b^d = O(b^d)$

فضا ؟؟ O(bd)

بهینگی؟بله اگر گام cost برابر یک باشد. می تواند برای به وجود آوردن درخت uniform-cost بازنگری شود. به عنوان مثال ، مقایسه ی عددی برای b=10 و d=5 ، برای دورترین برگ سمت راست :

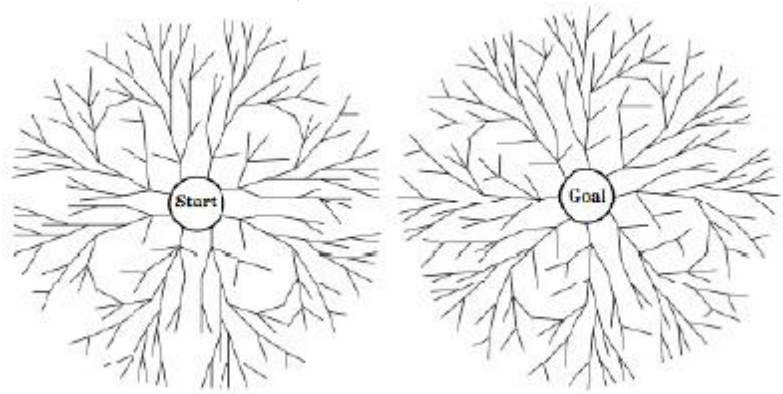
N(IDS) = 50 + 400 + 3000 + 20000 + 100000 = 123450

N(BFS) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100

IDS (جستجوی عمیق شونده ی تکراری) بهتر عمل می کند ، چون که گره های دیگر درون عمق d توسعه داده نمی شوند. BFS (جستجوی اول سطح) می تواند برای مدت زمانی که یک گره تولید می شود بازنگری شود.

جستجوی دوطرفه [3]

- دو جستجوی شبیه را اجرا می نماید
  - یکی در جهت وضعیت اولیه و دیگری در خلاف جهت و از طرف وضعیت هدف
  - در زمانی که دو جستجو به هم می رسند جستجو خاتمه می یابد



خصوصیات جستجوی دو طرفه

- در صورتی که هر دو جستجو اول سطح باشند کامل است
- زمان : $O(b^{d/2})$
- فضا : $O(b^{d/2})$

[1] intelligent backtracking
[2] forward checking
[3] bidirectional search

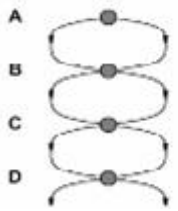
- در صورتی که هر دو روش اول سطح باشند بهینه می باشد
- اشکالات [1]
  - تولید قبلی ها (معکوس کردن عملگرها)
  - هماهنگی میان دو جستجو
  - یک جستجو باید گره ها را در حافظه نگهداری نماید (برای بررسی این که آیا گره ها قبلا ملاقات شده اند یا نه)

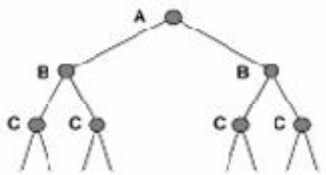
خلاصه ی الگوریتم ها

مقیاس [2]	اول سطح	با هزینه س یکسان	اول عمق	عمق محدود شده	عمیق شونده ی تکراری
کامل بودن ؟؟	بله *	بله *	نه	بله ، در صورتی که l>=d باشد	بله
زمان ؟؟	b^d+1	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
فضا ؟؟	b^d+1	$b^{\lceil C^*/\epsilon \rceil}$	b m	bl	bd
بهینگی ؟؟	بله *	بله	نه	نه	بله *

b : فاکتور انشعاب
d : عمق کم عمق ترین راه حل
m : بیشینه ی عمق درخت جستجو
l : محدودیت عمق

حالت (وضعیت) های تکرار شده - عدم موفقیت در پیداکردن حالت های تکرار شده می تواند یک مساله ی خطی را تبدیل به یک مساله ی نمایی کند .





- توسعه ی وضعیت هایی که به تازگی ملاقات شده اند
  - بیش از یک مسیر برای وضعیت یکسان وجود دارد
  - عملیات قابل برگشت هستند به عنوان مثال پیداکردن مسیر و پازل ۸ تایی
- راه حل : نگهداری تاریخچه

[1] drawbacks
[2] Criterion

- گره های ملاقات شده (توسعه داده شده) در closed list نگهداری می شوند
- گره جاری اگر به تازگی در closed list بوده ، حذف می شود

- مسایل
  - بهینگی ممکن است از بین برود (همه ی گره ها در حافظه نگهداری می شوند)

جستجوی گراف

function Graoh-Search(problem,fringe) یک راه حل یا عدم موفقیت را برمی گرداند .
یک مجموعه ی خالی را به closed تخصیص بده .
fringe ← Insert(Make-Node(Initial-State[problem]),fringe)
در حلقه کارهای زیر را انجام بده
در صورتی که fringe خالی باشد عدم موفقیت را بر گردان
node ← Remove-Front(fribge)
در صورتی که Goal-Test(problem,State[node]) موفقیت آمیز بود ، گره (node) را برگردان
در صورتی که State[node] در closed نبود
State[node] را به closed اضافه نما
InsertAll(Expand(node,problem),fringe) را در fringe بریز
پایان الگوریتم

خلاصه - فرمول بندی مساله ، معمولا به در کنار هم قرار دادن جزییات دنیای واقعی برای تعریف یک فضای حالت که بتواند به طور عملی به وجود آید نیاز دارد. تنوع روش ها یا استراتژی های جستجوی ناآگاهانه : جستجوهای عمیق شونده ی تکراری [1] فقط به صورت فضای خطی استفاده می شوند و زمان به مراتب بیش تری نسبت به سایر الگوریتم های ناآگاهانه ندارند. جستجوی گرافی می تواند به مراتب موثرتر از جستجوی درختی باشد .

- فرمول بندی یک مساله ی جستجو
  - وضعیت اولیه
  - آزمون هدف
  - عملیاتی که باید انجام شوند
  - تابع جانشین
  - هزینه ی مسیر
- منجر به جستجو در یک فضای حالت با استفاده از یک درخت جستجو می شود .
- الگوریتم ها :
  - جستجوی اول سطح
  - جستجوی اول عمق
  - جستجوی با هزینه ی یکسان
  - جستجوی با عمق محدود شده
  - جستجوی عمیق کننده ی تکراری

[1] iterative deeping search

فصل چهارم

جستجوی آگاهانه (مکاشفه ای)

جستجوی آگاهانه (مکاشفه ای)
فصل چهارم

خلاصه ی ریوس مطالب
جستجوی اول - بهترین [1]
جستجوی A^*
ابتکارات [2]**(اکتشافات)**

یادآوری : روش های ناآگاهانه

- جستجوی اول سطح (FIFO)
  - جستجوی لایه ای
- جستجوی با هزینه ی یکسان
  - از هزینه ها به جای مراحل استفاده می کند
- جستجوی اول عمق (LIFO)
  - تا جایی که می تواند به عمق می رود
- جستجوی با عمق محدود شده (الگوریتم بازگشتی دارد)
  - تا موقع محدودیت داده شده به عمق می رود
- جستجوی عمیق کننده ی تکراری
  - جستجوهای عمقی محدود شده با افزایش محدودیت ها می باشند
  - می توانند برای یک جستجوی با هزینه ی یکسان بازنگری شوند
- جستجوی دو طرفه (دو جستجوی اول سطح می باشند)

مرور : جستجوی درختی
تابع Tree-Search(problem,fringe) ، یک راه حل یا عدم موفقیت را برمی گرداند
fringe ← Insert(Make-Node(Initial-State[problem]),fringe
کارهای زیر را انجام بده
در صورتی که ریشه خالی است عدم موفقیت [3] را برگردان

[1] Best-first
[2] Heuristics
[3] failure

node ← Remove-Front(fringe)
در صورتی که Goal-Test[problem] به کار گرفته شده برای State(node) موفقیت آمیز بود ، node را برگردان
fringe ← InsertAll(Expand(node,problem),fringe)

آگاهانه کردن جستجو

- الگوریتم های قبلی ، می توانستند راه حل ها را پیدا کنند ، اما ناکارامد بودند .
  - تعدادی نمایی از گره ها ، توسعه داده می شدند .
- با استفاده کردن از دانش در مورد ساختار مساله ، می توانیم کارایی را بهبود ببخشیم .
- دو اخطار : ما باید ، دانش در مورد مساله را از جایی دیگر بگیریم .
- این دانش باید درست باشد .

چرا از دانش در یک مساله استفاده می نماییم ؟

- روش های ناآگاهانه فقط از اطلاعات قابل دسترس از تعریف مساله استفاده می نمایند
  - بدترین حالت پیچیدگی زمانی ، در موارد به صورت نمایی می باشد
  - در اکثر موارد این روش های ناآگاهانه ناکارآمد می باشند
- ما ممکن در صورتی که چیزهایی در مورد مساله بدانیم بهتر عمل کنیم
  - بدترین حالت در مواقع نمایی می باشد
  - اما در بیش تر موارد الگوریتم رفتاری بهتر را از خود نمایش می دهد
- ایده ی اساسی : توسعه ی گره (اکتشاف درخت) توسط یک تابع ارزیابی [1] انجام می شود

تابع ارزیابی

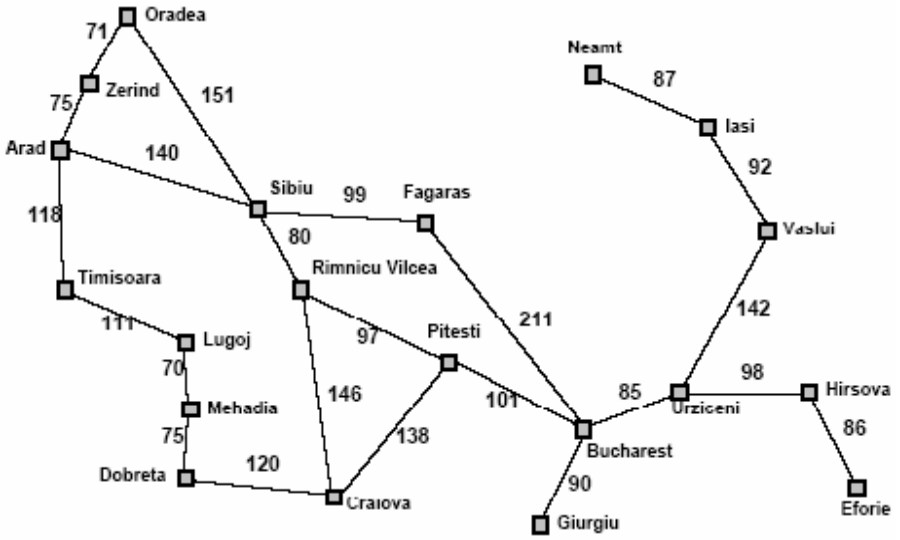
- یک تابع ارزیابی f(n) ، شایستگی یک وضعیت هدف را تخمین می زند
  - یک وضعیت با هر گره درخت جستجو پیوند برقرار می کند
  - بنابراین ، تابع می تواند برای گره های درخت جستجو استفاده شود
- گره ای که دارای کم ترین مقدار ارزیابی است در میان گره های کاندید انتخاب می شود
  - ما به دنبال ارزان ترین راه حل می باشیم
- پیاده سازی : fringe یک صف مرتب شده توسط تابع ارزیابی می باشد
- جستجوی با هزینه ی یکسان را به یاد بیاورید
  - گره ها براساس مجموع هزینه ی مسیر ، توسعه داده می شوند

[1] evaluation function

- یک صف اولویت ، برای پیاده سازی جستجوی با هزینه ی یکسان استفاده می شود .
- هزینه ی مسیر ، مثالی از یک تابع ارزیابی می باشد .
- روش **جستجوی اول – بهترین** [1]
  - اول ، بهترین گره را برای رسیدن به هدف توسعه دهید
- ما در ادامه نمی دانیم که این گره هنوز بهترین است یا نه
  - اگر ما این مطلب را بدانیم ، آن گاه ما نیازی به عمل جستجو نداریم (در صورتی که شما راه خود را بدانید شما بر روی یک طرح کار نخواهید کرد)
- گره ای را که به نظر می رسد بهترین باشد را توسعه دهید
- تابع ارزیابی h(n) می باشد
  - هزینه ی تخمین زده شده از ارزان ترین مسیر از n تا گره هدف

پس ایده ی جستجوی اول – بهترین استفاده از یک تابع ارزیابی می باشد . موارد خاص: جستجوی حریصانه [2] ، جستجوی A^*

مساله ی کشور رومانی با ارزیابی مراحل به صورت کیلومتر -



آراد	۳۶۶	فاگاراس	۱۷۸	مهادیا [3]	۲۴۱	سیبیو	۲۵۳
بخارست	۰	گیورگیو [6]	۷۷	نمت [5]	۲۳۴	تیمیسوارا [4]	۳۲۹
کرایوا [1]	۱۶۰	هیرسوا [9]	۱۵۱	ارادیا [8]	۳۸۰	ارزیسنی [7]	۸۰

[1] best-first search
[2] greedy search
[3] Mehadia
[4] Timisoara
[5] Neamt
[6] Giurgiu
[7] Urziceni
[8] Oradea
[9] Hirsova

۱۹۹	واسلوی [۲]	۹۸	پیتستی [۳]	۲۲۶	یاسی	۲۴۲	دوبرتا [۴]
۳۷۴	زرینـد	۱۹۳	ریمنیکو ویلسیا	۲۴۴	لوگوج [۵]	۱۶۱	افری [۶]

- **توجه کنید اکتشاف (که به صورت جدولی در سمت راست تصویر بالا داده شده است) جزیی از تعریف اولیه ی مساله نمی باشد**
- **در این مورد به صورت یک جدول خارجی داده شده است**

مسافت خطی مستقیم به شهر بخارست

جستجوی حریصانه

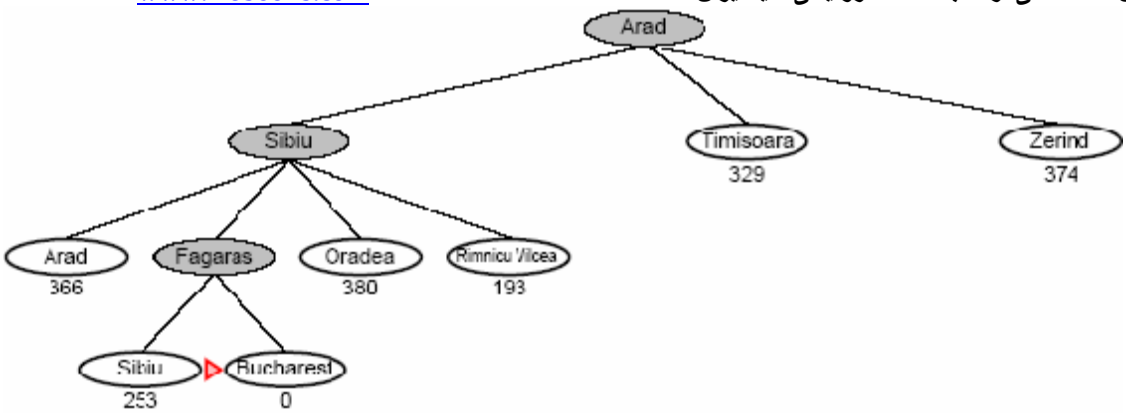
- ارزیابی ، فقط از مکاشفه استفاده می نماید f(n)=h(n)
- الگوریتم ، گرهی را که به نظر می رسد به هدف نزدیک تر است را توسعه می دهد .

تابع ارزیابی h(n) (ابتکاری) = تخمین ارزش از n تا نزدیک ترین هدف

$h_{SLD}(n)$ = فاصله ی خطی مستقیم از n تا شهر بخارست

مثالی از جستجوی Greedy

Craiova [1]
Vaslui [2]
Pitesti [3]
Dobreta [4]
Lugoj [5]
Eforie [6]

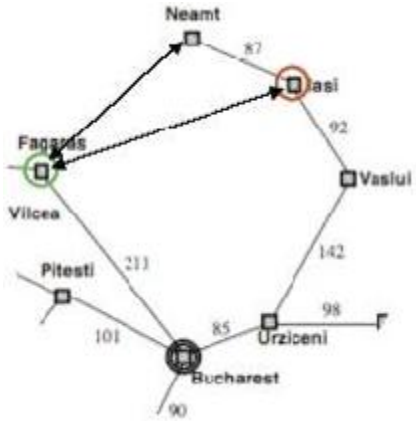


خصوصیات جستجوی حریصانه

کامل بودن ؟؟ نه ممکن است در حلقه ها گیر کند ، مثلا ، با شهر Oradea به صورت هدف ، داریم :

lasi → Neamt → lasi → Neamt →

- به عنوان مثالی دیگر ، از یاسی [1] به فاگاراس
  - میان یاسی و نمت [2] حلقه به وجود می آید
  - راه حل : اجتناب از وضعیت های ملاقات شده



- در فضاهای جستجوی محدود با تست وضعیت تکرار شده **کامل** است

زمان ؟؟ $O(b^m)$ ، اما با یک ابتکار صحیح می توان بهبودی قابل توجهی به دست آورد .

فضا ؟؟ $O(b^m)$ همه ی گره ها را در حافظه نگه می دارد . که b ، فاکتور انشعاب و m عمق درخت جستجو می باشد

بهینگی ؟؟ نه ، یک مسیر را به سوی هدف دنبال می نماید

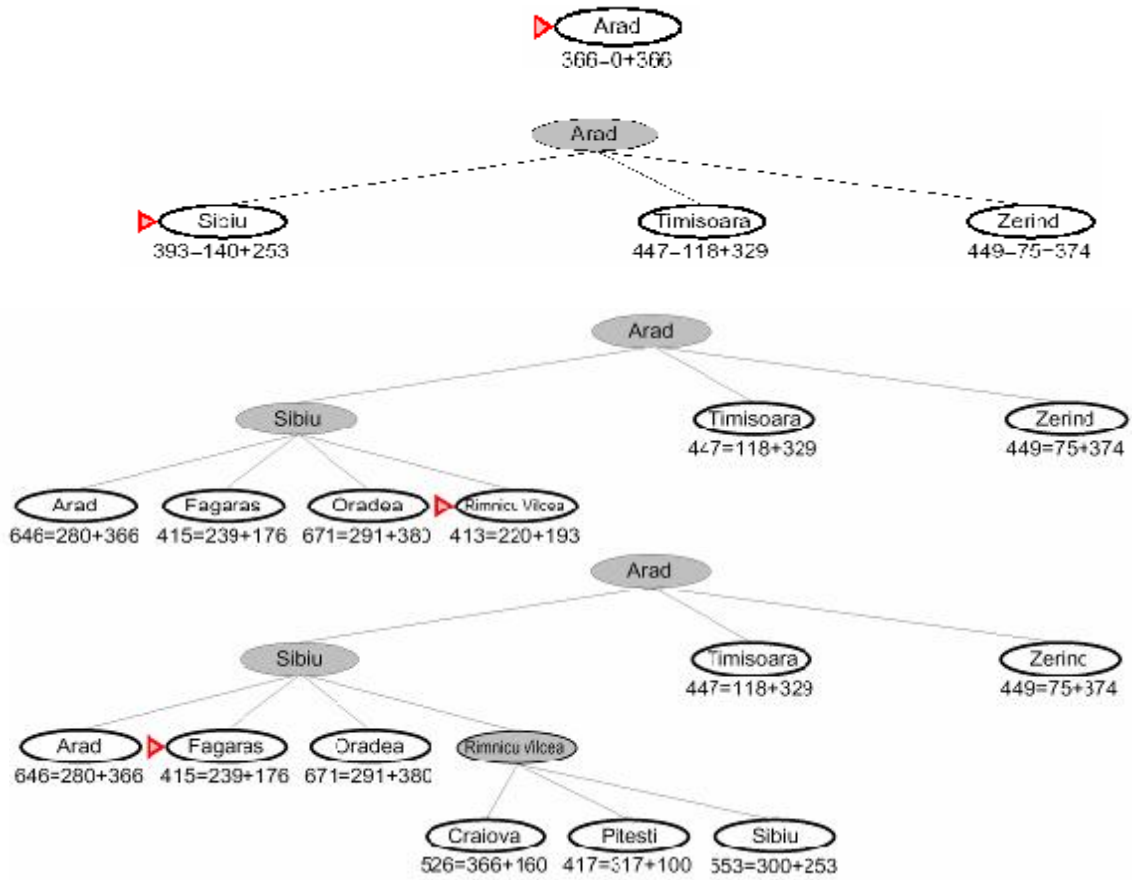
- شبیه اول – عمق می باشد

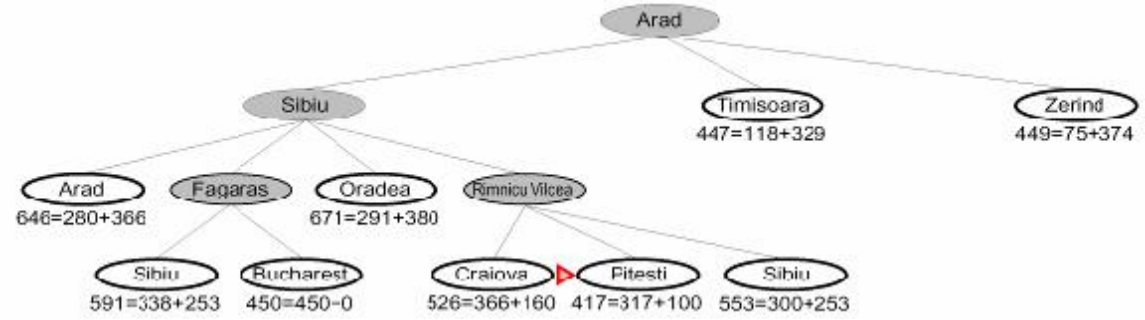
جستجوی A^*

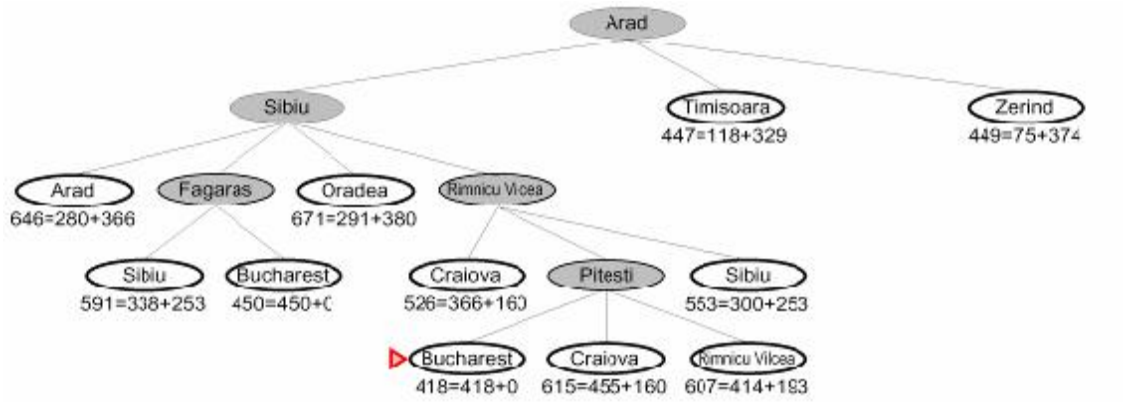
[1] Iasi
[2] Neamt

- مساله ای که در مورد اول – بهترین حریصانه وجود دارد ، این است که هزینه ی مسیر تا کنون مورد توجه قرار نگرفته است .
- ایده : از توسعه دادن مسیرهایی که پرهزینه هستند اجتناب می کند.
- تابع ارزیابی $f(n)=g(n)+h(n)$ می باشد . $g(n)$ = هزینه ای که تا کنون برای رسیدن به n صرف شده است . $h(n)$ = هزینه ی تخمین زده شده برای n . $f(n)$ = کل هزینه ی تخمین زده شده ی مسیر از n تا هدف .
- جستجوی A^* از یک روش ابتکاری مجاز برای جستجو استفاده می کند.
- جستجوی A^* ترکیب جستجوی با هزینه ی یکسان و جستجوی حریصانه می باشد .
- **قضیه : جستجوی A^* ، جستجویی بهینه یا مطلوب است .**

مثالی برای جستجوی A^*







بهینگی جستجوی A^*

- برای تضمین بهینگی مکاشفه نباید هزینه ی رسیدن به هدف را زیاد برآورد نماییم
- مکاشفه ی قابل قبول $h(n) \le h^*(n)$ ($h^*(n)$ هزینه ی واقعی رسیدن به هدف می باشد)
- در مسیریابی ، فاصله ی خطی مستقیم قابل قبول می باشد
- قابلیت قبول بهینگی را فقط با الگوریتم جستجوی عمومی تضمین می کند
  - تست w/o برای وضعیت های تکرار شده انجام می شود

الگوریتم

تابع General-Search(problem,Queuing-Fn) یک راه حل یا عدم موفقیت را برمی گرداند

fringe ← make-queue(make-node(initial-state[problem]))
در حلقه کارهای زیر را انجام بده
در صورتی که fringe خالی می باشد عدم موفقیت را برگردان
node ← Remove-Front(fringe)
در صورتی که Goal-Test[problem] به کار گرفته شده برای وضعیت (گره) موفق شد ، گره را برگردان
fringe ← Queuing-Fn(fringe,Expand(node,Operators[problem]))

تابع Graph-Search(problem,Queuing-Fn) یک راه حل یا شکست را برمی گرداند
closed ← Φ

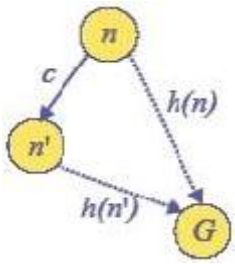
fringe ← make-queue(make-node(initial-state[problem]))
در حلقه کارهای زیر را انجام بده
در صورتی که fringe خالی است عدم موفقیت را برگردان
node ← Remove-Front(fringe)
در صورتی که Goal-Test[problem] به کار رفته برای State(node) موفقیت آمیز بود node را برگردان
در صورتی که $node \notin closed$ آن گاه
node را به closed اضافه نما
Queuing-
fringe ← Fn(fringe,Expand(node,Operators[problem]))

سازگاری (یکنوایی) اکتشافات

- راه حل : مطمین شوید که مسیر بهینه ی وضعیت های تکرار شده اولین باری است که توسعه داده می شود
- این مورد می تواند با انتخاب اکتشاف صحیح انجام شود
- سازگاری (یکنوایی)

$h(n) \leq c + h(n')$

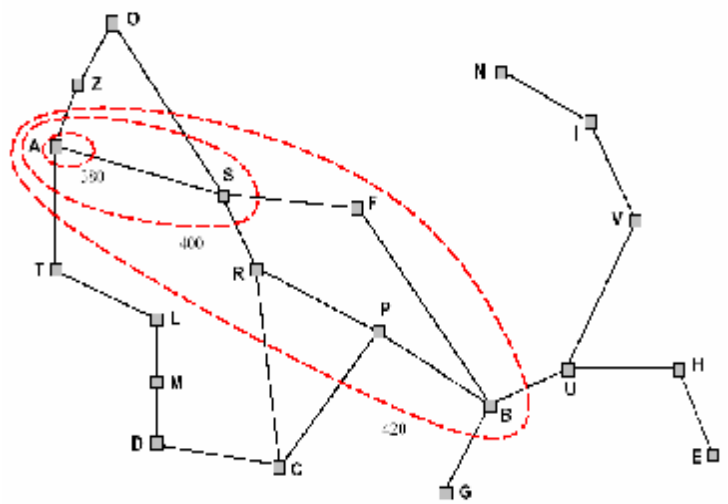
- نا برابری مثلثی



$f(n') = g(n') + h(n') = g(n) + c + h(n') \geq g(n) + h(n) = f(n)$

- ارزیابی در طول هر مسیر غیرکاهشی می باشد

بهینه سازی A^* (مفیدتر) - مقدمه : A^* گره ها را با نظم افزایشی مقدار f توسعه می دهد. به تدریج شکل f (f-contours) گره ها را افزایش می دهد . شکل i دارای همه ی گره های با $f = f_i$ در جایی که $f_i < f_{i+1}$ می باشد.



خصوصیات

A^*

کامل بودن ؟؟

بله ، مگر این که تعداد نامحدودی گره با f(n)<=f(G) وجود داشته باشد.

زمان ؟؟به صورت نمایی می باشد
فضا ؟؟تمام گره ها را در حافظه نگه می دارد و به صورت نمایی می باشد
بهینه ؟؟بله – تا زمانی که f_i تمام نشده است نمی تواند f_{i+1} را توسعه بدهد.

- تصور نمایید C^* هزینه ی مسیر راه حل بهینه باشد
  - A^* تمام گره های با $f(n) < C^*$ را توسعه می دهد.
  - A^* بعضی از گره هایی که $f(n) = C^*$ است را توسعه می دهد.
  - A^* گره هایی که $f(n) > C^*$ است را توسعه نمی دهد.
- **هرس** [1] **:** حذف بدل ها بدون امتحان
- **A^* به طور موثری برای هر تابع ارزیابی بهینه می باشد**
  - در میان الگوریتم هایی که جستجو را از ریشه توسعه می دهند
  - الگوریتم بهینه ی دیگری برای توسعه ی تعداد گره ها ی کم تر ضمانت نشده است
  - یک الگوریتم ممکن است که راه حل بهینه را در صورتی که همه ی گره های با $f(n) < C^*$ را توسعه ندهد ، گم کند

بهترکردن A^*

- A^* یکی از الگوریتم های کلیدی در هوش مصنوعی می باشد
- اما دارای اشکال در حافظه ی مورد نیاز می باشد
  - تمام گره های ملاقات شده را در حافظه نگه می دارد
  - برای مسایل با ساختار بزرگ ، عملی نمی باشد
- ایده ی اساسی : به کار گیری ایده ی عمیق شونده ی تکراری
  - شبیه مورد اول عمق
  - از تابع ارزیابی (f) به جای عمق استفاده می نماید
- A^* عمیق شونده ی تکراری [2]
  - جستجوی اول بهترین بازگشتی [3]
  - A^* با حافظه ی محدود ساده شده [4]

A^* عمیق شونده ی تکراری

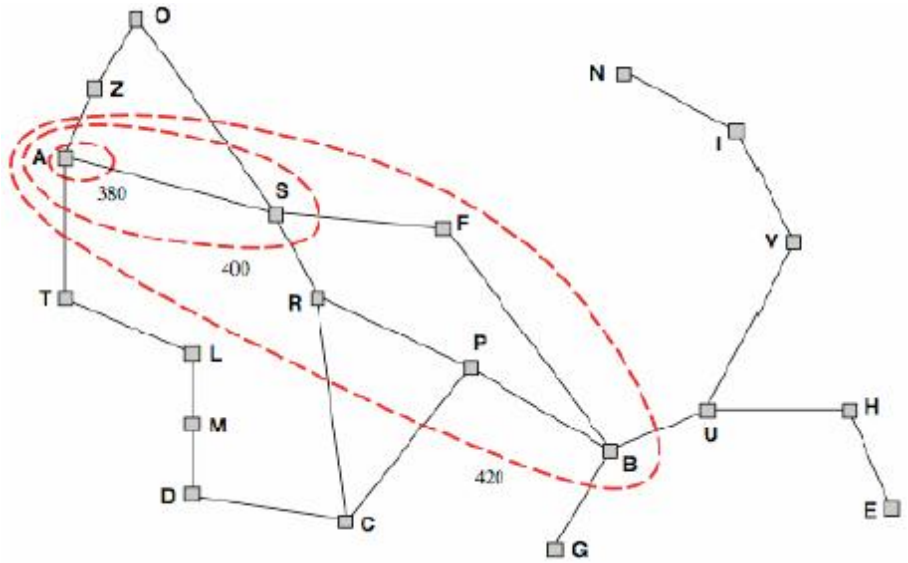
۱. با یک limit برابر با f-value حالت ابتدایی شروع نمایید
۲. یک جستجوی اول عمق را براساس حداقل هزینه و هرس گره ها با ارزیابی بزرگ تر از limit انجام بده
۳. اگر هدف پیدا نشد ، limit را به کم ترین f-value پیدا شده در جستجوی قبلی که از limit قبلی تجاوز می کند افزایش بده و مرحله ی ۲ را مجددا انجام بده

---

[1] pruning
[2] iterative-deepening A^* (ID A^*)
[3] Recursive Best-First Search (RBFS)
[4] Simplified Memory-bounded A^* (SM A^*)

- به صورت مجانبی با A^* ساده برابر می باشد
- فضا : به صورت خطی می باشد (شبیه اول عمق)
- لازم نیست که برای وضعیت های تکرار شده امتحان شود

* ایده ی اساسی : هر جستجوی اول عمق تا حد یک حدفاصل به وجود می آید



جستجوی اول – بهترین بازگشتی

- جستجوی اول – بهترین را با فضای خطی تقلید می نماید
- شبیه اول – عمق بازگشتی می باشد
  - بازگشت را با نگهداری جریان f-cost بهترین مسیر جایگزین از هر گره جَد محدود می نماید
  - در صورتی که گره جاری از این مقدار بیش تر می شود ، بازگشت به مسیر جایگزین برمی گردد
  - به این صورت حرکت به عقب [1] f-cost گره با بهترین f-cost زیر درخت را جایگزین می نماید
    - اجازه می دهد که که توسعه ی مجدد مسیر را در گذشته به یاد داشته باشیم

پیاده سازی جستجوی اول – بهترین بازگشتی

تابع RBFS-Search(problem) یک راه حل یا عدم موفقیت را برمی گرداند
RBFS(problem,make-node(initial-state[problem]),∞) را برگردان

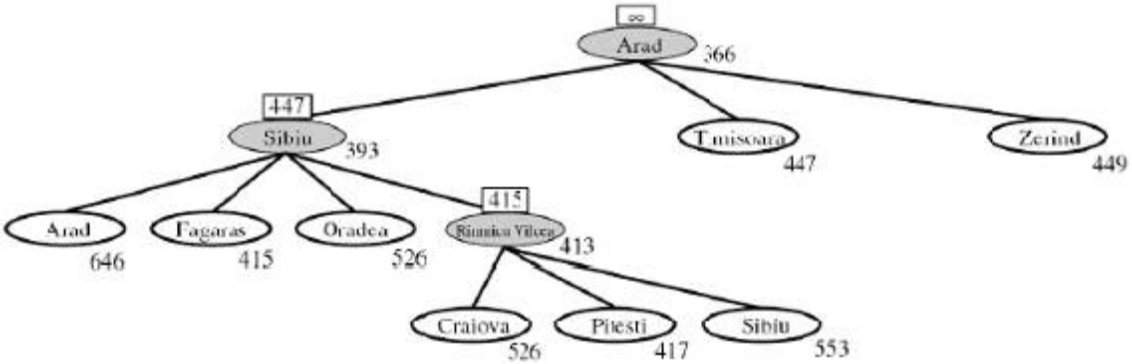
تابع RBFS(problem,node,f_limit) یک راه حل یا عدم موفقیت و یک f_limit جدید را برمی گرداند
در صورتی که Goal-Test[problem] به کار گرفته شده برای State(node) موفق شد ، گره را برگردان

[1] backtracking

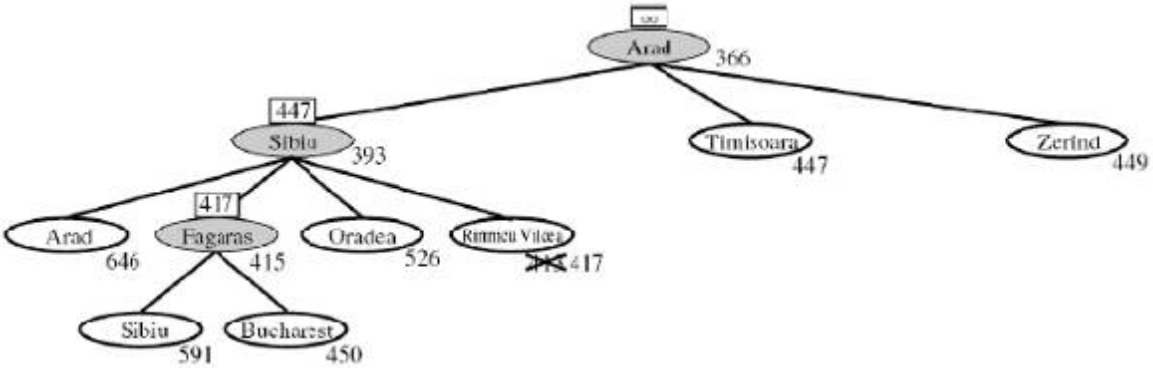
successors ← Expand(node,problem)
در صورتی که successors خالی بود عدم موفقیت را برگردان
برای هر s در successors ، ***f[s] ← max(g(s)+h(s),f[node])*** ***(معادله ی pathmax برای غیریکنوایی)***
تکرار کن
کم ترین گره f[ ] را در میان successors ← best
در صورتی که f[best]>f_limit شکست و f[best] را برگردان
در میان successors دومین مقدار کمینه را ← alternative
RBFS(problem,best,min(f_limit,alternative)) (فراخـــــــوان)
بازگشتی) ← **result,f[best]** (f[best] بهترین زیرگره ها را نگه می دارد)
در صورتی که result≠failure می باشد result را برگردان

مثال RBFS

a) بعد از توسعه ی آراد ، سیبیو و ریمنیکو ویلسیا [1]

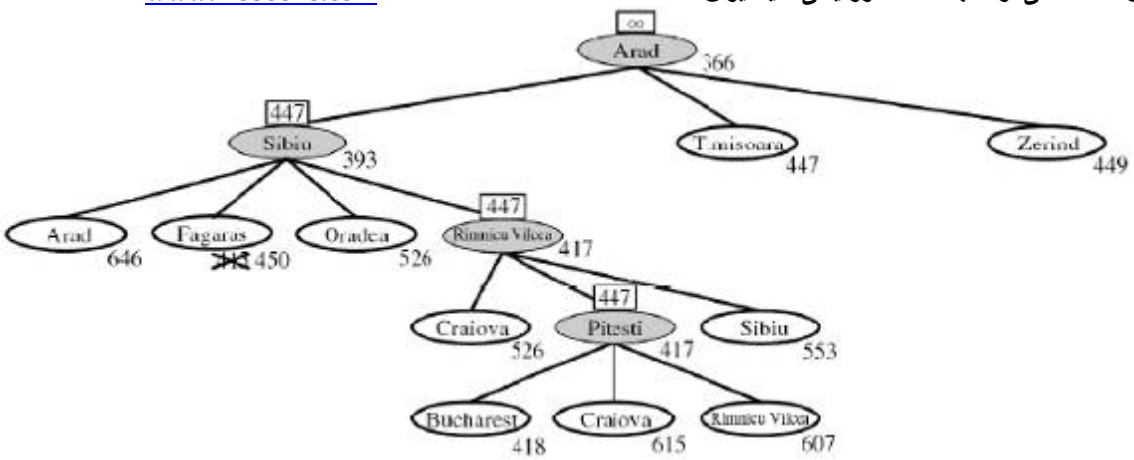


b) بعد از برگشت به سیبیو و توسعه ی فاگاراس



c) بعد از برگشت به عقب به ریمنیکو ویلسیا و توسعه ی پیتستی

[1] Rimnicu Vilcea



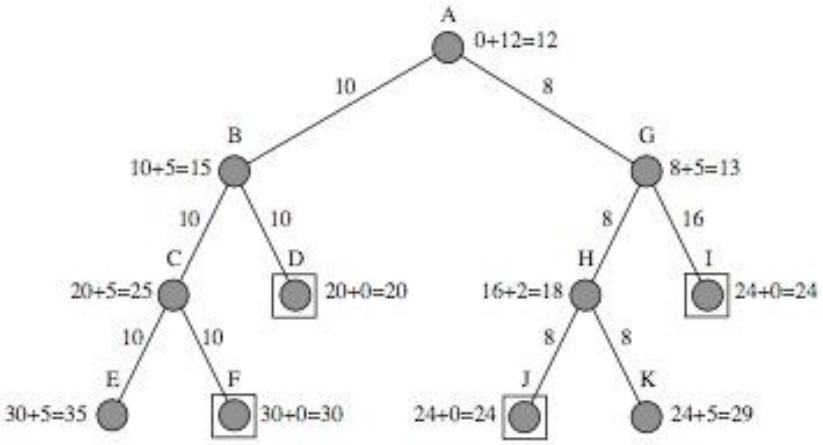
A^* با حافظه ی محدود ساده شده

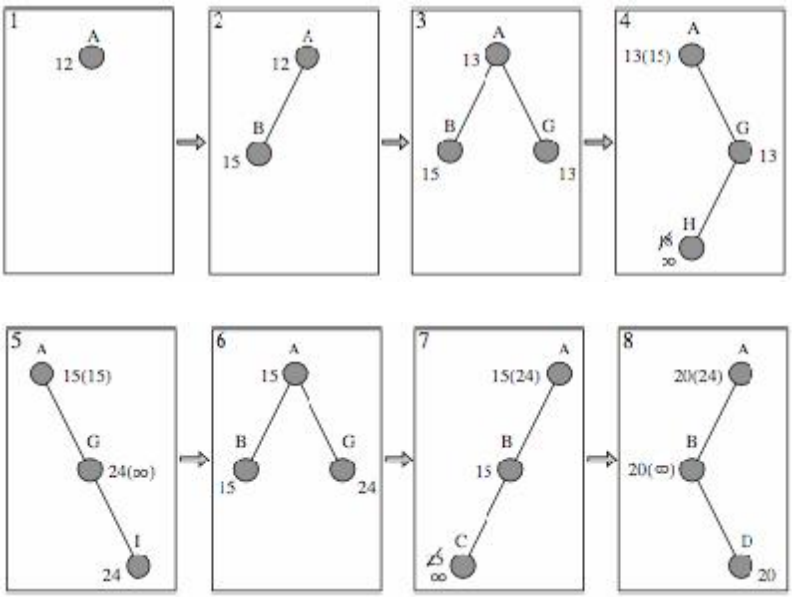
- A^* با یک مقدار حافظه ی ثابت می باشد
  - گره های کم تری تولید مجدد می شوند
- هر اندازه از حافظه را که در دسترس می باشد به کار می برد
- از وضعیت های تکرار شده تا حدی که حافظه اجازه می دهد جلوگیری می نماید
- در صورتی که صف برای نگهداری کم عمق ترین مسیر راه حل مناسب باشد کامل است
- در صورتی که صف برای نگهداری کم عمق ترین مسیر راه حل بهینه مناسب باشد بهینه است
  - در غیر این صورت ، بهترین راه حل که می تواند با توجه به حافظه قابل دسترس باشد را برمی گرداند
- با حافظه ی کافی (برای هر درخت جستجوی کامل) رفتار A^* را تکرار می نماید
- گره ها را از صف در زمانی که یک جایگزین باید تولید شود اما فضای کافی وجود ندارد منشعب می کند
  - گره های فراموش شده
  - گره های بدون امید [1] (با f-cost بالا یا گره های قدیمی) را منشعب می کند
  - هزینه ی بهترین مسیر زیردرخت های فراموش شده در اجداد نگهداری می شود
- مسیرهای بلندتر از صف دارای هزینه ی ∞ می باشند
- گاهی اوقات A^* با حافظه ی محدود ساده شده در تعویض میان دو مسیر در صف مناسب مردد می شود (کوبیدگی [2])
- مسایل قابل حل در A^* (با حافظه ی نامحدود) برای SMA^* لجوجانه عمل می کنند

---

[1] unpromise
[2] thrashing

مثال برای SMA^*
اندازه ی صف = ۳





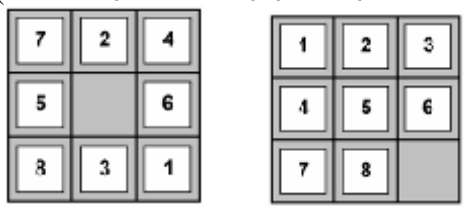
اکتشافات

- تابع مکاشفه ای خوب کلیدی برای یک الگوریتم جستجو می باشد
- تا کنون ما به فاصله ی اقلیدسی [1] توجه می کردیم
  - برای مساله ی مسیریابی خوب است
- اما ما مکاشفات را برای مسایل جدید چگونه پیدا کنیم ؟
  - راه استانداردی وجود ندارد ، در بیش تر موارد باید مکاشفات به صورت دستی پیدا شوند
- **راحت کردن** [2] یک مساله اغلب تذکراتی را دارد

Euclidean [1]
relaxing [2]

- مثلا فاصله ی اقلیدسی محدودیتی که شما برای ماندن روی یک مسیر از یک جا به جای دیگر دارید را راحت (آرام) می کند

ابتکارات قابل قبول [1] - برای پازل ۸- تایی : $h_1(n)$ = تعداد کاشی های در جای نادرست گذاشته شده . $h_2(n)$ = مجموع فاصله تا جزیره ی مانهاتان [2] (به تعداد کاشی (خانه) های در جای درست قرار داده شده توجه نمایید)



حالت نهایی **حالت اولیه**

$h_1(S) = ??6$

$h_2(S) = ??4+0+3+3+1+0+2+1=14$

مکاشفه و کارایی

- ما به راهی برای اندازه گیری کیفیت یک مکاشفه نیاز داریم
- یک **روش فاکتور انشعاب موثر** [3] می باشد (یک اجرا از مساله ای که داریم)
  - N تعداد گره ی تولید شده
  - d عمق راه حل
- ما فاکتور انشعاب را نمی دانیم ، اما می توانیم یک درخت یکسان را با N+1 گره به وجود آوریم

$$N+1 = 1+b^*+(b^*)^2+\ldots+(b^*)^d$$

  - اکتشافات خوب دارای b^* نزدیک به ۱ هستند

تسلط [4]

- مکاشفات می توانند برای این که کدام بر دیگری برتری دارد مقایسه شوند .
- در صورتی که به ازای هر n : $h_2(n) \geq h_1(n)$ (هر دو قابل قبول باشند) ، h_2 بر h_1 برتری دارد یا غلبه می کند و برای جستجو بهتر است. ارزیابی های جستجوی مطلوب :

$d = 14$

$IDS = 3,473,941$ گره , $A^*(h_1) = 539$ گره , $A^*(h_2) = 113$ گره

d=24

$IDS \approx 54,000,000,000$ گره , $A^*(h_1) = 39,135$ گره , $A^*(h_2) = 1,641$ گره

[1] admissible heuristics
[2] Manhattan
[3] effective branching factor
[4] Dominance

به ازای هر h_a و h_b ابتکاری داده شده ، $h(n) = \max(h_a(n), h_b(n))$ قابل قبول است و بر h_a و h_b غلبه می کند.

مسایل آرام شده [1]

- h_1 و h_2 راه حل هایی برای نوع آرام شده (ساده شده) جدول معما [2] هستند
  - در صورتی که قانون های جدول معمای ۸ – تایی برای این که یک کاشی بتواند به هر کجا حرکت کند ساده شده باشند ، در این صورت h_1 کوتاه ترین راه حل را می دهد
  - در صورتی که قانون ها برای این که یک کاشی بتواند به هر کاشی مجاور حرکت کنند آرام شده باشند ، آن گاه h_2 کوتاه ترین راه حل را ارایه می نماید
- مسایل آرام شده : مساله ی با محدودیت های کم تر در عملیات هستند
  - ابتکارات قابل قبول برای مساله ی اصلی می توانند از راه حل بهینه برای مساله ی آرام شده مشتق شوند
  - نکته ی کلیدی ، این که ، هزینه ی ارزیابی بهینه بودن راه حل یک مساله ی آرام شده ، بزرگ تر از هزینه ی ارزیابی بهینه بودن راه حل یک مساله ی واقعی نیست.
- می توانیم از چند مکاشفه به صورت همزمان استفاده کنیم

$$h(n) = \max(h_1(n),...,h_k(n))$$

- در صورتی که اجزا قابل قبول باشند ، هنوز قابل قبول می باشد

مثال معروف : مساله ی فروشنده ی دوره گرد ، کوتاه ترین گشت [3] که از همه ی شهرها دقیقا یک بار ملاقات کند را پیدا کنید. کوتاه ترین درخت پوشا [4] می تواند در $O(n^2)$ محاسبه شود و یک حد کوتاه تر ، در کوتاه ترین گشت است .

خلاصه - توابع مکاشفه ای ، هزینه ی کوتاه ترین مسیر را تخمین می زنند. ابتکارات خوب می توانند به طور چشمگیری هزینه ی جستجو را کاهش دهند. جستجوی اول بهترین و حریصانه ی اول - بهترین پایین ترین h را توسعه می دهد و کامل نیست و همیشه هم بهینه نیست . جستجوی A^* پایین ترین g+h را توسعه می دهد و کامل و بهینه می باشد و همچنین بهینگی موثری دارد (تا حد شکستن گره ها و برای جستجوی مستقیم) و ابتکارات قابل قبول می توانند از راه حل مستقیم مسایل مجاز مشتق شوند.

[1] Relaxed problems
[2] puzzle
[3] tour
[4] spanning tree

فصل پنجم

الگوریتم های جستجوی محلی

الگوریتم های جستجوی محلی [1]
فصل پنجم

رئوس مطالب

- ✓ **تپه نوردی (بالا رونده از تپه)** [2]
- ✓ **گرم و سرد کردن شبیه سازی شده** [3]
- ✓ **الگوریتم های ژنتیکی ، پیدایشی یا تکوینی** [4] **(به طور خلاصه** [5]**)**
- ✓ **جستجوی محلی در فضاهای پیوسته (خیلی به طور خلاصه)**

مسایل بهینه سازی

- برخی از ساختارهای ترکیبی که ما در تلاش برای بهینه سازی آن ها هستیم
- محدودیت ها باید ارضا شوند
- تابع هزینه باید بهینه شود
  - با تابع ارزیابی مکاشفه ای سردرگم نشوید

- اغلب پیدا کردن راه حل ... آسان است
- ... اما پیداکردن بهترین راه حل سخت می باشد
  - پیداکردن همه ی راه حل های ممکن غیر ممکن می باشد

- ما دست کم ، یک راه حل خوب را می خواهیم

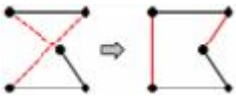
الگوریتم های بهبود یافته ی تکراری [6] - در تعدادی از مسایل بهینه سازی ، مسیر ، نامربوط است ؛ وضعیت (حالت) هدف ، خودش راه حل است . بنابراین ، برای فضای حالت ، مجموعه ای از پیکربندی های کامل ؛ پیکربندی بهینه و TSP را پیدا کنید ، یا ارضای

---

[1] local search algorithms
[2] Hill-climbing
[3] Simulated annealing
[4] Genetic algorithms(GA)
[5] briefly
[6] iterative improvement algorithms

محدودیت ها ، پیکربندی و برنامه ی زمانی را پیدا کنید. در موارد مشابه ، شما می توانید از الگوریتم های بهبود یافته ی تکراری استفاده نمایید ؛ یک حالت جاری منفرد یا تکی را نگهداری نمایید و برای بهبود آن تلاش نمایید. در فضای ثابت ، جستجوی برخط به اندازه ی جستجوی برون خط مناسب است .

مثال : مساله ی فروشنده ی دوره گرد - با هر گشت کامل شروع کنید و معاوضه های دو به دو را انجام دهید.



مثال : وزیر های شطرنج n – تایی [1] ، n وزیر را در یک صفحه ی شطرنج n×n بدون این که هیچ یک از دو وزیر در سطر [2] ، ستون [3] همانند قرار گیرند یا به طور مورب [4] همانند قرار گیرند ، قرار دهید. برای کاهش تعداد تعارض ها یا ناسازگاری ها ، یک وزیر را حرکت دهید. تقریبا همیشه مسایل وزیرهای چند گانه تقریبا به سرعت برای هر n بزرگ حل می شوند ، مثلا برای n = یک میلیون

جستجو در مساله ی بهینه

دو نوع روش وجود دارد

- **سازنده** [5]
  - از یک وضعیت ابتدایی شروع کنید و به طرف یک راه حل حرکت نمایید
  - به عنوان مثال در مورد مساله ی مسافرت شخص دوره گرد : از یک شهر شروع نمایید و یک مسیر را با ملاقات همه ی شهرها به دست آورید
- **تکراری**
  - با یک راه حل شروع نمایید : غلط یا غیربهینه
  - آن را به طرف یک راه حل بهینه ببرید
  - به عنوان مثال در مورد مسافرت شخص دوره گرد : با یک گشت شروع کنید و هزینه ی آن را با تعویض شهرها بهبود دهید

چرا از روش سازنده استفاده نمی کنیم ؟

- هزینه فقط به راه حل بستگی دارد ، نه به چگونگی آن
- جستجوی آگاهانه (مثل A^*) وقتی که ما یک راه حل برای همه ی مسیر راه حل داریم خوب کار می کند
- جستجوی ناآگاهانه را استفاده نمی کنیم (چون فضای حالت خیلی بزرگ می باشد)

جستجوی محلی

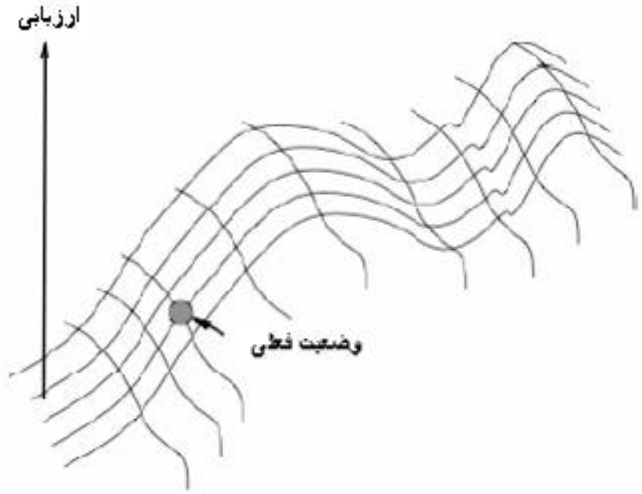
- تا حالا ، الگوریتم هایی که ما بررسی کرده ایم ، تمام مسیر از وضعیت اولیه به وضعیت نهایی را نگهداری می کنند .
- این باعث مصرف حافظه ی زیاد / فضای زیاد در مورد مسایل بزرگ می شود .

---

[1] n-queens
[2] row
[3] column
[4] diagonal
[5] constructive

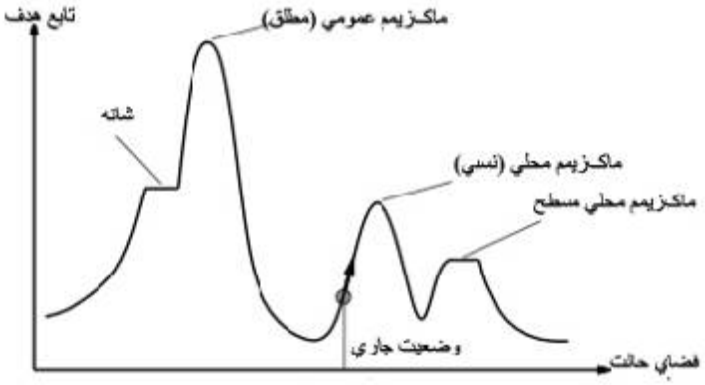
- برای برخی از مسایل ، اطلاعات مسیر ، ضروری (حیاتی) می باشد
  - مسیر یابی
  - پازل ۸ – تایی
  - ما هدف را می دانیم ، اما این که چگونه به آن برسیم را نمی دانیم .
- برای دسته ای دیگر از مسایل ، ممکن است ما نگران چگونگی رشته ی عملکردها نباشیم .
  - پیدا کردن راه حل بهینه ، چیزی است که مهم می باشد .
  - زمانبندی
  - طرح VLSI
  - پنهان شناسی (رمزنویسی) [1]
- در این موارد ، ما می توانیم با اطمینان ، برخی از اطلاعات مسیر را حذف نماییم .
- یک الگوریتم جستجو که فقط از وضعیت جاری استفاده می کند ، یک *الگوریتم جستجوی محلی* نام دارد .
- مزایا :
  - به حافظه ی ثابت ، نیازمندیم
  - می توانیم راه حل هایی باورنکردنی را در مورد فضاهای بزرگ پیدا نماییم .
- معایب :
  - معمولا ، بهینه نمی باشد – فقط بهینه های محلی ، پیدا خواهند شد .
  - می تواند به علت کمبود حافظه ، مردد باشد .

- **مقایسه** [2] **:** فضای حالت شبیه روی زمین دارای تپه ها و دره ها می باشد
- ارزیابی توسط تابع هزینه (ارزیابی) داده شده است

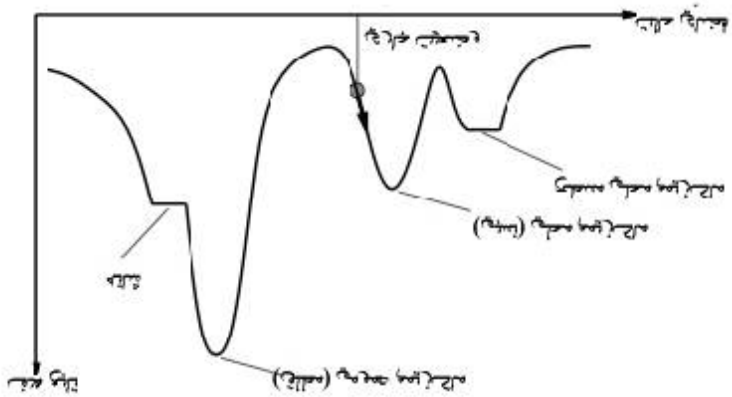


- فضای حالت یک بعدی [3] برای مثال ها
  - آسان تر تصویر می شود (در کل جند بعدی هستند)

[1] cryptography
[2] analogy
[3] one-dimensional



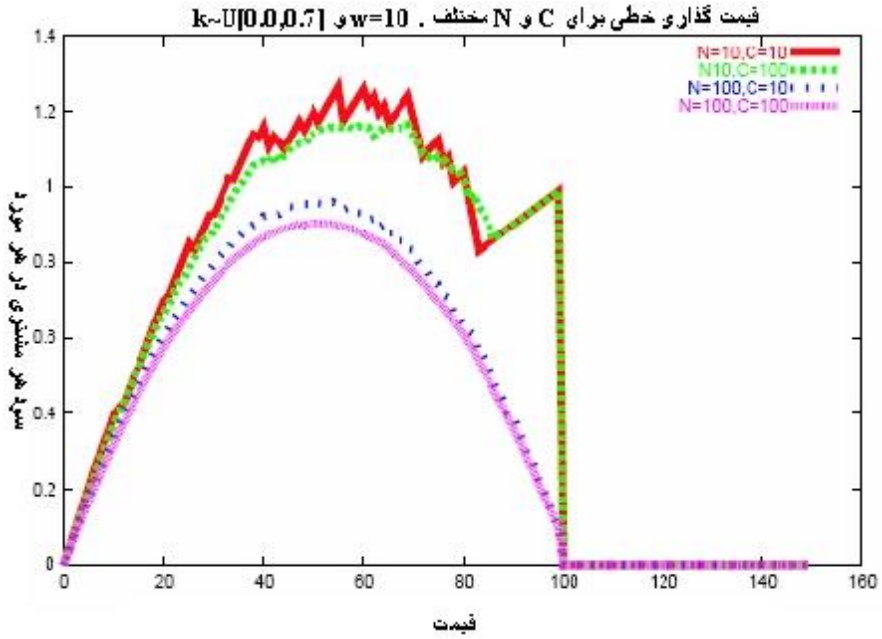
- ماکزیمم و مینیمم قابل تعویض اند
  - فقط مقدار را معکوس نمایید (با استفاده از -)



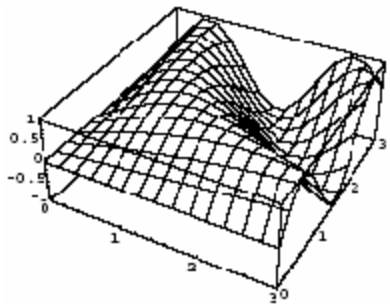
چشم انداز [1] جستجو

- جستجوی محلی ، معمولا برای مسایل بهینه سازی ، مفید می باشد
- "پارامترهایی را پیدا نمایید که o(x) بیشینه / کمینه باشد "
- این ، در جایی که فضای حالت ، ترکیبی از پارامترها می باشد ، یک مساله می باشد .
- در صورتی که n پارامتر وجود داشته باشد ، ما می توانیم یک فضای n+1 بعدی را تصور نماییم ، که n بعد اول ، پارامترهای تابع می باشند و n+1 امین بعد ، *تابع هدف* [2] می باشد .
- ما این فضا را یک چشم انداز جستجو می نامیم
  - بهینه ها روی تپه ها هستند
  - گودی ها ، راه حل های ضعیف هستند .
- یک چشم انداز یک بعدی :

[1] landscape
[2] objective function

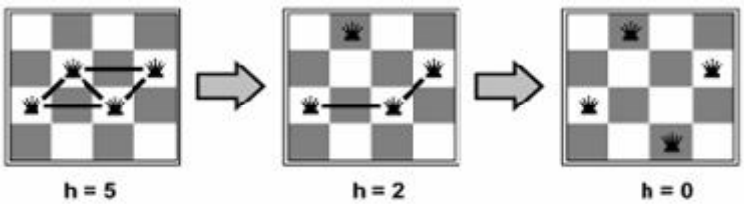


- یک چشم انداز دوبعدی :



- دورنماها ، یک نمایش خوب برای الگوریتم های جستجوی محلی هستند .
- بیایید بالارفتن از یک تپه را تصور نماییم .
- یک راه را برای تشخیص مسایل آسان از مسایل سخت ، ارایه می کند .
  - o آسان : قله های کم ، سطوح صاف ، بدون برآمدگی (خط الراس) / مسطح (فلات)
  - o سخت : تعداد قله های زیاد ، سطوح دندانه دار یا ناپیوسته (گسسته) و مسطح

الگوریتم بالا رونده از تپه



- ساده ترین شکل جستجوی محلی ، جستجوی تپه نوردی می باشد .
- خیلی ساده : در هر نقطه ، به جانشینان (همسایه ها) نگاه کنید و در جهت بهتر ، حرکت نمایید .
- خیلی شبیه به جستجوی حریصانه می باشد .
- به حافظه ی خیلی کمی نیاز دارد .

- فلات می تواند سبب این شود که الگوریتم ، بی هدف ، سرگردان باشد .

جستجوی محلی در حساب دیفرانسیل و انتگرال [1]

- ریشه های یک معادله f(x)=0 را به دست آورید ، f ، تشخیص پذیر می باشد .
- با فرض یک x_1 ، $f(x_1)$ و $f'(x_1)$ را به دست آورید .
- از خط تانژانت برای $f(x_1)$ استفاده نمایید تا x_2 را انتخاب نمایید (شیب= $f'(x_1)$) .
- تکرار نمایید $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_1)}$
- این یک جستجوی تپه نوردی می باشد .
- برای موارد مسطح ، خیلی خوب می باشد .

- **جستجوی محلی حریصانه ی قرمز**

تابع Hill-Climbing(problem) یک وضعیت را برمی گرداند
current ← Initial-State(problem)
در حلقه کارهای زیر را انجام بده
بالاترین جانشین مقداردهی شده ی current را در neighbour قرار بده
در صورتی که Value(neighbour)≤Value(current) بود current را برگردان
current ← neighbour

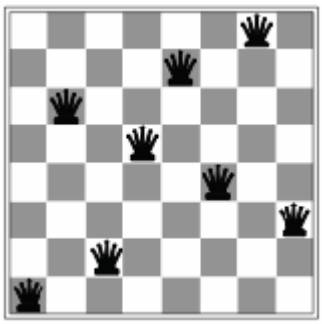
الگوریتم (به بیان دیگر) :
تابع Hill-Climbing(problem) یک حالت که یک ماکزیمم محلی است را برمی گرداند
ورودی ها ، problem که یک مساله است می باشد
متغیّرهای محلی current و neighbor که هر کدام یک گره هستند ، می باشند.
current ← Make-Node(Initial-State[problem])
کارهای زیر را انجام بده
بالاترین مقداردهیsuccessor از current ← neighbor (مقدار بالا ترین جانشین current را در neighbor قرار می دهد)
اگر Value[current] ≤ Value[neighbor] می باشد State[current] را برگردان
current ← neighbor
پایان حلقه
پایان الگوریتم

شروع مجدد تصادفی الگوریتم بالا رونده از تپه بر ماکزیمم محلی غلبه می کند .
حرکت های غیرمستقیم تصادفی از شانه های [2] حلقه در مسطح ترین جا می گریزند

[1] calculus
[2] shoulders

مثال جستجوی تپه نوردی

- فرمول بندی کامل برای ۸ – وزیر
  - تابع جانشین : یک وزیر را به مربع دیگر در همان ستون ببرید (۸*۷=۵۶ جانشین)
  - هزینه : تعداد جفت هایی که در حال حمله به یکدیگر هستند
- **کمینه سازی [1] مساله (سرپایینی)**



هزینه ی کمینه ی محلی = ۱

هزینه ی فعلی =۱۷

خصوصیات تپه نوردی

- پیاده سازی آن آسان می باشد
- مقدار کمی از حافظه را استفاده می نماید
- تپه نوردی ، زیاد تحت تاثیر شکل فضای حالت است
  - با ماکزیمم محلی کم و plateaux خوب می باشد
  - برای فضاهای جوجه تیغی (دندانه دار) مانند بد است

- در موارد زیر دچار مشکل می شود
  - شانه ها
  - ماکزیمم محلی
  - خط الراس [2] ها
- در ماکزیمم محلی نمی تواند قله ی بالاتر را ببیند
- در شانه ، راهی به طرف بیرون را نمی تواند پیدا نماید
- در خط الراس ها
  - برای حرکت ، شما باید به طرف پایین حرکت نمایید
  - حرکت های بد را مجاز می داند

تنوعات تپه نوردی

- **حرکت های غیرمستقیم [3]** : روی کف [1] (جای مسطح) حرکت می نماید

---

[1] minimisation
[2] ridge
[3] sideways move

- در حلقه نمی افتد (یک تعداد بیشینه از حرکات را ثبت می کند)

- **تپه نوردی اتفاقی :** در میان حرکت های به طرف سربالایی یکی را به تصادف انتخاب می نماید
- **اولین انتخاب تپه نوردی :** اولین حرکت سربالایی را انتخاب می کند
- **تپه نوردی با شروع مجدد تصادفی :** چند تپه نوردی از وضعیت های اولیه ی انتخاب شده به صورت تصادفی
  - تولید وضعیت های تصادفی می تواند غیر جزیی باشد

به عبارت بهتر ،

بهبود روش تپه نوردی

- تپه نوردی ، می تواند خوشایند باشد
  - برنامه نویسی آن آسان می باشد
  - به فضای کمی نیازمند می باشد
  - ممکن است ما ، روشی بهتر را نداشته باشیم .
- چگونه این روش ، بهبود می یابد ؟
- تپه نوردی تصادفی – به صورت تصادفی حرکت های بالای تپه را انتخاب نمایید
- تپه نوردی با شروع مجدد تصادفی
- تا زمانی که به یک بهینه برسید ، ادامه دهید
- یک وضعیت اولیه را به صورت تصادفی ، انتخاب نمایید
- مجددا ، اجرا نمایید .
- بعد از n تکرار ، بهترین راه حل را نگهداری نمایید .

گرم و سرد کردن شبیه سازی شده

- ضعف روش تپه نوردی ، این می باشد که هرگز به طرف پایین تپه حرکت نمی کند .
- شبیه به جستجوی حریصانه ، نمی تواند "پشتیبان [۲]" داشته باشد .
- گرم و سرد کردن شبیه سازی شده ، تلاشی برای برطرف نمودن این موارد می باشد .

ایده : گریختن از ماکزیمم محلی با اجازه دادن به بعضی از حرکت های بد یا نادرست می باشد . اما به تدریج تعداد و فراوانی [۳] حرکت های بد کاهش می یابد.

- تپه نوردی را با حرکت تصادفی ترکیب می نماید

- **سرد و گرم کردن** : فلزات را با حرارت دادن آن ها و رساندن دمای آن ها به دمایی بالا و سردکردن تدریجی آن ها سخت می کنند

- یک توپ پینگ پنگ را به عمیق ترین جای یک سطح دارای گودی بیندازید
  - سطح را برای خارج کردن توپ از مینیمم محلی تکان دهید

---

[1] plateau
[2] back up
[3] frequency

- خارج کردن توپ به سختی خارج کردن آن از کمینه ی مطلق نمی باشد

- سرد و گرم کردن شبیه سازی شده :
  - با ارتعاشات شدید شروع نمایید (در دمایی زیاد) و سپس به تدریج ارتعاشات را کم نمایید (با کم تر کردن دما)
  - از کمینه ی محلی با اجازه دادن به برخی از حرکت های بد بگریزید
  - اما به تدریج تعداد و فراوانی آن ها را کاهش دهید

الگوریتم :

تابع Simulated-Annealing(problem,schedule) یک راه حل را برمی گرداند ورودی ها ، یک مساله و schedule که یک نگاشت از زمان به دما می باشد ، هستند .

متغیر های محلی عبارتند از : current ، که یک گره است و next ، که یک گره است و T ، که یک پروب کنترل کننده ی دما از مراحل پایین تر است .

current ← Make-Node(Initial-State[problem])

برای مقادیر t = 1 تا ∞ کارهای زیر را انجام بده :

T ← schedule[t]

در صورتی که T=0 بود گره ی current را برگردان

یک جانشین به تصادف انتخاب شده از گره ی current را به گره ی next نسبت بده

ΔE ←Value[next] – Value[current]

در صورتی که $\Delta E > 0$ بود محتوای گره ی next را در گره ی current بریز

در غیر این صورت ، محتوای گره ی next را فقط با احتمال $e^{\Delta(E/T)}$ در گره ی current کپی کن

پایان الگوریتم

- همیشه حرکت های خوب را می پذیرد
- احتمال پذیرش یک حرکت بد
  - به صورت نمایی با "بدی " حرکت کاهش پیدا می نماید
  - به صورت نمایی با "دمای" T کاهش می یابد
- یک راه بهینه با احتمال نزدیک به یک را پیدا می نماید ، اگر از دما به آهستگی کاسته شود

خصوصیات گرم و سرد کردن شبیه سازی شده - در دمای ثابت T ، حالت کاری ممکن است به توزیع بولتزمان [1] برسد.

[1] Boltzman

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T با سرعت کمی کاهش داده می شود ، در نتیجه ، همیشه به بهترین حالت x^* می رسد . زیرا برای T های کوچک داریم:

$$e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} >> 1$$

آیا این الزاما یک ضمانت قابل قبول است ؟؟

در بسیاری از موارد در قالب بندی VLSI[1] ، زمانبندی خطوط هوایی و غیره به کار می رود. گرم و سرد کردن شبیه سازی شده در صورتی که T به اندازه ی کافی با سرعت کم تر ، کاسته شود کامل و بهینه می باشد . به سادگی به الگوریتم تپه نوردی اضافه می شود . ضعف این روش در انتخاب یک زمانبندی خنک کننده ی خوب می باشد .

جستجوی پرتومحلی [2]

ایده : به جای یک ، حالت های k را نگه می دارد ؛ بالاترین مقدار k را در بین همه ی جانشینان انتخاب می کند . همانند جستجوهای k که در حالت موازی اجرا می شوند ، نمی باشد . جستجو می کند تا حالت های خوب دیگر جستجو ها را پیدا کند و آن ها را به هم پیوند دهد.

مساله : اغلب کامل است ، تمام حالت های k در نهایت در بالای تپه ی محلی شبیه قرار می گیرند .

ایده : جانشینان k را به طور تصادفی انتخاب می نماید . مثال نزدیک انتخاب طبیعی را مشاهده نمایید.

- تنوع جستجوی پرتو
  - یک روش برمبنای مسیر که به چند مسیر " اطراف " مسیر جاری نگاه می کند
- k وضعیت را در حافظه به جای یک وضعیت نگهداری می نماید
  - با k گره ی به طور تصادفی انتخاب شده شروع می کند
  - تمام جانشینان تولید می شوند ، در صورتی که هدف پیدا شود متوقف می شود
  - در غیر این صورت تعداد k تا از بهترین جانشینان انتخاب می شوند
- اطلاعات میان وضعیت ها می توانند مشترک شوند
  - به وعده داده شده ترین ناحیه ها حرکت می نماید
- جستجوی پرتو محلی اتفاقی تعداد k جانشین را به صورت تصادفی انتخاب می نماید
  - به وسیله ی احتمالی که توسط تابع ارزیابی تشخیص داده می شود

الگوریتم های ژنتیکی

[1] مخفف Very Large Scale Integration می باشد ، مجتمع سازی در مقیاس بسیار بزرگ ، تجمع مقیاس بسیار وسیع ، قطعه ی روی یک آی سی (IC)
[2] local beam search

- الگوریتم های ژنتیکی می توانند به صورت یک شکل از جستجوی موازی تپه نوردی ، تصور شوند .
- ایده ی اساسی :
  - برخی از راه حل ها را به صورت تصادفی ، انتخاب نمایید .
  - برای به دست آوردن راه حل های جدید ، بهترین قطعات راه حل ها را با هم ترکیب نمایید .
  - تکرار نمایید .
- جانشین ها ، تابعی از دو وضعیت می باشند

الگوریتم های ژنتیکی = جستجوی پرتوی محلی اتفاقی + تولید جایگزین هایی از جفت حالت ها

- جستجوی پرتو محلی به شکل " انتخاب طبیعی " : جانشینان (موجود تولید شده توسط والدین [1]) یک وضعیت (زیستمند [2]) نسل بعدی را با توجه به مقدارشان (سازگاری [3]) تولید می کنند

- GA ها : الگوریتم های نمونه بعد از تکامل زیستی
- تنوع جستجوی پرتو اتفاقی
  - وضعیت های جانشین به صورت تنوعاتی از دو وضعیت والد تولید می شوند ، نه فقط یکی

اصطلاحات الگوریتم ژنتیکی

- **جمعیت** [4]
  - به صورت اولیه مجموعه ای از k وضعیت تولید شده به صورت تصادفی
  - مجموعه ای از وضعیت های مشتق شده
- **نسل** [5]
  - جمعیت در یک نقطه از زمان
  - معمولا ، انتشار [6] برای تمام جمعیت همزمان [7] می شود
- **فرد** [8]
  - یک عنصر از جمعیت
  - به صورت یک رشته با الفبای محدود توصیف می شود
    - دودویی [9] ، حروف [10] ، ارقام [11]

---

[1] offspring
[2] organism
[3] fitness
[4] population
[5] generation
[6] propagation
[7] synchronize
[8] individual
[9] binary
[10] letters
[11] digits

- برای همه ی جمعیت یکسان می باشد

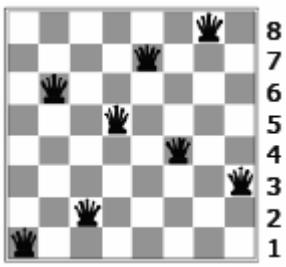
- **تابع شایستگی** [1]
  - تابع ارزیابی در واژه ی جستجو می باشد
  - مقادیر بالاتر منجر به شانس های بهتر برای تولید مجدد می شوند
- **شایستگی :** خوبی یک راه حل
- **ژنوم** [2] - یک راه حل یا وضعیت
- **ویژگی / ژن** [3] - یک پارامتر یا متغیر وضعیت

یک الگوریتم ژنتیکی پایه

یک جمعیت تصادفی را تولید کن و آن را در pop قرار بده
مادامی که انجام نشده است ، کارهای زیر را انجام بده
برای هر p در pop
p را مورد ارزیابی قرار بده
برای i=1 تا اندازه ی pop کارهای زیر را انجام بده
راه حل های تصادفی را از pop ، انتخاب نما
راه حل ها crossover نما
راه حل ها را تغییر بده
راه حل های جدید را در pop قرار بده

مثال : ۸ – وزیر

- یک وزیر در هر ستون
  - وضعیت یک ۸ – چندتایی [4] از ارقام از ۱ تا ۸ می باشد
  - هر رقم ردیف وزیر را نشان می دهد
  - شایستگی تعداد جفت های غیر حمله کننده می باشد (معکوس شده
)



صفحه
فرد ۳ ۸ ۴ ۷ ۵ ۲ ۶ ۱
شایستگی = ۱

[1] fitness function
[2] genome ؛ تمامي تركيب ژنتيكي يک فرد يا جمعيتي كه بوسيله كروموزومها به ارث مي رسد .
[3] trait/gene
[4] tuple

رمزکردن یک مساله

- فرض کنید که ما می توانیم مساله امان را به صورت یک رشته ی بیتی ، رمز نماییم .
- مثال : ۸ – وزیر . برای هر ستون ، ما از سه بیت برای رمز نمودن ردیف وزیر استفاده می کنیم = ۲۴ بیت .
  - 100 101 110 000 101 001 010 110 = 4 5 6 0 5 1 2 6
- ما با تولید رشته های بیتی تصادفی شروع می کنیم ، سپس آن ها را با توجه به تابع شایستگی (تابعی برای بهینه نمودن) مورد ارزیابی قرار می دهیم .

تولید راه حل های جدید

- تابع جانشین ما با کار برروی دو راه حل ، کار می کند .
- این کار ، crossover نام دارد .
- دو راه حل را به صورت تصادفی انتخاب نمایید .
- روش اول : انتخاب شایستگی مناسب
  - تمام شایستگی ها را جمع نمایید
  - احتمال انتخاب s = شایستگی s ، تقسیم بر مجموع شایستگی
- یک نقطه ی (مکان هندسی) تصادفی را در رشته های بیتی انتخاب نمایید .
- تکه ی اول b1 را با تکه ی دوم b2 برای به وجود آوردن دو رشته بیتی جدید ، با هم ادغام نمایید .

مثال crossover

s1: (100 101 110) (000 101 001 010 110) = 4 5 6 0 5 1 2 6
s2: (001 000 101) (110 111 010 110 111) = 1 0 5 6 7 2 6 7

مکان هندسی را برابر ۹ قرار دهید

s3 = (100 101 110) (110 111 010 110 111) 4 5 6 6 7 2 6 7
s4 = (001 000 101) (000 101 001 010 110) 1 0 5 0 5 1 2 6

سپس ، جهش را اعمال نمایید . با احتمال m (برای m کوچک) به صورت تصادفی ، یک بیت در راه حل را انتخاب نمایید . بعد از تولید یک جمعیت جدید از همان اندازه به صورت جمعیت قدیمی ، جمعیت قبلی را حذف نمایید و دوباره شروع نمایید . crossover قطعات راه حل های موفق به صورت جزیی را با هم ترکیب می کند . ژن های نزدیک تر به هم ، برای ماندن با هم در رشته ی نسل ها ، بهتر می باشند . این کار ، رمز کردن را مهم می کند . جهش : راه حل های جدید را به جمعیت اعمال می کند . در صورتی که یک ژن از جمعیت اولیه گم شده بود ، crossover نمی تواند آن را تولید کند ، در غیر این صورت ، مکان هندسی درون ژن را قرار می دهیم .

اصول الگوریتم ژنتیکی

- **تولید مجدد توسط دورگه** [1]
  - دو قسمت کردن [2] والدین در یک نقطه ی دورگه ی داده شده و ترکیب شده

[1] crossover
[2] split

- دو وضعیت جدید را تولید می نماید

- **نقطه ی دورگه**
  - در پیشرفت تصمیم می گیرد و اغلب به صورت تصادفی انتخاب می شود
  - برای هر دو والد باید یکسان باشد
  - باید نسل های قابل قبول را تولید نماید

- **جهش** [1] **:** تولید مشخصه های جدید که در والدها وجود ندارد
  - تغییرات تصادفی اجزای رمزکننده در نسل
  - توسط احتمال کنترل شده است
- **الگو** [2]
  - اجزای مفید یک راه حل می توانند در میان نسل ها باقی بمانند

انتخاب مناسب ترین hypothese ها

- انتخاب شایستگی مناسب

$$P(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{k} Fitness(h_j)}$$

می تواند منجر به تولید چند کپی منتشرشده شود

- **انتخاب مسابقه ای** [3] :
  ۱. دو فرد را به صورت تصادفی و با احتمال یکسان انتخاب نمایید
  ۲. با یک احتمال ثابت مناسب تر را انتخاب نمایید
- **انتخاب ردیف** [4] **:**
  ۱. همه ی hypothese ها را براساس شایستگی مرتب نمایید
  ۲. احتمال انتخاب با ردیف متناسب است

دورگه

- مرکب از تکه های ترکیب کننده ی افراد برای ساختن افراد جدید است (مثال هایی با رمزکننده ی دودویی)
- دورگه ی تک نقطه ای : یک نقطه ی دورگه را انتخاب نمایید و دورگه ها را از آنجا بردارید و قطعه ها را تعویض نمایید . به عنوان مثال :

0111110 — 101|1110

دورگه

1010101 — 011|0101

گام بعد :

---

[1] mutation
[2] schema
[3] tournament
[4] rank

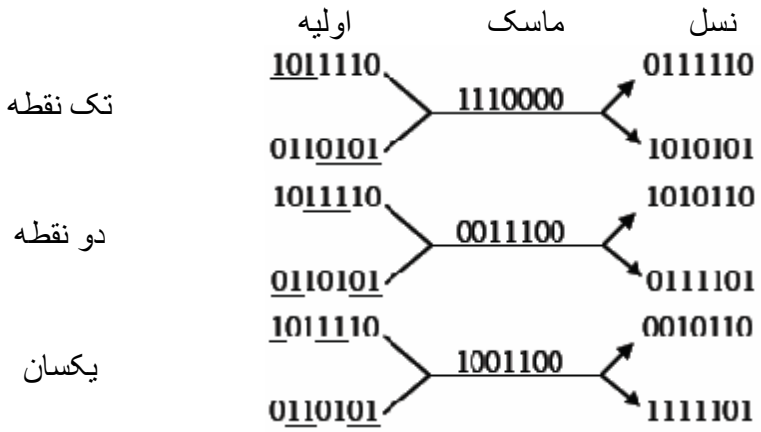
0111110
دورگه
1010101
گام بعدی :
0111110
دورگه
1010101
گام بعدی :
0111110
دورگه
1010101
گام بعدی :
0111110
دورگه
1010101
گام بعدی :
0111110
دورگه
1010101

101|1110
011|0101
101|1110
011|0101
101|1110
011|0101
101|1110
011|0101

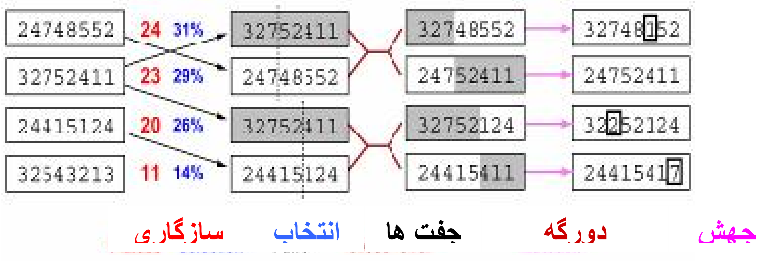
- پیاده سازی : از یک ماسک دورگه (رشته ی دودویی) استفاده نمایید
  - در مثال 1110000 می باشد
- دو والـــــــــــــد h_1 و h_2 ارایـــــــــــــه شـــــــــــــده :

$$(h_1 \wedge m) \vee (h_2 \wedge \neg m), (h_1 \wedge \neg m) \vee (h_2 \wedge m)$$

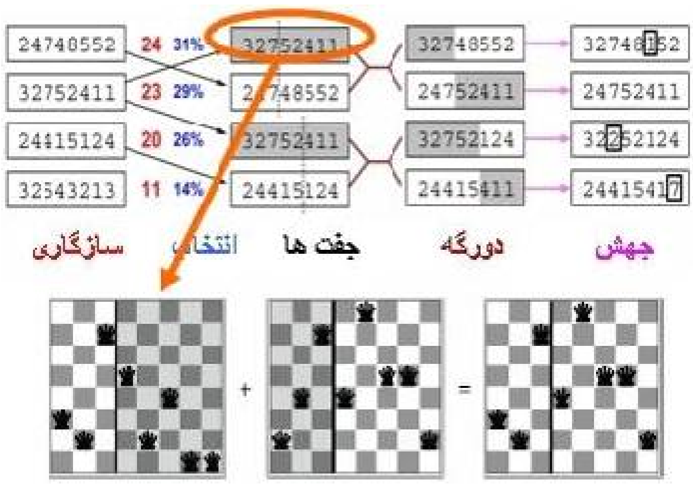
- دورگه در واژه های ماسک های اختیاری



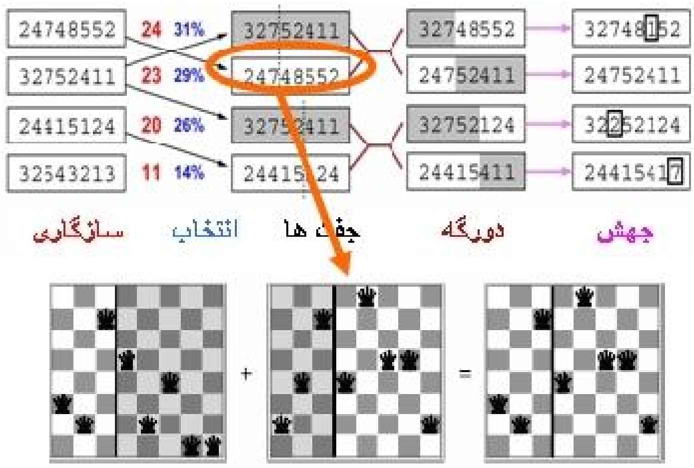
مثال : ۸ – وزیر



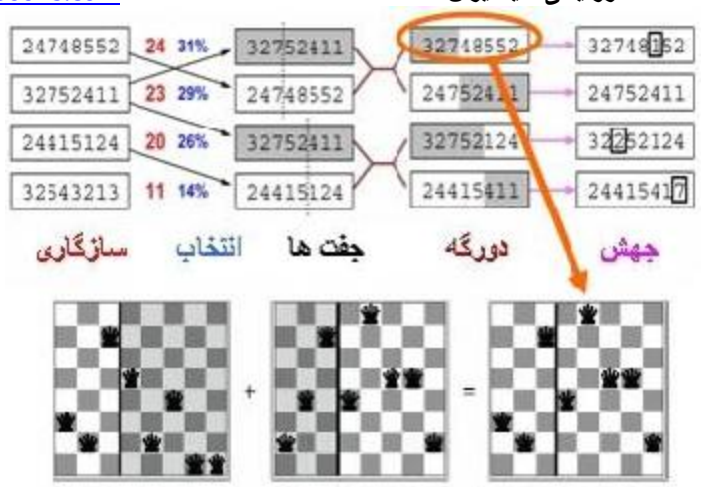
گام بعدی :



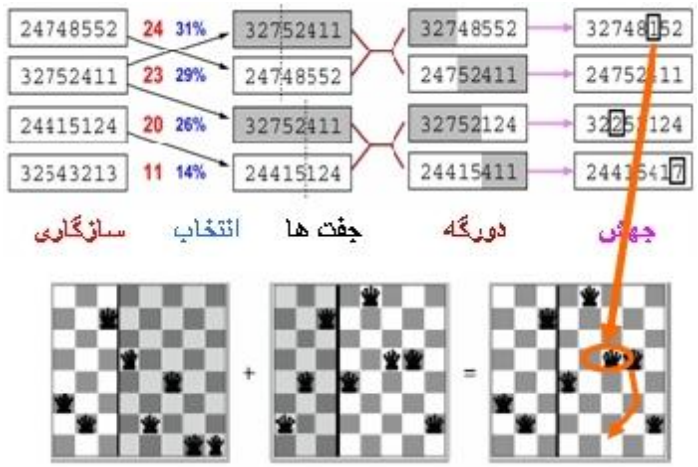
گام بعدی :

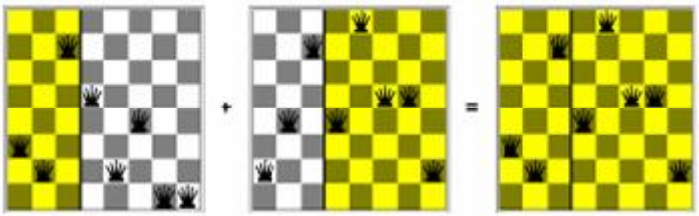


گام بعدی :



گام بعدی :





الگوریتم های ژنتیکی به حالت های رمزگذاری شده به صورت رشته نیاز دارند (GP ها از برنامه ها استفاده می کنند) . دورگه به زیر رشته های iff ، که اجزایی معنی دار هستند کمک می کند.

GA ها ≠ تکامل ، عوامل تکامل [1] واقعی ، تشکیلات [2] تکراری را رمز گذاری (encode) می کند.

الگوریتم های ژنتیکی به صورت جستجو

- وضعیت ها : راه حل های ممکن
- عملگرهای جستجو : جهش ، دورگه ، انتخاب

[1] genes
[2] machinery

- جستجوی موازی ، از آنجایی که چند راه حل در موازی به کار گرفته می شوند (جمعیت ها)
- تپه نوردی در تابع شایستگی ، اما بدون سراشیبی
- جهش و دورگه باید به ما اجازه دهند که از کمینه ی محلی خارج شویم
  - وابسته به گرم و سرد کردن شبیه سازی شده
- **نکته : برنامه نویسی ژنتیکی** وابسته به الگوریتم ژنتیکی می باشد
  - فردها برنامه ها هستند به جای رشته ها

کاربردهای الگوریتم ژنتیکی

- اغلب برای مسایل بهینه سازی مورد استفاده قرار می گیرد
  - آرایش مدار ، طراحی سیستم ، زمانبندی
- **خاتمه**
  - به اندازه ی کافی برای پیداکردن راه حل ، مناسب می باشد
  - بهبود قابل توجهی در مورد چند نسل ندارد
  - محدودیت زمانی

فضاهای حالت پیوسته - تصور کنید که می خواهیم جای سه فرودگاه را در کشور رومانی مشخص نماییم: فضای حالت شش بعدی توسط $(x_1, y_2), (x_2, y_2), (x_3, y_3)$ تعریف می شود . تابع هدف : $f(x_1, y_2, x_2, y_2, x_3, y_3)$ = مجموعه ای از فواصل مربعی از هر شهر به نزدیک ترین فرودگاه می باشد . متدهای مجزا [1] ، فاصله های پیوسته را به فاصله های مجزا تبدیل می نمایند. گرادیان تجربی با تغییرات $\pm\delta$ در هر وضعیت متناسب است. گرادیان از فرمول زیر محاسبه می شود:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

با توجه به رابطه ی $x \leftarrow x + \alpha \nabla f(x)$ می توانیم مقدار f را افزایش (کاهش) دهیم ، گاهی از اوقات می توان این معادله را به درستی برای حالت $\nabla f(x) = 0$ در صورتی که فقط یک شهر داشته باشیم حل نماییم . در روش نیوتن – رفسون [2] (1664-1690) برای حل $\nabla f(x) = 0$ در جایی که $H_{ij} = \partial^2 f / \partial x_i \partial x_j$ از فرمول زیر استفاده می شود:

$$x \leftarrow x - H_f^{-1}(x) \nabla f(x)$$

[1] discretization methods
[2] Newton-Raphson

فصل ششم

مسایل ارضای محدودیت

مسایل ارضای محدودیت [1]
فصل ششم

رئوس مطالب

- **مثال های مسایل ارضای محدودیت**
- **جستجوی معکوس لیست [2] برای مسایل ارضای محدودیت**
- **ساختار مساله و تجزیه [3] ی مساله**
- **جستجوی محلی برای مسایل ارضای محدودیت**

مسایل ارضای محدودیت

مساله ی جستجوی استاندارد : حالت ، یک " جعبه ی سیاه [4] " است – هر ساختمان داده ای قدیمی که از تست اهداف ، ارزیابی و جانشین پشتیبانی می کند .

مساله ی ارضای محدودیت : حالت ، توسط **متغیرهای** X_i و با **مقدارهای** درون **دامنه ی** D_i تعریف می شود. تست هدف ، مجموعه ای از محدودیت های معین کننده ی محدودیت های مجاز از مقادیر زیرمجموعه ی متغیر ها می باشد. یک **زبان ارایه ی رسمی** [5] مثال ساده ای از آن می باشد ، الگوریتم های **همه منظوره** [6] ی مفید را با توانایی بیش تر از الگوریتم های جستجوی استاندارد سبب می شود .

مثال نقشه ی رنگی : متغیرها عبارتند از : WA,NT,Q,NSW,V,SA,T . دامنه ها : $D_i = \{red, green, blue\}$

محدودیت ها : نواحی مجاور ، باید رنگ های متفاوتی داشته باشند. به عنوان مثال $WA \neq NT$ (اگر زبان این مطلب را اجازه بدهد) ، یا $(WA,NT) \in$

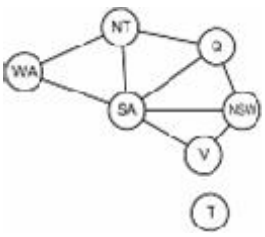
[1] Constraint Satisfaction Problems (CSPs)
[2] backtracking
[3] decomposition
[4] black box
[5] formal representation language
[6] general-purpose

{(red,green),(red,blue),(green,red),(green,blue),…}

گام بعدی :

راه حل ارایه شده همه ی شرایط را لحاظ می کند ، به عنوان مثال ،

{WA=red,NT=green,Q=red,
NSW=green,V=red,SA=blue,T=green}



گراف محدودیت [1]

مساله ی ارضای محدودیت دودویی [2] **:** هر محدودیت حداکثر دو متغیر را تشریح می نماید.

گراف محدودیت : گره ها ، متغیرها هستند ، کمان ها [3] ، محدودیت ها را نشان می دهند. الگوریتم های همه منظوره ی مساله ی ارضای محدودیت از ساختار درختی برای بالا بردن سرعت جستجو استفاده می کنند . توجه نمایید که شهر تاسمانیا یک زیر مساله ی مستقل است !

تنوع مسایل ارضای محدودیت

متغیرهای مجزا (گسسته) ،

- در دامنه های محدود ؛ اندازه d است ← (در نتیجه) $O(d^n)$ و انتساب ها کامل است .
  - به عنوان مثال ، مسایل ارضای محدودیت بولین [4] (NP – کامل)
- در دامنه های نامحدود (integer ها ، string ها (رشته ها) و .....)
- به عنوان مثال در زمانبند کار [5] ، متغیرها روزهای شروع / پایان برای هر کار هستند .

---

[1] Constraint graph
[2] Binary CSP
[3] arcs
[4] Boolean CSPs
[5] job scheduling

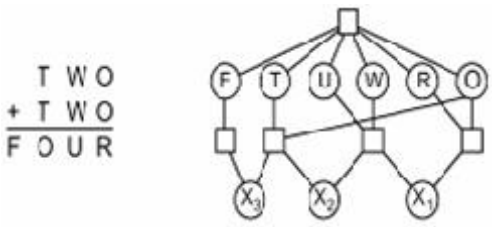
- به یک زبان محدود [1] نیازمند است ، به عنوان مثال ، $StartJob_1 + 5 \leq StartJob_3$
- محدودیت های خطی قابل حل و غیر خطی غیر قابل تصمیم گیری .

متغیرهای پیوسته [2] عبارتند از :

- به عنوان مثال ، زمان های شروع / پایان برای مشاهدات تلسکوپ [3] هابل [4]
- محدودیت های خطی قابل حل در چند زمانی به وسیله ی متدهای LP[5]

تنوع محدودیت ها

محدودیت های منحصر به فرد [6] ، شامل یک متغیر منفرد می شوند ، به عنوان مثال ، $SA \neq WA$.



محدودیت های دودویی ، شامل جفت هایی از متغیرها می باشند ، به عنوان مثال ، $SA \neq WA$.

محدودیت های مرتبه ی بالاتر ، شامل تعداد سه یا بیش تر متغیر می باشند ، به عنوان مثال ، محدودیت های با ستون های پنهان (رمزی) [7] .

برتری ها (محدودیت های نرم) ، به عنوان مثال ، رنگ قرمز بهتر از سبز است و اغلب قابل ارایه توسط یک مقدار برای هر متغیر منتسب می باشد ← (در نتیجه) مسایل بهینه سازی محدود به وجود می آیند .

مثال : مسایل رمزی

متغیرها : $F\ T\ U\ \ W\ R\ O\ X_1\ X_2\ X_3$. دامنه ها : {۰،۱،۲،۳،۴،۵،۶،۷،۸،۹} .

محدودیت ها : alldiff(F,T,U,W,R,O) و X_1 .O+O=R+10 و غیره .

مسایل ارضای محدودیت دنیای واقعی

مسایل انتساب ، به عنوان مثال ، چه کسی درس می دهد و چه کلاسی

مسایل برنامه ی زمانی ، کدام کلاس ارایه می شود ، چه زمانی و کجا ؟

پیکربندی سخت افزاری - صفحات گسترده [8] - زمان بند کننده ی انتقال - زمان بند کننده ی مرکز تولید .

---

[1] constraint language
[2] continuous variables
[3] Telescope
[4] Hubble
[5] برنامه نویسی خطی (Linear Programming)
[6] unary
[7] cryptarithmetic column constraints
[8] برنامه ای که امکان اجرای محاسبات روی چندین ستون از اعداد را برقرار کند ، spreadsheet نام دارد

توجه کنید که تعدادی از مسایل دنیای واقعی شامل متغیرهای با مقدار واقعی هستند .

فرمول بندی جستجوی استاندارد ، بیایید به صورت ساده شروع کنیم ، برخورد ساده ای داشته باشیم و سپس آن را اثبات نماییم . تا کنون حالت ها توسط مقادیر منتسب تعریف می شدند

- حالت اولیه : انتساب تهی ، {}
- تابع جایگزین : یک مقدار را به یک متغیر فاقد مقدار شده منتسب می کند که با انتساب فعلی تعارض یا ناسازگاری نداشته باشد. در نتیجه در صورتی که انتساب های مجاز وجود نداشته باشد ، رد می شود.
- آزمایش یا تست هدف : انتساب جاری کامل باشد .

۱) این برای همه ی مسایل ارضای محدودیت ، یکسان است ! 

۲) هر راه حل در عمق n با n متغیر به نظر می رسد ← (در نتیجه) از جستجوی اول عمق استفاده نمایید .

۳) مسیر نامربوط است ، بنابراین می توان از فرمول بندی با حالت کامل استفاده نمود.

۴) در عمق l ، داریم : b=(n-l)d

از این رو تعداد برگ ها برابر است با : !!! $n!d^n$ 

جستجوی معکوس لیست- انتساب متغیرها جابجایی پذیر می باشد ، توجه نمایید که ، [WA=red , NT=green] همانند [NT=green , WA=red] می باشد . فقط لازم است توجه نمایید که انتساب ها برای یک متغیر منفرد در هر گره می باشد ، در نتیجه b=d و تعداد d^n گره وجود دارد . جستجوی اول عمق برای مسایل ارضای محدودیت با انتساب های متغیر منفرد ، جستجوی معکوس لیست نام دارد. جستجوی معکوس لیست ، اساس الگوریتم ناآگاهانه برای مسایل ارضای محدودیت می باشد . می توان n- وزیر را برای $n \approx 25$ حل نمود .

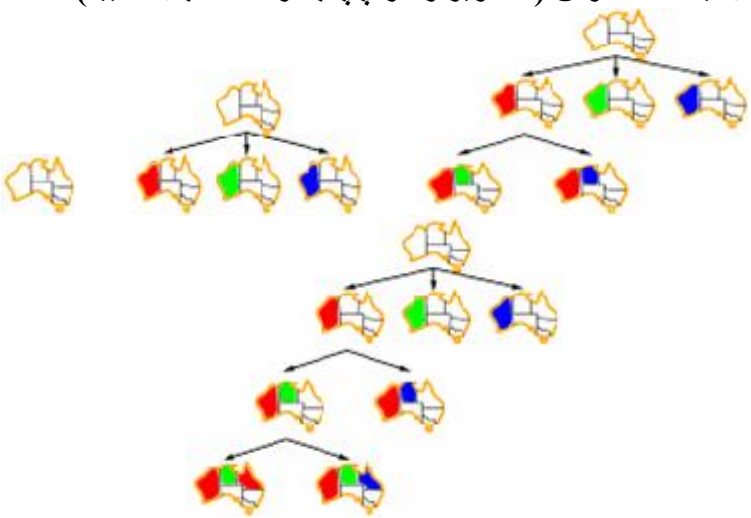
تابع Backtracking-Search(csp) ، راه حل یا عدم موفقیت را برمی گرداند
Recursive-Backtracking({},csp) را برگردان

تابع Recursive-Backtracking(assignment,csp) ، soln یا عدم موفقیت را برمی گرداند .

اگر انتساب (assignment) کامل است انتساب را برگردان
var ← Select-Unassigned-Variable(Variable[csp],assignment,csp)
برای هر مقدار (value) در Order-Domain-Values(var,assignment,csp) کارهای زیر را انجام بده
در صورتی که value با assignment ارایه شده در Constraint[csp] سازگار است
{ var = value } را به انتساب (assignment) اضافه نما
result ← Recursive-Backtracking(assignment,csp)
در صورتی که result ≠ failure است ، result را برگردان
{var = value} را از انتساب [1] بردار
عدم موفقیت را برگردان

[1] assignment

مثال لیست معکوس (تصاویر را از چپ به راست دنبال نمایید)



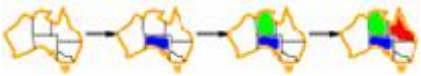
بهبود بخشیدن به کارایی لیست معکوس - متدهای همه منظوره می توانند افزایش زیادی در سرعت داشته باشند:

۱. کدام متغیر باید به بعدی نسبت داده شود ؟
۲. متغیرها به چه نظمی چیده شوند ؟
۳. آیا می توانیم خطای اجتناب ناپذیر را سریع پیدا نماییم ؟
۴. می توانیم سودی از ساختار مساله داشته باشیم؟

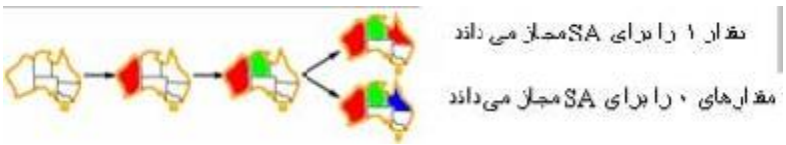
کم ترین مقادیر باقی مانده - کم ترین مقادیر باقی مانده [1] : متغیری را با کمترین مقادیر مجاز انتخاب نمایید .

a. قرمز دارای محدودترین مقدار می باشد
b. متغیری با کم ترین مقادیر مجاز را انتخاب نمایید

پیداکردن درجه - متغیرها را در طول MRV به طور تصادفی بشکنید . کشف درجه : متغیری با بیشترین محدودیت در متغیرهای باقی مانده را انتخاب نمایید .



کم ترین مقدار محدود کننده - یک متغیر ارایه شده ، کم ترین مقدار محدود کننده را انتخاب می نماید

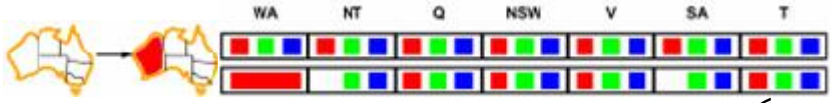


ترکیب کردن این ابتکارات ، داشتن ۱۰۰۰ وزیر را امکان پذیر می سازد .

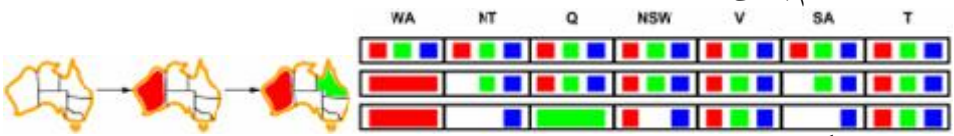
بررسی مستقیم [2] - ایده : مسیر باقی مانده با مقدارهای مجاز را برای متغیرهای منتسب نشده نگهداری نمایید . جستجو را هنگامی که هر متغیر مقدارهای مجاز ندارد خاتمه دهید.

[1] Minimum Remaining Values (MRV)
[2] forward checking

گام بعدی :

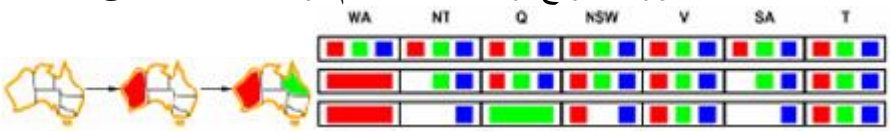


گام بعدی :



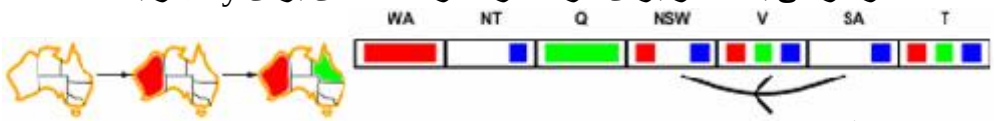
گام بعدی :

توزیع محدود - بررسی اطلاعات توزیع های مستقیم از متغیرهای منتسب به متغیرهای منتسب نشده ، به صورت سریع برای همه ی عدم موفقیت ها عمل نمی کند :

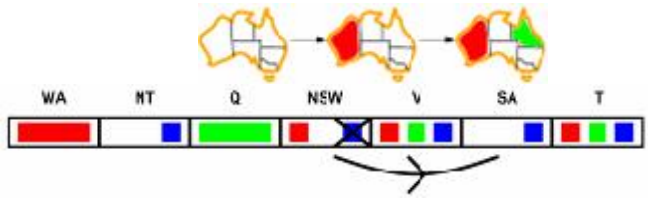


NT و SA هر دو نمی توانند آبی باشند ! توزیع محدود ، به صورت تکراری ، محدودیت ها را به صورت محلی اجرا می کند .

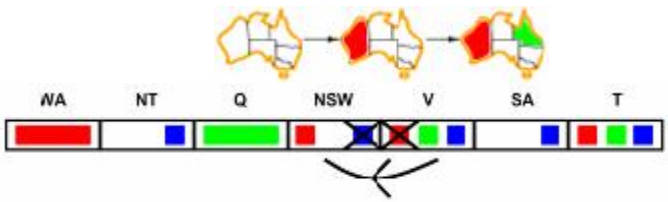
سازگاری قوس [1] - ساده ترین توزیع ، سازگاری هر قوس را سبب می شود . $X \rightarrow Y$ سازگار می باشد اگر برای هر مقدار x از X تعدادی برای y مجاز باشند .



گام بعدی :

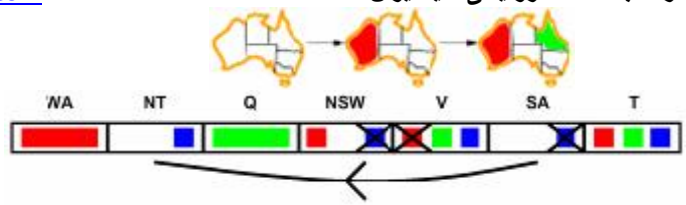


گام بعدی :



اگر X یک مقدار را از دست داد ، همسایه های X باید دوباره بررسی شوند.

[1] Arc consistency (AC)



اگر X یک مقدار را از دست داد ، همسایه های X باید دوباره بررسی شوند. سازگاری قوس زودتر از بررسی مستقیم ، عدم موفقیت پیدا می کند . می تواند به صورت یک پیش پردازنده و یا بعد از هر انتساب اجرا شود .

الگوریتم سازگاری قوس

تابع AC-3(csp) ، CSP و شاید دامنه های کاهش داده شده را برگرداند

ورودی ها ، CSP که یک مساله ی ارضای محدودیت دودویی با متغیرهای $\{X_1, X_2, ..., X_n\}$ می باشد

متغیرهای محلی ، queue ، که یک صف از قوس ها که همه به صورت اولیه قوس هایی در csp هستند

مادامی که queue خالی نیست کارهای زیر را انجام بده

$(X_i, X_j) \leftarrow$ Remove-First(queue)

اگر Remove-Inconsistent-Values(X_i, X_j) آن گاه

برای هر X_k در Neighbours(X_i) کارهای زیر را انجام بده

(X_k, X_i) را به queue اضافه کن

تابع Remove-Inconsistent-Values(X_i, X_j) موفقیت های صحیح iff را برمی گرداند

remove ← false

برای هر x در $Domain[X_i]$ کارهای زیر را انجام بده

در صورتی که هیچ مقداری در $Domain[X_j]$ اجازه نمی دهد که (x,y) محدودیت $X_i \leftrightarrow X_j$ را ارضا کند آن گاه x را از $Domain[X_i]$ حذف کن ؛ true ← removed

removed را برگردان

$O(n^2d^3)$ می تواند به $O(n^2d^2)$ کاهش داده شود.

سازگاری قوس فقط مقادیر ناسازگار را برمی دارد : فضای جستجو را هرس می کند .

مثال : X≠Y و X≠Z و Y≠Z با دامنه ی {۱و۲} سازگاری قوس می باشد اما راه حل نیست .

سازگاری قوس هدایتی [1]

- سازگاری قوس ، تجدید نظر را تکرار می کند (تعداد دفعات وابسته به قوس ها و دامنه ها می باشد)
- سازگاری قوس هدایتی : فقط یک بار بازنگری می کند
  - یک ترتیب متغیر را بازبینی می کند (<)

[1] Directional Arc Consistency (DAC)

- مساله ارضای محدودیت یک سازگاری قوس هدایتی است اگر هر قوس (X_i, X_j) که $X_i < X_j$ سازگاری قوس باشد
  - سازگاری قوس کلی تر از سازگاری قوس هدایتی است
  - برای درخت ها مناسب می باشد

سازگاری مسیر [1]

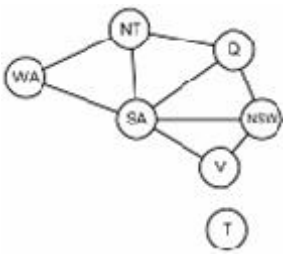
- یک مسیر $(X_1,...,X_m)$ یک مسیر سازگار است اگر برای هر جفت از مقادیر x_1 و x_m راضی کننده ی همه ی محدودیت های x_1 و x_m باشد ؛ یک انتساب برای $X_2,...,X_{m-1}$ که راضی کننده ی محدودیت های (X_i, X_{i+1})باشد وجود دارد
- مساله ی ارضای محدودیت با مسیر سازگار است در صورتی که هر مسیر ، سازگار باشد
  - در واقع ، شما فقط سازگاری همه ی مسیرهای با طول ۲ را احتیاج دارید
  - سازگاری مسیر و سازگاری قوس
    - سازگاری مسیر جفت های مقادیر را برمی دارد
    - محدودیت ها را صریح می سازد (به عنوان مثال ، X<Y و $Y<Z \Leftarrow X+1<Z$

K – سازگاری [2]

k – سازگاری ، اَشکال سازگاری را تعمیم می دهد

- مساله ی ارضای محدودیت ، k- سازگار است اگر برای هر مجموعه ی k-1 متغیری با انتساب سازگار و یک مقدار سازگار بتواند به هر k انتساب داده شود
  - در صورتی که هر متغیر با خودش سازگار باشد ۱- سازگار می باشد
  - ۲- سازگار ، سازگاری قوس می باشد
  - ۳- سازگار ، سازگاری مسیر می باشد
- k – سازگار قوی است در صورتی که
  - k – سازگار و k-1 – سازگار و ... و ۱- سازگار باشد

ساختار مساله [3]



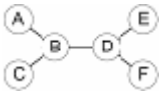
تاسمانیا و مین لند [4] زیر مساله هایی مستقل هستند. به صورت اجزای متصل شده به گراف محدودیت قابل تعریف اند.

تصور کنید که هر زیر مساله دارای متغیرهای c تا حد مجموع n باشد . در بد ترین حالت ، هزینه ی راه حل برابر است با $n/c.d^c$ و به صورت خطی بر حسب مقدار n می باشد . به عنوان مثال ، برای $n = 80 , d = 2 , c = 20$ داریم : 2^{80}= چهار بیلیون سال با سرعت ده میلیون گره بر ثانیه . 4.2^{20}= چهار دهم ثانیه با سرعت ده میلیون گره بر ثانیه .

مسایل ارضای محدودیت درختی ساختیافته [5]

---

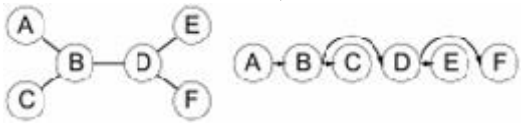
[1] Path Consistency (PC)
[2] K-consistency
[3] problem structure
[4] mainland
[5] Tree-structured CSPs



قضیه : در صورتی که گراف محدودیت هیچ حلقه ای نداشته باشد ، مساله ی ارضای محدودیت می تواند در زمان $O(nd^2)$ حل شود . این زمان را در جایی که در بدترین حالت ، زمان برابر $O(d^n)$ است مقایسه نمایید . این خصوصیت همچنین برای استدلال های منطقی و احتمالاتی به کار برده می شود : یک مثال مهم ، ارتباط میان محدودیت ها و پیچیدگی استدلال است.

الگوریتمی برای مسایل ارضای محدودیت درختی ساختیافته

۱ . متغیری را به صورت ریشه انتخاب نمایید ، متغیرها را از ریشه به برگ ها مرتب نمایید تا این که هر گره مقدم والد آن به صورت منظم در آید ،

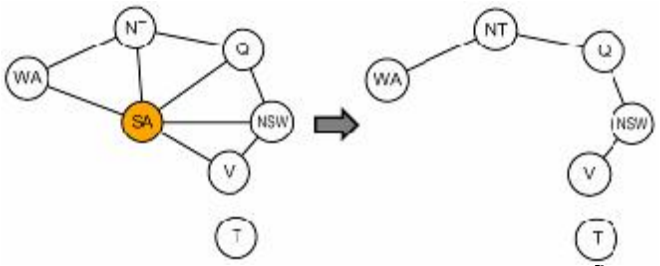


۲ . برای j از n تا ۲ (به صورت کاهشی) ، تا بع $(Parent(X_j), X_j)$ RemoveInconsistent را به کار ببرید

۳ . برای j از ۱ تا n ، X_j را بدون تناقض با $Parent(X_j)$ منتسب نمایید.

مسایل ارضای محدودیت درختی ساختیافته ی تقریبی [1]

شایسته سازی [2] : معرفی یک متغیر و هرس کردن دامنه ی همسایه های آن .



شایسته سازی برش [3] : معرفی (در همه ی موارد) یک مجموعه از متغیرها تا این که گراف محدودیت ، به یک درخت تبدیل شود. اگر اندازه ی برش c باشد ، در نتیجه ، زمان اجرای $O(d^c.(n-c)d^2)$ را خواهیم داشت ، برای c کوچک خیلی سریع است.

درخت تجزیه [4]

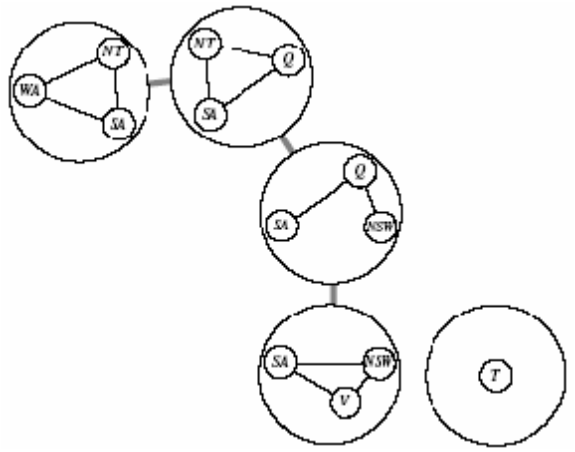
- تجزیه ی مساله به زیر مساله های متصل
  - ساختار ۳ – شکلی
  - زیرمساله ها = گره ها
- احتیاجات
  - برای هر متغیر ، لااقل یک گره
  - متغیرهای پیوسته باید لااقل در یک گره با هم به نظر برسند

---

[1] Nearly tree-structured CSPs
[2] Conditioning
[3] Cutset conditioning
[4] Tree decomposition

- متغیری که به نظر می رسد در دو گره است باید در هر گره دیگر در مسیر وجود داشته باشد

مثال درخت تجزیه



- محدودیت میان گره ها سازگاری متغیرها را سبب می شود
- تجزیه منحصر به فرد نمی باشد
  - زیر مساله ها باید تا حد امکان کوچک باشند
- زیرمساله ها به صورت مستقل حل می شوند
- ... سپس به وسیله ی متغیرهای مشترک به هم چسبانده [1] می شوند
- در صورتی که یک زیرمساله دارای هیچ راه حلی نباشد آن گاه کل مساله دارای هیچ راه حلی نمی باشد
- در صورتی که اندازه ی بزرگ ترین مساله ، w باشد ، آن گاه مساله می تواند در $O(nd^w)$ حل شود

الگوریتم های تکراری برای مسایل ارضای محدودیت - بالا رونده از تپه ، گرم و سرد کردن شبیه سازی شده ی معمولی با حالت های کامل کار می باشد ، توجه کنید که همه ی متغیرها نسبت داده شده اند . برای به کاربردن با مسایل ارضای محدودیت ، باید حالت های مجاز با محدودیت های عملگرهای ناخوشایند ، دوباره به مقدار متغیرها نسبت داده شوند.

انتخاب متغیر : هر متغیر ناسازگار را به صورت تصادفی انتخاب نمایید .

ابتکار انتخاب مقدار با کم ترین ناسازگاری : مقداری را که از کم ترین محدودیت ها تخلف می کند را انتخاب نمایید . توجه نمایید ، در بالا رونده از تپه ، $h(n)$ = شماره ی مجموع محدودیت های متخلف می باشد .

مثال : چهار وزیر

تصور نمایید که یک وزیر در هر ستون وجود دارد .

متغیرها : Q_1 و Q_2 و Q_3 و Q_4

دامنه ها : $D_i = \{1,2,3,4\}$

محدودیت ها :

$Q_i \neq Q_j$ (نمی توانند در ردیف یکسان باشند)

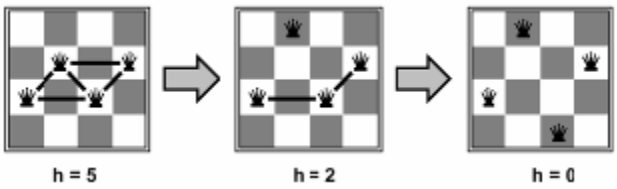
$|Q_i - Q_j| \neq |i - j|$ (در یک قطر هم نمی توانند باشند)

هر محدودیت را به مجموعه ای از مقادیر مجاز برای متغیرشان ترجمه نمایید

[1] glued

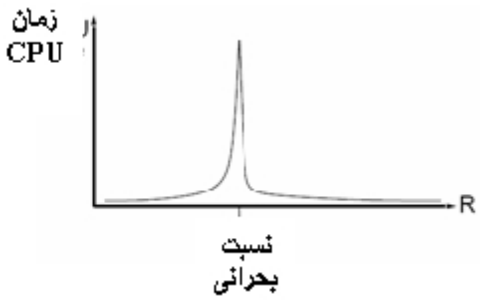
به عنوان مثال ، برای (Q_1, Q_2) مقادیر عبارتند از (۳و۱) و (۴و۱) و (۴و۲) و (۱و۳) و (۱و۴) و (۲و۴)

حالت ها (وضعیت ها) : چهار وزیر در چهار ستون ($4^4 = 256$ حالت) | عملگرها : وزیر را در ستون حرکت دهید | تست هدف : هیچ حمله ای وجود نداشته باشد . | ارزیابی : h(n) = تعداد حملات



عمل با کمترین ناسازگاری - حالت اولیه ی تصادفی ارایه شده ، می تواند n – وزیر را در ثابت زمانی تقریبی دلخواه n با احتمال زیاد حل کند ($n = 10{,}000{,}000$) . نمودهای همانند می توانند برای هر مساله ی ارضای محدودیت به صورت تصادفی صحیح باشند **به جز** در یک محدوده ی باریک از نسبت زیر : **R = تعداد محدودیت ها / تعداد متغیرها**

محدودیت های دودویی



- بیش تر تکنیک های گفته شده فقط با محدودیت های دودویی کار می کنند
- تعدادی از مسایل به طور طبیعی به صورت محدودیت های n تایی [1] رمزگذاری می شوند ، به عنوان مثال : X+Y=Z
- گره با سازگاری زوج ، دودویی نمی باشد ؛ به عنوان مثال $X \leq 12$
  - محدودیت های یکانی به سادگی می توانند با کاهش دامنه به کار گرفته شوند
  - $D_X = \{1,5,24\}$ و $X \leq 12 \leftarrow D'_X = \{1,5\}$

کاهش محدودیت های n تایی

- هر مساله ی ارضای محدودیت می تواند به یک مساله ی ارضای محدودیت دودویی معادل تبدیل شود
- چند روش وجود دارد ؛ در این جا ما **رمزکننده ی متغیر پنهان** [2] را نشان می دهیم
  - محدودیت های n تایی با متغیرهای جدید ارایه می شوند

[1] n-ary
[2] hidden variable encoding

- دامنه ی متغیرهای جدید ، چندتایی [1] (n تایی) هستند
- محدودیت های دودویی متغیرهای " اصلی " را به اجزای چندتایی ها بچسبانید

- مثال : X+Y=Z ، با $X,Y,Z \in N$
  - متغیر جدید U با دامنه ی $D_U = \{(x_1,x_2,x_3) \in N^3 \mid x_1 + x_2 = x_3\}$
  - از محدودیت $\pi_i(0,0)$ برای چسباندن مقدار I اُم استفاده نمایید ؛ به عنوان مثال

$$\pi_2(U,Y) = \{((x_1,x_2,x_3),x_2) \in N^3 \times N\}$$

سختی مساله ی ارضای محدودیت

- مساله ی ارضای محدودیت NP – کامل و سخت است
- از الگوریتم های مکاشفه ای استفاده کنید (همان طوری که در درس مطرح شده است)
  - در بدترین حالت ، پیچیدگی به صورت نمایی می باشد
  - اما در مواردی دارای رفتار خوبی می باشد
- الگوریتم های مکاشفه ای ، کلاس هایی از مسایل نرم و رام شدنی را تعریف می نماید
  - دارای زمان چندجمله ای با محدودیت هایی در مسایل است
  - به عنوان مثال ، مسایل با ساختار درختی

برنامه نویسی براساس محدودیت

- محدودیت ها ، یک راه طبیعی ارایه کننده ی مسایل می باشند
- مساله ی ارضای محدودیت ، در یک زبان برنامه نویسی ظاهر شده است
  - دارای فرمول بندی اعلانی (اظهاری) می باشد
  - برنامه نویسی با زبان های منطقی نظیر پرولوگ انجام می شود
  - دارای محیط اختصاصی می باشد (مثل موزارت [2])
  - پرولوگ به صورت های تجاری و رایگان ارایه شده است (به عنوان مثال SICStus یا GNU-Prolog)
- برای اطلاعات بیش تر نگاه کنید به
  - راهنمای برخط برای برنامه نویسی محدود :
    http://kti.mff.cuni.cz/~Ebartak/constraints/
  - آرشیو محدودیت ها :
    http://www.cs.unh.edu/ccc/archive

خلاصه

[1] tuple
[2] Mozart

مسایل ارضای محدودیت ، نوع خاصی از مسایل هستند : حالت ها توسط مقدارهای یک مجموعه ی ثابت از متغیر ها تعریف می شوند . و هدف آزمایش ، توسط محدودیت هایی بر مقدار متغیرها تعریف می شود . پیمایش معکوس = جستجوی اول - عمق با یک متغیر انتساب داده شده به هر گره . نظم دهنده ی متغیر و انتخاب ابتکاری مقدارها به طور بسیار مطلوبی به ما کمک می کند . بررسی مستقیم ، از انتساب هایی که خرابی آینده را به وجود می آورند یا تضمین می کنند جلوگیری می کند . پخش محدود ، کاری اضافی را برای مقدارهای محدود و کشف ناسازگاری ها ، انجام می دهد . ارایه ی مساله ی ارضای محدودیت ، تحلیل ساختار مساله را اجازه می دهد . درخت ساختیافته ی مسایل ارضای محدودیت می تواند در زمانی که به صورت خطی می باشد ، حل شود . ناسازگاری های کمینه ی تکراری ، معمولا در مطالعه موثر است .

فصل هفتم

مسایل ارضای محدودیت و برنامه نویسی ارضای محدودیت

فصل هفتم
مساله ی ارضای محدودیت و برنامه نویسی ارضای محدودیت [1]

مساله ی بهینه سازی محدودیت [2]

- مساله ی بهینه سازی محدودیت
  - مساله ی ارضای محدودیت استاندارد
    - مجموعه ای از متغیرها با دامنه های مرتبط
    - یک مجموعه از محدودیت ها
  - به اضافه ی تابع هدف نگاشت کننده ی هر راه حل به یک مقدار عددی
    - راه حل : انتساب کامل محدودیت های ارضا کننده
    - بیشینه سازی یا کمینه سازی مساله

- به عنوان مثال : رنگ آمیزی نقشه
  - هزینه وابسته به رنگ های استفاده شده دارد
  - چاپ با چند رنگ دارای هزینه ی بیش تری می باشد

الگوریتم های مسایل بهینه سازی محدودیت

- الگوریتم ساده : همه ی راه حل ها را به حساب می آورد
  - با استفاده از تمام روش هایی که تا کنون توضیح داده شده است
  - زمانی که تعداد زیادی راه حل وجود داشته باشد بد است (به عنوان مثال ، n – وزیر)
    - بهبودهای الگوریتم مساله ی ارضای محدودیت برای اجتناب از جستجو در شاخه های بدون راه حل طراحی می شود

- ایده ی کلیدی : شاخه ها را با زیر راه حل های بهینه هم هرس نمایید
  - مکاشفه برای هرس فضای جستجو

---

[1] Constraint Satisfaction Programming
[2] Constraint Optimisation Problem (COP)

مکاشفه ها

- تابع هدف ، انتساب های کامل را ارزیابی می کند
- تابع مکاشفه ای ، انتساب های جزیی را ارزیابی می نماید
  - به مقادیر عددی ، منتسب می کند
- تخمین تابع هدف
  - به همه ی راه حل های توسعه دهنده ی یک انتساب جزیی توجه نمایید
  - بهترین مقدار این راه حل ها را تخمین می زند
- انتساب های با مقادیر مکاشفه ای بد را توسعه ندهید
  - هرس اضافی
  - هرس استاندارد مساله ی ارضای محدودیت هم می تواند به کار گرفته شود

منشعب کردن و محدود کردن [1]

- الگوریتم پیمایش معکوس ، نگهداری جریان
  - بهترین راه حل تاکنون
  - محدوده ی بالایی (ارزیابی بهترین راه حل)

الگوریتم کمینه سازی

- مقداردهی اولیه
- روال بازگشتی [2]
  - وقتی که یک راه حل پیدا می شود
    - با بهترین راه حل نگهداری شده آن را مقایسه کن
  - حلقه ی انتخاب مقدار
    - اگر مفدار مکاشفه کوچک تر از حد بالایی باشد ، مراجعه کن
- بازده [3] وابسته است به
  - تابع مکاشفه ای
  - موفقیت اولیه ی راه حل های خوب

- در صورتی که بهینه شناخته می شود
  - حد بالا را با مقدار بهینه مقداردهی نما
  - متوجه پایان باشید

- حد قابل قبول
  - با یک راه حل زیر بهینه راضی می باشد
  - به اندازه ی کافی به یک مقدار ارایه شده نزدیک باشد
  - خاتمه در زمانی است که یک راه حل دارای ارزیابی کوچک تر از این مقدار است

تابع مکاشفه ای

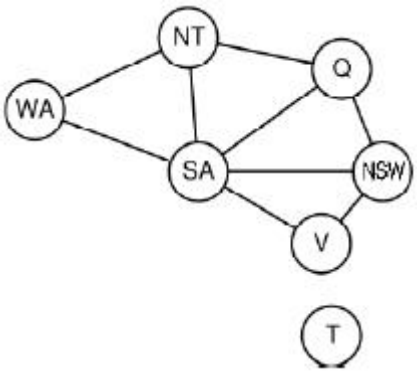
---

[1] Branch and Bound
[2] Recursive procedure
[3] efficiency

- به دو حریف نیازمند است
  - به اندازه ی تعداد شاخه ها ی ممکن هرس نمایید
    - مکاشفه ، مقادیر بالاتر را ارایه می نماید
  - از هرس کردن شاخه های دارای راه حل های بهینه خودداری نمایید
- مشابه مساله جستجوی آگاهانه می باشد (به عنوان مثال A^*)

- مکاشفه ، باید تابع بهینه را دست کم بگیرد
  - برای بیشینه سازی ، دست بالا بگیرد

مساله ی ارضای محدودیت به صورت برنامه نویسی منطقی




```
rgb(red).
rgb(green).
rgb(blue).

colourable([WA,SA,NT,Q,NSW,V,T])                    :-
rgb(WA),rgb(SA),rgb(NT),rgb(Q),rgb(NSW),rgb(V),rgb(T),

WA\=NT,NT\=Q,Q\=NSW,NSW\=V,SA\=WA,SA\=NT,SA\=Q,SA\=NSW,
SA\=V.
```

- هر مساله ی ارضای محدودیت با دامنه ی محدود می تواند به صورت یک شرط صریح تک با برخی واقعیات زمینه بیان شود
  - هر اتصال [1] یک محدودیت در متغیرهای شامل شده می باشد

- بنابراین : مساله ی ارضای محدودیت را به زبان پرولوگ بنویسید و اجازه بدهید که مفسر [2] آن را حل کند

- مساله : روش اول – عمق پرولوگ در بیش تر موارد ، بهترین نمی باشد

برنامه نویسی منطقی محدود

[1] conjunct
[2] interpreter

- کتابخانه هایی از الگوریتم های مساله ی ارضای محدودیت به پرولوگ استاندارد اضافه شده اند

- معمولا دارای نام CLP (دامنه) می باشند
  - CLP(FD) در دامنه های محدود
  - CLP(B) در مورد بولین ها
  - CLP(Q,R) در مورد عقلانیات و واقعیات

- با استفاده از مکانیزم محلی [1] پرولوگ برای توسعه ی زبان پیاده سازی شده است

SWI پرولوگ و CLP

کتابخانه های در دسترس

- CLP/bounds : حل کننده ی محدودیت دارای محدوده های صحیح [2]
  - محدوده ی متغیرها عضو متغیرهای نوع Integer می باشد
  - زیر مجموعه ای از SICStus(FD) می باشد
- clpr : برنامه نویسی منطقی محدود در واقعیات [3]

مثال : مساله ی رمزی

SEND+MORE=MONEY

```
:- use_module(library('clp/bounds')).
cryptarithmetic(S,E,N,D,M,O,R,Y) :-
Digits= [S,E,N,D,M,O,R,Y],
Carries = [C1,C2,C3,C4],
Digits in 0..9,
Carries in 0..1,
M              #=              C4,
O + 10 * C4 #= M + S + C3,
N + 10 * C3 #= O + E + C2,
E + 10 * C2 #= R + N + C1,
Y + 10 * C1 #= E + D,
M #>= 1,
S #>= 1,
all_different(Digits),
label(Digits).
```

منابع :

---

native [1]
Integer [2]
Constraint Logic Programming over Reals [3]

فصل هشتم

تئوری بازی ها

تئوری بازی ها [1]
فصل هشتم

رئوس مطالب

- **بازی ها**
- **بازی کامل** [2]
  - **تصمیم گیری های مینیماکس** [3]
  - **هرس** [4] **α-β**
- **محدودیت منابع و ارزیابی تقریبی** [5]
- **بازی های تصادفی** [6]
- **بازی های با اطلاعات غیر کامل** [7]

محیط های چندعاملی

- الگوریتم های جستجویی که ما تا حالا دیده ایم ، یک محیط تک عاملی را فرض می کنند .
- عامل در حال کار در یک جهان بی اثر [8] می باشد
- در مواردی که یک عامل باید دیگر عامل ها را به حساب بیاورد چه طور ؟
- محیط های رقابتی
  - شطرنج ، چکرز ، go و ...
  - حراجی ها [9] ، فروشگاه های برخط
- محیط های شرکتی (تعاونی) [10]
  - عامل های یک تیم فوتبال

---

[1] Game playing
[2] Perfect play
[3] Minimax decisions
[4] pruning
[5] Resource limits and approximate evaluation
[6] Games of chance
[7] Games of imperfect information
[8] neutral
[9] auctions
[10] cooperative

تئوری بازی ها – تئوری بازی ها ، شاخه ای از علم ریاضیات / علم اقتصاد می باشد که به اثرات متقابل میان چند عامل می پردازد . روش هایی را برای تشخیص رفتار بهینه ، تایین می نماید .

فرق ها :

بازی های با اطلاعات کامل : عامل ها به همه ی دانش در مورد محیط ، دسترسی دارند . ما این محیط را محیط کاملا قابل مشاهده می نامیم .

بازی های با اطلاعات ناقص : عامل باید در مورد جهان یا حریفش ، استنتاج نماید . ما این محیط را محیط قابل مشاهده به صورت جزیی می نامیم . مثال ها ، بازی های شانسی ، بیش تر برهم کنش های دنیای واقعی و برخی از حراجی ها می باشند .

مفروضات تئوری بازی ها :

عقلانیت کامل – عملکردها ، همیشه کار درست را انجام خواهند داد .

محاسبه ی نامحدود – عامل ها همیشه قادر به تشخیص این هستند که چه کاری برای انجام دادن ، درست می باشد .

همان طور که دیده ایم ، این مفروضات در برخی از موارد ، مصداق ندارند (درست نمی باشند) . به هر حال ، ما هنوز از برخی از ایده های تئوری بازی ها برای کمک در مورد تصمیم گیری در مورد این که چه کاری برای انجام ، درست می باشد ، استفاده می نماییم . بنابراین ، یک عامل چگونه در یک محیط چند عاملی چگونه تصمیم گیری نماید که چه کار باید بکند ؟

روش های بهینه

- بیایید با ساده ترین نوع بازی ها شروع کنیم .
  - دو نفره
  - اطلاعات کامل
  - رقابتی به صورت کامل – یکی می برد و دیگری می بازد
- ما می توانیم این بازی را به صورت یک مساله ی جستجو تعریف نماییم .

تئوری بازی ها یکی از قدیمی ترین مباحثی است که در هوش مصنوعی راجع به آن زیاد مطالعه شده است . دلیل این مطلب آن است که :

- ◊ افراد بازی ها را دوست دارند ! و خوب می توانند با بازی ها کار کنند .
- ◊ اغلب ، بازی ها به صورت یک نماینده ی هوش دیده می شوند .
- ◊ یک توضیح واضح از محیط می باشند .
- ◊ اما فضاهای حالت ، خیلی بزرگ و پیچیده اند ، بنابراین برخی اوقات اطلاعات اتفاقی و ناکار آمد می باشند و این باعث به وجود آمدن چالش در مساله می شود .
- ◊ وقتی که هوش مصنوعی خوب کار می کند بازی ها واضح و صریح می باشند .

بازی ها برای هوش مصنوعی مانند جایزه ای بزرگ برای مسابقه ی طراحی اتومبیل هستند .

مسایل جستجوی بازی ها – در بازی ها حرکت حریف [1] غیر قابل پیش بینی می باشد در نتیجه راه حل ، یک استراتژی مشخص کننده ی یک حرکت ، برای هر جواب ممکن حریف است . محدودیت های زمانی موجود برای پیداکردن هدف ، نا مطلوب یا نا خوشایند هستند ، در نتیجه ، باید تخمین زده شوند .

تاریخچه : کامپیوتر به خطوط ممکن بازی توجه می کند (Babbage,1846) | الگوریتمی برای بازی کامل (Zermelo,1912;Von Neumann,1944) | وسعت محدود ، ارزیابی

[1] opposite

تقریبی (Zuse , 1945 ; Wiener , 1948 ; Shannon , 1950) | برنامه ی شطرنج اولیه (Turing , 1951) | آموزش ماشینی [1] برای بهبود بخشیدن به درستی ارزیابی (Samuel , 1952-57) | بازبینی یا هرس برای اجازه دادن به جستجوی عمیق تر (McCarthy , 1956)

انواع بازی ها

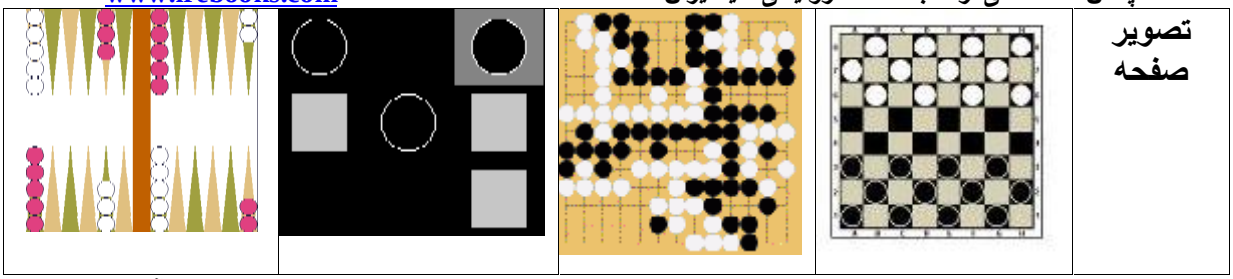
	قطعی	شانسی
اطلاعات کامل	شطرنج[2]،بازی چکرز[3]،go	تخته نرد[4]
اطلاعات ناقص	تیک تا تو کور[5]	پل[6]،پوکر[7]

شطرنج : بازی ای است که بر روی یک صفحه انجام می شود و برای دو بازیگر است که شانزده مهره را با توجه به قوانینی معین حرکت می دهند . هدف مات کردن شاه طرف مقابل می باشد . **بازی چکرز :** صفحه ی بازی چکرز برای دو نفر می باشد که هر نفر دارای دوازده مهره می باشد ؛ هدف پریدن و دستگیر نمودن مهره های حریف می باشد . **بازی go :** بازی ای بر روی صفحه برای دو بازیگر می باشد که شمارنده هایی را روی یک شبکه قرار می دهند ، هدف محاصره کردن و سپس دستگیر کردن شمارنده های حریف می باشد . [8] **تیک ـ تاک ـ تو کور :** بازی تیک – تاک – تو دارای دو بازیگر می باشد . برای هر دو بازیگر هدف این است که اولین کسی باشند که سه شی ء همانند را در یک ردیف ، ستون یا قطر قرار می دهند . صفحه ی این بازی دارای یک شبکه ی سه در سه می باشد . بنابراین دارای نه خانه می باشد . [9] **تخته نرد :** بازی ای بر روی صفحه است و دو بازیگر دارد . هر بازیگر دارای پانزده مهره می باشد که آن ها را در بیست و چهار خانه ی مثلثی شکل با توجه به عدد ظاهر شده روی دو تاس حرکت می دهد . [10] **پل :** یک بازی کارتی حقّه ای برای چهار نفر می باشد که این چهار نفر به صورت دو گروه دو نفری با هم بازی می کنند و در هر طرف هر گروه مقابل هم می نشیند . بازی پل دارای دو مرحله می باشد : پیشنهاد و بازی . [11] **پوکر :** بازی ای کارتی می باشد ، محبوب ترین بازی از یک دسته از بازی ها به نام بازی های همچشمی می باشد که در آن بازیگران با کارت هایی کاملا پنهان یا اندکی پنهان روی یک چیزی شرط بندی می کنند ، سپس چیزی که شرط بندی روی آن انجام شده است به بازیگر یا بازیگران باقی مانده ای که دارای بهترین ترکیب کارت ها هستند جایزه داده می شود . [12] در زیر تصویر صفحه ی چند بازی ای که هم اکنون معرفی نمودیم مشاهده می نمایید :

نام بازی	چکرز	go	تیک – تاک – تو کور	تخته نرد

[1] machine learning
[2] chess
[3] checkers
[4] backgammon
[5] blind tictactoe
[6] bridge
[7] poker
[8] wordnet.princeton.edu/perl/webwn
[9] http://www.seeingwithsound.com/tictactoe.htm
[10] wordnet.princeton.edu/perl/webwn
[11] en.wikipedia.org/wiki/Bridge_game
[12] en.wikipedia.org/wiki/Poker

تصویر صفحه



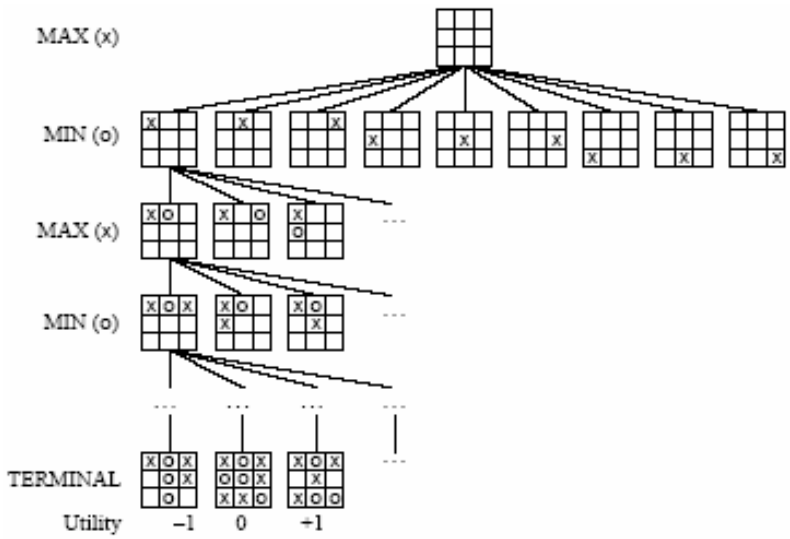
تئوری بازی به صورت جستجو – به بازی دو نفره ای با اطلاعات کامل توجه نمایید . آیا ما می توانیم این بازی را به صورت مسایل جستجو فرمول بندی نماییم ؟

- ◊ وضعیت صفحه ی بازی ، وضعیت جستجوی مساله می باشد .
- ◊ حرکت های مجاز ، عملگرها هستند .
- ◊ وضعیت های پایانی وضعیت هایی هستند که درآن ها بازی را برده ایم یا بازنده شده ایم یا بازی بی نتیجه مانده است .
- ◊ ما می توانیم به بردن عدد ۱+ ، به باختن عدد ۱- و به بازی بی نتیجه مانده عدد ۰ را نسبت دهیم .
- ◊ ما می خواهیم یک روش (یک راه برای انتخاب حرکت ها) که بازی را می برد پیدا نماییم .

چالش در جستجوی بازی

- ◊ یک حریف وجود دارد .
- ◊ حریف بداندیش است – چیزهای خوب را برای خود می خواهد و چیز های بد را برای شما می خواهد .
- ◊ ما باید تصمیم گیری حریفمان را شبیه سازی نماییم .
- ◊ واژگان تخصصی : یک بازیگر بیشینه وجود دارد (که بیش ترین نفع را برای خودش می خواهد) و یک بازیگر کمینه وجود دارد که کم ترین نفع را برای خودش می خواهد .

درخت بازی – در حالت کلی ، ما می توانیم درخت بازی را برای هر بازی به صورت زیر رسم نماییم .



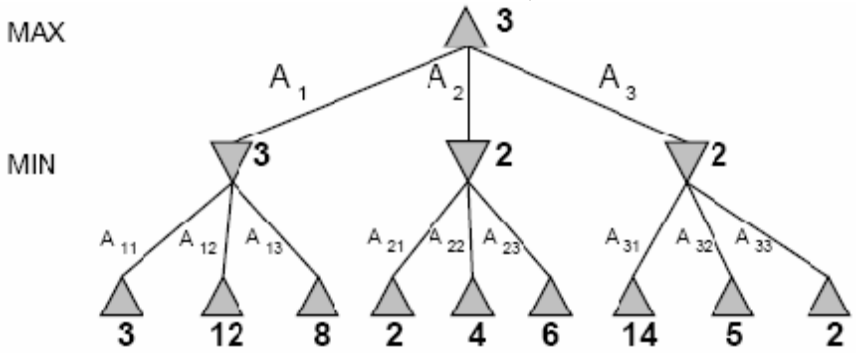
مینیماکس – یک روش برای تایین حرکت های خوب در تیوری بازی های کامپیوتری می باشد که با انتخاب حرکتی که باخت بازیگر را کم می کند می تواند پیش بینی هایی که انتظار داریم را

برآورده نماید ، البته با فرض این که بازیگر مقابل هم همان روش [مورد انتظار ما] [1] را انتخاب می نماید (و بنابراین برای به و جود آوردن بیش ترین باخت تلاش می نماید) . امکان برد یا باخت برای یک رشته از حرکت ها معمولا با استفاده از یک تابع ارزیابی استاتیک به کار گرفته شده برای وضعیت بازی ، و در پایان رشته تشخیص داده می شود . [2]

بازی کامل قطعی [3] ، بازی های با اطلاعات کامل می باشند .

ایده : حرکت با وضعیت بالاترین مقدار مینیماکس = بهترین بازدهی ممکن برخلاف بهترین بازی .

به عنوان مثال ، برای بازی دو نفره داریم :



الگوریتم مینیماکس

تابع Minimax-Decision(state) یک عملکرد را برمی گرداند

ورودی : state ، که حالت جاری بازی می باشد

a ای را که در Actions(state) بیشینه کننده ی Min-Value(Result(a ,state)) می باشد را برگردان

تابع Max-Value(state) یک مقدار مفید را یرمی گرداند

در صورتی که تابع Terminal-Test(state) دارای مقدار بازگشتی درست می باشد Utility(state) را برگردان

$v \leftarrow -\infty$

برای a,s در تابع Successor(state) ، $v \leftarrow Max\ (v,Min\text{-}Value(s))$

مقدار v را برگردان

تابع Min-Value(state) یک مقدار مجاز را برمی گرداند

در صورتی که Terminal-Test(state) درست است ، Utility(state) را برگردان

$v \leftarrow \infty$

برای a,s درون Successors(state) ، $v \leftarrow Min(v,Max\text{-}Value(s))$

v را برگردان

خصوصیات مینیماکس

کامل بودن ؟؟ فقط در صورتی که درخت محدود باشد (شطرنج قوانین معینی برای این کار دارد).

NB یک استراتژی محدود حتی در یک درخت نامحدود هم می تواند وجود داشته باشد !

بهینه بودن ؟؟ بله

پیچیدگی زمانی ؟؟ $O(b^m)$

---

[1] مترجم

[2] www.informatics.susx.ac.uk/books/computers-and-thought/gloss/node1.html

[1] Deterministic در کامپیوتر ، نتیجه ی فرایندی که به موقعیت های اولیه ورودی ها بستگی دارد

پیچیدگی فضا ؟؟ O(bm) (جستجوی اول عمق)

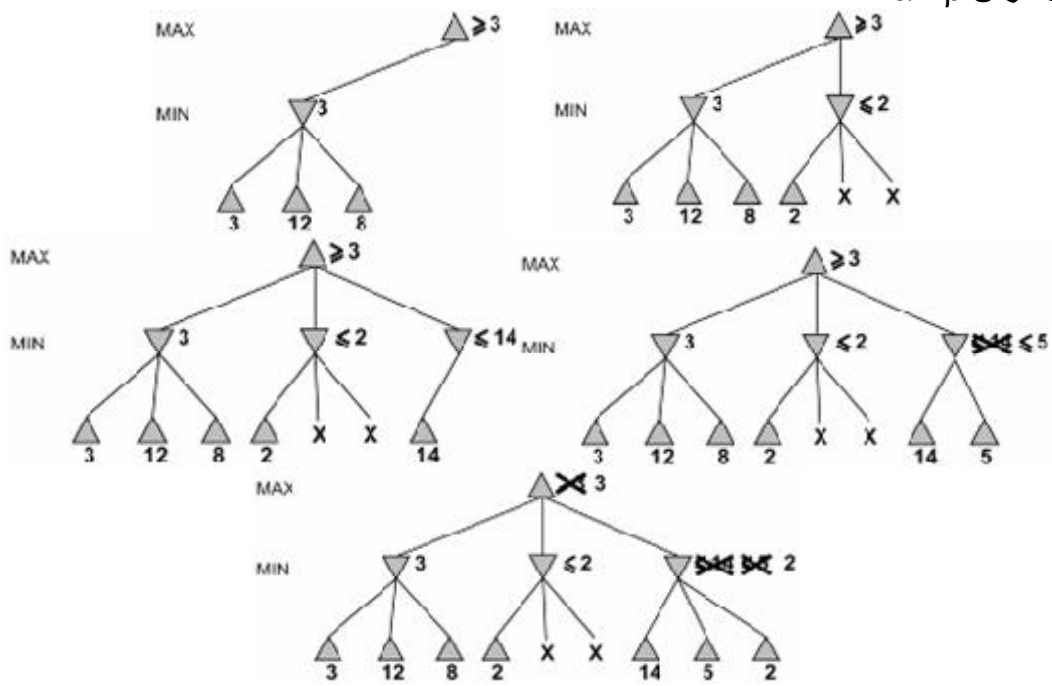
برای شطرنج ، $b \approx 35$ ، $m \approx 100$ (برای بازی های مستدل) ← راه حل دقیق کاملا اجرا نشدنی است .

اما آیا ما احتیاج داریم که هر مسیر را پیدا نماییم ؟

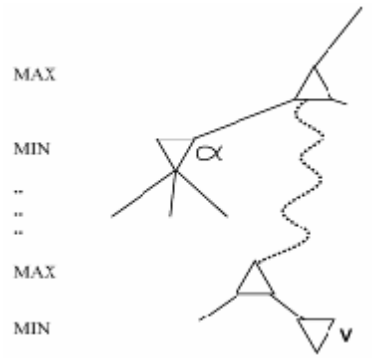
هرس α - β

- ◊ روشی استاندارد برای بازی های قطعی و با اطلاعات کامل می باشد .
- ◊ ایده ی این روش شبیه هرس آلفا می باشد : اگر مسیری به نظر می رسد بدتر از مسیری که ما داریم می باشد ، آن را حذف کن !
- ◊ الگوریتم این روش شبیه مینیماکس می باشد ، اما جریان برگ با مقدار بهترین را برای بازیگر ما (آلفا) و بازیگر حریف (بتا) نگهداری می کند .
- ◊ در صورتی که بهترین حرکت تغییر نکرد ، با وجود این که ما می توانیم آن را با جستجو پیدا نماییم ، آن گاه ما دیگر نباید جستجو نماییم .

مثال هرس α - β



چرا این روش α-β نامیده شده است ؟



α بهترین مقداری است (برای MAX) که تا کنون برای مسیر جاری پیدا شده است . اگر V بد تر از α باشد ، MAX از آن جلوگیری خواهد کرد ← آن شاخه را هرس نمایید . β را هم به همین صورت برای MIN تعریف نمایید .

الگوریتم α-β

تابع Alpha-Beta-Decision(state) یک عمل را بر می گرداند
a ای را که در Actions(state) بیشینه کننده ی Min-Value(a,state)) می باشد را برگردان

تابع Max-Value(state,α,β) یک مقدار مجاز را بر می گرداند
ورودی ها ، state که حالت جاری در بازی می باشد
α مقدار بهترین برای MAX در طول مسیر به state می باشد .
β مقدار بهترین برای MIN در طول مسیر به state می باشد .
در صورتی که Terminal-Test(state) درست می باشد ، Utility(state) را برگردان
V← - ∞
برای a,s در تابع Successors(state) کارهای زیر را انجام بده
v ← Max(v,Min-Value(s,α,β))
در صورتی که v ≥β باشد ، v را برگردان
α←Max(α,v)
انتهای حلقه
v را برگردان

تابع Min-Value(state, α,β) یک مقدار مجاز را بر می گرداند
شبیه مقدار Max-Value ، اما با قوانین رزرو شده ی α و β

خصوصیات α-β - بازبینی یا هرس بر روی نتیجه ی نهایی اثری ندارد . حرکت خوب و منظم اثر هرس را بهبود می بخشد . با " منظم کننده ی کامل " ، پیچیدگی زمانی = $O(b^{m/2})$ ← (درنتیجه) عمق جستجو را مضاعف می کند . در نتیجه می تواند به سادگی به عمق هشت برسد و بازی شطرنج خوبی را ارایه نماید . یک مثال ساده از مقداری که در مورد آن استدلال می کنیم ، جایی است که محاسبات با هم ارتباط دارند (یک شکل از فرا استدلال) . متاسفانه 35^{50} هنوز غیر ممکن است !

محدودیت منابع - فرض کنید ما صد ثانیه زمان داریم و می توانیم 10^4گره در ثانیه کشف کنیم درنتیجه در صد ثانیه (هر حرکت) 10^6گره را می توانیم کشف نماییم .

مشی یا روش استاندارد [1] :

- از تست برش [2] به جای تست پایانه [3] استفاده نمایید ، به محدودیت عمق توجه نمایید.
- از ارزیابی به جای فایده یا سود استفاده نمایید .

به تابع ارزیابی [4] که شرایط مطلوب را تخمین می زند توجه نمایید . تصور کنید که ما ۱۰۰ ثانیه وقت داریم و سرعت کاوش 10^4 گره بر ثانیه باشد ← 10^6 گره در هر حرکت ≈ $35^{8/2}$ ← α-β به عمق هشت می رسد ← برنامه ی شطرنج بالنسبه خوب می باشد .

---

[1] Standard approach
[2] Cutoff-Test
[3] Terminal-Test
[4] evaluation function

توابع ارزیابی – یک تابع ارزیابی ، تابعی است که خوبی یک وضعیت را اندازه گیری می نماید (به عنوان مثال ، شانس برنده شدن در آن وضعیت را اندازه می گیرد)

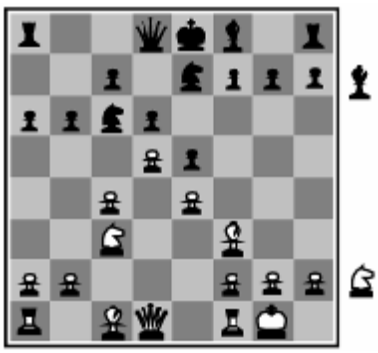
- ◊ می تواند توسط طراح ارایه شود یا با آزمایش به دست آید .
- ◊ اگر ترکیبات صفحه بتواند به صورت مستقل ارزیابی شود آن گاه یک انتخاب خوب یک تابع توزیع شده ی خطی می باشد :

$w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$ ، s وضعیت صفحه ی بازی می باشد .

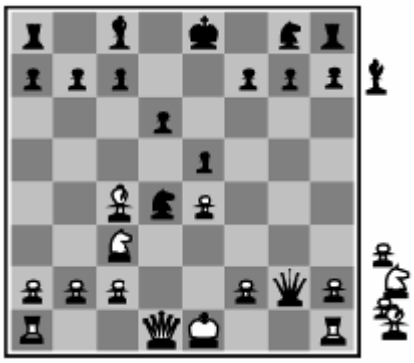
f_i تعداد یک نوع مهره بر روی صفحه ی بازی می باشد (مثلا فیل) و w_i ارزش مهره ها (مثلا یک برای سرباز و سه برای فیل)

توجه کنید :

- ◊ برنامه های پیشرفته از ترکیبات غیرخطی هم استفاده می نمایند .
- ◊ ترکیبات و توزیع ها جزیی از قوانین شطرنج نمی باشند .



با حرکت سیاه وضعیت سفید بهتر می شود



با حرکت سفید سیاه برنده می شود

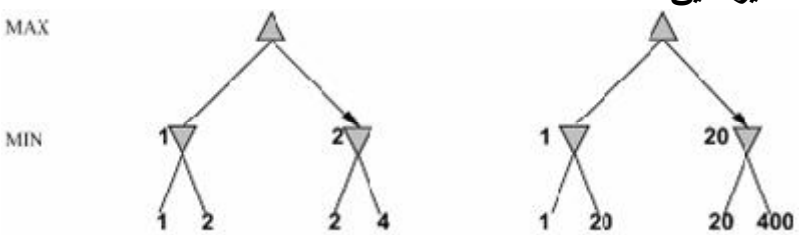
برای شطرنج ، معمولا توزیع خطی [1] جمع عناصر [2] زیر است :

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$$

توجه کنید با $w_1 = 9$

$f_1(s)$ = تعداد وزیران سفید – تعداد وزیران مشکی و .....

انحراف [3] : مقادیر دقیق



رفتار ، تحت هر تغییر یکنواخت Eval ثابت می ماند . تنها چیزهای نظم دهنده : بازدهی در بازی های قطعی به صورت یک تابع مفید ترتیبی عمل می نماید .

بازی های قطعی در عمل [4]

---

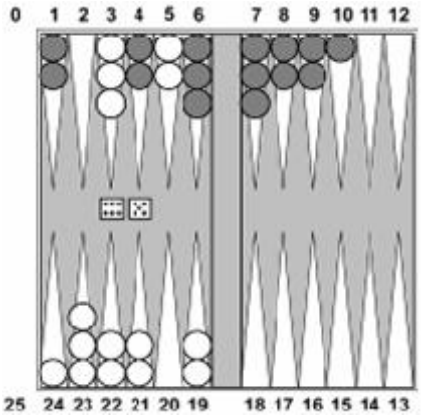
[1] linear weighted
[2] features
[3] digression
[4] deterministic games in practice

- بازی چکرز : چینوک [1] به چهل سال قهرمانی قهرمان جهان ماریون تینزلی [2] در سال ۱۹۹۴ خاتمه داد. از پایگاه داده ی مشخص کننده ی پایان بازی شطرنج برای تعریف تمام بازی برای وضعیت های شامل هشت یا کم تر مهره در صفحه استفاده شد ، یک مجموعه از ۲۴۷و۴۰۱و۷۴۸و۴۴۳ حالت. Deep Blue قهرمان شطرنج جهان ، گری کاسپاروف [3] را در شش بازی در سال ۱۹۹۷ شکست داد. Deep Blue ، دویست میلیون حالت را در هر ثانیه جستجو می کرد و از یک ارزیابی خیلی ماهرانه استفاده می کرد و از متدهای فاش نشده برای توسعه دادن تعدادی از خطوط جستجو تا ۴۰ تا استفاده می نمود.

اتلو [4] : قهرمانانی که خیلی خوب هستند نمی پذیرند که با کامپیوتر رقابت نمایند .

Go : قهرمانان نمی پذیرند که با کامپیوتر رقابت کنند . در این بازی ، b>300 ، بنابراین بیش تر برنامه ها از الگوی براساس دانش برای پیشنهاد حرکت های ممکن استفاده می نمایند .

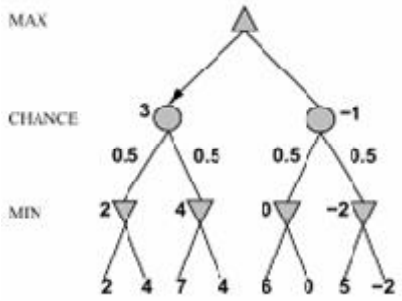
بازی های غیرقطعی : تخته نرد



بازی های غیرقطعی در حالت کلی

در بازی های غیر قطعی ، شانس به وسیله ی تاس [5] و برهم زدن کارت [6] معرفی می شود.

مثال ساده شده با سکه ی دورویه [7] :



الگوریتمی برای بازی های غیرقطعی

◊ از آمار و احتمالات داریم :

---

[1] Chinook
[2] Marion Tinsley
[3] Gary Kasparov
[4] Othello
[5] dice
[6] card-shuffling
[7] coin-flipping

مقدار مورد انتظار ار یک متغیر تصادفی x میانگینی از مقدارهای ممکن x می باشد و توسط احتمالات رخ داده ی P(x) توزیع شده است :

$$E(x) = \sum_{x \in X} xP(x)$$

◊ الگوریتم Expectminimax : شبیه Minimax می باشد ، تفاوت آن در این است که از مقدارهای مورد انتظار در گره های شانسی استفاده می نماید .

◊ EXpectminimax بازی کامل را ارایه می نماید .

مینیماکس بازی کامل را ارایه می کند . ما همچنین باید از گره های شانس استفاده نماییم :

...

در صورتی که حالت (state) یک گره ماکزیمم است ، بالاترین ExpectiMinimax-Value از Successors(state) را برگردان

در صورتی که حالت (state) یک گره مینیمم است ، کم ترین ExpectiMinimax-Value از Successors(state) را برگردان

در صورتی که حالت (state) یک گره شانسی است ، میانگین ExpectiMinimax-Value از Successors(state) را برگردان

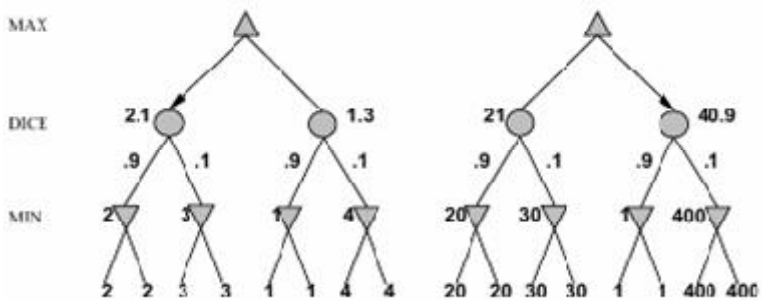
...

بازی های غیرقطعی در عمل

نقش تاس b را افزایش می دهد : بیست و یک نقش ممکن با دو تاس وجود دارد . تخته نرد ≈ بیست حرکت مجاز دارد (می تواند با نقش ۱-۱ ، ۶۰۰۰ باشد)

$depth\ 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$ با افزایش عمق ، احتمال رسیدن به یک گره ارایه شده (داده شده) جمع می شود (افزایش می یابد) ← مقدار پیش بینی شده ، کاهش می یابد . بازبینی یا هرس α-β به مراتب کم تر موثر است . TDGammon از جستجوی عمق دو + ارزیابی (Eval) خیلی خوب استفاده می کند ≈ در سطح قهرمانی جهان قرار دارد .

انحراف : مقادیر دقیق



رفتار [1] فقط به وسیله ی انتقال خطی مثبت [2] Eval نگه داشته می شود. از این رو Eval باید متناسب با بازدهی مورد انتظار باشد

بازی های با اطلاعات ناقص - توجه کنید که در بازی های کارتی با کارت های با مقدار اولیه ی متضاد کارت ها ناشناخته می باشند . معمولا ما می توانیم یک احتمال را برای هر مقدار ممکن محاسبه نماییم . بازی سیمز [3] فقط دارای یک نقش بزرگ تاس ، آن هم در ابتدای بازی است.

---

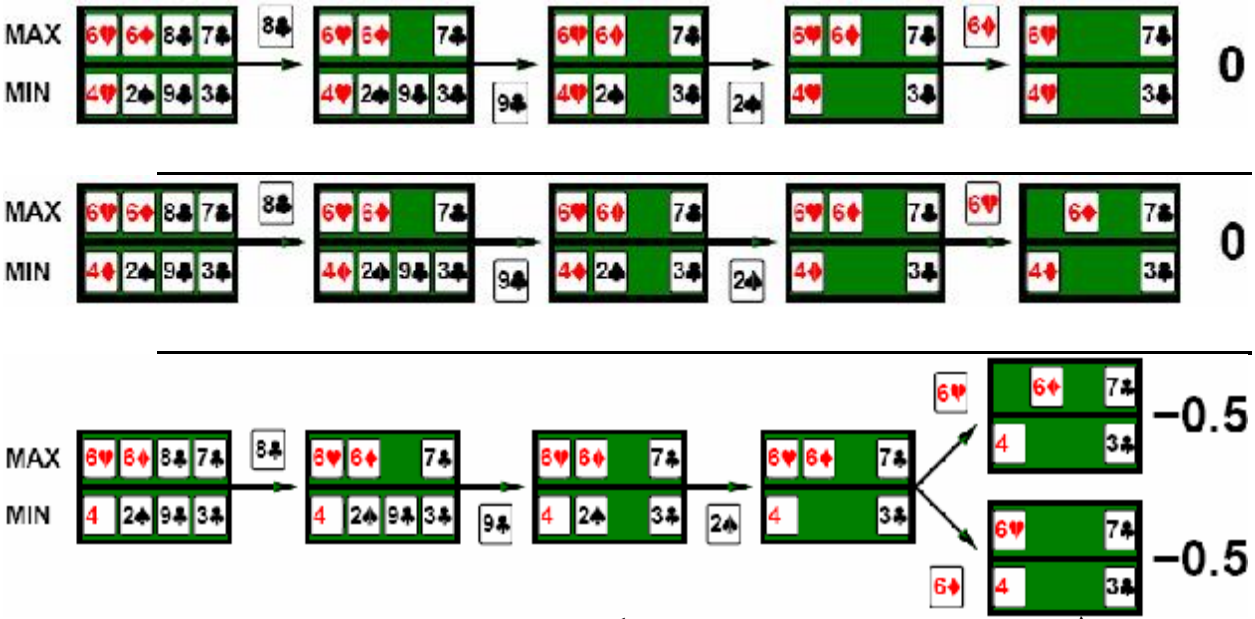
[1] Behaviour
[2] positive linear
[3] Seems

ایده : مقدار هر عمل را در هر مرحله از مینیماکس محاسبه نمایید و سپس عمل را با بالاترین مقدار مورد انتظار برای هر مقدار انتخاب نمایید . نکته ی مهم : اگر یک عمل برای همه ی مقادیر ، بهینه باشد ، آن عمل بهینه می باشد. GIB در حال حاضر بهترین برنامه ی پل می باشد ، ایده ی آن تقریبا به صورت زیر است :

۱) تولید صد مقدار سازگار با اطلاعات پیشنهاد شده

۲) انتخاب عملی که به صورت میانگین با بیش ترین حقه ها ما را برنده می سازد

مثال - پل / whist (نوعی بازی ورق) / hearts hand با چهار کارت و در صورتی که در ابتدا Max بخواهد بازی نماید به صورت زیر می باشد :



تحلیل مناسب [1] - تصور کنید که مقدار یک عمل ، میانگینی از مقادیر آن در همه ی حالت های واقعی که **اشتباه** هستند می باشد. با توانایی دید [2] جزیی [3] ، مقدار یک عمل وابسته به حالت اطلاعات یا حالت تصور [4] عاملی که در آن است ، می باشد. می توانیم یک درخت حالت های اطلاعاتی را تولید و جستجو کنیم . موارد زیر منجر به رفتارهای هوشمند می شوند :

- ✓ عمل کردن با اطلاعات دریافت شده
- ✓ علامت دهی برای همتای آن
- ✓ عمل کردن به صورت تصادفی برای کمینه کردن فاش سازی اطلاعات

Deep Blue

- Deep Blue ، یک مثال خوب از ترکیب توان محاسباتی مدرن و تکنیک ها (روش ها) ی هوش مصنوعی می باشد . در سال ۱۹۹۷ میلادی ، Deep Blue گری گاسپاروف (بهترین شطرنج باز بشری) را مغلوب نمود . و این کار ، یک هدف هوش مصنوعی از دهه ی ۵۰ میلادی بود . Deep Blue یک ابررایانه ی 32-Node RS/6000 می باشد . برنامه ی Deep Blue به زبان سی نوشته شده بود و از جستجوی الفا – بتا استفاده می

---

[1] Proper analysis
[2] observability
[3] partial
[4] belief state

نمود و دارای سخت افزار مخصوص محاسبه کننده ی توابع ارزیابی بود . ارزیابی ها ، دویست میلیون حالت در ثانیه بود که ۱۰۰ – ۲۰۰ بیلیون وضعیت را در سه دقیقه ، ارزیابی می کند . میانگین فاکتور انشعاب در شطرنج ۳۵ می باشد . فضای حالت در حدود 10^{70} می باشد . مقدار زیادی از دانش با دامنه ی مخصوص افراد برای به وجود آوردن دقت در توابع ارزیابی به کار گرفته شد . از پایگاه داده های استادان بزرگ شطرنج در بازی های قبلی و همچنین از تعداد زیادی کتاب های پیشرفته استفاده شده بود .

منابع :

http://www.inf.unibz.it/~tessaris/teaching/AI/handouts/games.pdf
http://aima.eecs.berkeley.edu/slides-pdf /chapter06.pdf
http://www.cs.usfca.edu/~brooks/F03classes/ai/lectures/adversarial-search.pdf

فصل نهم

عامل های منطقی

عامل های منطقی [1]
فصل نهم

ریوس مطالب

- **عامل ها براساس دانش [2]**
- **دنیای Wumpus**
- **منطق در حالت کلی – مدل ها و دارایی ها [3]**
- **منطق گزاره ای [4](بولین)**
- **تساوی [5] ، صحت [6] ، توانایی راضی کردن [7]**
- **قوانین استنتاج [8] و اثبات قضیه [9]**
  - **زنجیره ی مستقیم [10]**
  - **زنجیره ی معکوس [11]**
  - **تحلیل [12]**

پایگاه های دانش

الگوریتم های مستقل از دامنه [14] ←
محتوای با دامنه ی مشخص [15] ←

موتور استنتاج [13]
پایگاه دانش

---

[1] Logical Agents
[2] Knowledge-based agents
[3] entailment
[4] Propositional
[5] Equivalence
[6] validity
[7] satisfiability
[8] Inference rules
[9] theorem proving
[10] forward chaining
[11] backward chaining
[12] resolution
[13] inference engine
[14] domain-independent algorithms
[15] domain-specific content

پایگاه دانش = (یعنی) مجموعه ای از **عبارات** به یک زبان **رسمی** [1] .
شیوه ی [2] اعلانی یا اظهاری برای ساختن یک عامل (یا یک سیستم دیگر) :
بگویید ( ) [3] که به چه چیزی برای دانستن نیاز دارد . سپس می تواند از خودش **سوال کند ( )** [4] که چه کاری را باید انجام دهد – جواب ها باید از KB یا پایگاه دانش گرفته شوند . عامل ها می توانند در **سطح دانش** [5] دیده شوند . توجه نمایید که **چه چیزی را آن ها می دانند** ، بدون این که بدانند چگونه به کار گرفته می شوند ، یا در **مرحله ی کاربرد** [6] به ساختمان های داده ای [7] در پایگاه دانش و الگوریتم هایی که آن ها را به کار می برند توجه می کنند .

یک عامل ساده ی براساس دانش

الگوریتم :

تابع KB-Agent(percept) یک عمل را برمی گرداند
متغیر KB که یک متغیر static است ، یک پایگاه دانش می باشد
t ، یک شمارنده با مقدار اولیه ی صفر و نشان دهنده ی زمان می باشد
Tell(KB,Make-Percept-Sentence(percept,t))
action ← Ask(KB,Make-Action-Query(t))
Tell(KB,Make-Action-Sentence(action,t))
t ← t+1
عمل (action) را برگردان

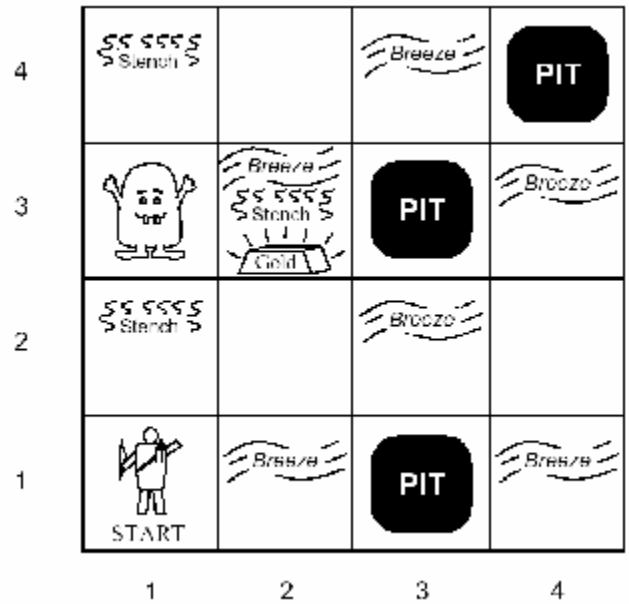
عامل باید قادر باشد :
حالت ها ، عملکرد ها و غیره را ارایه نماید
دریافت های ادراکی [8] جدید را یکی کند [9].
ارایه های درونی جهان را به روز نماید
خصوصیات جهان پنهان را استنباط [10] نماید
عملکردهای مناسب را استنباط نماید

توضیح PEAS دنیای wumpus

اندازه گیری عملکرد یا معیارکارایی

---

[1] formal
[2] approach
[3] Tell ( )
[4] Ask ( )
[5] knowledge level
[6] implementation level
[7] data structures
[8] percepts
[9] Incorporate
[10] deduce



طلا (۱۰۰۰+) ، مرگ (۱۰۰۰-) ، (منفی یک) در هر گام ، (۱۰-) برای استفاده از پیکان [۱]

محیط

مربع های نزدیک به wumpus بدبو [۲] هستند
مربع های نزدیک به چاله [۳] خوش هوا [۴] هستند
درخشش [۵] طلا در مربع همانند می باشد
تیراندازی [۶] ، wumpus را در صورتی که شما روبه روی آن باشید می کشد
تیراندازی فقط پیکان را مصرف [۷] می نماید
قلاب [۸] فقط طلا را در صورتی که در مربع یکسان باشد انتخاب می نماید
قطره های آزاد شونده ی طلا در مربع شبیه می باشند

محرک ها

چپ گرد [۹] ، راست گرد [۱۰] ، مستقیم [۱۱] ، قلاب [۱۲] ، آزاد [۱۳] ، تیراندازی [۱۴]

حس کننده ها

خوش هوا ، درخشش ، بدبو

خصوصیات دنیای wumpus

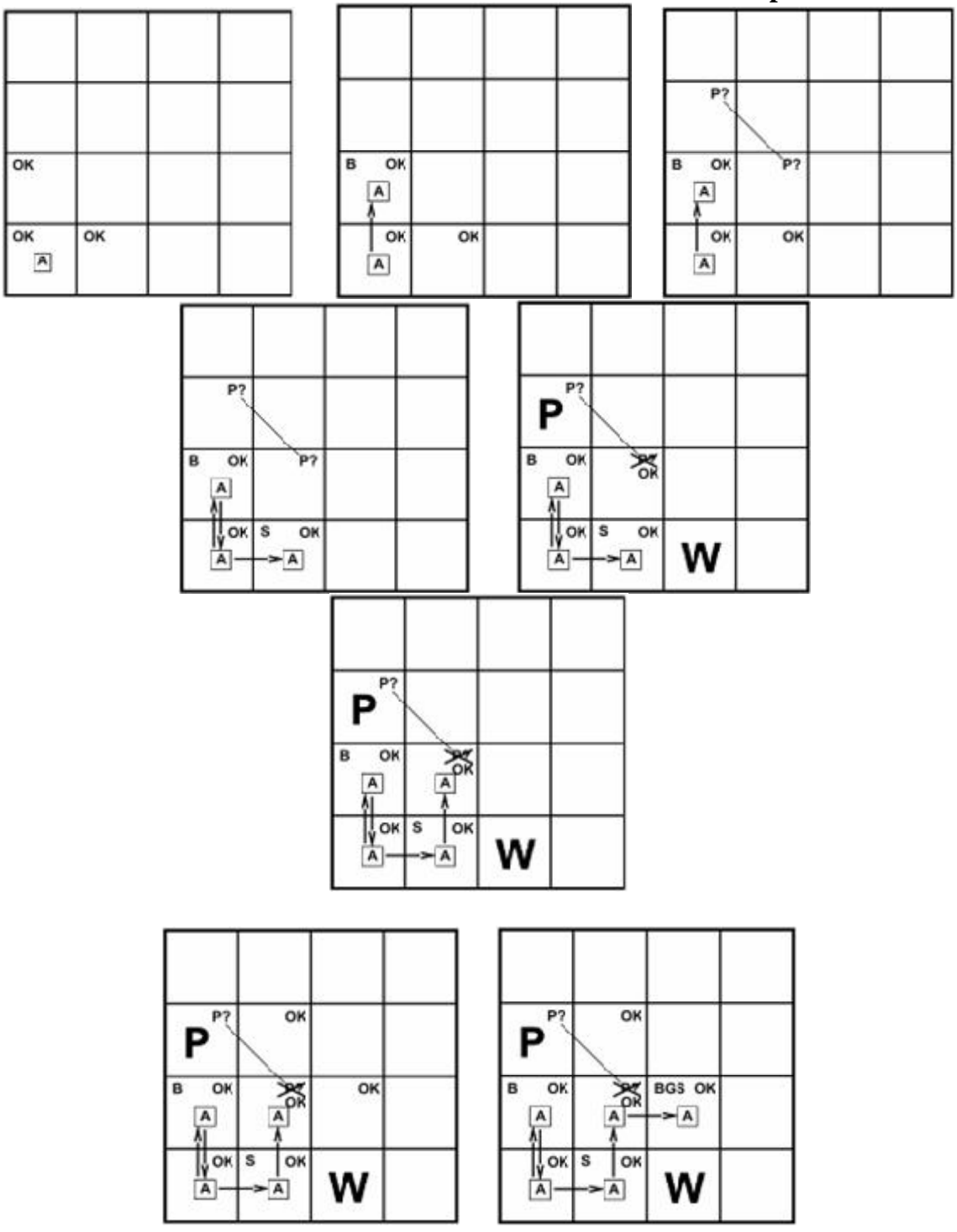
قابل مشاهده بودن ؟؟ نه – فقط دارای ادراک محلی است

---

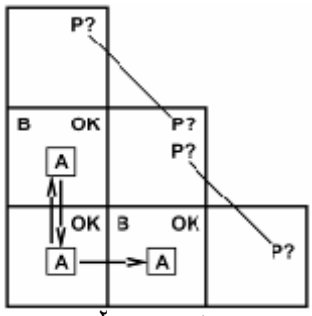
[1] arrow
[2] smelly
[3] pit
[4] breezy
[5] glitter
[6] glitter
[7] use up
[8] grabbing
[9] Left turn
[10] Right turn
[11] Forward
[12] Grab
[13] Release
[14] Shoot

قطعیت ؟؟ بله – نتایج دقیقا مشخص است
دوره ای نه – دارای ترتیب در سطح عملکردها است
ایستایی ؟؟ بله – wumpus و چاله ها نمی توانند حرکت نمایند
گسسته؟؟ بله
تک عاملی ؟؟ بله wumpus در اصل یک طرح طبیعی می باشد

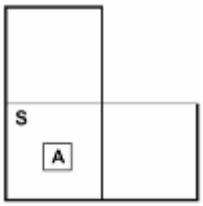
اکتشاف دنیای Wumpus



دیگر نقاط تنگنا [1]

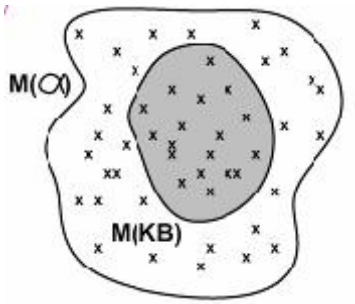


خوش هوایی در (۱و۲) و (۲و۱) ← هیچ عمل ایمنی [2] ای وجود ندارد



بوی بد در (۱و۱) می باشد ← نمی توانیم حرکت بکنیم
می توانیم از یک روش (استراتژی) اجباری استفاده نماییم :
مستقیما به جلو شلیک کنیم
اگر wumpus آنجا بود ← می میرد ← ایمن
اگر wumpus آنجا نبود ← ایمن

منطق در حالت کلی - منطق ها ، زبان هایی رسمی برای ارایه ی اطلاعات هستند ، از این رو نتایج می توانند خالی باشند . ظاهر [3] ، جمله ها را در زبان تعریف می کند . سمانتیک ها یا معناها [4] ، " معنی [5] " جملات را تعریف می کنند ؛ به تعریف درست یک جمله در جهان توجه نمایید ، به عنوان مثال ، در زبان ریاضی - $x+2\geq y$ یک جمله است ولی $x2+y>$ یک جمله نمی باشد . $x+2\geq y$ درست است در صورتی که عدد $x+2$ کم تر از y نباشد . $x+2\geq y$ در جهان در صورتی که $x=7,y=1$ باشد صحیح می باشد . $x+2\geq y$ در جهان در صورتی که $x=0,y=6$ غلط است .



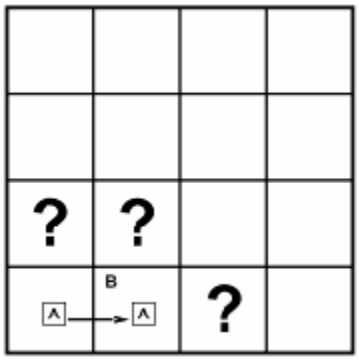
دنبال کردن [6] - معنای دنبال کردن این است که چیزی توسط دیگری دنبال شود : $KB \models \alpha$.
پایگاه دانش KB جمله ی α را دنبال می کند اگر و فقط اگر α در همه ی جهان ها در جایی که KB درست است ، درست باشد. به عنوان مثال ، $x+y=4$ ، $4=x+y$ را دنبال می نماید . دنبال کردن ، ارتباطی میان جملات است که براساس **سمانتیک ها یا معناها** می باشند. به عنوان مثال دیگر ، اگر پایگاه دانش ، دارای " پیراهن ، سبز رنگ می باشد " و "پیراهن ، خط دار (راه راه) می باشد " باشد ، در این صورت " پیراهن ، سبز رنگ است یا راه راه است را دنبال می نماید "

---

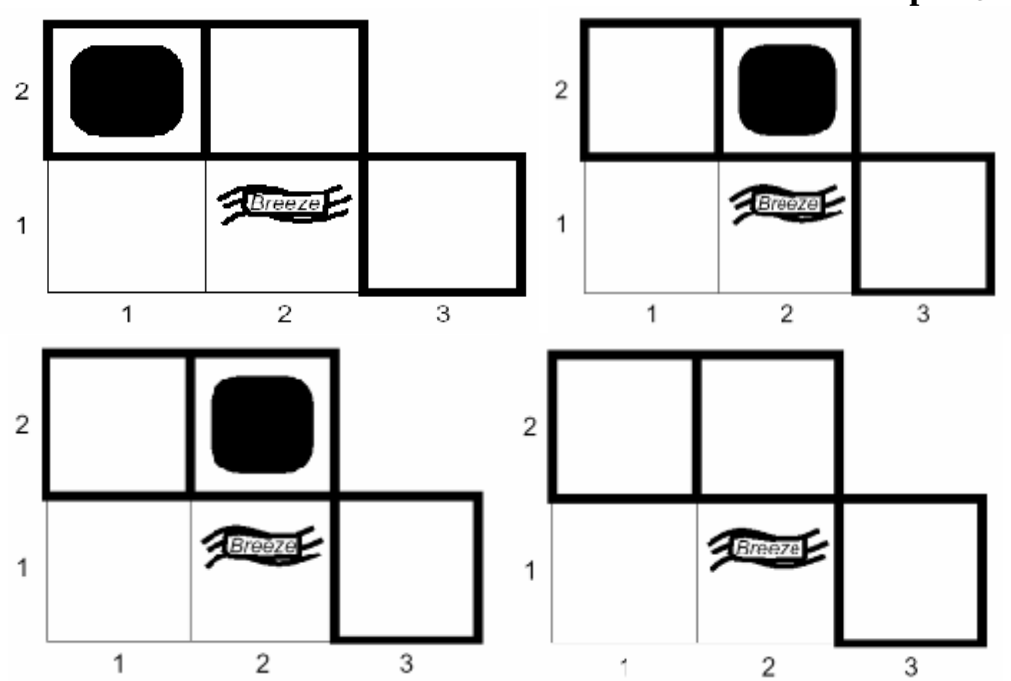
[1] tight spots
[2] tight spots
[3] Syntax
[4] Semantics
[5] meaning
[6] Entailment

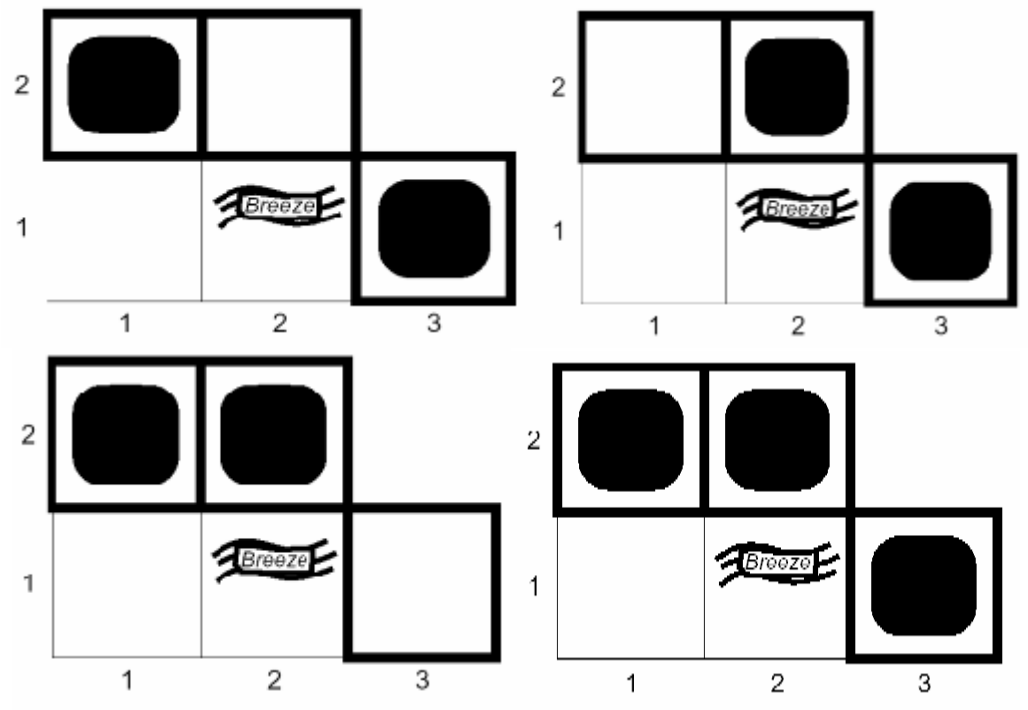
مدل ها - گوییم **m مدلی از** یک جمله ی **α** است در صورتی که **α** در **m** درست باشد. اگر M(α) مجموعه ای از تمام مدل های α باشد در این صورت KB ╞α است اگر و فقط اگر M(KB) زیر مجموعه یا مساوی M(α) باشد . به عنوان مثال ، **KB** = برندگان بزرگ و برندگان قرمز
α = برندگان بزرگ

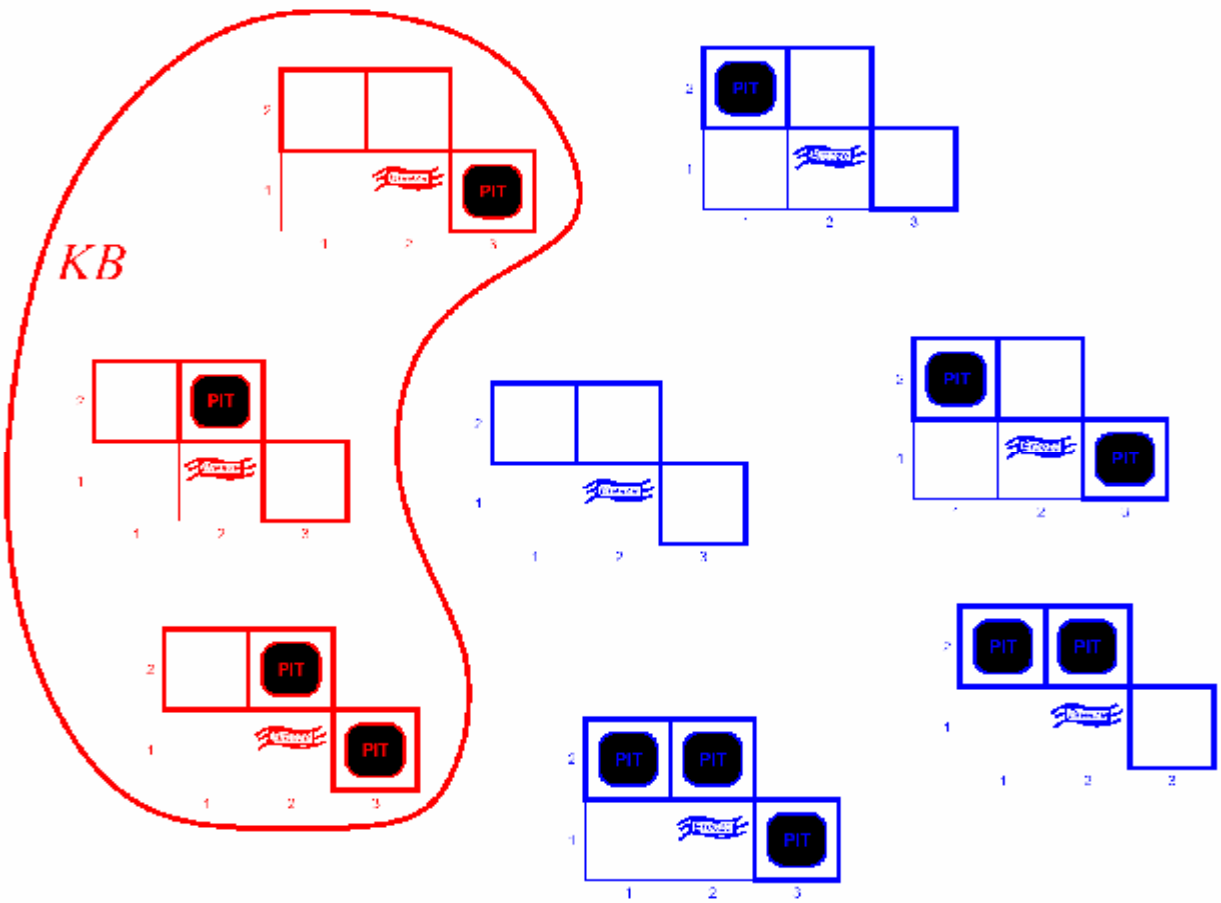
دنبال کردن در دنیای wumpus - وضعیت بعدی هیچ چیزی را در [۱و۱] پیدا نمی کند ، حرکت به راست در [۱و۲] خوش هواست . به مدل های ممکن برای ? ها توجه نمایید . این ها فقط گودال ها را به خود می گیرند . سه انتخاب بولین وجود دارد ← ۸ مدل ممکن است .



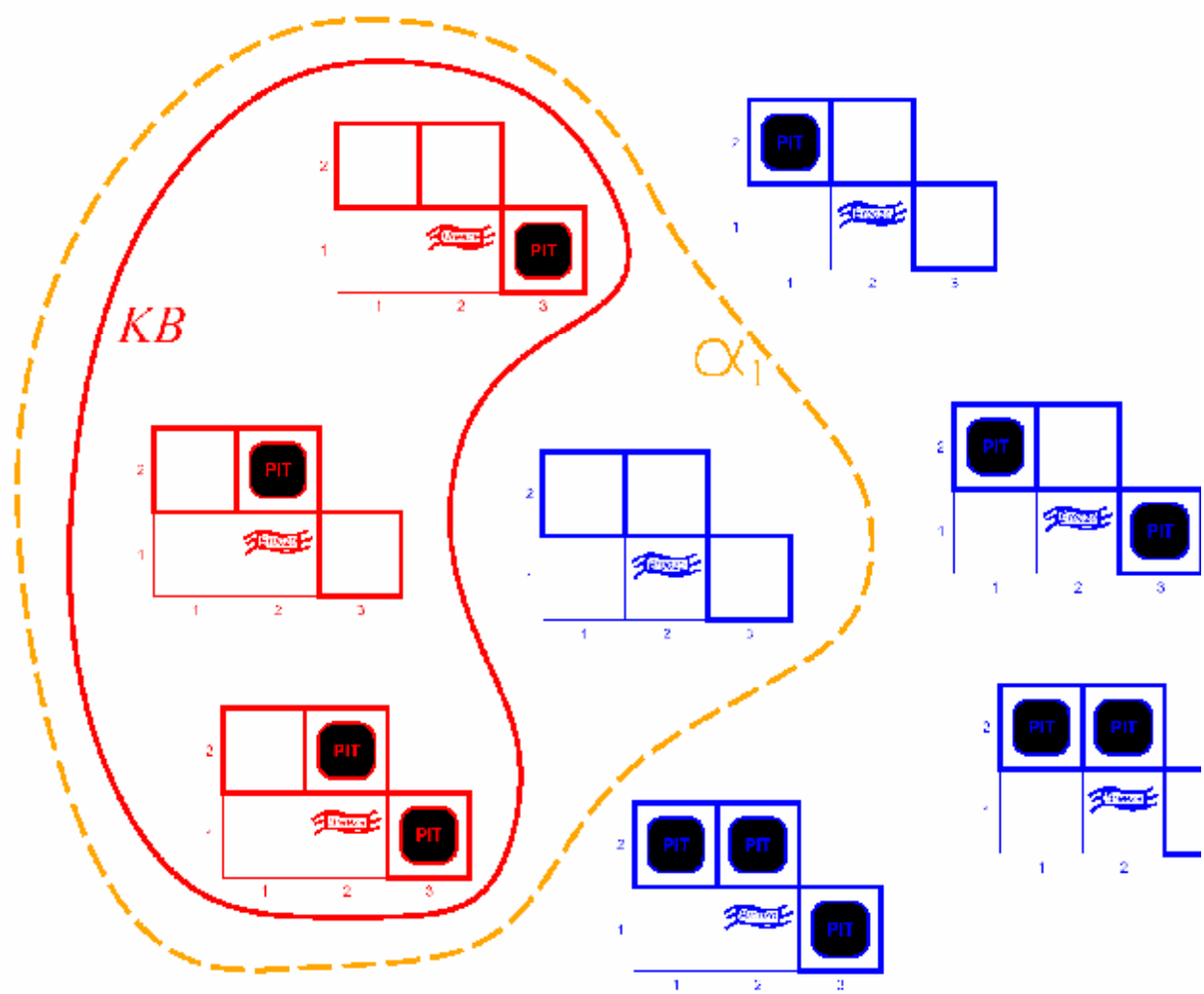
مدل های Wumpus





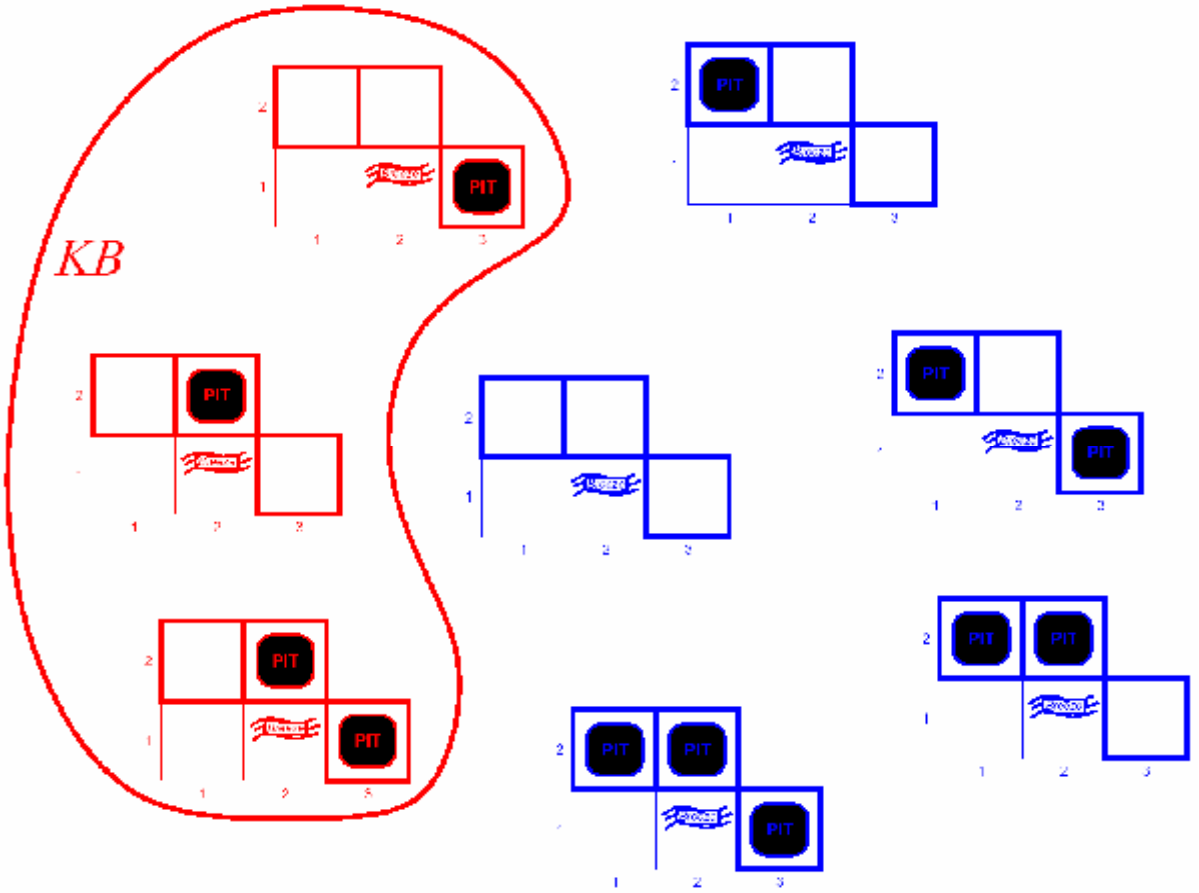


KB = قوانین دنیای wumpus + مشاهدات

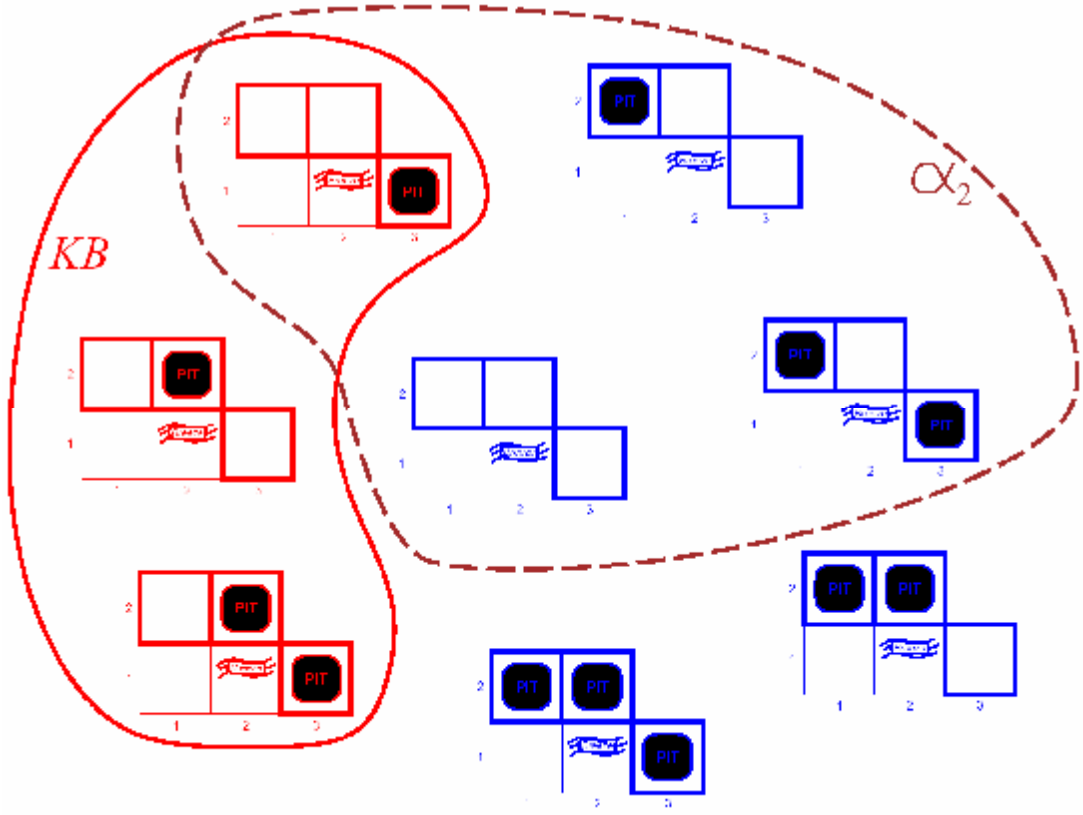


KB = قوانین دنیای wumpus + مشاهدات

α_1 = [۲و۱] ایمن است ، $KB \models \alpha_1$



KB = قوانین دنیای wumpus + مشاهدات



KB = قوانین دنیای wumpus + مشاهدات
α_2 = در [۲و۲] ایمن است و $KB \not\models \alpha_2$

استنتاج - $KB \mid_i \alpha$ = جمله ی α می تواند از KB توسط روال [۱] i مشتق شود . نتایج [۲] KB یک کومه [۳] می باشد ؛ α مانند یک سوزن تزریق کننده [۴] عمل می کند.

استلزام [۵] = سوزن تزریق کننده در کومه ؛ استنتاج = پیداکردن آن

صحت [۶] : i در صورتی صحیح است که ، در هر جایی که $KB \mid_i \alpha$ ، این درست باشد که $KB \vdash \alpha$. تمامیت [۷] : i در صورتی کامل است که ، در هر جایی که $KB \vdash \alpha$ ، عبارت $KB \mid_i \alpha$ برقرار باشد . پیش دید [۸] : ما می خواهیم یک منطقی را که به اندازه ی کافی رسا می باشد برای گفتن تقریبا هر چیزی که می خواهیم و برای این که بفهمیم کدام یک صحیح است و نتیجه ی کدام زیر برنامه کامل می باشد ، تعریف نماییم . پیش دید ، زیربرنامه ای است که ما می خواهیم به هر سوالی که کدام جواب توسط KB دنبال می شود ، جواب دهد.

منطق گزاره ای : ظاهر - منطق گزاره ای ، ساده ترین منطقی است که ایده های اساسی را توضیح می دهد . نمادهای گزاره ای P1,P2 و ..... می باشند . اگر S یک جمله باشد : $\neg S$ ،

---

[1] procedure
[2] consequences
[3] haystack
[4] needle
[5] Entailment
[6] soundness
[7] Completeness
[8] preview

عبارت نقیض [1] آن می باشد . عبارات S_1 و S_2 را در نظر بگیرید ، در این صورت $S_1 \wedge S_2$ ، AND (ترکیب فصلی [2] یا جدایی ؛ در صورتی که ورودی (ها) صحیح باشند ، نتیجه درست خواهد بود) این دو خواهد بود . عبارات S_1 و S_2 را در نظر بگیرید ، در این صورت $S_1 \vee S_2$ ، OR (ترکیب عطفی [3] ؛ در صورتی که از S_1 و S_2 یکی درست باشد ، نتیجه درست خواهد بود) این دو خواهد بود . عبارات S_1 و S_2 را در نظر بگیرید ، در این صورت $S_1 \Rightarrow S_2$ نتیجه گیری (استنباط [4]) خواهد بود . عبارات S_1 و S_2 را در نظر بگیرید ، در این صورت $S_1 \Leftrightarrow S_2$ استنباط دو شرطی (اگر و فقط اگر [5] ، S_1 اگر و فقط اگر S_2) خواهد بود .

منطق گزاره ای : سمانتیک ها یا معناها - هر مدل درست [6] / غلط [7] را برای هر نشان [8] مشخص می نماید .

$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
↓	↓	↓
درست	درست	غلط

(با این هشت نشان ، می توانیم هشت مدل داشته باشیم .) قوانینی برای ارزیابی درستی به وسیله ی یک مدل m :

$\neg S$ در صورتی که S غلط باشد ، صحیح است | $S_1 \wedge S_2$ در صورتی که S_1 **و** S_2 درست باشند ، درست است | $S_1 \vee S_2$ در صورتی که S_1 **یا** S_2 درست باشند ، درست است | $S_1 \Rightarrow S_2$ در صورتی که S_1 غلط باشد **یا** S_2 درست باشد ، درست است . توجه نمایید که در صورتی که S_1 درست باشد **و** S_2 غلط باشد ، غلط است . | $S_1 \Leftrightarrow S_2$ در صورتی که $S_1 \Rightarrow S_2$ درست باشد **و** $S_2 \Rightarrow S_1$ هم درست باشد ، درست است . پردازش بازگشتی ساده ، یک عبارت اختیاری ساده را ارزیابی می نماید ، به عنوان مثال ،

$$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$$

جدول صحت برای رابطه ها

P	**Q**	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
غلط	درست	درست	غلط	غلط	درست	درست
غلط	غلط	درست	غلط	درست	درست	غلط
درست	درست	غلط	غلط	درست	درست	غلط
درست	غلط	غلط	درست	درست	غلط	درست

جملات دنیای wumpus

" چاله ها باعث ایجاد هوای خوش در خانه های اطراف می شوند "

$P_{i,j}$ در صورتی که یک چاله در [i,j] وجود دارد ، درست می باشد .

$B_{i,j}$ در صورتی که یک هوای خوش (breeze) در [i,j] وجود دارد ، درست می باشد .

---

[1] negation
[2] conjunction
[3] disjunction
[4] implication
[5] biconditional
[6] true
[7] false
[8] symbol

$\neg P_{1,1}$

$\neg B_{1,1}$

$B_{2,1}$

" چاله ها باعث ایجاد هوای خوش در خانه های اطراف می شوند "

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

" یک خانه خوش هوا است *اگر و فقط اگر* یک چاله در اطرافش وجود داشته باشد "

جداول صحت برای استنتاج

KB	R_5	R_4	R_3	R_2	R_1	$P_{3,1}$	$P_{2,2}$	$P_{2,1}$	$P_{1,2}$	$P_{1,1}$	$B_{2,1}$	$B_{1,1}$
غلط	غلط	درست	درست	درست	درست	غلط	غلط	غلط	غلط	غلط	غلط	غلط
غلط	غلط	درست	غلط	درست	درست	درست	غلط	غلط	غلط	غلط	غلط	غلط
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
غلط	درست	درست	غلط	درست	درست	غلط	غلط	غلط	غلط	غلط	درست	غلط
درست	درست	درست	درست	درست	درست	درست	غلط	غلط	غلط	غلط	درست	غلط
درست	درست	درست	درست	درست	درست	غلط	درست	غلط	غلط	غلط	درست	غلط
درست	درست	درست	درست	درست	درست	درست	درست	غلط	غلط	غلط	درست	غلط
غلط	درست	درست	غلط	غلط	درست	غلط	غلط	درست	غلط	غلط	درست	غلط
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
غلط	درست	غلط	درست	درست	غلط	درست	درست	درست	درست	درست	درست	درست

در شمارش سطرها (با انتساب های مختلف به سطرها) ، در صورتی که KB در سطر درست است ، از α هم صرف نظر نمایید .

استنتاج با شمارش

شمارش Depth-first همه ی مدل ها صحیح و کامل است .

تابع TT-Entails?(KB,α) درست یا غلط را برمی گرداند

ورودی ها : KB ، که یک پایگاه دانش می باشد و یک جمله در منطق گزاره ای می باشد و α که یک صف می باشد و به صورت یک جمله ی منطق گزاره ای می باشد

یک لیست از سمبل های گزاره ای در KB و α ← symbols

TT-Check-All(KB,α,symbols,[ ]) را برگردان

تابع TT-Check-All(KB,α,symbols,model) درست یا غلط را برمی گرداند

در صورتی که Empty?(symbols) صحیح می باشد ،

در صورتی که PL-True?(KB,model) دارای ارزش درست است ، PL-True?(α,model) را برگردان .

در غیر این صورت درست (true) را برگردان

در غیر این صورت

P ← First(symbols);rest ← Rest(symbols)

TT-Check-All(KB,α,rest,Expand(P,true,model)) و TT-Check-All(KB,α,rest,Expand(P,false,model)) را برگردان

$O(2^n)$ برای n سمبل می باشد ؛ مساله به صورت co-NP-complete می باشد .

تساوی های منطقی - دو عبارت در صورتی معادل منطقی می باشند که در مدل های یکسان برابر باشند :

$\alpha \equiv \beta$ اگر و فقط اگر $\alpha \models \beta$ و $\beta \models \alpha$ باشد

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ جابه جایی پذیری[1] $\wedge$

$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ جا به جایی پذیری $\vee$

$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ شرکت پذیری[2] $\wedge$

$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ شرکت پذیری $\vee$

$\neg(\neg\alpha) \equiv \alpha$ حذف نقیض دوگانه[3]

$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ مفهوم مخالف[4]

$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$ حذف استنتاج[5]

$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ حذف شرط دو طرفه[6]

$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ قانون دمورگان[7]

$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ قانون دمورگان

$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ توزیع پذیری[8] $\wedge$ روی $\vee$

$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ توزیع پذیری $\vee$ روی $\wedge$

صحت و قابلیت ارضا - یک عبارت در صورتی **درست** است که در **همه ی** مدل ها درست باشد و $True, A \vee \neg A, A \Rightarrow A, (A \wedge (A \Rightarrow B)) \Rightarrow B$. صحت مربوط می شود به نتیجه گیری از راه قضیه ی قیاس[9] : $KB \models \alpha$ اگر و فقط اگر $(KB \Rightarrow \alpha)$ مجاز باشد . یک عبارت راضی کننده است اگر در برخی از مدل ها درست باشد . $A \vee B, C$ یک جمله ناراضی کننده است اگر در هیچ مدلی درست نباشد . $A \wedge \neg A$

قابلیت ارضا وابسته به نتیجه ی به دست آمده از راه زیر است: $KB \models \alpha$ اگر و فقط اگر $(KB \wedge \neg\alpha)$ ناراضی کننده باشد . α را به وسیله ی برهان خلف[10] اثبات نمایید .

متدهای اثبات[11] - به طور کلی متدهای اثبات به دو دسته تقسیم می شوند:

- ✓ **استفاده از قوانین استنتاج**
  - صحت تولید عبارات جدید از قدیم

---

[1] commutativity
[2] associativity

[3] double-negation elimination
[4] contraposition
[5] implication
[6] biconditional elimination
[7] De Morgan
[8] distributivity

[9] Deduction Theorem

[10] reductio ad absurdum

[11] Proof methods

- اثبات = به کار بردن یک ترتیب از قوانین استنتاج
می توانیم از قوانین استنتاج به صورت عملگرهایی در یک الگوریتم جستجوی استاندارد استفاده نماییم
- معمولا به ترجمه ی عبارات به صورت یک فرم نرمال نیازمندیم

✓ **بررسی مدل** [1] شمارش جدول صحت (معمولا به صورت نمایی از n می باشد) پیمایش لیست معکوس را بهبود می بخشد .

زنجیره ی مستقیم و معکوس

فرم شیپوری [2] (محدودشده)

KB = پیوند شروط شیپوری

شرط شیپوری =

- سمبل قضیه ؛ یا
- اتصال سمبل ها ← سمبل

مثال : $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

روش برجستگی ها [3] برای فرم شیپوری : کامل برای KB های شیپوری

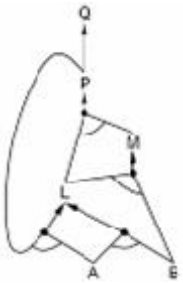
$$\frac{\alpha_1,\ldots,\alpha_n,\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

می تواند با زنجیره ی مستقیم یا معکوس استفاده شود. این الگوریتم ها خیلی طبیعی هستند و در زمان **خطی** اجرا می شوند .

زنجیره ی مستقیم

- ایده : تا زمانی که پرس و جو وجود دارد ، هر چیزی که در KB مجاز است را به KB اضافه کن .

$$P \Rightarrow Q, L \wedge M \Rightarrow P, B \wedge L \Rightarrow M, A \wedge P \Rightarrow L, A \wedge B \Rightarrow L, A, B$$



الگوریتم زنجیره ی مستقیم

تابع PL-FC-Entails?(KB,q) درست یا غلط را برمی گرداند

ورودی ها ، KB ، که پایگاه دانش ، یک مجموعه از شروط گزاره ای شیپوری

q ، صف ، که یک سمبل گزاره ای می باشد

متغیرهای محلی [4] : count ، یک جدول شاخص گذاری شده توسط شرط ، به صورت اولیه حاوی تعداد permis ها می باشد

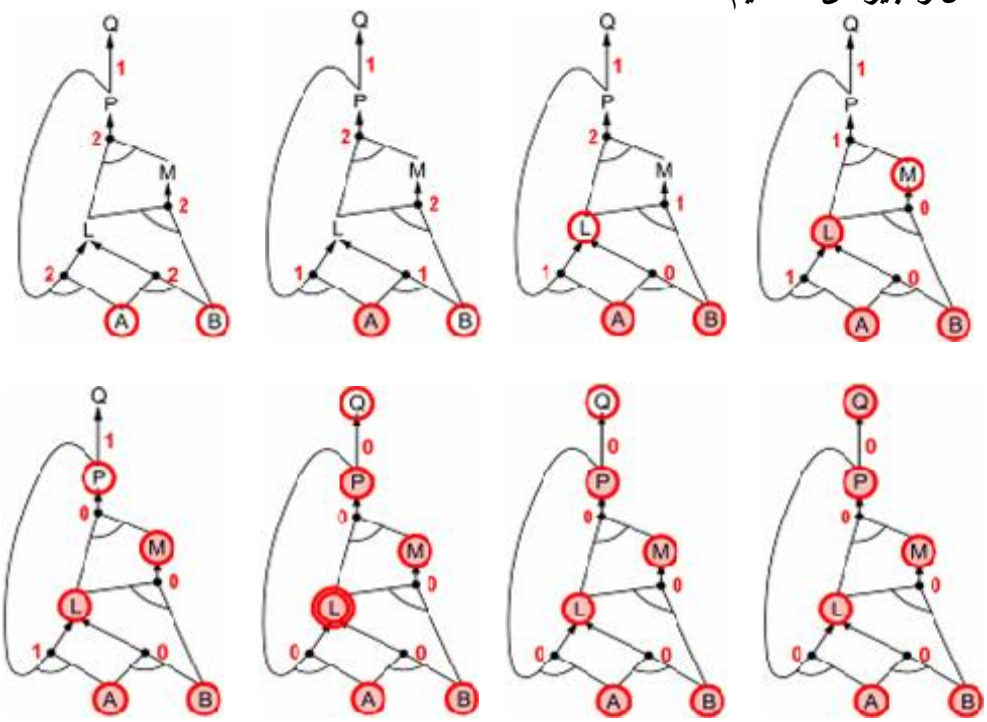
inferred یک جدول شاخص گذاری شده توسط سمبل ، هر ورودی دارای مقدار اولیه ی false می باشد

---

[1] Model checking
[2] Horn Form
[3] Modus Ponens
[4] local variables

agenda ، لیستی از سمبل ها ، به صورت اولیه سمبل ها در KB شناخته می شوند
مادامی که agenda خالی نمی باشد کارهای زیر را انجام بده
p← Pop(agenda)
وگرنه برای inferred[p] کارهای زیر را انجام بده
inferred[p] ← true
برای هر شرط شیپوری c که در p قضیه به دست می آید کارهای زیر را انجام بده
count[c] را کاهش بده
در صورتی که count[c]=0 می باشد کارهای زیر را انجام بده
در صورتی که Head[c]=q می باشد true را برگردان
Push(Head[c],agenda)
false را برگردان

مثال زنجیره ی مستقیم



اثبات تمامیت

FC یا زنجیره ی مستقیم ، هر عبارت اتمیک که توسط KB به وجود آمده است را مشتق می کند .

۱. FC در جایی که هیچ عبارت اتمیک جدیدی مشتق نمی شود به یک **نقطه ی ثابت** [1] می رسد
۲. توجه کنید که حالت نهایی به صورت یک مدل m ، منتسب کننده ی درست / غلط به سمبل ها می باشد
۳. هر شرط در KB اصلی ، در m درست است

اثبات : تصور نمایید که یک شرط $a_1 \wedge ... \wedge a_k \Rightarrow b$ در m غلط می باشد

[1] fixed point

بنابراین $a_1 \wedge ... \wedge a_k$ در m درست است و b در m غلط می باشد ، بنابراین الگوریتم به یک نقطه ی ثابت نمی رسد !

۴. از این رو m مدلی از KB می باشد

۵. در صورتی که KB|=q ، q در هر مدل KB ، شامل m درست می باشد .

ایده ی کلی : ساختن هر مدل KB با صحت نتیجه گیری و بررسی α می باشد .

زنجیره ی معکوس

ایده : برای ثابت کردن q توسط BC یا زنجیره ی معکوس .

بررسی کنید که آیا q قبلا شناخته شده است ، یا توسط BC ، همه ی مفروضات شامل برخی از قوانین منتج کننده ی q را ثابت کنید .

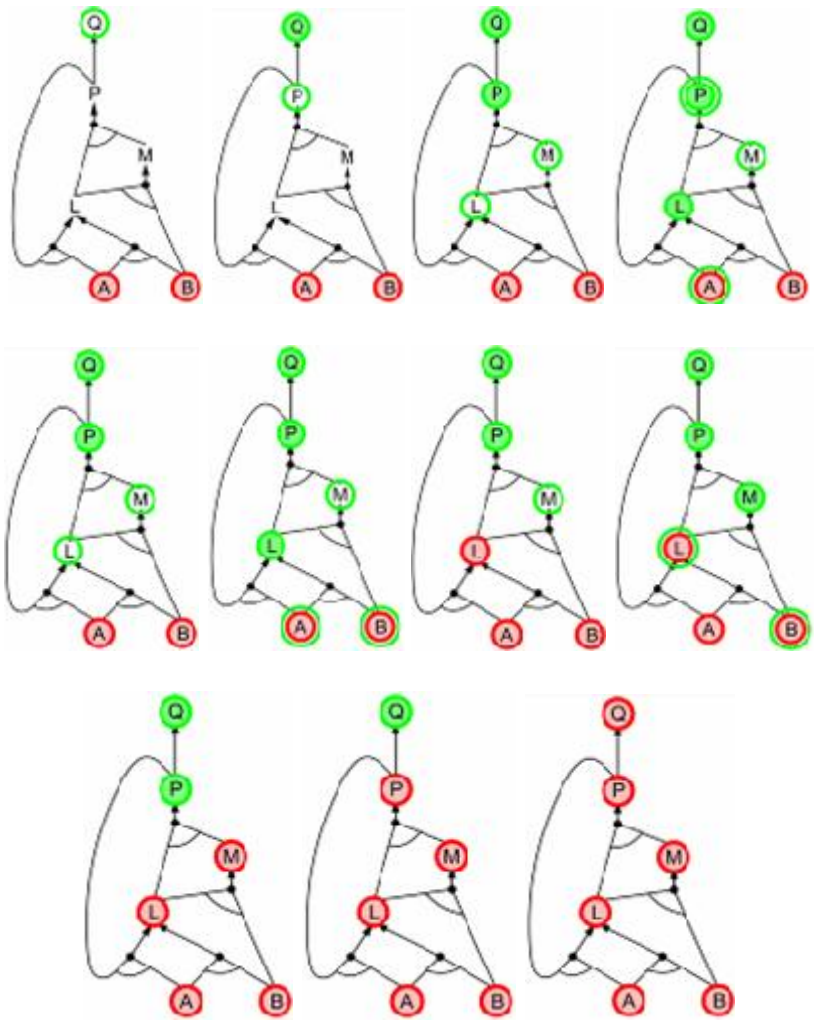
اجتناب از حلقه ها : بررسی کنید که آیا اهداف فرعی قبلا در پشته ی هدف وجود داشته اند .

اجتناب از کارهای تکراری : بررسی کنید که آیا اهداف فرعی جدید ،

۱) قبلا به صورت درست ثابت شده اند یا

۲) قبلا مردود شده اند

مثال زنجیره ی معکوس



زنجیره ی مستقیم و معکوس

FC مشتق شده از داده می باشد . به صورت خودکار و ناخود آگاه پردازش می شود . مثال : شناخت شی ء و روال های تصمیم گیری . ممکن است تعدادی کار را که به هدف نامربوطند انجام دهیم . BC مشتق شده از هدف می باشد و برای حل مساله مناسب می باشد . مثال : کلیدهای من کجا هستند ؟ چگونه می توانم اطلاعات یک برنامه ی PhD را بگیرم ؟ پیچیدگی BC می تواند به مراتب کم تر از پیچیدگی خطی در اندازه ی KB باشد .

تحلیل - صورت نرمال ربط دهنده [1]

ربط [2] بی ربط های واقعی [3]

شرایط

مثال : $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

قانون تحلیل نتیجه (برای CNF) : برای منطق گزاره ای کامل می باشد .

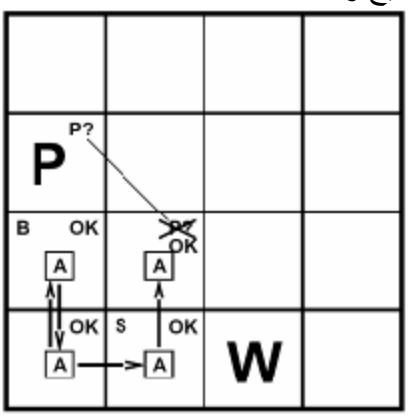
$$\frac{\lambda_1 \vee ... \vee \lambda_k , m_1 \vee ... \vee m_n}{\lambda_1 \vee ... \vee \lambda_{i-1} \vee \lambda_{i+1} \vee ... \vee \lambda_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n}$$

در جایی که λ_i و m_j متمم واقعی هستند .

مثال :

$$\frac{P_{1,3} \vee P_{2,2} , \neg P_{2,2}}{P_{1,3}}$$

تحلیل برای منطق گزاره ای صحیح و کامل است .



تبدیل به CNF - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

۱. حذف کردن $\Leftrightarrow$ ، جایگزین کردن $\alpha \Leftrightarrow \beta$ با $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

[1] CNF یا Conjunctive Normal Form
[2] conjunction
[3] disjunction of literals

۲. حذف کردن $\Leftarrow$ ، جایگزین کردن $\alpha \Leftarrow \beta$ با $\neg\alpha \vee \beta$

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

۳. $\neg$ را با استفاده از قوانین دمورگان و قرینه سازی دوگانه حرکت دهید:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

قوانین توزیع و گسترش را روی $\vee$ و $\wedge$ به کار ببرید:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

تحلیل الگوریتم ـ اثبات به روش برهان خلف ، نشان دهید $KB \wedge \neg\alpha$ نامناسب است .

تابع PL-Resolution(KB,α) درست یا غلط را برمی گرداند

ورودی ها : KB که پایگاه دانش است ، یک عبارت در منطق گزاره ای می باشد

α یک پرس و جو است ، یک عبارت در منطق گزاره ای می باشد .

مجموعه ای از شرایط CNF ارایه شده توسط $KB \wedge \neg\alpha \leftarrow$ clauses

new $\leftarrow \{\}$

در حلقه کارهای زیر را انجام بده

برای هر C_i, C_j در clauses کارهای زیر را انجام بده

$$resolvents \leftarrow PL - \text{Re}\, solve(C_i, C_j)$$

در صورتی که resolvents شامل شرط خالی است درست را برگردان

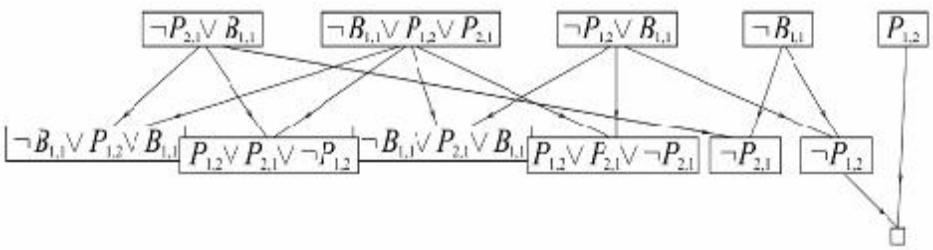
$$new \leftarrow new \cup resolvents$$

در صورتی که new زیر مجموعه یا مساوی clauses می باشد false را برگردان

$$clauses \leftarrow clauses \cup new$$

مثال تحلیل

$$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1} \quad \alpha = \neg P_{1,2}$$



خلاصه ـ عامل ها یا مولفه های منطقی ، نتیجه را برای یک پایگاه دانش ، برای به دست آوردن اطلاعات جدید و به وجود آوردن تصمیم گیری ها به کار می گیرند . اجزای اصلی منطق :

- ظاهر : ساختار رسمی عبارات
- سمانتیک ها : صحت عبارات مدل های wrt
- دنبال کردن : صحت لازم یک عبارت ارایه شده ی دیگر
- نتیجه یا استنتاج : عبارت های مشتق شده از عبارات دیگر
- صحت : مشتقات فقط عبارت های حبس شده را تولید می نمایند
- کمال یا تمامیت : مشتقات می توانند همه ی عبارت های دنبال شده را تولید نمایند

دنیای wumpus به توانایی برای ارایه کردن جزیی و اطلاعات منفی شده و توضیح داده شده توسط موارد و غیره احتیاج دارد . زنجیره ی مستقیم و معکوس زمانبندی خطی می باشند و

برای شرایط شیپوری ، کامل هستند . تحلیل ، برای منطق گزاره ای ، کامل می باشد . منطق گزاره ای ، قدرت بیان را کم می کند.

منابع :

http://www.uni.koblenz.de
http://aima.eecs.berkeley.edu/slides-pdf /chapter07.pdf

فصل دهم
منطق گزاره ای

ارایه

- تا حالا ، ما در مورد جستجو که یک ابزار توجه کننده به امکانات تناوبی است ، صحبت کرده ایم .
- بحث دیگر ، ارایه ی این احتمال ها می باشد .

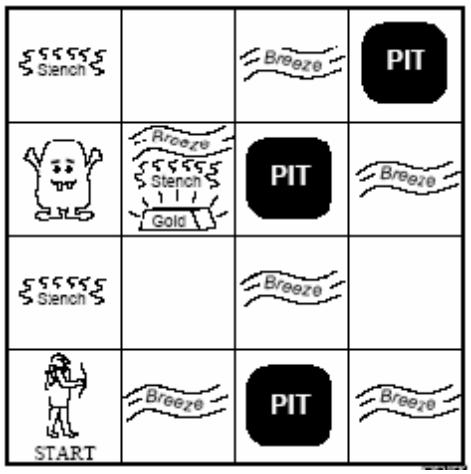
منطق

- منطق ، روشی کلاسیک برای ارایه ی دانش عامل می باشد .
- به ما اجازه می دهد که عامل ها را به صورت اعلانی (اظهاری) در موارد زیر برنامه ریزی نماییم و توصیف کنیم .
  - در موارد ارتباط های میان اشیاء .
  - دانش به صورت واقعیات و ارتباطات ، بیان می شود

- عامل ها از یک پایگاه دانش که به آن ها برای استدلال در مورد یک مساله اجازه می دهد ، استفاده می کنند .

- این کار در برخی مواقع برنامه نویسی در *سطح دانش* [1] نامیده می شود .
  - واقعیات مشخص شده توسط یک عامل و اهداف را مشخص می نماید .
  - عملکردها انتخاب می شوند ، زیرا آن ها به اهداف ، دست می یابند .
  - پیاده سازی ، خلاصه می شود .
- استوارت راسل [2] و پیترنورویگ [3] از دنیای Wumpus به صورت یک دامنه ی مثالی ، استفاده می کنند .
- محیط : شبکه ی ۴*۴ از خانه ها
  - طلا دریک خانه قرار دارد و wumpus در خانه ی دیگر
  - چاله ها هم در برخی از خانه ها وجود دارند
- عملکردها : حرکت مستقیم ، حرکت به چپ ، حرکت به راست ، شلیک پیکان (نیزه) ، چنگک (گیرنده ی) طلا .
- حسگرها : حس بوی بد [4] ، حس بوی خوب [5] ، حس کننده ی طلا ، حس دیوار ، شنیدن کشته شدن wumpus .
- هدف : بیشینه نمودن کارایی ، که به معنی پیداکردن طلا به سرعت و بدون روبه رو شدن با wumpus و افتادن در چاله می باشد .

دنیای Wumpus



منطق

- یک پایگاه دانش تشکیل شده از *عباراتی* که واقعیاتی در مورد جهان را مطرح می کنند .
- برخی از تعریف ها :

---

[1] knowledge level
[2] http://www.cs.berkeley.edu/~russell
[3] http://norvig.com
[4] perceive stench
[5] perceive breeze

- ظاهر (نحو یا گرامر) : مشخص می کند که آیا یک عبارت به صورت درست ساخته شده است .
  - در ریاضیات ، عبارت x+2=5 به صورت گرامری درست می باشد ، اما x+=3 به صورت گرامری درست نمی باشد .
- معنا (سمانتیک) : مشخص می نماید که یک عبارت در چه موقعی درست یا غلط می باشد . معنای x+2=5 در زمانی درست است که x=3 باشد و در غیر این صورت غلط می باشد . عبارت های منطقی ، باید درست یا غلط باشند ؛ "درجه ی درستی ، وجود ندارد" .

- مدل : یک مدل ، یک انتساب مقادیر به هر یک از متغیرهای وضعیت ما می باشد .
  - یک مدل برای دنیای روباه و جوجه ها ، باید مقادیر nfoxesLeft ، nfoxesRight ، nchickensLeft ، nchickensRight و boatPos را مشخص نماید .
  - ما اغلب می خواهیم مدلی را پیدا نماییم که عبارتی درست یا غلط را بسازد .
- دنبال کردن [1] : ایده ای است که یک عبارت به صورت منطقی از عبارت دیگر دنباله روی می کند یا نه .
  - به صورت a|=b نوشته می شود
  - به صورت تکنیکی (مطابق اصول فنی) ، می گوید که : برای همه ی مدل هایی که a درست است ، b هم درست می باشد .
  - a+2=5|=a=3
- دنبال کردن به ما اجازه خواهد داد که استدلال نماییم و واقعیات جدیدی را به پایگاه دانش عامل مان اضافه نماییم .

استدلال

- یک پایگاه دانش ، مدلی را که به ما اجازه می دهد استدلال را انجام دهیم ، اضافه می کند .
  - برای یک مجموعه از عبارت های داده شده ، برخی از انتساب مقادیر را به متغیرها اضافه می کنیم ، چه می توانیم نتیجه بگیریم ؟
- دنبال کردن به ما می گوید که یک عبارت ، می تواند مشتق شود .
- الگوریتمی که فقط عبارت های دنبال شده را مشتق می نماید ، درست یا صحیح [2] نام دارد .
  - اشتباه نمی کند یا عبارت های نادرست را نتیجه گیری نمی کند .
- الگوریتمی که می تواند همه ی عبارت های دنبال شده را مشتق نماید ، کامل [3] نام دارد .
  - در صورتی که یک عبارت ، دنبال شود ، یک الگوریتم کامل ، آن را سرانجام استنتاج خواهد کرد .

منطق گزاره ای

- منطق گزاره ای ، منطقی خیلی ساده می باشد .
  - برای مثال ها خوب می باشد
  - به صورت محاسباتی (کامپیوتری) قابل پیاده سازی می باشد .

---

[1] entailment
[2] sound
[3] complete

- در توانایی ارایه ، محدود شده است .

- واژگان (راسل و نورویگ ، این ها را عبارت های اتمیک می گویند) از یک سمبل منفرد تشکیل شده اند .
  - BoatOnRight ، 3FoxesLeft
- یک عبارت پیچیده ، مجموعه ای از واژه های به هم پیوسته توسط $\Leftrightarrow,\Rightarrow,\neg,\wedge,\vee$ می باشد .
  - $3FoxesLeft \wedge 3ChickensRight \wedge BoatRight$
  - $3\ foxesLeft \wedge 2ChickensLeft \wedge BoatRight \Rightarrow ChickensEaten$
- $A \wedge B$ - AND – عبارت در صورتی درست می باشد که هم A و هم B هر دو درست باشند .
- $A \vee B$ - OR – عبارت در صورتی درست می باشد که یا A یا B و یا هر دو درست باشند .
- $\neg A$ (NOT) – عبارت در صورتی درست است که A غلط باشد .
- $A \Rightarrow B$ (دلالت یا استلزام [1]) عبارت در صورتی درست می باشد که A غلط باشد یا B درست باشد .
- $A \Leftrightarrow B$ (هم ارزی [2]) – عبارت در صورتی درست می باشد که A و B دارای ارزش درستی یکسان باشند .

منطق گزاره ای – استلزام یا استنباط

- استلزام ، یک طرح منطقی مفید می باشد .
- عبارت $A \Rightarrow B$ در صورتی درست می باشد که :
  - A درست باشد و B درست باشد .
  - A غلط باشد .
- مثال : اگر الان هوا بارانی باشد ، آن گاه الان هوا ابری می باشد .
- $A \Rightarrow B$ معادل است با $\neg A \vee B$
- استلزام به ما امکان استنتاج را می دهد .

منطق گزاره ای – تعریف های بیش تر

- هم ارزی منطقی : دو عبارت به صورتی منطقی هم ارز هستند اگر آن ها برای مجموعه مدل های یکسان ، درست باشند .
  - $P \wedge Q$ به صورت منطقی هم ارز با $\neg(\neg P \vee \neg Q)$ می باشد
- صحت [3] (همانگویی [4]) : عبارتی دارای همانگویی می باشد که برای همه ی مدل ها درست باشد .
  - $A \vee \neg A$
- تناقض [5] : عبارتی که در همه ی مدل ها غلط (نادرست) می باشد .

---

[1] implies
[2] equivalence
[3] validity
[4] tautology
[5] contradiction

- $A \wedge \neg A$
- قابلیت ارضا [1] : عبارتی دارای این خصوصیت می باشد که برای برخی از مدل ها درست باشد .
  - $3\ foxesleft \vee 2\ foxesleft$ در برخی از جهان ها درست می باشد .
  - اغلب مساله ی ما به دنبال پیداکردن مدلی است که یک عبارت را درست (غلط) نماید :
  - این راه حل مساله ی ما خواهد بود .

استدلال منطقی

- استدلال منطقی با استفاده از عبارات موجود در پایگاه دانش عامل ، برای استنتاج عبارت های جدید ، جلو می رود .
- قوانین استنتاج
  - روش اثبات [2]
    - $A, A \Rightarrow B$ ، B را نتیجه می دهد
  - حذف AND
    - $A \wedge B$ ، A را نتیجه می دهد .
  - معرفی OR
    - A ، $A \vee B$ را نتیجه می دهد
  - مفهوم مخالف [3] : $A \Rightarrow B$ می تواند به صورت $\neg B \Rightarrow \neg A$ بازنویسی شود
  - منفی کردن دوباره : $\neg(\neg A) = A$
  - توزیع :
    - $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
    - $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
  - قضیه ی دمورگان :
    - $A \vee B$ به صورت $\neg(\neg A \wedge \neg B)$ بازنویسی می شود
    - یا $A \wedge B \Leftrightarrow \neg(\neg A \vee \neg B)$

استنتاج به صورت جستجو

- سپس ما می توانیم از جستجوی قبلی خوب اول سطح برای انجام استنتاج و تشخیص این که آیا یک عبارت توسط یک پایگاه دانش دنبال می شود استفاده نماییم .
- ایده ی اساسی : با عبارت های موجود در پایگاه دانش مان شروع نمایید .
- عملکردها به کار بردن استنتاج می باشند .
  - برای مثال ، تصور نمایید که ما $A \rightarrow B$ (1 ، $B \Rightarrow C$ (2 و A(3 را داریم .
  - یک عمل ممکن این است که ما روش اثبات را برای ۱ و ۳ برای استنتاج B به کار ببریم .
  - سپس ما می توانیم روش های اثبات را دوباره برای استنتاج C به کار ببریم .

---

[1] satisfiability
[2] modus ponens
[3] contraposition

- جستجوی ما می تواند به روش اول سطح (که همه ی استنتاج های ممکن از پایگاه دانش اصلی می باشد) و اول عمق و یا در مواقعی با استفاده از هر دو روش ادامه یابد .
- نتیجه ی این جستجو ، دلیل یا گواه یا اثبات نام دارد .

تحلیل

- قوانین قبلی ، درست هستند ولی الزاما کامل نمی باشند .
- خوشبختانه ، قانون کاملی برای استنتاج وجود دارد
- این قانون ، تحلیل نام دارد .
- قانون تحلیل پایه شبیه زیر می باشد :
  - $A \vee B$ و $\neg A \vee C$ به ما اجازه می دهند که $B \vee C$ را استنتاج نماییم .
  - A یا درست است و یا درست نمی باشد . اگر A درست باشد ، آن گاه C هم باید درست باشد .
  - اگر A غلط باشد ، آن گاه B باید درست باشد .
- این مطلب می تواند به مواردی با هر طولی تعمیم داده شود .

فرم نرمال ربط دهنده

- تحلیل به صورت مجزا کار می کند
- این به این معنی است که پایگاه دانش ما باید به این فرم (شکل) باشد .
- فرم نرمال ربط دهنده ، اجتماعی از شرط هایی که مجزا هستند می باشد .
- $(A \vee B \vee C) \wedge (D \vee E \vee F) \wedge (G \vee H \vee I) \wedge \ldots$
- هر عبارت منطق گزاره ای می تواند به فرم نرمال ربط دهنده تبدیل شود .

دستورالعمل فرم نرمال ربط دهنده

۱. هم ارزی را حذف نمایید
   - $A \Leftrightarrow B$ می شود $A \Rightarrow B \wedge B \Rightarrow A$
۲. استلزام را حذف نمایید
   - $A \Rightarrow B$ می شود $\neg A \vee B$
۳. $\neg$ را به درون ، حرکت دهید ، با استفاده از نقیض دوگانه و قوانین دمورگان
   - $\neg(\neg A)$ می شود A
   - $\neg(A \wedge B)$ می شود $(\neg A \vee \neg B)$
۴. عبارت های تودرتو را توزیع نمایید .
   - $(A \vee (B \wedge C))$ می شود $(A \vee B) \wedge (A \vee C)$

اثبات با ردکردن [1]

- در صورتی که پایگاه دانش شما به صورت فرم نرمال ربط دهنده می باشد ، می توانید تحلیل را با رد کردن انجام دهید .
- ایده ی اساسی : ما می خواهیم نشان دهیم که A درست می باشد .

[1] refutation

- $\neg A$ را به پایگاه دانش وارد نمایید و تلاش کنید که به یک تناقض برسید .

عبارات شیپوری [1]

- اثبات قضیه ی تحلیل استاندارد به صورت تشریحی ، مشکل می باشد .
- اما ، اگر ما خودمان را به یک قطعه محدود نماییم ، کارمان ساده خواهد شد .
- یک *عبارت شیپوری* یک فصل است که در آن حداکثر یک لفظ مثبت وجود دارد .
  - $\neg A \vee \neg B \vee \neg C \vee D$
  - $\neg A \vee \neg B$
- این ها می توانند به صورت استنتاج هایی با یک تالی (نتیجه) نوشته شوند .
  - $A \wedge B \wedge C \Rightarrow D$
  - $A \wedge B \Rightarrow False$
- عبارت های شیپوری ، مبنا (پایه) ی برنامه نویسی منطقی می باشند .

زنجیره ی مستقیم [2]

- ایده ی کلی : از پایگاه دانش شروع نمایید و دایما روش اثبات را به کار ببرید تا تمام واقعیات ممکن را به دست آورید .
- این در برخی از موارد استنتاج مشتق شده از داده ها [3] نام دارد .
- با دامنه ی دانش شروع کنید و ببینید دانش به شما چه می گوید
- این برای به دست آوردن واقعیات یا قوانین جدید بسیار مفید می باشد
- برای اثبات درستی یا غلط بودن یک عبارت معین ، کم تر مفید می باشد
  - جستجو به سمت یک هدف نمی رود
- Jess و CLIPS از زنجیره ی مستقیم استفاده می کنند

زنجیره ی معکوس [4]

- زنجیره ی معکوس از هدف شروع می کند و " به طرف عقب" و به سمت شروع حرکت می کند .
- مثال : اگر ما بخواهیم که نشان دهیم که A ، دنبال می شود ، باید عبارتی را پیدا نماییم که تالی آن A می باشد .
- سپس تلاش کنید که عبارت های پیشین را اثبات نمایید .
- این برخی اوقات استنتاج مشتق شده از پرس و جو [5] نامیده می شود
- برای اثبات یک پرس و جوی مشخص ، مفیدتر می باشد ، زیرا که بر یک هدف ، تمرکز می کند .
- احتمال به وجود آوردن اطلاعات جدید و ناشناخته ، کم تر می باشد .
- تحلیل ابزار – اهداف [1] ، یک نوع استدلال شبیه می باشد .

---

[1] horn clauses
[2] Forward Chaining
[3] data-driven reasoning
[4] Backward Chaining
[5] query-driven reasoning

- پرولوگ از زنجیره ی معکوس استفاده می کند .

مزیت های منطق گزاره ای

- اعلانی یا اظهاری بودن – دانش می تواند از استنتاج ، مجزا باشد
- می تواند اطلاعات جزیی را به کار بگیرد .
- می تواند عبارت های پیچیده تر را به صورت ساده تر تولید نماید .
- مکانیزم های صحیح و کامل دارد (برای عبارت های شیپوری ، کارامد می باشد)

معایب منطق گزاره ای

- افزایش تشریحی در تعداد لفظ ها
- راهی برای تشریح ارتباط های میان اشیا وجود ندارد
- راهی برای بیان کیفیت ، در مورد اشیا وجود ندارد .
- منطق مرتبه ی اول ، روشی برای رسیدگی کردن به این مسایل می باشد .

منابع :

http://www.cs.usfca.edu

[1] means-ends analysis

فصل یازدهم

منطق مرتبه ی اول به بیان ساده

منطق مرتبه ی اول [1] به بیان ساده
فصل یازدهم

- منطق مرتبه ی اول ، تمام خصوصیات منطق گزاره ای را داراست
- منطق مرتبه ی اول براساس این ایده است که جهان مرکب از دو نوع موجودیت [2] می باشد : اشیا [3] و ارتباطات [4]
- اشیا ، معمولا به نام ها و چیزها اشاره می نمایند ، به عنوان مثال ، جورج بوش [5] و خورشید
- ارتباطات معمولا به خصوصیات [6] و ویژگی ها [7] اشاره می کنند ، به عنوان مثال ، جورج بوش در حیات است (زنده است) . خورشید ، داغ است . لیلی پاتر [8] ، مادر هری پاتر [9] است .
- تفسیر زبان طبیعی
  - " هر ... " یا " همه ... " ، برای مثال :
    - " هر شخصی دارای یک والد است "
    - " همه ی پرندگان پرواز می نمایند "
  - این مطالب اظهار می کنند که همه ی اشیا دارای یک ویژگی معین هستند
- همچنین ، تفسیر زبان طبیعی
  - " برخی ... "
  - برای مثال :
    - " برخی از پرندگان نمی توانند پرواز نمایند "
    - " برخی از افراد از شطرنج لذت می برند "

---

[1] First-Order Logic(FOL)
[2] entity
[3] objects
[4] relations
[5] George Bush
[6] properties
[7] attributes
[8] Lily Potter
[9] Harry Potter

- این مطالب اظهار می کنند که لااقل ، یک شی دارای یک خصوصیت معین می باشد

واژگان

- یک زبان مرتبه ی اول شامل :
۱. یک مجموعه از ثابت ها [1] ؛ که با حروف کوچک مشخص می شوند می باشد ، نظیر a ، b ، c و ...
۲. به طور حسّی ، یک ثابت نام یک شی می باشد
۳. یک مجموعه از متغیرها ، معمولا با حروف کوچک مشخص می شوند ، نظیر x ، y ، z و ...
۴. یک مجموعه از سمبل های تابعی ، معمولا با حروف کوچک نمایش داده می شوند ، نظیر f ، g ، h و ...
۵. هر سمبل تابعی با یک ارزش صحیح و مثبت پیوند داده می شود (به عبارت دیگر ، هر سمبل تابعی دارای یک مقدار صحیح و مثبت می باشد)
- مجموعه ای از واژگان با قوانین [2] زیر تعریف می شود :
۱. یک ثابت ، یک واژه می باشد
۲. یک متغیر ، یک واژه می باشد
۳. یک عبارت [3] $f(t_1,...,t_n)$ در صورتی یک واژه است که یک سمبل تابعی و هر t_i واژه باشند
۴. هیچ چیز دیگری واژه نمی باشد

گزاره ها [4]

- یک زبان مرتبه ی اول همچنین شامل
  - یک مجموعه از گزاره ها که معمولا با حروف بزرگ مشخص می شوند است ، نظیر P ، Q ، R
  - گزاره ها خصوصیات اشیا را نمایش می دهند

رابط ها [5] و کمیت سنج ها (سورها) [6]

- رابط های " استاندارد " منطق گزاره ای به ارث برده می شوند ، نظیر $\rightarrow, \neg, \wedge, \vee$ و ...
- دو کمیت سنج :
  - $\forall$ ، بخوانید " برای همه " ؛ که سور عمومی [7] نامیده می شود
  - $\exists$ ، بخوانید " وجود دارد " ؛ که سور وجودی [8] نام دارد
- به طور معمول ، سورها با متغیر ها و متغیرها با سورها نظیر می شوند

یک فرمول مرتبه ی اول

---

[1] constants
[2] rules
[3] expression
[4] predicates
[5] Connectives
[6] Quantifiers
[7] universal quantifier
[8] existential quantifier

- در صورتی که P یک سمبل گزاره ای دارای ارزش n و $t_1,...,t_n$ واژگان باشند ، آن گاه $P(t_1,...,t_n)$ یک فرمول منطق مرتبه ی اول می باشد
- در صورتی که φ و ψ فرمول باشند ، آن گاه $\varphi \vee \psi$ ، $\varphi \wedge \psi$ ، $\neg\varphi$ و $\varphi \to \psi$ هم فرمول هستند
- در صورتی که φ یک فرمول باشد و x یک متغیر باشد ، آن گاه $\forall x(\varphi)$ و $\exists x(\varphi)$ هم فرمول هستند
  - فرمول دیگری وجود ندارد

متغیرهای آزاد و محدود [1]

- ساختار فرمول مرتبه ی اول ، کامل نمی باشد زیرا فرمول هایی نظیر عبارت زیر را مجاز می داند
  - $\forall x : P(a)$
  - P(x) x یک متغیر آزاد نامیده می شود

معنی یک فرمول منطق مرتبه ی اول

- معنی $\neg, \vee, \wedge$ و $\to$ از منطق گزاره ای به ارث برده می شود ، به عنوان مثال در صورتی که φ و ψ دو فرمول باشند آن گاه $\varphi \vee \psi$ در صورتی درست می باشد که لااقل یکی از φ یا ψ درست باشند
- در زمانی که سور ها وجود دارند ، صحّت یک فرمول باتوجه به دامنه ی سخن تشخیص داده می شود
- یک دامنه که معمولا با حرف D نمایش داده می شود فقط یک مجموعه می باشد ، یک مجموعه که می تواند مثل موارد زیر باشد
  - اعداد طبیعی {...،۳،۲،۱}
  - مجموعه ای از افراد
- به فرمول $\forall x : \varphi$ و یک دامنه ی D توجه نمایید
- فرمول $\forall x : \varphi$ در دامنه ی D درست می باشد اگر و فقط اگر $\varphi(x \mid d)$ برای هر $d \in D$ درست باشد
  - $\varphi(x \mid d)$ فرمولی گرفته شده از φ بعد از جایگزین نمودن هر وقوع x در φ با d می باشد
- به فرمول $\forall x : has\_chair(x)$ و یک دامنه ی D تشکیل شده از دانش آموزانی که در حال حاضر در OMB 31 هستند توجه نمایید
- فرمول $\forall x : has\_chair(x)$ در دامنه ی D درست است اگر و فقط اگر
  - $has\_chair(Sanjeev)$ درست باشد
  - $has\_chair(John)$ درست باشد
  - ..... برای هر دانشجوی درون OMB 31
- به فرمول $\exists x : \varphi$ و دامنه ی D توجه نمایید
- فرمول $\exists x : \varphi$ در دامنه ی D درست است اگر وفقط اگر $\varphi(x \mid d)$ برای برخی از $d \in D$ درست باشد

[1] Free and Bound variables

- $\varphi(x \mid d)$ فرمولی گرفته شده از φ بعد از جایگزین نمودن هر وقوع x در φ با d می باشد

- به فرمول $\exists x : s\tan ding(x)$ و یک دامنه ی D مرکب از افرادی که در حال حاضر در OMB 31 هستند توجه نمایید
- فرمول $\exists x : s\tan ding(x)$ در دامنه ی D درست است اگر و فقط اگر لااقل یک شخص در OMB 31 ایستاده باشد و بنابراین standing(Lecturer) درست و داریم : $\exists x : s\tan ding(x)$ درست می باشد
- یک فرمول ممکن است در برخی از دامنه ها درست باشد اما در برخی دیگر نادرست باشد
  - به فرمول $\forall x \exists y : (x = y^2)$ توجه نمایید
  - در دامنه ی اعداد حقیقی مثبت این عبارت درست می باشد
  - در دامنه ی همه ی اعداد حقیقی این عبارت غلط می باشد ؛ اگر x منفی باشد ، x نمی تواند با هر y^2 ای برابر باشد ، از آن جایی که $y^2 \geq 0$ می باشد
- یک فرمول که در همه ی دامنه ها درست است یك همانگویی [1] (همیشه درست) می باشد
- یک فرمول که در همه ی دامنه ها غلط است یک خلاف گویی [2] (تناقض)

ارایه ی نسبت خانوادگی

- منطق مرتبه ی اول یک زبان ارایه ی قدرتمند برای دانش می باشد
- به ارایه ی نسبت خانوادگی به زبان انگلیسی توجه نمایید
- گزاره ها ممکن است شامل موارد زیر باشند :
  - Parent(_ , _)
  - Child(_ , _)
  - Grandparent(_ , _)
  - Sibling(_ , _)
  - Male(_)
  - Female(_)
  - - = (_ , _)
- راهی کلی (یکسره) برای تبدیل انگلیسی به منطق مرتبه ی اول وجود ندارد
- نظریه های [3] غیردقیق [4] سادگی [5] و اختصار [6] وجود دارد
  - " James پدر Harry است "
    - $Parent(James, Harry) \wedge Male(James)$
  - " Lily مادر Harry است "
    - $Parent(Lily, Harry) \wedge Female(Lily)$

ارتباطات خانوادگی

---

[1] tautology
[2] contradiction
[3] notions
[4] imprecise
[5] simplicity
[6] conciseness

- " مردان [1] و زنان [2] کاملا متمایز هستند
  - $\forall x : (Male(x) \leftrightarrow \neg Female(x))$
- $p \leftrightarrow q$ خوانده می شود " p معادل با q است و کوتاه نویسی برای $((p \rightarrow q) \wedge (q \rightarrow p))$ می باشد
- برادر و خواهر [3] افرادی دقیقا متمایز هستند که دارای یک والد مشترک می باشند
  - $\forall x \forall y(Sibling(x, y) \leftrightarrow (\neg = (x, y) \wedge \exists p (Parent(p, x) \wedge Parent(p, y)))$

منطق مرتبه ی اول و منطق گزاره ای

- آیا $\forall$ لازم است ؟ بله
- به فرمول $\forall x : P(x)$ توجه نمایید
  - برای یک دامنه ی محدود $D = \{a_1, ..., a_n\}$ این عبارت دقیقا در زمانی درست است که $P(a_i)$ برای $1 \le i \le n$ درست باشد
    - $\forall x : P(x) \equiv P(a_1) \wedge ... \wedge P(a_n)$
  - برای یک دامنه ی نامحدود $D = \{a_1, ..., a_n, ...\}$ این عبارت دقیقا در زمانی درست است که $P(a_i)$ برای هر i در عبارت زیر درست باشد
    - $\forall x : P(x) \equiv P(a_1) \wedge ... \wedge P(a_n) \wedge ...$
  - اما یک فرمول نمی تواند دارای یک طول بی نهایت باشد
- آیا $\exists$ لازم است ؟ بله
- به فرمول $\exists x : P(x)$ توجه نمایید
  - برای یک دامنه ی محدود $D = \{a_1, ..., a_n\}$ این عبارت دقیقا در زمانی درست است که $P(a_i)$ برای $1 \le i \le n$ درست باشد
    - $\exists x : P(x) \equiv P(a_1) \vee ... \vee P(a_n)$
  - برای یک دامنه ی نامحدود $D = \{a_1, ..., a_n, ...\}$ این عبارت دقیقا در زمانی درست است که $P(a_i)$ برای هر i در عبارت زیر درست باشد
    - $\exists x : P(x) \equiv P(a_1) \vee ... \vee P(a_n) \vee ...$
  - اما یک فرمول نمی تواند دارای یک طول بی نهایت باشد

نقیض و سورها

- یك ارتباط میان $\forall$ با $\wedge$ و $\exists$ با $\vee$ داده شده است ، داریم :
  - $\neg \forall x : P(x) \equiv \exists x : \neg P(x)$
  - $\forall x : \neg P(x) \equiv \neg \exists x : P(x)$
  - $\neg \forall x : \neg P(x) \equiv \exists x : P(x)$
  - $\forall x : P(x) \equiv \neg \exists x : \neg P(x)$

مثال برای نقیض و سورها

- به گزاره ی male(_) در دامنه ی دانش آموزان درون OMB 31 توجه نمایید

---

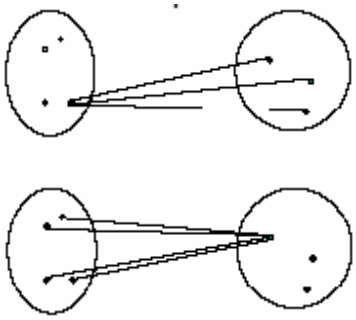
[1] males
[2] females
[3] siblings

- معنای $\neg\forall x : male(x)$ این است که موردی وجود ندارد که در آن هرکس در OMB 31 ، مرد باشد
- این مطلب معادل با این عبارت است که کسی در OMB مرد نمی باشد
- ظاهرا ، $\exists x : \neg male(x)$ می باشد

سایر معادل ها

- $\exists x \exists y : P(x, y) \equiv \exists y \exists x : P(x, y)$
- $\forall x \forall y : P(x, y) \equiv \forall y \forall x : P(x, y)$

- به دامنه ی $G1 \cup G2$ از دو گروه افراد G1 و G2 توجه نمایید
- Group1(_) و Group2(_) گزاره ها ی یگانی هستند که نشان می دهند که به ترتیب کسی در گروه G1 و G2 هست
- Know(_ , _) یک گزاره ی دودویی نشان دهنده ی این که دو نفر همدیگر را می دانند می باشد
  - $\forall x \forall y(Group1(x) \wedge Group2(y) \rightarrow Know(x, y))$



$\forall y \forall x(Group1(x) \wedge Group2(y) \rightarrow Know(x, y))$

تحلیل مرتبه ي اول

- روال تحلیل براي منطق گزاره ای برای منطق مرتبه ی اول توسعه می یابد
- سور وجودی توسط یک پردازه به نام اِسکِلِمیزاسیون [1] حذف می شود
- فرض مقدم [2] و نتیجه ی منفی شده (نقیض شده) ، به شرط های مرتبه ی اول با سور عمومی حرکت داده شده به طرف چپ فرمول ، تبدیل می شوند

منبع :

http://www.cse.unsw.edu.au/~cs9020/2006s1/FOL_Simple.pdf

[1] Skolemization
[2] premise

فصل دوازدهم

منطق مرتبه ی اول

منطق مرتبه ی اول
فصل دوازدهم

ریوس مطالب

- **چرا منطق مرتبه ی اول**
- **ظاهر ومعنای منطق مرتبه ی اول**
- **بازی با عبارات**
- **دنیای wumpus در منطق مرتبه ی اول**

موافقین و مخالفین منطق گزاره ای

منطق گزاره ای ، اظهاری (اعلانی) است : اجزای گرامری با واقعیات برابرند . منطق گزاره ای ، به اطلاعات جزیی / فصلی / منفی شده اجازه می دهد (برخلاف بیش تر ساختار های داده ای و پایگاه های داده ای) . منطق گزاره ای ، **ترکیبی** [1] می باشد: مفهوم $B_{1,1} \wedge P_{1,2}$ از $B_{1,1}$ و از $P_{1,2}$ مشتق می شود . مفهوم در منطق گزاره ای ، **مستقل از متن** [2] است . (برخلاف زبان طبیعی ، که مفهوم وابسته به متن می باشد) . منطق گزاره ای محدودیت زیادی در قدرت بیان دارد (برخلاف زبان طبیعی) . مثال ، ما نمی توانیم بگوییم " چاله ها باعث هوای مطلوب در خانه های مجاور می شوند " به جز با نوشتن یک عبارت برای هر خانه .

منطق مرتبه ی اول - از آنجایی که منطق گزاره ای ، محتویات واقعی دنیا را به خود می گیرد ، منطق گزاره ای (مثل زبان طبیعی) محتویات جهان را به خود می گیرد .

اشیاء : مردم ، خانه ها ، اعداد ، تئوری ها ، رونالد مک دونالد [3] ، رنگ ها ، بازی های بیسبال [4] ، جنگ ها ، قرن ها و .....

وابستگی ها (ارتباطات یا روابط) : قرمز ، گرد [5] ، ساختگی [6](جعلی یا قلابی) ، اول [7] (عمده) ، ساختمان چند طبقه [1] و .....

---

[1] compositional
[2] context-independent
[3] Ronald McDonald
[4] baseball
[5] round
[6] bogus
[7] prime

- برادر ... ، بزرگ تر از ، درون ... ، قطعه ای از ... ، به رنگ ، اتفاق افتاده بعد از ، داشتن ، آمدن میان و .....
- **عملکردها [2] (تابع ها) :** پدر ... ، بهترین دوست ، تصدی ... ، یکی بیش تر از ، پایان .....

منطق در حالت کلی

زبان	الزام هستی شناسی [3]	الزام معرفت شناسی [4]
منطق گزاره ای	واقعیات	**درست / غلط / ناشناخته**
منطق مرتبه ی اول	واقعیات ، اشیاء ، رابطه ها	**درست / غلط / ناشناخته**
منطق زمانی [5]	واقعیات ، اشیاء ، رابطه ها ، زمان ها	**درست / غلط / ناشناخته**
تیوری احتمال [6]	واقعیات	**اندازه ی اعتقاد [7]**
منطق فازی [8]	**واقعیات + درجه ی درستی [9]**	**مقدار فاصله ی شناخته شده [10]**

ظاهر منطق مرتبه ی اول : عناصر اساسی

- ثابت ها مثل : KingJohn,2,UCB,Koblenz,C,…
- گزاره ها مثل : Brother,>,=,...
- توابع مثل : Sqrt,LeftLegOf,…
- متغیر ها مثل : x,y,a,b,...
- رابط ها مثل : $\wedge,\vee,\neg,\Rightarrow,\Leftrightarrow$
- تساوی : $=$
- کمیت سنج ها (سورها) : $\forall,\exists$

عبارات اتمیک [11]

- عبارت اتمیک = $predicate(term_1,...,term_n)$ یا $term_1 = term_2$

واژه = $function(term_1,...,term_n)$ ، یا ثابت است و یا متغیر می باشد

مثال :

Brother(KingJohn,RichardTheLionheart) >
(Length(LeftLegOf(Richard)),Length(LeftLegOf(KingJohn)))

---

[1] multistoried
[2] functions
[3] Ontological Commitment
[4] Epistemological Commitment
[5] Temporal logic
[6] Probability logic
[7] degree of belief
[8] Fuzzy logic
[9] degree of truth
[10] known interval value
[11] Atomic

در عبـــــارت Brother(KingJohn,RichardTheLionheart) ، Brother گـــــزاره ، KingJohn ثابـــــــت ، RichardTheLionheart ثابـــــــت ، KingJohn واژه ، RichardTheLionheart واژه و کل عبارت یک عبارت اتمیک می باشد .

> (Length(LeftLegOf(Richard)),Length(LeftLegOf(KingJohn)))

در عبارت فوق ، > گزاره ، Length تابع ، LeftLegOf تابع ، Richard ثابت ، Length تابع ، مجددا LeftLegOf تابع ، KingJohn ثابت ، (Length(LeftLegOf(Richard)) واژه ، Length(LeftLegOf(KingJohn)) واژه و کل عبارت (> (Length(LeftLegOf(Richard)),Length(LeftLegOf(KingJohn)))) ، عبارت اتمیک می باشد .

عبارات پیچیده - عبارات پیچیده از عبارات اتمیک با استفاده از رابط ها ساخته می شوند :

$$\neg S, S_1 \wedge S_2, S_1 \vee S_2, S_1 \Rightarrow S_2, S_1 \Leftrightarrow S_2$$

(همانند منطق گزاره ای)

مثال :

$$Sibling(KingJohn, Richard) \Rightarrow Sibling(Richard, KingJohn)$$

$$> (1,2) \vee \leq (1,2)$$

$$> (1,2) \wedge \neg > (1,2)$$

در عبارت فوق ، Sibling ، گزاره مـی باشد . KingJohn واژه است . Richard واژه است . Sibling طـرف راسـت گـزاره اسـت . Richard طـرف راسـت واژه اسـت . KingJohn طـرف راسـت واژه اسـت . Sibling(KingJohn,Richard) ، عبـارت اتمیـک اسـت . Sibling(Richard,KingJohn) عبـارت اتمیـک اسـت و کـل عبـارت ، عبـارت پیچیـده مـی باشد .

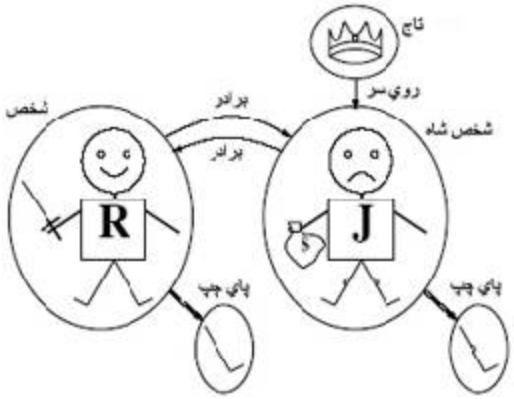
صحت در منطق مرتبـه ی اول - عبـارت هـا بـا رعایـت یـک مـدل و یـک تفسیر [1] ، درسـت مـی باشند. مدل شامل اشیای بزرگ تر یا مساوی یـک (عناصـر دامنـه) و رابـط هـای میـان آن هـا می باشد (contains≥1) . تفسیر موارد مراجعه را برای موارد زیر تایین می نماید :

سمبل های ثابت [2] ← اشیاء
سمبل های گزاره ای [3] ← رابط ها
سمبل های تابعی [4] ← رابط های تابعی [5]

یک عبارت اتمیک $predicate(term_1,...,term_n)$ در صورتی درست است که اشیاء ارجـاع شده به آن توسط $term_1,...,term_n$ با ارتباط های ارجاع شده توسط گزاره در ارتباط باشند .

مثالی برای مدل های منطق مرتبه ی اول

[1] interpretation
[2] constant symbols
[3] predicate symbols
[4] function symbols
[5] functional relations



مثال صحت - به تفسیری به صورت زیر توجه نمایید :

ریچارد [1] ← فردی دلیر [2] است

جان [3] ← پادشاهی بد

برادر ← ارتباط برادری [4]

تحت این تفسیر ، Brother(Richard,John) فقط در مورد **ریچارد فردی دلیر است** درست است و **جان پادشاهی بد است** در این مدل دارای ارتباط برادری هستند.

مدل های بسیار فراوان منطق مرتبه ی اول

می توانیم مدل های منطق مرتبه ی اول را برای یک پایگاه دانش ارایه شده ، شمارش نماییم :

برای هر تعداد از عناصر دامنه ی n از ۱ تا ∞

برای هر k-ary گزاره ی P_k در لغت [5]

برای هر k-ary ممکن در ارتباط با n شیء

برای هر سمبل ثابت C در لغت

برای هر انتخاب مورد مراجعه برای C از n شی ء ...

محاسبه ی حبس برای مدل شمارشی منطق مرتبه ی اول آسان نیست !

سور عمومی - همه در برکلی [6] زرنگ [7] هستند : $\forall x : At(x, Berkeley) \Rightarrow Smart(x)$

$\forall xP$ در یک مدل m درست است در صورتی که P به ازای هر x ممکن در مدل درست باشد .

$$(At(KingJohn, Berkeley) \Rightarrow Smart(KingJohn)) \wedge (At(Richard, Berkeley) \Rightarrow Smart(Richard))$$
$$\wedge (At(Berkeley, Berkeley) \Rightarrow Smart(Berkeley)) \wedge ...$$

جلوگیری از یک اشتباه مشترک - معمولا ، $\Leftarrow$ پیوند اصلی با $\forall$ می باشد . اشتباه مشترک : استفاده از $\wedge$ به صورت پیوند اصلی با $\forall$: $\forall x : At(x, Berkeley) \wedge Smart(x)$ معنای آن این است که " همه در برکلی هستند و همه باهوش هستند "

تعیین خواص وجودی [8]

---

[1] Richard
[2] Lionheart
[3] John
[4] brotherhood relation
[5] vocabulary
[6] Berkely
[7] smart
[8] Existential quantification

- $\exists \langle \text{var}\, iables \rangle \langle sentence \rangle$

شخصی در Stanford باهوش است : $\exists x At(x, S\tan ford) \wedge Smart(x)$

$\exists x P$ در یک مدل m در صورتی که P با x در برخی از اشیای ممکن در مدل موجود باشد درست است

به طور کلی با جدایی های مثالی P برابر می باشد :

$$(At(KingJohn, S\tan ford) \wedge Smart(KingJohn))$$
$$\vee (At(Richard, S\tan ford) \wedge Smart(Richard))$$
$$\vee (At(S\tan ford, S\tan ford) \wedge Smart(S\tan ford))$$

دیگر مشکلات مشترک - معمولا ، $\wedge$ ارتباط اصلی با $\exists$ می باشد

اشتباه مشترک : استفاده از $\Leftarrow$ به صورت اتصال اصلی با $\exists$:

$$\exists x At(x, S\tan ford) \Rightarrow Smart(x)$$

در صورتی درست است که کسی در استنفورد [1] نباشد !

خصوصیات کمیت سنج ها (سورها)

$\forall x \forall y$ مثل $\forall y \forall x$ می باشد (چرا ؟؟) . $\exists x \exists y$ مثل $\exists y \exists x$ می باشد (چرا ؟؟) . $\exists x \forall y$ مثل $\forall y \exists x$ نمی باشد .

$\exists x \forall y Loves(x, y)$ ؛ یعنی : " شخصی وجود دارد که همه را در جهان دوست دارد "

$\forall y \exists x Loves(x, y)$ ؛ یعنی : " همه در جهان با لااقل یک فرد دوست می شوند "

دوگانگی کمیت سنج ها [2] : هر چیزی می تواند با استفاده از دیگری بیان شود :

$$\forall x Likes(x, IceCream) \equiv \neg \exists x \neg Likes(x, IceCream)$$
$$\exists x Likes(x, Broccoli) \equiv \neg \forall x \neg Likes(x, Broccoli)$$

بازی با عبارات

برادران هم نژاد [3] هستند را به این صورت بیان می نماییم $\forall x, y Brother(x, y) \Rightarrow Sibling(x, y)$

" برادری (هم نژادی) " [رابطه ای] [4] متقارن [5] است

$$\forall x, y : Sibling(x, y) \Leftrightarrow Sibling(y, x)$$

مادر یکی والده ی یکی دیگر است :

$$\forall x, y : Mother(x, y) \Leftrightarrow (Female(x) \wedge Parent(x, y))$$

اولین عموزاده بچه ای هم نژاد با والد است

$$\forall x, y : FirstCou\sin(x, y) \Leftrightarrow \exists p, psParent(p, x) \wedge Sibling(ps, p) \wedge Parent(ps, y)$$

تساوی

$term_1 = term_2$ تحت یک تفسیر ارایه شده صحیح است اگر و فقط اگر $term_1$ و $term_2$ به شی ء همانند ارجاع نمایند .

مثال : ۱=۲ و $\forall x \times (Sqrt(x), Sqrt(x)) = x$ خوشایند هستند و ۲=۲ مجاز می باشد .

مثال ، تعریف کامل **هم نژادی** در واژه های **والد**

---

[1] Stanford
[2] Quantifier duality
[3] Sibling
[4] مترجم
[5] symmetric

$$\forall x, y : Sibling(x, y) \Leftrightarrow [\neg(x = y) \wedge \exists m, f \neg(m = f) \wedge$$
$$Parent(m, x) \wedge Parent(f, x) \wedge Parent(m, y) \wedge Parent(f, y)]$$

محاوره با پایگاه های دانش منطق مرتبه ی اول

تصور نمایید یک مولفه ی دنیای wumpus در حال استفاده از یک پایگاه دانش منطق مرتبه ی اول می باشد و یک بوی خوب [1] و یک بوی بد [2] را در t=5 استشمام می کند [3] (اما تابش نمی کند) :

Tell(KB,Percept([Smell,Breeze,None],5))
Ask(KB, ∃a Action(a,5))

سوال : آیا پایگاه دانش هر عمل مشخص را در t=5 حبس می نماید ؟ جواب : بله ،

زیر وضعیت (لیست الزام آور [4]) ← {a/Shoot}

یک عبارت ارایه شده ی S و یک زیر وضعیت б و Sб نتیجه ی دو راهه ی б به S را مشخص می نماید ؛ مثال :

S=Smarter(x,y) , σ={x/Hillary,y/Bill} , Sσ=Smarter(Hillary,Bill)

Ask(KB,S) برخی / همه ی б هایی که در آن $KB \models S\sigma$ را بر می گرداند .

پایگاه دانش برای دنیای wumpus

" گزاره [5] "

$\forall b, g, t$ Percept([Smell,b,g],t) ⇒ Smelt(t)

$\forall s, b, t$ Percept([s,b,Glitter],t) ⇒ AtGold(t)

بازتاب [6] :

$\forall t$ AtGold(t) ⇒ Action(Grab,t)

بازتاب با حالت یا وضعیت ورودی [7] : ما می توانیم یک طلا را از قبل داشته باشیم ؟

$$\forall t AtGold(t) \wedge \neg Holding(Gold, t) \Rightarrow Action(Grab, t)$$

Holding(Gold,t) نمی تواند مشاهده شود ← نگهداری جریان تغییرات ، ضروری می باشد .

استنباط خصوصیات پنهان

خصوصیات محل ها :

$$\forall x, t : At(Agent, x, t) \wedge Smelt(t) \Rightarrow Smelly(x)$$
$$\forall x, t : At(Agent, x, t) \wedge Breeze(t) \Rightarrow Breezy(x)$$

خانه های نزدیک یک چاله [8] بدبو هستند :

قانون تشخیصی [9] --- از اثر ، استنباط می نماید .

$$\forall y : Breezy(y) \Rightarrow \exists x Pit(x) \wedge Adjacent(x, y)$$

قانون سببی [10] - اثر را از مورد ، استنباط می نماید .

$$\forall x, y : Pit(x) \wedge Adjacent(x, y) \Rightarrow Breezy(y)$$

---

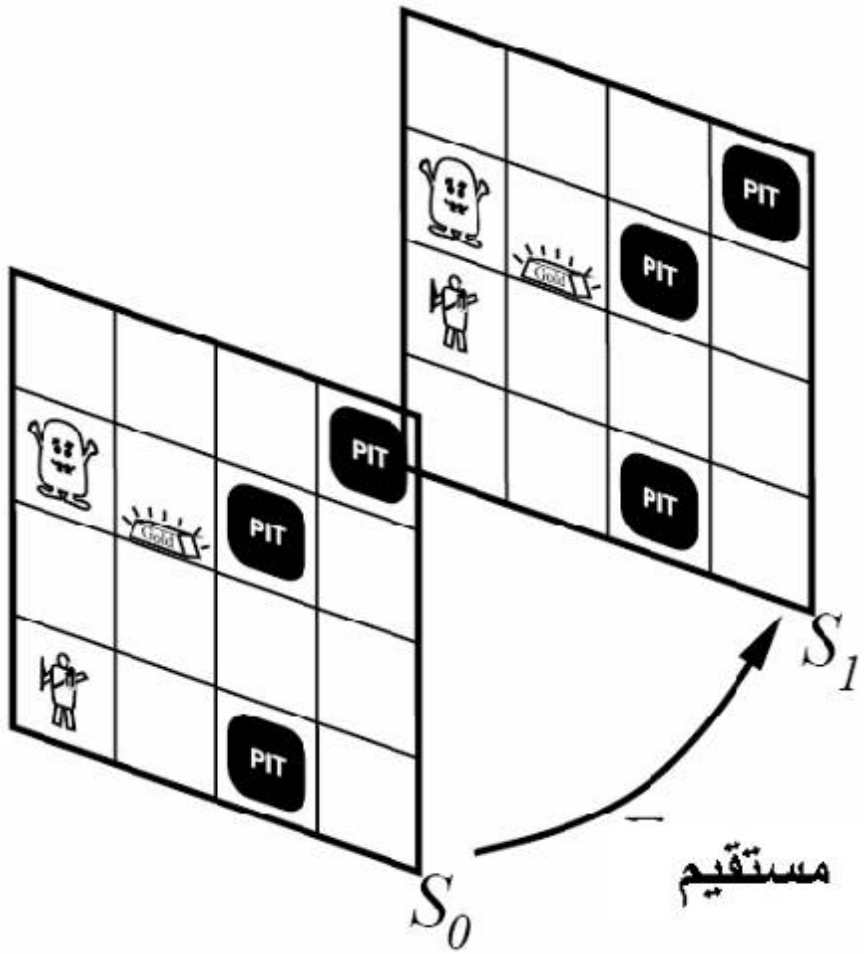
[1] smell
[2] breeze
[3] preceive
[4] binding list
[5] Perception
[6] Reflex
[7] reflex with internal state
[8] pit
[9] Diagnostic rule
[10] Causal rule

هیچ کدام از این ها کامل نیستند --- مثال : قانون سببی نمی گوید که آیا خانه های دور از چاله ها می توانند بدبو باشند .

تعریف برای **تابع بدبویی**

$$\forall y : Breezy(y) \Leftrightarrow [\exists x : Pit(x) \wedge Adjacent(x, y)]$$

نگهداری جریان تغییر - واقعیات به صورت ذاتی در وضعیت ها [1] نگهداری می شوند . مثال ، Holding(Gold,Now) ترجیحا فقط از Holding(Gold) استفاده می کنیم . محاسبه ی وضعیت [2] راهی برای دادن تغییر در منطق مرتبه ی اول می باشد : یک آرگومان وضعیت را برای هر تابع یا گزاره اضافه می نماید . مثال : **حالا** در Holding(Gold,Now) یک وضعیت را مشخص نمایید . وضعیت ها با تابع نتیجه متصل می شوند . تابع Result(a,s) وضعیتی که نتایج انجام دهنده a در s دارند را دارد .



تشریح عملکردها I

قضیه ی " اثر " [3] تغییراتی را که به وسیله ی عملکرد ، اعمال شده را تشریح می نماید .

$$\forall s AtGold(s) \Rightarrow Holding(Gold, Result(Grab, s))$$

قضیه ی " قاب " [4] ثابت هایی که به وسیله ی عملکرد اعمال شده اند را تشریح می نماید .

---

[1] situations
[2] Situation calculus
[3] "Effect" axiom
[4] "Frame" axiom

$\forall s HaveArrow(s) \Rightarrow HaveArrow(\text{Re}sult(Grab, s))$

مساله ی قاب [1] **:** یک راه حل مطلوب برای به کار گیری ثابت را پیدا نمایید :

(a) نمایش – جلوگیری از قضیه ی قاب ها

(b) استنتاج – جلوگیری از " copy – overs " تکرار شده برای نگهداری جریان حالت

مساله ی صفت [2] : توضیحات درست عملکردهای واقعی نیازمند اخطارهای [3] بی پایان [4] در صورتی که طلا متغیر (بی ثبات) [5] یا ... باشد .

مساله ی انشعاب [6] **:** عملکردهای واقعی ، دارای تعدادی نتایج [7] فرعی [8] که درباره ی خاک طلا ، پوشیدن [9] و پاره کردن [10] دستکش ها [11] و ... می باشند .

تشریح عملکردها II

قضایای حالت جانشین [12] ، مسایل قاب ارایه شده را حل می نمایند . هر قضیه **درباره** ی یک **گزاره** می باشد . P درست بعد از آن می باشد $\Leftrightarrow$ (اگر و فقط اگر) یک عملکرد P را درست کند $\vee$ (یا) P همیشه درست بوده و هیچ عملکردی آن را غلط نکند . برای نگهداری طلا :

$$\forall a,s\ Holding(Gold,Result(a,s)) \Leftrightarrow [(a=Grab \wedge AtGold(s)) \vee (Holding(Gold,s) \wedge a \neq Release)]$$

ساختن طرح ها

وضعیت اولیه در پایگاه دانش :

$$At(Agent,[1,1],S_0)$$
$$At(Gold,[1,2],S_0)$$

سوال : اگر $Ask(KB, \exists s Holding(Gold, s))$ ، آن گاه در چه وضعیتی دارایی ما طلا خواهد بود ؟

جواب : $\{s / \text{Re}sult(Grab, \text{Re}sult(Forward, S_0))\}$ توجه کنید که اگر مستقیم بروید ، در این صورت طلا را خواهید داشت . فرض بر این است که عامل با توجه به طرح های شروع کننده در S_0 هست و S_0 فقط وضعیت تشریح شده در پایگاه دانش می باشد .

ساخت طرح ها : روشی بهتر - طرح ها را به صورت عملکردهای متوالی $[a_1, a_2, ..., a_n]$ ارایه نمایید . PlanResult(p,s) نتیجه ی اجرای p در s می باشد . سپس پرسش $Ask(KB, \exists p Holding(Gold, Plan\text{Re}sult(p, S_0)))$ دارای راه حل {p/[Forward,Grab]} خواهد بود .

تعریف PlanResult به صورت Result :

---

[1] Frame problem
[2] Qualification problem
[3] caveats
[4] endless
[5] slippery
[6] Ramification problem
[7] consequence
[8] secondary
[9] wear
[10] tear
[11] gloves
[12] Successor-state

$$\forall s : Plan\,\mathrm{Re}\,sult([], s) = s$$

$$\forall a, p, s : Plan\,\mathrm{Re}\,sult([a \mid p], s) = Plan\,\mathrm{Re}\,sult(p, \mathrm{Re}\,sult(a, s))$$

سیستم های طراحی [1] ، استدلال کننده های [2] تک منظوره [3] اند که برای انجام این نوع از استدلال ها به روشی ، به مراتب مناسب تر از یک استدلال کننده ی همه منظوره [4] طراحی شده اند .

خلاصه ـ منطق مرتبه ی اول :

- اشیا و رابطه ها سمانتیک ها (معناها) ی قدیمی هستند .
- ظاهر : ثابت ها ، توابع ، گزاره ها ، برابری و کمیت سنج ها می باشند .

توانایی بیان افزایش داده شده [5] : برای تعریف دنیای wumpus مناسب می باشد

محاسبه ی وضعیت [6] :

- عرف های تشریح کننده ی عملکردها و تغییر در منطق مرتبه ی اول (First Order Logic=FOL) .
- می توان طرح ها را به صورت نتیجه در یک پایگاه دانش محاسبه ی وضعیت فرمول بندی نمود .

منابع :

http://aima.eecs.berkeley.edu/slides-pdf/chapter08.pdf
http://www.uni.koblenz.de

[1] Planning systems
[2] reasoners
[3] special-purpose
[4] general-purpose
[5] increased expressive power
[6] situation calculus

فصل سیزدهم

استنتاج در منطق مرتبه ی اول

استنتاج در منطق مرتبه ی اول [1]
فصل سیزدهم

ریوس مطالب

- ☑ **کاهنده ی استنتاج منطق مرتبه ی اول به استنتاج گزاره ای**
- ☑ **یک شکلی [2]**
- ☑ **برجستگی های روش تعمیم یافته [3]**
- ☑ **زنجیره ی مستقیم و معکوس**
- ☑ **برنامه نویسی منطقی [4]**
- ☑ **تحلیل [5]**

مروری بر منطق مرتبه ی اول

- به یاد بیاورید که منطق گزاره ای موارد زیر را پیوند می دهد :
  - o واقعیاتی در مورد جهان که یا درست است و یا غلط
  - o این واقعیات می توانند برای ساخت جمله ها با استفاده از رابط های منطقی $(\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow)$ استفاده شوند
  - o مثال : $P_{1,1} \vee P_{2,2}$.
- منطق مرتبه ی اول ، موارد زیر را پیوند می دهد :
  - o اشیا و ارتباط ها (نسبت ها) ی میان اشیا و ارتباط های آن ها ، به شکل عبارات اتمیک
  - o عبارات اتمیک می توانند با رابط های منطقی متصل شوند
  - o ما می توانیم متغیرها و سورها را با ارجاع به اشیای نامحدود معرفی نماییم
  - o مثال : $Siblings(Bart, Lisa), \forall x : Likes(x, M\arg e) \Rightarrow Likes(x, \hom er)$

---

[1] Inference in First-order Logic
[2] unification
[3] Generalized Modus Ponens
[4] Logic programming
[5] Resolution

تاریخچه ی مختصری از استدلال - سال ۴۵۰ قبل از میلاد مسیح / پیروان فلسفه ی رواقیون [۱] / منطق گزاره ای و(شاید) استدلال | سال ۳۲۲ قبل از میلاد مسیح / ارسطو / قیاس منطقی [۲] (قوانین استدلال) و سورها | سال ۱۵۶۵ / کاردانو [۳] تیوری احتمال [۴] (منطق گزاره ای [۵] + عدم قطعیت [۶]) | سال ۱۸۴۷ / بول [۷] / منطق گزاره ای (دوباره) | سال ۱۸۷۹ / فرگ [۸] / منطق مرتبه ی اول | سال ۱۹۲۲ / ویتجنستین [۹] / اثبات توسط جدول صحت | سال ۱۹۳۰ / گدل [۱۰] / الگوریتم کاملی برای منطق مرتبه ی اول وجود دارد | سال ۱۹۳۰ / هربرند [۱۱] / الگوریتم کاملی برای منطق مرتبه ی اول | سال ۱۹۳۱ / گدل / الگوریتم کاملی برای محاسبه وجود ندارد | سال ۱۹۶۰ / دیویس-پوتنام [۱۲] / الگوریتمی کاربردی برای منطق گزاره ای | سال ۱۹۶۵ / رابینسن [۱۳] الگوریتمی کاربردی برای منطق مرتبه ی اول – تحلیل .

استنتاج در منطق مرتبه ی اول

- آیا ما می توانیم نوع های یکسان استنتاج را با منطق مرتبه ی اول ، همان طوری که با منطق گزاره ای انجام می دادیم ، انجام دهیم ؟
- بله ، با برخی از جزییات اضافی
- به نگهداری جریان اتصال های متغیر [۱۴] نیاز داریم (جانشینی [۱۵])

برداشتن سورها

معرفی عمومی [۱۶]

- ما همیشه می توانیم یک واژه ی زمینه را با یک متغیر جانشین نماییم
  - $\forall x \, LivesIn(x, Springfield) \Rightarrow knows(x, Homer)$
  - از آنجایی که این عبارت برای همه ی x ها صحیح می باشد ، ما می توانیم {x=Bart} را جایگزین نماییم
  - $LivesIn(Bart, Springfield) \Rightarrow knows(Bart, Homer)$

پس : $\frac{\forall \upsilon : \alpha}{Subst(\{\upsilon / g\}, \alpha)}$ برای هر متغیر v و واژه ی زمینه ی g برقرار می باشد .

مثال ، نتیجه های $\forall x : King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ عبارتند از :

---

[1] Stoics
[2] syllogisms
[3] Cardano
[4] probability theory
[5] propositional logic
[6] uncertainty
[7] Boole
[8] Frege
[9] Wittgenstein
[10] Godel
[11] Herbrand
[12] Davis/Putnam
[13] Rabinson
[14] variable bindings
[15] substitution
[16] Universal instantiation (UI)

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$
$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$$

M

معرفی وجودی [1]

- همچنین ما می توانیم به شی ای که یک عبارت را راضی می کند یک نام بدهیم .
  - $\exists x : LivesIn(x, Springfield) \wedge knows(x, Homer)$
  - ما می دانیم که این باید برای لااقل یک شی نگهداری شود . بیایید این شی را k بنامیم .
  - $LivesIn(k, Springfield) \wedge knows(k, Homer)$
  - k یک ثابت اِسکلِم نامیده می شود
  - k باید استفاده نشده باشد – ما باید یک راه ارجاع به یک شی وجودی را ارایه نماییم
- یک بار ما سورها را برداشته ایم ، ما می توانیم از قانون های استنتاج گزاره ای استفاده نماییم .

پس ، برای هر عبارت α و متغیر v و سمبل ثابت k که در جای دیگری در پایگاه دانش وجود ندارد داریم :

$$\frac{\exists \upsilon \alpha}{Subst(\{\upsilon / k\}, \alpha)}$$

مثال : نتایج $\exists x Crown(x) \wedge OnHead(x, John)$

$$Crown(C_1) \wedge OnHead(C_1, John)$$

C_1 به وجود آمده از یک سمبل ثابت جدید به نام ثابت اسکلم [2] می باشد . مثالی دیگر : از $\exists x : d(x^y)/dy = x^y$ ما نتیجه می گیریم ، $d(e^y)/dy = e^y$ e) ای که به وجود آمده است یک سمبل ثابت جدید می باشد) .

معرفی وجودی می تواند در برخی از موارد برای **اضافه** نمودن عبارات جدید به کار رود ؛ پایگاه دانش جدید ، منطقا با قبلی برابر می باشد . معرفی وجودی می تواند برای **جایگزین کردن** عبارت موجود به کار گرفته شود ؛ پایگاه دانش جدید با قبلی **برابر نیست** و در صورتی خوشایند است که پایگاه دانش قبلی خوشایند باشد .

گزاره بندی [3]

- ما می توانیم هر عبارت وجودی را با یک نوع اسکلمایز شده جایگزین نماییم
- برای عبارت های با سور عمومی ، در هر جایگزین ممکن جایگزین نمایید
- این به ما اجازه می دهد که از قانون های استنتاج گزاره ای استفاده نماییم .
- مساله : خیلی ناکارامد می باشد
- تا سال ۱۹۶۰ وضعیت همین جور بود .

---

[1] Existential instantiation (EI)
[2] Skolem constant
[3] propositionalization

تبدیل به استدلال گزاره ای ـ تصور نمایید که پایگاه دانش فقط شامل عبارات زیر باشد :
$\forall$ x: King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x) , King(John) , Greedy(John) , Brother(Richard,John)
معرفی عبارت عمومی در **همه ی موارد ممکن است** و داریم :
King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John) , King(Richard) $\wedge$ Greedy(Richard) $\Rightarrow$ Evil(Richard) , King(John) , Greedy(John) , Brother(Richard,John)
پایگاه دانش جدید گزاره ای می باشد : سمبل های گزاره ای عبارتند از :
King(John) , Greedy(John) , Evil(John) , King(Richard)
و .....
تبدیل
ـ ادعا : یک عبارت زمینه به وسیله ی پایگاه دانش جدید در صورتی که به وسیله ی پایگاه دانش اصلی به وجود آمده باشد ، موجود می شود .
ـ ادعا : هر پایگاه دانش منطق مرتبه ی اول می تواند برای حفظ موجودیت ، به صورت گزاره ای بیان شود .
ـ ایده : پایگاه دانش را به صورت گزاره ای و پرس و جو بیان نمایید ، تحلیل را به کار ببرید و نتایج را برگردانید .
ـ مساله : با سمبل های توابع ، واژگان زمینه به صورت نامحدود وجود خواهند داشت .
مثال : Father(Father(Father(John)))
قضیه ی هربرند (۱۹۳۰) : در صورتی که عبارت α توسط یک پایگاه دانش منطق مرتبه ی اول به وجود بیاید ، توسط یک زیرمجموعه ی محدود از پایگاه دانش های گزاره ای موجود می شود .
ایده : برای n = 0 تا بینهایت کارهای زیر را انجام بده
یک KB گزاره ای با عمق n بسازید
بررسی نمایید که آیا α توسط این KB تولید می شود
قضیه ی تورینگ و چورچ [1](۱۹۳۶) : به وجود آمدن [2] در منطق مرتبه ی اول تقریبا قابل تصمیم گیری [3] می باشد .
مسایل با گزاره ها ـ گزاره ، برای تولید تعداد زیادی عبارات که به هم نامربوط هم هستند به کار می رود .
مثال :
$\forall$ x King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x) , King(John) , $\forall$ y Greedy(y) , Brother(Richard,John)
حاصل به نظر می رسد که Evil(John) باشد ، اما گزاره بندی ، تعداد زیادی واقعیات نظیر Greedy(Richard) که نامربوط هستند را به وجود می آورد . با گزاره های k - تایی p و n های ثابت ، $p.n^k$ معرفی وجود خواهد داشت . با سمبل های تابعی ، معرفی های با مقادیر به مراتب بدتری وجود خواهد داشت !
یک شکلی

[1] Turing,Church
[2] entailment
[3] semidecidable

- کلید یک شکلی این است که ما فقط می خواهیم جایگزین هایی برای عباراتی که به ما کمک می کنند چیزهایی را ثابت نماییم پیدا کنیم .
- برای مثال ، اگر ما بدانیم :
  - $\forall x : LivesIn(x, Springfield) \wedge WorksAt(x, PowerPlant) \Rightarrow know(x, Homer)$
  - $\forall y : LivesIn(y, Springfield)$
  - $WorksAt(MrSmithers, PowerPlant)$
- ما باید قادر باشیم مستقیما نتیجه بگیریم knows(MrSmithers,Homer)
- جایگزینی : {x/MrSmithers,y/MrSmithers}

پس در مثال قبلی ، در صورتی که بتوانیم یک زیر وضعیت θ را پیدا کنیم ، می توانیم بلافاصله استدلال نماییم King (x) و Greedy (x) با King (John) و Greedy(y) مطابقت می کنند .

$\theta = \{x/John, y/John\}$, $Unify(\alpha,\beta)= \theta$ اگر $\alpha\theta = \beta\theta$

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}
Knows(John,x)	Knows(x,OJ)	fail

استاندارد کردن از روی هم افتادن[1] متغیرها جلوگیری می کند ، مثال : $Knows(z_{17}, OJ)$

روش کلی برجستگی ها [2]

- این بحث یک شکل کلی روش برجستگی ها می باشد .
- ایده ی اساسی : بیایید فرض کنیم :
  - یک استنتاج به شکل $P_1 \wedge P_2 \wedge ... \wedge P_i \Rightarrow Q$ را داریم
  - عبارات به صورت $P_1', P_2', ..., P_i'$ می باشند
  - اگر مجموعه ای از جانشین ها به صورت زیر وجود داشته باشند

$$P_1 = P_1', P_2 = P_2', ..., P_n = P_n'$$

- سپس ما می توانیم جانشین را به کار بگیریم و روش برجستگی ها را برای به دست آوردن Q به کار بگیریم .
- این تکنیک استفاده از جانشین ها برای جفت عبارات بالا برای به دست آوردن استنتاج ، ***یک شکلی*** نام دارد .
- پردازه ی استنتاج ما حالا یکی از پیداکننده های جانشین هایی که به ما اجازه خواهند داد عبارات جدید را مشتق کنیم می باشد .
- الگوریتم یگانی : دو عبارت را می پذیرد و مجموعه ای از جانشین هایی که عبارات را یکتا می سازند را برمی گرداند .
  - WorksAt(x,PowerPlant),WorksAt(Homer,PowerPlant) ، {x/Homer} را تولید می نماید .

---

[1] overlap
[2] Generalized Modus Ponens (GMP)

- WorksAt(x,PowerPlant),WorksAt(Homer,y) ، {x/Homer,y/PowerPlant} را تولید می نماید
- WorksAt(x,PowerPlant),WorksAt(FatherOf(Bart),y) ، {x/FatherOf(Bart),y/PowerPlant} را تولید می کند .
- WorksAt(x,PowerPlant),WorksAt(Homer,x) غلط می باشد – x نمی تواند هم به Homer اشاره کند و هم به PowerPlant .
- این آخرین عبارت یک مساله است فقط به خاطر این که ما خواسته ایم x را در هر دو عبارت استفاده نماییم .
- ما می توانیم x را با یک متغیر منحصر به فرد (x_{21}) که یک عبارت است جایگزین نماییم .
  - این کار استانداردسازی نامیده می شود .

- اگر بیش از یک جانشین وجود داشته باشد که بتوانند دو عبارت که به نظر می رسد یکسان هستند را بسازند چطور ؟
  - Sibling(Bart,x) و Sibling(y,z)
  - می تواند {Bart/y,x/z} یا {x/Bart,y/Bart,z/Bart} را تولید نماید
- اولین یکسان سازی کلی تر از دومی است - الزام های کم تری را سبب می شود .
- ما می خواهیم کلی ترین یکسان کننده ها را در زمانی که استنتاج را انجام می دهیم پیدا نماییم .

الگوریتم یکسان سازی

- برای یگانه کردن دو عبارت ، به صورت بازگشتی جلو بروید .
- اگر هر دو عبارت تک منغیری است ، یک یگانه سازی که متغیر را به یک ثابت متصل می کند را پیدا نمایید .
- در غیر این صورت یکانگی درون واژه ی اول را ، که توسط باقی مانده ی هر عبارت دنبال شده است را صدا بزنید .
  - $Sibling(x, Bart) \wedge PlaysSax(x)$ و Sibling(Lisa,y)
  - ما می توانیم Sibling(x,Bart) و Sibling(Lisa,y) را با {x/Lisa,y/Bart} یگانه نماییم .
- پردازه ی پیدا کردن مجموعه ی کاملی از جانشین ها ، یک پردازه ی جستجو می باشد .
  - وضعیت ، لیستی از جانشین ها می باشد
  - تابع جانشین ، لیستی از یکسان سازی های بالقوه و لیست های جانشین جدید است .

پس در مورد یکسان سازی داریم :

$$\frac{p_1', p_2', ..., p_n', (p_1 \wedge p_2 \wedge ... \wedge p_n \Rightarrow q)}{q\theta}$$

در جایی که $p_i'\theta = p_i\theta$ برای همه ی i ها برقرار باشد . به عنوان مثال ، اگر King(John) ، p_1' می باشد . King(x) ، p_1 می باشد . Greedy(y) ، p_2' می باشد . Greedy(x) ، p_2 می

باشد . {x/John,y/John} ، Θ می باشد و Evil(x) ، q می باشد و Evil(John) ، qΘ می باشد .

روش کلی برجستگی با شرط های نامحدود پایگاه دانش استفاده می شود (**دقیقا** با یک لفظ مثبت) . تمام متغیر ها ، کمیت های عمومی را به خود می گیرند .

اثبات روش کلی برجستگی - باید نشان بدهیم که : $p_1',..., p_n', (p_1 \wedge ... \wedge p_n \Rightarrow q) \models q\theta$

مشروط بر این که برای همه ی i ها ی موجود ، $p_i'\theta = p_i\theta$ باشد .

اثبات : با UI برای هر شرط نامحدود p داریم $p \models p\theta$

$$1.(p_1 \wedge ... \wedge p_n \Rightarrow q) \models (p_1 \wedge ... \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge ... \wedge p_n\theta \Rightarrow q\theta)$$

$$2.p_1',..., p_n' \models p_1' \wedge ... \wedge p_n' \models p_1'\theta \wedge ... \wedge p_n'\theta$$

۳. از ۱ و ۲ نتیجه می شود که qθ توسط روش های معمولی دنبال می شود .

مثال : پایگاه دانش - قانون می گوید که این یک جنایت است که یک آمریکایی به ملت های دشمن ، اسلحه بفروشد . کشور نونو [1] یک دشمن آمریکا می باشد و دارای تعدادی موشک می باشد و تمام این موشک ها توسط کلنل وست [2] که یک آمریکایی است به این کشور فروخته شده است . ثابت کنید که کلنل وست یک جنایت کار می باشد .

... برای یک آمریکایی این جنایت است که به ملل دشمن اسلحه بفروشد :

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Cri\min al(x)$$

کشور نونو دارای تعدادی موشک می باشد ... . توجه کنید :

$$\exists x Owns(Nono, x) \wedge Missle(x) : Missle(M_1), Owns(Nono, M_1)$$

... تمام این موشک ها توسط کلنل وست فروخته شده اند .

$$\forall x Missle(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

موشک ها اسلحه هستند : $Missle(x) \Rightarrow Weapon(x)$

$Enemy(x,America) \Rightarrow Hostile(x)$

وست آمریکایی است ... American(West).

کشور نونو یک دشمن آمریکا است : Enemy(Nono,America)

زنجیره ی مستقیم

- ایده ی اساسی : با واقعیات و قوانین شروع کنید (استنباط ها)
- دایما روش برجستگی ها را به کار ببرید تا واقعیات جدید بتوانند مشتق شوند .
- به شرط های نامحدود احتیاج داریم
  - o استنباط های با شرط های مثبت در گذشته
  - o واقعیت های مثبت
  - o Jess از یک نوع کلی شرط های نامحدود استفاده می نماید .

الگوریتم زنجیره ی مستقیم (به بیان دیگر)

تابع FOL-FC-Ask(KB,α) یک تعویض یا غلط را بر می گرداند

مادامی که new خالی است کارهای زیر را انجام بده

new ←{}

---

[1] Nono
[2] Colonel West

برای هر عبارت r در KB موارد زیر را انجام بده

$(p_1 \wedge ... \wedge p_n \Rightarrow q) \leftarrow Standardize-Apart(r)$

برای هر θ در $(p_1 \wedge ... \wedge p_n)\theta = (p_1' \wedge ... \wedge p_n')\theta$ برای برخی از $p_1', ..., p_n'$ در KB

$q' \leftarrow Subst(\theta, q)$

در صورتی که q' تغییر نام یک عبارت موجود در KB یا new نبود کارهای زیر را انجام بده

q' را به new اضافه کن

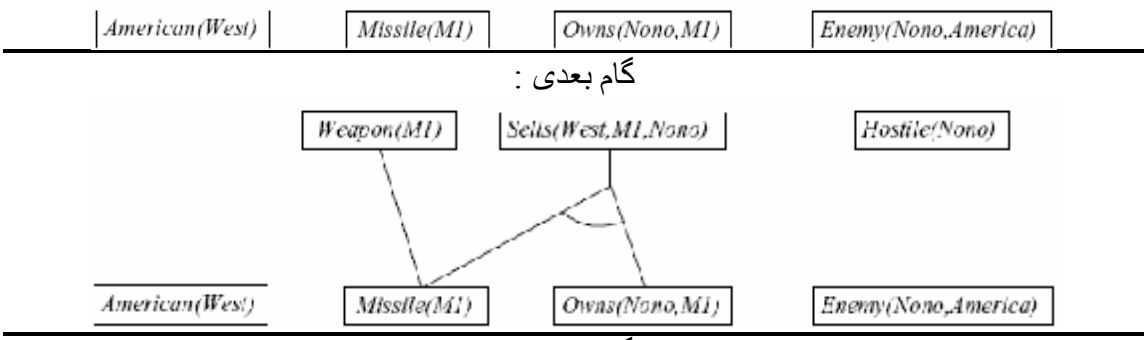
$\phi \leftarrow Unify(q', \alpha)$

در صورتی که ϕ ، fail نمی باشد ، ϕ رابرگردان

new را به KB اضافه کن

false را برگردان

اثبات زنجیره ی مستقیم

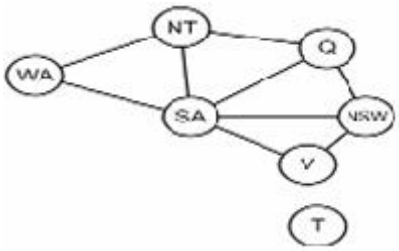


خصوصیات زنجیره ی مستقیم

- زنجیره ی مستقیم ، بی عیب می باشد ، از آنجایی که از روش برجستگی ها استفاده می نماید
- برای شرط های محدود ، زنجیره ی مستقیم ، کامل می باشد .
- خیلی شبیه به جستجوی اول – بهترین عمل می کند
- این الگوریتم اساسی خیلی هم کارامد نیست ،
  - همه ی یکی کننده های ممکن را پیدا می کند ولی گران است
  - هر قانون در هر تکرار مجددا چک می شود .
  - واقعیاتی که به هدف منجر نمی شوند تولید می شوند .

زنجیره ی مستقیم به طور گسترده ای در ***پایگاه داده های قیاسی*** [1] استفاده می شود . یک سیستم پایگاه داده ی قیاسی ، سیستمی پایگاه داده ای است که می تواند قیاس را براساس قوانین و واقعیات نگهداری شده پایگاه دانش قیاسی انجام دهد . سیستم های پایگاه دانش قیاسی : اساسا به قوانین و واقعیات رسیدگی می کنند . از یک زبان اعلانی (نظیر پرولوگ) برای مشخص کردن این قوانین و واقعیات استفاده نمایید . از یک موتور استنتاج که می تواند قوانین و واقعیات جدید را از آن هایی که ارایه شده اند قیاس نماید استفاده نمایید . [2]

مثال تطبیق دهنده ی سخت [3]



$$Diff(wa,nt) \wedge Diff(wa,sa) \wedge Diff(nt,q) Diff(nt,sa) \wedge Diff(q,nsw) \wedge Diff(q,sa) \wedge Diff(nsw,v) \wedge Diff(nsw,sa) \wedge Diff(v,sa) \Rightarrow Colorable()$$

Diff(Red,Blue) Diff(Red,Green) Diff(Green,Red)
Diff(Green,Blue) Diff(Blue,Red) Diff(Blue,Green)

( ) Colorable در صورتی که CSP راه حلی برای CSP ها ی شامل 3SAT در مورد خاص داشته باشد استنتاج می شود ، با وجودی که تطبیق به صورت NP-hard می باشد .

زنجیره ی معکوس

- ایده ی اساسی : به طرف معکوس ، از هدف به واقعیاتی که باید برای هدف نشان داده شوند حرکت می کند
- از روش برجستگی ها به صورت معکوس برای تمرکز دادن جستجو روی پیدا کردن شرط هایی که می توانند منجر به هدف شوند استفاده می کند .
- همچنین از شرط های نامحدود استفاده می نماید .
- جستجو به روش اول – عمق جلو می رود

الگوریتم زنجیره ی معکوس

تابع FOL-BC-Ask(KB,goals,θ) مجموعه ای از تعویض ها را بر می گرداند
ورودی ها : KB ، که یک پایگاه دانش می باشد
goals ، لیستی از پیوندهای شکل دهنده ی یک صف (θ قبلا به کار گرفته شده است)
θ ، تعویض جاری ، به صورت اولیه ، تعویض خالی می باشد { }
متغیرهای محلی : answers ، که مجموعه ای از تعویض ها می باشد ، و به صورت اولیه تهی ({ }) می باشد .

---

[1] deductive databases
[2] en.wikipedia.org/wiki/Deductive_database
[3] hard matching

در صورتی که goals تهی است مجموعه ی {θ} را برگردان

$q' \leftarrow Subst(\theta, First(goals))$

برای هر عبارت r موجود در KB

در جایی که $S\tan dardize - Apart(r) = (p_1 \wedge ... \wedge p_n \Rightarrow q)$ و $\theta' \leftarrow Unify(q, q')$ برقرار می باشد .

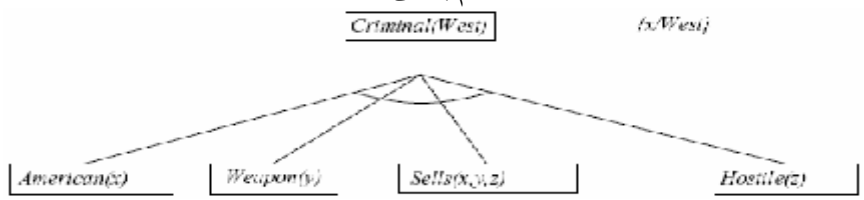
$new - goals \leftarrow [p_1, ..., p_n \mid \text{Re}\, st(goals)]$

$answers \leftarrow FOL - BC - Ask(KB, new - goals, Compose(\theta', \theta)) \cup answers$

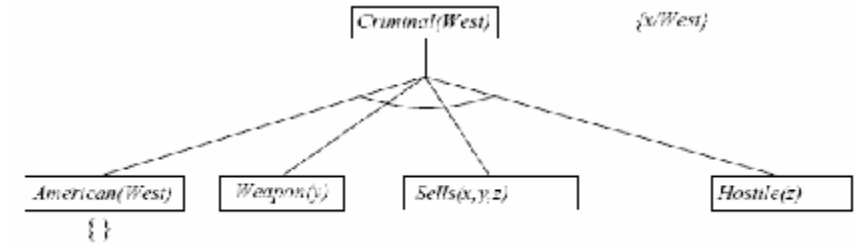
answers را برگردان

مثال زنجیره ی معکوس

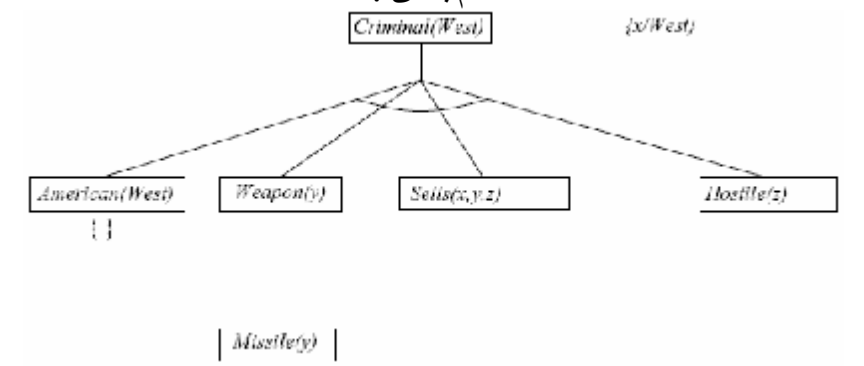
گام بعدی :



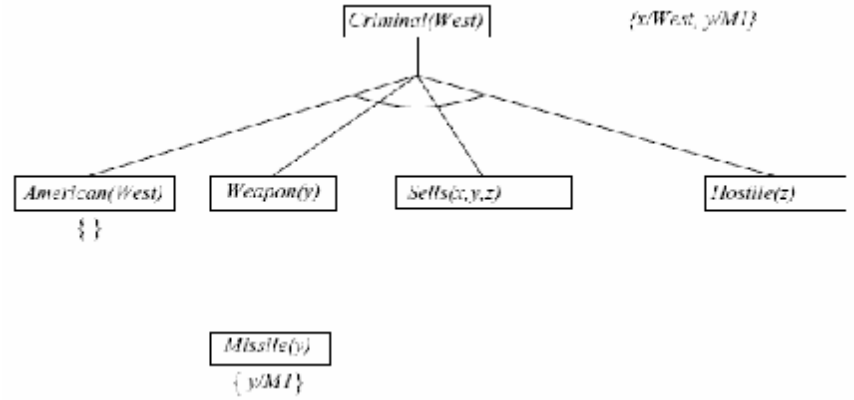
گام بعدی :



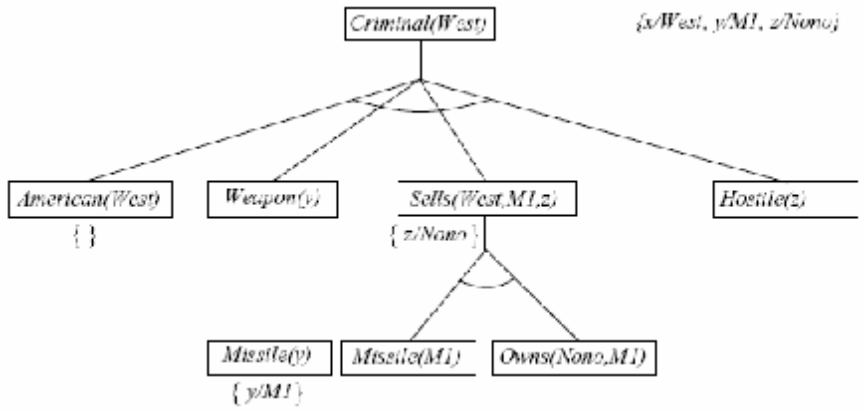
گام بعدی :



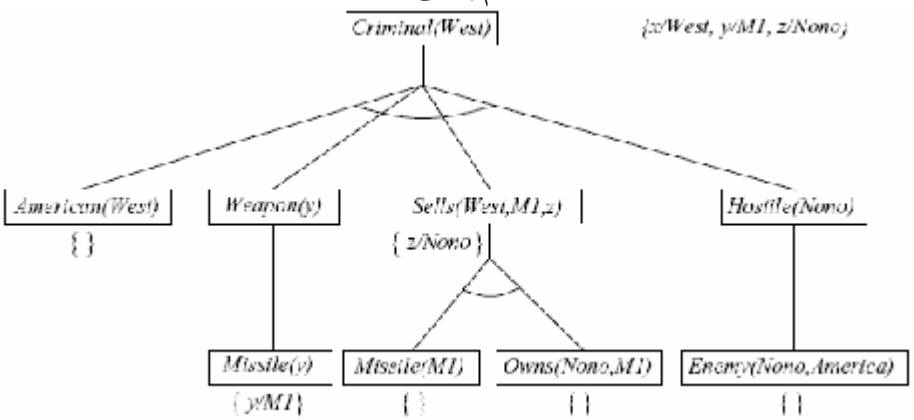
گام بعدی :



گام بعدی :



گام بعدی :



خصوصیات زنجیره ی معکوس

- زنجیره ی معکوس از روش اول – عمق استفاده می کند
- این به این معنی است که از وضعیت های تکرار شده رنج می برد .
- همچنین این روش کامل نمی باشد
- می تواند برای سیستم های برمبنای پرس و جو [1] خیلی موثر باشد

این روش برای برنامه نویسی منطقی [1] به طور گسترده ای استفاده می شود .

[1] query – based systems

برنامه نویسی منطقی

	برنامه نویسی منطقی	برنامه نویسی معمولی
۱	تعریف کردن مساله	تعریف کردن مساله
۲	سر هم بندی کردن اطلاعات	سر هم بندی کردن اطلاعات
۳	شکستن	پیدا کردن راه حل
۴	رمز گشایی اطلاعات درون KB	راه حل برنامه
۵	رمزگشایی اجزای مساله به صورت واقعیات	رمزگشایی اجزای مساله به صورت داده ها
۶	به وجود آوردن پرس و جوها	به کار بردن برنامه برای داده ها
۷	پیدا نمودن واقعیات غلط	برطرف کردن خطاها به صورت رویه ای

سیستم های پرولوگ[۲] - اساس : زنجیره ی معکوس با عبارت های شیپوری + زنگ ها و سوت ها [۳] . سیستم های پرولوگ ، به صورت گسترده ای در اروپا و ژاپن استفاده می شود (اساس پروژه ی نسل پنجم می باشد) .

برنامه = مجموعه ای از شرط ها $head : -literal_1,...literal_n.$

برنامه = مجموعه ای از شرایط = محدوده ی - - محدوده ی واقعی ۱ ($-literal_1$) ..... محدوده ی واقعی n ($literal_n$)

criminal(X):- american(X),weapon(Y),sells(X,Y,Z),hostile(Z)

در پرولوگ جستجوی اول - عمق از چپ به راست زنجیره ی معکوس می باشد .

مثال های پرولوگ

- جستجوی اول - عمق از یک حالت شروع X :

```
dfs(X) :- goal(X) .
dfs(X): - successor(X,S),dfs(S).
```

نیازی به حلقه روی S وجود ندارد : successor برای همه موفق است .

اضافه نمودن دو لیست برای تولید کردن سومی :

```
append([],Y,Y).
append([X|L],Y,[X|Z]) : - append(L,Y,Z).
```

سوال : ? append(A,B,[1,2]) - جواب ها :

A=[] B=[1,2]
A=[1] B=[2]
A=[1,2] B=[]

تحلیل : خلاصه

- تحلیل را در منطق گزاره ای به یاد بیاورید :

---

[1] logic programming
[2] زبان برنامه نویسی سطح بالایی که از عملیات منطقی برای هوش مصنوعی و برنامه های بازیابی داده استفاده می کند
[3] bells and whistles

- $(A \vee C) \wedge (\neg A \vee B) \Rightarrow (B \vee C)$
- تحلیل در منطق مرتبه ی اول به طریقی مشابه عمل می کند .
- لازم است که عبارات به فرم CNF باشند .

- نوع مرتبه ی اول کامل :

$$\frac{\lambda_1 \vee ... \vee \lambda_k, m_1 \vee ... \vee m_n}{(\lambda_1 \vee ... \vee \lambda_{i-1} \vee \lambda_{i+1} \vee ... \vee \lambda_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)\theta}$$

در جایی که : $Unify(\lambda_i, \neg m_j) = \theta$

برای مثال ،

$\neg Rich(x) \vee Unhappy(x)$

Rich(Ken)

Unhappy(Ken)

با $\theta = \{x / Ken\}$ مراحل تحلیل را برای $CNF(KB \wedge \neg\alpha)$ به کار ببرید ؛ برای منطق مرتبه ی اول آن را کامل نمایید .

تبدیل به صورت نرمال CNF - هر کس که همه ی حیوانات را دوست می دارد کسی او را دوست می دارد :

$$\forall x[\forall y Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y Loves(y, x)]$$

۱. استلزام[1] ها را حذف کنید

$$\forall x[\neg\forall y \neg Animal(y) \vee Loves(x, y)] \vee [\exists y Loves(y, x)]$$

۲. $\neg$ را به درون حرکت دهید :

$\neg\forall x, p \equiv \exists x \neg p, \neg\exists x, p \equiv \forall x \neg p$:

$$\forall x[\exists y \neg(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y Loves(y, x)]$$

$$\forall x[\exists y \neg\neg Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y Loves(y, x)]$$

$$\forall x[\exists y Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y Loves(y, x)]$$

۳. متغیرهای استاندارد : هر کمیت ، باید از یک متغیر جدا استفاده نماید ؛

$$\forall x[\exists y Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z Loves(z, x)]$$

۴. عبارت های وجودی را اسکلمایز نمایید

$$\forall x[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

۵. سورهای عمومی را رها نمایید :

$$[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

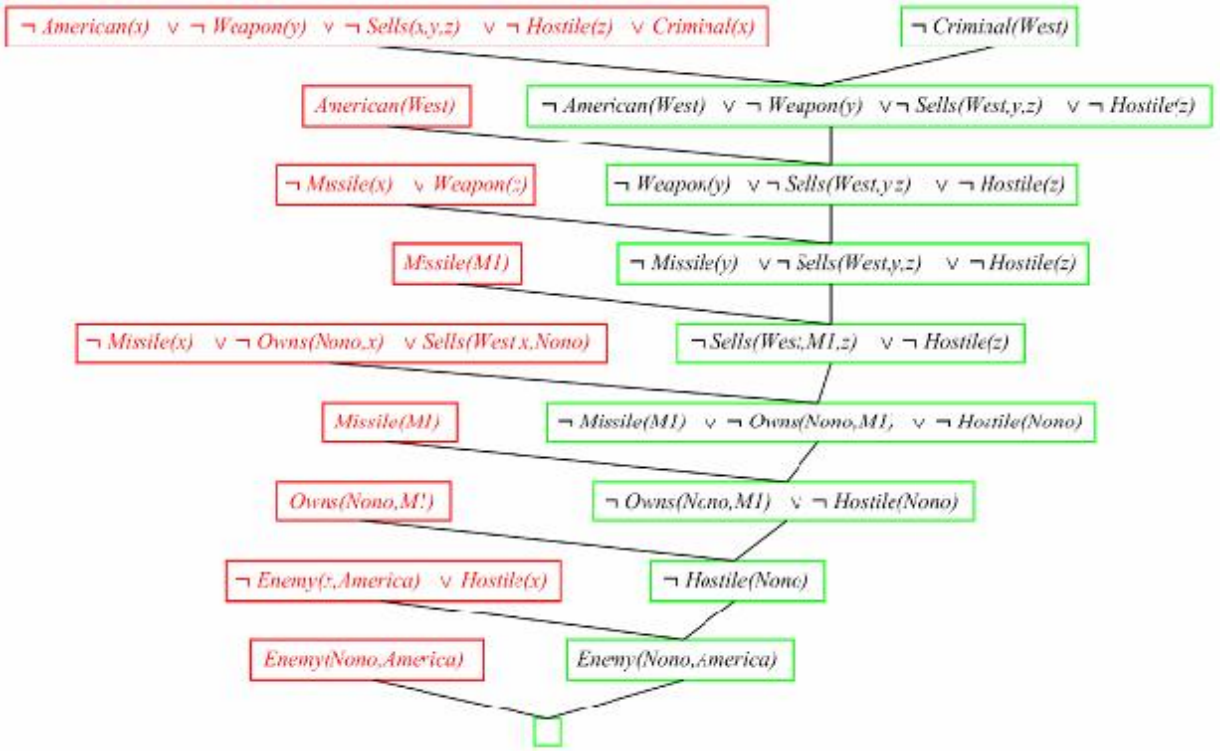
۶. $\wedge$ را روی $\vee$ توزیع نمایید :

$$[Animal(F(x)) \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)]$$

اثبات تحلیل : شرط های نامحدود

---

[1] implication
در کامپیوتر ، عملی منطقی با ساختار
IF-THEN
اگر A,B درست باشند ، آن گاه تابع AND این دو هم درست است



زنجیره ی مستقیم کارآمد :

- به یاد بیاورید که زنجیره ی مستقیم پایه تلاش می کند که هر قانون را با واقعیات داده شده در هر تکرار منطبق نماید .
  - این کار تحت عنوان " واقعیات پیداکننده ی قوانین [1] " شناخته می شود .
- در عمل ، این خیلی کارآمد نمی باشد
  - پایگاه دانش به طور موثری میان تکرارها تغییر نمی کند .
  - تعداد کمی از واقعیات در هر مرحله نمایش داده می شوند
- همچنین ، تعدادی از قوانین دارای جوانب چپی [2] هستند

شبکه ی رت [3]

- رت تطبیق های جزیی گذشته را نگهداری می نماید و این اطلاعات را در سرتاسر تکرارها نگهداری می کند .
- فقط واقعیات جدید در برابر اثر چپی یک قانون چک می شوند
  - " قوانین پیداکننده ی واقعیات "
- کامپایلرهای رت ، یک مجموعه از قوانین درون یک شبکه ، در جایی که گره ها در شبکه تست ها یا وضعیت های در یک طرف چپی را ارایه می کنند هستند .
- به صورتی که واقعیات ، این گره ها را منطبق می کنند و عملیات در شبکه منتشر می شود .

---

[1] rules finding facts
[2] left – hand sides
[3] rete

مثال شبکه ی رت : با فرض قوانین زیر ، شبکه به چه صورتی خواهد بود ؟

```
(defrule drink-beer
(thirsty homer)
=>
(assert (drink-beer homer)))
(defrule go-to-moes
(thirsty homer)
(at homer ~moes)
=>
(assert (at homer moes)))
(defrule happy-homer
(drink-beer homer)
(at homer moes)
(friday)
=>
(assert (happy homer)))
```

حل :

- گره های ورودی تکی یک واقعیت را دریافت می کنند ، آن را تست می کنند و منتشر می کنند .
- دو گروه از گره های ورودی و واقعیت های یگانه ، هر والد را منطبق می کنند .
- ما همچنین می توانیم گره ها را میان قوانین به اشتراک بگذاریم و کارایی را افزایش دهیم
- (به کامپایل نگاه کنید) در جس ساختار شبکه ی تولید شده توسط یک قانون نشان داده شده است .
  - +1+1+1+1+1+2+2+t ، شش گره ی تک ورودی و دو گره دو ورودی جدید را به اضافه ی یک گره ی پایانی جدید را نشان می دهد
  - =1=1=1=1+2+t ، چهار گره ی مشترک تک ورودی را به اضافه ی یک گره دو ورودی جدید و یک گره پایانی را نشان می دهد .

سيستم هاي خبره ي عملي

- شبكه ي رت الهام بخش تمام يك نسل از سيستم های خبره می باشد
  - یکی از موفق ترین روش های هوش مصنوعی می باشد .
  - با روشی که خبره های بشر مسایل را توضیح می دهند تطبیق می کند .
- این مطلب را امکان پذیر می سازند که سیستم های براساس قانون [1] را با مقیاس بزرگ [2] بسازیم .
  - پیدا کننده های ته نشین های روغن [3] ، کنترل کننده های کارخانه ها [1] ، تشخیص دهنده های بیماری [2] ، رفع عیب کننده ی شبکه ها [3] و ... .

---

[1] rule – based systems
[2] large - scale
[3] locating oil deposits

- همچنین ، بیش تر سیستم های خبره دارای یک مفهوم عدم قطعیت [4] هستند

منابع :

http://www.cs.usfca.edu/~brooks/F03classes/ai/lectures/inference.pdf
http://aima.eecs.berkeley.edu/slides-pdf/chapter09.pdf

[1] controlling factories
[2] diagnosing illness
[3] troubleshooting networks
[4] uncertainty

فصل چهاردهم

عدم قطعیت

عدم قطعیت [1]
فصل چهاردهم

ریوس مطالب

- **عدم قطعیت**
- **احتمال [2]**
- **ظاهر و معناها**
- **استدلال (استنتاج)**
- **استقلال [3] و قانون بیز [4]**

عدم قطعیت

- در تعدادی از محیط های جالب عامل ها ، عدم قطعیت یک نقش مرکزی را ایفا می کند .
- عملکردها ممکن است دارای اثرات غیر قطعی باشند
  - پرتاب یک پیکان به هدف ، بازیابی یک صفحه ی وب
- عامل ها ممکن است وضعیت درست جهان را ندانند .
- ارتباطات میان واقعیت ها ممکن است به صورت قطعی نباشند .
  - گاهی اوقات در زمانی که هوا ابری می باشد باران می بارد .
- عامل های عقلانی نیازمند رسیدگی کردن به عدم قطعیت می باشند .

منطق و عدم قطعیت

- ما به زودی خواهیم دید که چگونه منطق را با عدم قطعیت به کار ببریم
  - $Studies(Bart) \vee WatchesTV(Bart)$
  - $Hungry(Homer) \Rightarrow Eats(Homer, HotDog) \vee Eats(Homer, Pie)$
  - $\exists x : Hungry(x)$
- متاسفانه ، نگرش منطقی دارای برخی اشکالات می باشد .

---

[1] uncertainty
[2] Probability
[3] Independence
[4] Bayes' Rule

ضعف های منطق [1]

- تایین همه ی نتیجه های ممکن .
  - " اگر الان ترک کنم به موقع آن جا خواهم بود ، در غیر این صورت زمین لرزه خواهد آمد ، من گاز را تمام خواهم کرد ، یا یک تصادف خواهد شد ... "
- ما ممکن است همه ی نتیجه های ممکن را ندانیم .
  - " در صورتی که یک بیمار دارای دندان درد باشد ، وی ممکن است یک حفره [2] در دندان خود داشته باشد ، یا ممکن است دارای بیماری لثه [3] باشد ، یا ممکن است دلیلی دیگر داشته باشد که ما چیزی در مورد آن نمی دانیم . "
- ما هیچ راهی برای صحبت در مورد احتمال حوادث نداریم .
  - " امروز ممکن است که من با روشنایی برخورد کنم . "

کمیت و کیفیت

- منطق ، یک نگرش کمی به عدم قطعیت را ارایه می دهد
  - ما می توانیم بگوییم که یک رخداد ، عمومی تر از دیگری است ، یا این که چیزی ممکن است .
  - در مواردی که ما آماری نداریم و یا این که می خواهیم به صورت مجرد تر بحث کنیم ، مفید می باشد .
- احتمال به ما اجازه می دهد که به صورت کیفی بحث کنیم
  - ما مقادیر محسوس را به شانس رخدادن یک رویداد انتساب می دهیم و مقادیر محسوس جدید را براساس مشاهدات می گیریم .

عدم قطعیت و عقلانیت

- تعریف ما از عقلانیت را به یاد بیاورید :
  - یک عامل عقلانی ، عاملی است که با بیشینه نمودن معیارکارایی عمل می نماید .
- ما چگونه این عامل را در یک جهان با عدم قطعیت تعریف نماییم ؟
- ما خواهیم گفت که یک عامل دارای یک سود برای نتیجه های مختلف می باشد و نتیجه ها دارای یک احتمال رویداد هستند .
- سپس یک عامل می تواند به هر کدام از نتایج ممکن و سود آن ها و احتمال رخدادن نتیجه توجه نماید و عملی را که دارای بالاترین سود مورد انتظار می باشد را انتخاب نماید .
- تیوری ترکیب کننده ی برتری ها پیرامون نتیجه ها با احتما ل رخداد یک نتیجه ، *تیوری تصمیم گیری* [4] نام دارد .

مثال : بیایید عملکرد A_t را ترک کردن فرودگاه ، t دقیقه قبل از پرواز قرار دهیم . آیا در زمان A_t به موقع آن جا خواهم بود ؟

---

[1] weaknesses with logic
[2] cavity
[3] gum
[4] decision theory

مسایل :

۱) مشاهده پذیری جزیی [۱] (حالت جاده ، خلبانان دیگر هواپیما ها و .....)
۲) حس گرهای پر خش [۲](اغتشاش) (گزارش های ترافیک KCBS)
۳) عدم قطعیت در نتایج عملکرد [۳] (تایر مسطح و .....)
۴) پیچیدگی بیش از حد [۴] طرح ریزی [۵] و پیش بینی ترافیک [۶]

بنابراین یک شیوه ی منطقی یا با

۱) ریسک های دروغ [۷] است مثل : " با A_{25} به موقع خواهم رسید "
یا
۲) منتهی می شود به نتایجی که خیلی برای تصمیم گیری ضعیفند :

" A_{25} در صورتی مرا به موقع خواهد رساند که تصادفی در راه رخ ندهد و باران نبارد و لاستیک اتومبیل من بی عیب باشد و ..... " (A_{1440} معقولانه است که مرا به موقع برساند اما من باید در طول شب در فرودگاه بمانم .....)

متدهایی برای به کارگیری عدم قطعیت

منطق پیش فرض یا غیر یکنواخت [۸] : فرض کنید که اتومبیل من دارای تایر صاف نباشد فرض کنید که A_{25} کار می کند مگر این که با تناقض [۹] ثابت شود که کار نمی کند

.

برآمدها : چه مفروضاتی معقولانه می باشد ؟ چگونه تناقض را به کار ببریم ؟ قوانینی برای طفره رفتن از عوامل (فاکتورها) :

$$A_{25} \ \alpha_{0.3} \ AtAirportOnTime$$
$$Sprinkler \ \alpha_{0.99} \ WetGrass$$
$$WetGrass \ \alpha_{0.7} \ Rain$$

برآمدها : مسایل با ترکیب ، آیا **آبپاش** [۱۰] سبب **بارندگی** می شود ؟؟

اساس احتمال

- یک احتمال بر یک تصور که یک طرح یا گزاره [۱۱] ، درست است دلالت می کند
  - P(BartStudied) = 0.01
  - P(Hungry(Homer)) = 0.99
- خود گزاره ممکن است درست یا غلط باشد
- متفاوت است که بکوییم عبارت به صورت جزیی درست می باشد .

---

[1] partial observability
[2] noisy sensors
[3] uncertainty in action outcomes
[4] immense
[5] modelling
[6] predicting traffic
[7] risks falsehood
[8] nonmonotonic
[9] contradiction
[10] sprinkler
[11] proposition

- o " بارت [1] کوتاه [2] است " – این عبارت از نوع درست است ، چون که " کوتاه " ، واژه ای مبهم [3] می باشد .
- تصور وضعیت یک عامل ، یک نمایش از احتمال ارزش هر گزاره جالب است .

مقادیر تصادفی [4]

- یک متغیر تصادفی ، متغیر یا گزاره ای است که مقدارش ناشناخته است .
- دارای دامنه ای از مقادیر است که می تواند آن ها را به خود بگیرد .
- این متغیرها می توانند از موارد زیر باشند :
  - o بولین (درست و غلط) – مثل : Hungry(Homer) و isRaining
  - o گسسته – مقادیر از یک دامنه ی قابل شمارش گرفته می شوند .
    - دما [5] : < گرم [6] ، خنک [7] ، ملایم [8] > و پیش بینی وضع هوا : < آفتابی [9] ، ابری [10] ، بارانی >
  - o پیوسته – مقادیر ، می توانند از یک فاصله گرفته شوند ، نظیر : [۱و۰]
    - سرعت [11] ، زمان ، مکان [12]
- بیش تر تمرکز ما بر روی مورد گسسته خواهد بود .

پس ، یک متغیر تصادفی ، یک تابع از فضای نمونه برای برای بعضی از محدوده ها می باشد ، مثال : real ها یا Boolean ها . مثال : Odd(1)=true

رویدادها (رخدادها) ی اتمیک [13]

- ما می توانیم گزاره ها را با استفاده از رابط های منطقی استاندارد و به کارگیری اجتماع و اشتراک با هم ترکیب نماییم
  - o $P(Hungry(Homer) \wedge \neg Study(Bart)$
  - o $P(Brother(Lisa, Bart) \vee Sister(Lisa, Bart)$
- یک عبارت که یک مقدار ممکن را برای هر متغیر نامعین مشخص می کند یک *رویداد اتمیک* [14] نام دارد .
  - o رویدادهای اتمیک دو به دو ناسازگار (مانعه الجمعی) [15] هستند .
  - o مجموعه ای از همه ی رویدادهای اتمیک شامل تمام جزییات [16] هستند .
  - o یک رویداد اتمیک ، درستی یا نادرستی هر گزاره را پیش بینی می کند .

---

[1] Bart
[2] short
[3] vague
[4] random values
[5] temperature
[6] hot
[7] cool
[8] mild
[9] sunny
[10] overcast
[11] velocity
[12] position
[13] Atomic Events
[14] atomic event
[15] mutually exclusive
[16] exhaustive

- رویدادهای اتمیک در تشخیص درستی موارد با چند متغیر نامعین ، مفید می باشند .

نکته : منطق فازی [1] از درجه ی قطعیت (درجه ی درستی) [2] استفاده می کند نه عدم قطعیت [3] .
احتمال ـ خلاصه ی اثرات تاکید شده ی احتمال ،
تنبلی [4] : ناکارامد بودن برای شمارش استثنایات ، کمیت سنج ها و ..... .
نادانی [5] : نبود واقعیت های مربوط ، شرایط اولیه و .....

احتمال پیشین [6] ـ احتمال های گذشته ، یا غیرشرطی گزاره ها :
مثال : P(Cavity=true)=0.1 و P(Weather=sunny)=0.72 . برابر است با تصور قدیمی برای ورود هر مدرک جدید . توزیع احتمال ، مقادیر را برای همه ی انتساب های ممکن ، ارایه می دهد :

$$P(Weather) = \langle 0.72, 0.1, 0.08, 0.1 \rangle$$

توجه نمایید که مجموع اعداد برابر یک می باشد (۰,۷۲+۰,۱+۰,۰۸+۰,۱=۱,۰۰) . توزیع پیوسته ی احتمال برای مجموعه ای از متغیرهای تصادفی ، احتمال هر رویداد اتمیک را در متغیرهای تصادفی ارایه می کند . P(Weather,Cavity) برابر است با یک ماتریس ۴*۲ از مقادیر

Weather=	sunny	rain	cloudy	snow
Cavity=true	0.144	0.02	0.016	0.02
Cavity=false	0.576	0.08	0.064	0.08

هر پرسش درباره ی یک دامنه ، می تواند به وسیله ی توزیع پیوسته ، پاسخ داده شود ، زیرا هر رویداد ، مجموعه ای از فضای نمونه می باشد .

- احتمال پیشین یک گزاره ، احتمال یک مقدار در غیاب هر اطلاعات دیگری می باشد .
  - P(Study)=0.5 ، P(Overcast)=0.4 ، P(Rain)=0.1
- ما همچنین می توانیم احتمالات ترکیب های متغیرها را لیست نماییم
  - $P(Rain \wedge Humid) = 0.1$ ، $P(Rain \wedge \neg Humid) = 0.1$ ،
    $P(Overcast \wedge Humid) = 0.2$ ، $P(Overcast \wedge \neg Humid) = 0.2$ ،
    $P(Sunny \wedge Humid) = 0.15$ ، $P(Sunny \wedge \neg Humid) = 0.25$
- این ، یک *توزیع پیوسته ی احتمال* [7] نام دارد
- برای متغیرهای پیوسته ، ما نمی توانیم مقادیر را بشماریم
- در عوض ، ما از یک تابع پارامتری استفاده می نماییم .
  - $P(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{-x^2}{2}} dz$ (توزیع نرمال [1])

---

[1] Fuzzy logic
[2] degree of truth
[3] uncertainty
[4] laziness
[5] ignorance
[6] prior probability
[7] joint probability distribution

تصمیم گیری تحت شرایط عدم قطعیت - تصور نمایید موارد زیر را داریم :

$P(A_{25}|$ =(مرا به موقع برساند ۰,۰۴ $P(A_{90}|$ =(مرا به موقع برساند ۰,۷۴ $P(A_{120}|$ =(مرا به موقع برساند ۰,۹۵

$P(A_{1440}|$ =(مرا به موقع برساند ۰,۹۹۹

کدام را باید انتخاب نماییم ؟ وابسته به صلاح دید های من ، برای از دست ندادن پرواز ، وضعیت فرودگاه و ..... می باشد .

قضیه ی سود [2] ، برای ارایه و استدلال صلاح دیدها استفاده می شود .

قضیه ی تصمیم گیری [3] = قضیه ی سود + قضیه ی احتمال

مبانی احتمال - با یک مجموعه از فضای ساده ی Ω شروع می کنیم ؛ مثال ، شش حالت ممکن یک طاس . اگر $\omega \in \Omega$ می باشد .

$$0 \le P(\omega) \le 1$$

$$\sum_{\omega} P(\omega) = 1$$

مثال : P(1)=P(2)=P(3)=P(4)=P(5)=P(6)=1/6

یک رویداد A زیرمجموعه ای از Ω می باشد

$$P(A) = \sum_{\{\omega \in A\}} P(\omega)$$

(احتمال این که نقش طاس کم تر از ۴ باشد = احتمال ۱ آمدن + احتمال ۲ آمدن + احتمال ۳ آمدن = 1/6+1/6+1/6 = ۱/۲)

استنتاج با توزیع های پیوسته ی احتمال

- آسان ترین راه انجام استنتاج احتمالی این است که یک جدول ارایه کننده ی توزیع پیوسته ی احتمال را نگهداری نماییم .
- به هر متغیر مستقل نگاه کنید و احتمال وابسته را پیدا نمایید .

	Humidity=High Outlook=Overcast	Humidity=High Outlook=Sunny	Humidity=Normal Outlook=Overcast	Humidity=Normal Outlook=Sunny
Rain	0.1	0.05	0.15	0.05
$\neg Rain$	0.2	0.15	0.1	0.2

- ما همچنین می توانیم توزیع پیوسته ی احتمال را برای تشخیص *احتمال نهایی* [4] متغیر وابسته ، با جمع کردن همه ی راه های وابسته به متغیر که می توانند درست باشند استفاده نماییم .
  - P(Rain)=0.1+0.05+0.15+0.05=0.35
- مساله ای که با استفاده از توزیع پیوسته ی احتمال برای استنتاج وجود دارد چیست ؟

---

[1] normal distribution
[2] Utility theory
[3] Decision theory
[4] marginal probability

P ، یک **توزیع تصادفی** را برای هر متغیر تصادفی نظیر **X** ، القا می نماید :

$$P(X = x_i) = \sum_{\{\omega:X(\omega)=x_i\}} P(\omega)$$

مثال : P(Odd=true)=P(1)+P(3)+P(5)=1/6+1/6+1/6=1/2

گزاره ها - یک گزاره را به صورت رویداد (مجموعه ای از فضاهای نمونه) در جایی که گزاره صحیح می باشد ، تصور نمایید . متغیر های تصادفی بولین A و B داده شده اند : رویـــــــداد a = مجموعه ای از فضاهای نمونه ای به شرط آن که :

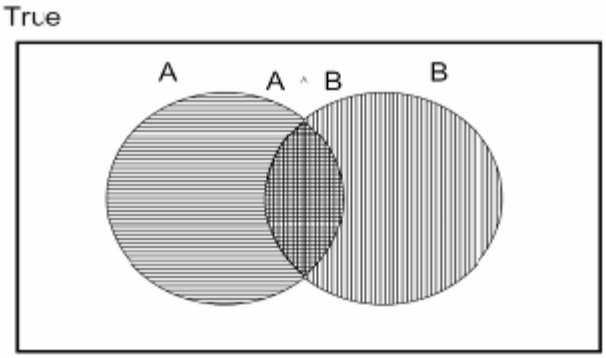
$A(\omega)=true$

رویداد $\neg a$ = مجموعه ای از فضاهای نمونه ای به شرط آن که $A(\omega)=false$

رویداد $a \wedge b$ = جاهایی که $A(\omega)=true$ و $B(\omega)=true$ می باشند .

اغلب در موارد هوش مصنوعی ، فضای نمونه ای به وسیله ی مقادیر یک مجموعه از متغیر های تصادفی **تعریف می شود** . توجه نمایید که فضای نمونه ای حاصلضرب کارتزین محدوده هایی از متغیر ها می باشد . با متغیر های بولین ، فضای نمونه = مدل منطق گزاره ای می باشد . مثال ، A = true ، B = false یا $a \wedge \neg b$. گزاره = اشتراک رویدادهای اتمیک که در آن رویداد صحیح می باشد . مثال :

$$(a \vee b) \equiv (\neg a \wedge b) \vee (a \wedge \neg b) \vee (a \wedge b) \Rightarrow P(a \vee b) = P(\neg a \wedge b) + P(a \wedge \neg b) + P(a \wedge b)$$



چرا از احتمال استفاده می کنیم ؟ تعریف ، بر این دلالت می کند که رویدادهای منطقی وابسته ی معین باید روابط احتمالاتی داشته باشند .

مثال : $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$

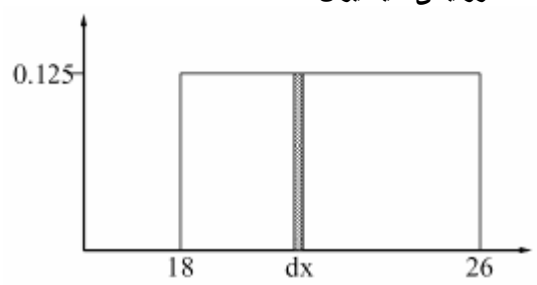
ظاهر گزاره ها - متغیر های تصادفی گزاره ای یا بولین . Cavity (آیا من کرم خوردگی دندان دارم ؟) . Cavity = true یک گزاره می باشد . متغیر های تصادفی گسسته (محدود یا نامحدود)

مثال : هوا [1] دارای یکی از حالت های آفتابی ، بارانی ، ابری و برفی [2] می باشد . هوا = بارانی ، یک گزاره می باشد . متغیر های تصادفی پیوسته [3] محدوده دار [4] یا فاقد محدوده [5] می باشند . مثال : دما = ۲۱,۶ ، همچنین Temp<22.0 هم مجاز می باشد.

احتمال برای متغیرهای پیوسته - بیان توزیع به صورت یک تابع پارامتری از مقدار :

---

[1] Weather
[2] snow
[3] Continuous random variables
[4] bounded
[5] unbounded

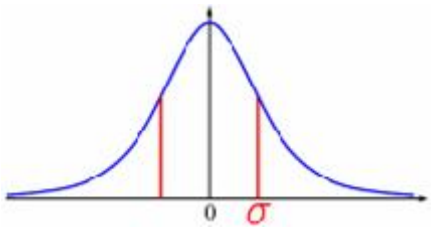


P(X=x)=U[18,26](x) = تراکم یک ریخت
مابین ۱۸ و ۲۶
P در این جا یک تراکم را نشان می دهد ؛ معنی واقعی P(X=20.5) = 0.125 به صورت زیر است :

$$\lim_{dx\to 0} P(20.5 \le X \le 20.5 + dx)/dx = 0.125$$

اتحاد گاوس [1]

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$



احتمال شرطی [2]

- سابقا ما با مشاهداتی درباره ی مقدار متغیرهای معین شروع کردیم . ما قبلا دیدیم که اگر هوا ابری باشد ، احتمال بارندگی بالا می رود .
- این یک احتمال شرطی نام دارد
- و به صورت P(Rain|Cloudy) نوشته می شود
- برای توزیع های شرطی توجه نمایید : اگر P(Cavity|Toothache) مورد نظر باشد و ما بیش تر بدانیم ، مثلا cavity ارایه شده باشد ، در این صورت داریم :
  P(cavity|toothache,cavity)=1
- تعریف احتمال شرطی :

$$P(a \mid b) = \frac{P(a \wedge b)}{P(b)}$$ ، با شرط $P(b) \neq 0$

- قانون ضرب [3] یک فرمول دیگر را ارایه می نماید :

$$P(a \wedge b) = P(a \mid b)P(b) = P(b \mid a)P(a)$$

در نوع کلی همه ی توزیع ها نگه داشته می شود ، مثال :
P(Weather,Cavity)=P(Weather|Cavity)P(Cavity)

---

[1] Gaussian density
[2] conditional probability
[3] Product rule

(به صورت یک مجموعه از تساوی های ۴*۲ آن را نگاه کنید ، نه به صورت ماتریس ضرب) .

مثال :

- P(Cloudy)=0.3
- P(Rain)=0.2
- $P(Cloudy \wedge Rain) = 0.15$
- $P(Cloudy \wedge \neg Rain) = 0.1$
- $P(\neg Cloudy \wedge Rain) = 0.1$
- $P(\neg Cloudy \wedge \neg Rain) = 0.65$
  - به صورت اولیه P(Rain)=0.2 .

$$P(Rain \mid Cloudy) = \frac{P(Rain \wedge Cloudy)}{P(Cloudy)} = \frac{0.15}{0.3} = 0.5$$

قانون زنجیره ای [1] - برای کاربرد موفقیت آمیز قانون ضرب به وجود آمده است :

$$P(X_1,...,X_n) = P(X_1,...,X_{n-1})P(X_n \mid X_1,...,X_{n-1})$$
$$= P(X_1,...,X_{n-2})P(X_{n_1} \mid X_1,...,X_{n-2})P(X_n \mid X_1,...,X_{n-1})$$
$$= ... = \prod_{i=1}^{n} P(X_i \mid X_1,...,X_{i-1})$$

استدلال با شمارش - با توزیع پیوسته شروع نمایید :

	دندان درد نگرفتن ($\neg toothache$)		دندان درد (toothache)	
	نگرفتن ($\neg catch$)	گرفتن (catch)	نگرفتن ($\neg catch$)	گرفتن (catch)
حفره ی دندان (cavity)	۰,۰۰۸	۰,۰۷۲	۰,۱۲	۰,۱۰۸
عدم حفره ی دندان ($\neg cavity$)	۰,۵۷۶	۰,۱۴۴	۰,۰۶۴	۰,۰۱۶

برای هر گزاره ی تهی (Ø) ، مجموع رویدادهای اتمیک در موارد صحیح برابر است با :

$$P(\phi) = \sum_{\omega:\omega|=\phi} P(\omega)$$

گام بعدی :

	دندان درد نگرفتن ($\neg toothache$)		دندان درد (toothache)	
	نگرفتن ($\neg catch$)	گرفتن (catch)	نگرفتن ($\neg catch$)	گرفتن (catch)

[1] Chain rule

پوسیدگی دندان (cavity)	۰,۱۰۸	۰,۱۲	۰,۰۷۲	۰,۰۰۸
عدم پوسیدگی دندان ($\neg cavity$)	۰,۰۱۶	۰,۰۶۴	۰,۱۴۴	۰,۵۷۶

$$P(toothache) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2$$

$$P(cavity \vee toothache) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$$

گام بعدی :

	دندان درد (toothache)		دندان درد نگرفتن ($\neg toothache$)	
	گرفتن (catch)	نگرفتن ($\neg catch$)	گرفتن (catch)	نگرفتن ($\neg catch$)
پوسیدگی دندان (cavity)	۰,۱۰۸	۰,۱۲	۰,۰۷۲	۰,۰۰۸
عدم پوسیدگی دندان ($\neg cavity$)	۰,۰۱۶	۰,۰۶	۰,۱۴۴	۰,۵۷۶

می توانیم همچنین احتمالات شرطی را محاسبه نماییم :

$$P(\neg cavity \mid toothache) = \frac{P(\neg cavity \wedge toothache)}{P(toothache)} = \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

نرمال سازی

	دندان درد (toothache)		دندان درد نگرفتن ($\neg toothache$)	
	گرفتن (catch)	نگرفتن ($\neg catch$)	گرفتن (catch)	نگرفتن ($\neg catch$)
پوسیدگی دندان (cavity)	۰,۱۰۸	۰,۱۲	۰,۰۷۲	۰,۰۰۸
عدم پوسیدگی دندان ($\neg cavity$)	۰,۰۱۶	۰,۰۶۴	۰,۱۴۴	۰,۵۷۶

مقسوم علیه [1] می تواند به صورت یک ثابت نرمال سازی α دیده شود .

$$P(Cavity \mid toothache) = \alpha P(Cavity, toothache)$$
$$= \alpha[P(Cavity, toothache, catch) + P(Cavity, toothache, \neg catch)]$$
$$= \alpha[\langle 0.108, 0.016\rangle + \langle 0.012, 0.064\rangle]$$
$$= \alpha\langle 0{,}12, 0.08\rangle = \langle 0.6, 0.4\rangle$$

[1] denominator

استدلال با شمارش - تصور نمایید X ، همه ی متغیر ها باشد . معمولا ، ما توزیع پیوسته ی قبلی **متغیرهای پرس و جوی** [1] **Y** را می خواهیم که توسط مقادیر e برای **متغیرهای ثابت** [2] **E** ارایه شده اند . تصور نمایید متغیرهای مخفی H=X-Y-E باشند . در این صورت ، مجموع تمام پیوند ها با جمع کردن متغیرهای مخفی به دست می آید :

$$P(Y \mid E = e) = \alpha P(Y, E = e) = \alpha \sum_h P(Y, E = e, H = h)$$

واژگان درون مجموعه ، تماما متصل می باشند زیرا Y و E و H با هم ، خروجی آن ها ، مجموعه ای از متغیرهای تصادفی می باشد .

مسایل آشکار :

۱) پیچیدگی زمانی در بدترین حالت در جایی که d بزرگ ترین می باشد برابر است با $O(d^n)$

۲) پیچیدگی فضا برای نگهداری توزیع پیوسته برابر است با $O(d^n)$

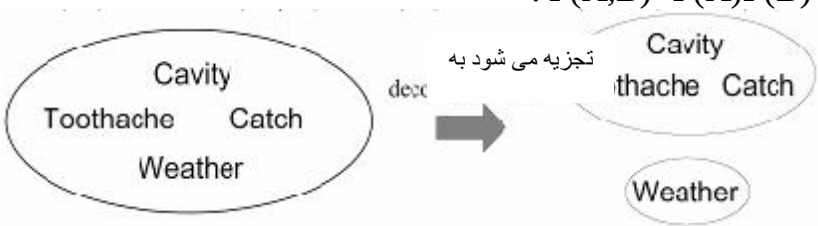
۳) چگونه می توان تعداد را برای ورودی های $O(d^n)$ پیدا نماییم ؟؟؟

استقلال [3]

- در برخی از موارد ، ما می توانیم چیزها را با توجه به این که یک متغیر دارای هیچ اثری بر روی دیگری نمی باشد ساده نماییم .
- برای مثال ، اگر ما یک متغیر چهارم DayOfWeek را به محاسبه ی بارندگی اضافه نماییم چه تغییری به وجود می آید ؟
- از آنجایی که روز هفته بر احتمال باران تاثیری نخواهد داشت ، ما داریم :

P(Rain|Cloudy,Monday)=P(Rain|Cloudy,Tuesday)…=P(Rain|Cloudy)

- ما گفتیم که DayOfWeek و Rain مستقل می باشند .
- ما سپس می توانیم توزیع پیوسته ی احتمال بزرگ تر را به زیر جدول هایی مجزا تقسیم نماییم .
- استقلال به ما کمک خواهد کرد که دامنه را به تکه های مجزا تقسیم نماییم .
- A و B در صورتی مستقل هستند که ، P(A|B)=P(A) یا P(B|A)=P(B) یا P(A,B)=P(A)P(B) .



P(Toothache,Catch,Cavity,Weather) = P(Toothache,Catch,Cavity)P(Weather)

۳۲ ورودی به ۱۲ تا کاهش داده می شود ؛ برای n سکه ی مستقل داریم : $2^n \to n$

استقلال مطلق ، خیلی مفید است ولی نادر یا کمیاب [4] می باشد . مثلا ، تراکم ، فیلد (زمینه ی) بزرگی با صد ها متغیر می باشد که هیچکدام مستقل نیستند . در این مورد چه کاری انجام بدهیم ؟

استقلال شرطی [1]

---

[1] query variables
[2] evidence variables
[3] Independence
[4] rare

- تصور کنید که ما می خواهیم بدانیم بیمار دارای یک حفره در دندان است . متغیرهای مشاهده شده ی ما toothache و catch می باشند .
- این ها به صورت اولیه مستقل نمی باشند – اگر بیمار دارای یک دندان درد باشد ، معمولا وی دارای یک حفره در دندان می باشد ، که احتمال دندان درد گرفتن را افزایش می دهد .
- ما این مطلب را به این صورت می نویسیم :

$$P(toothache \wedge catch \mid cavity) = P(toothache \mid cavity)P(catch \mid cavity)$$

- ما سپس از قضیه ی بیز استفاده می نماییم و این مطلب را به این صورت می نویسیم :

$$P(cavity \mid toothache \wedge catch) = \alpha P(toothache \mid cavity)P(catch \mid cavity)P(cavity)$$

- P(Toothache,Cavity,Catch) دارای $2^3 - 1 = 7$ ورودی مستقل می باشد . اگر من پوسیدگی دندان داشته باشم ، احتمال درد گرفتن به هوا وابستگی ندارد :

۱) P(catch|toothache,cavity)=P(catch|cavity)

مستقل بودن وقتی اتفاق می افتد که من پوسیدگی دندان نداشته باشم :

۲) P(catch|toothache,¬ cavity)=P(catch|¬ cavity)

Catch مستقل شرطی از دندان درد براثر پوسیدگی می باشد :

P(Catch|Toothache,Cavity)=P(Catch|Cavity)

حالت های برابری :

P(Toothache|Catch,Cavity)=P(Toothache|Cavity)

P(Toothache,Catch|Cavity)=P(Toothache|Cavity)P(Catch|Cavity)

گام بعدی- تمام توزیع های اتصال را با استفاده از قانون زنجیری بنویسید :

P(Toothache,Catch,Cavity)

=P(Toothache|Catch,Cavity)P(Catch,Cavity)

=P(Toothache|Cach,Cavity)P(Cach|Cavity)P(Cavity)

=P(Toothache|Cavity)P(Cach|Cavity)P(Cavity)

در بیش تر موارد ، استفاده از استقلال شرطی ، اندازه ی ارایه ی توزیع پیوسته را از حالت نمایی n به حالت خطی n کاهش می دهد.

استقلال شرطی ، پایه ای ترین و نیرومندترین دانش درباره ی محیط های نامعین می باشد .

قانون بیز

با استفاده از قانون ضرب داریم :

$P(a \wedge b) = P(a \mid b)P(b)$ و $P(a \wedge b) = P(b \mid a)P(a)$ در نتیجه ما می توانیم این تساوی ها را برابر هم قرار دهیم :

$P(a \wedge b) = P(a \mid b)P(b) = P(b \mid a)P(a)$ ← قانون بیز :

$$P(a \mid b) = \frac{P(b \mid a)P(a)}{P(b)}$$

یا به شکل توزیع

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} = \alpha P(X \mid Y)P(Y)$$

برای تشخیص احتمال تشخیصی [۲] از احتمال سببی [۱] مفید می باشد :

---

[1] conditional independence
[2] diagnostic probability

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} = \alpha P(X \mid Y)P(Y)$$

تصور نمایید M ، منتژیت [۲] باشد و S خشکی گردن [۳] باشد :

$$P(m \mid s) = \frac{P(s \mid m)P(m)}{P(s)} = \frac{0.8 \times 0.0001}{0.1} = 0.0008$$

نکته : احتمال ابتلا به بیماری منتژیت بسیار کم می باشد !

مثال برای قانون بیز :

- مفروضات زیر را در نظر بگیرید :
  - منتژیت باعث خشکی گردن در ۵۰% از بیماران می شود .
  - P(stiffNeck|Meningities)=0.5
  - احتمال منتژیت قدیمی برابر است با ۱/۵۰۰۰۰
  - P(meningitis)=0.00002
  - احتمال خشکی گردن قدیمی برابر است با ۱/۲۰
  - P(stiffNeck)=0.05
- حال یک بیمار با خشکی گردن مراجعه می نماید . احتمال این که وی دارای منتژیت باشد چقدر است ؟

$$P(meningitis \mid stiffNeck) = \frac{P(stiffNeck \mid meningitis)P(meningitis)}{P(stiffNeck)} = \frac{0.5 \times 0.00002}{0.05} = 0.0002$$

مثالی دیگر :

- فرض کنید جواب آزمایشگاه در مورد یک بیمار می آید و می گوید بیمار دارای سرطان [۴] می باشد . آیا ما باید این مطلب را باور نماییم ؟
- P(cancer)=0.008
- P(positiveTest|cancer)=0.98
- $P(positiveTest \mid \neg cancer) = 0.3$
- ما می خواهیم بدانیم که سرطان داشتن را باید بیش تر بپذیریم یا سرطان نداشتن را ؟
- ما می توانیم از مقسوم علیه برای مدت کوتاهی صرف نظر نماییم .
- P(positive|cancer)P(cancer)=0.98*0.008=0.0078
- $P(positive \mid \neg cancer) = 0.03 \times 0.992 = 0.298$
- ما می بینیم که بهتر است تصور نماییم بیمار دارای سرطان نمی باشد .
- ما می توانیم مقادیر دقیق تر را هم به دست آوریم :
- $$P(cancer \mid positive) = \frac{0.0078}{0.0078 + 0.0298} = 0.21$$

قانون بیز و استقلال شرطی

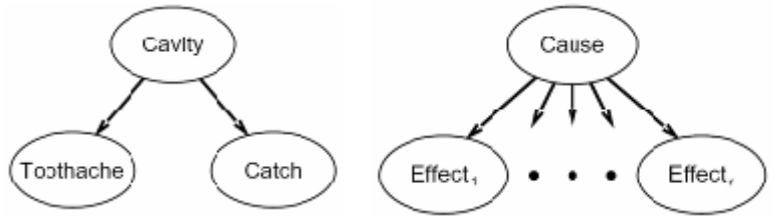
---

[1] causal probability
[2] meningitis
[3] stiff neck
[4] cancer

$$P(Cavity \mid toothache \wedge catch)$$
$$= \alpha P(toothache \wedge catch \mid catch \mid Cavity)P(Cavity)$$
$$= \alpha P(toothache \mid Cavity)P(catch \mid Cavity)P(Cavity)$$

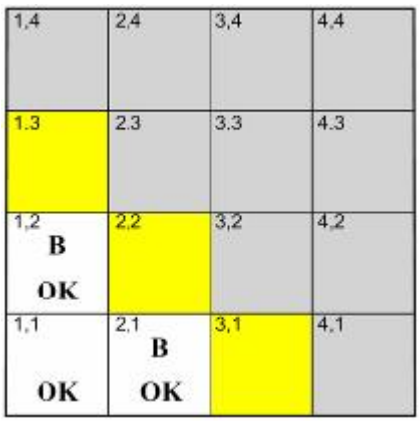
این مثالی ساده از قانون بیز می باشد :

$$P(Cause, Effect_1, ..., Effect_n) = P(Cause)\prod_i P(Effect_i \mid Cause)$$



مجموع تعداد پارامترها به صورت خطی و براساس n می باشد .

دنیای wumpus



$P_{ij} = true$ در صورتی که [i,j] شامل یک چاله باشد .

$B_{ij} = true$ در صورتی که [i,j] خانه ای بدبو باشد ، مدل احتمال فقط شامل خانه های $B_{1,1}, B_{1,2}, B_{2,1}$ می باشد .

تعریف مدل احتمال - توزیع پیوسته ی کامل به صورت زیر است :

$$P(P_{1,1}, ..., P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1})$$

به کار بردن قانون ضرب :

$$P(B_{1,1}, B_{1,2}, B_{2,1} \mid P_{1,1}, ..., P_{4,4})P(P_{1,1}, ..., P_{4,4})$$

(این راه را برای به دست آوردن P(Effect|Cause) انجام دهید.)

شرط اول : در صورتی که چاله ها در همسایگی خانه های بدبو باشند یک می باشد و در غیر این صورت صفر می باشد .

شرط دوم : چاله ها به طور تصادفی جایگزین می شوند ، برای هر خانه احتمال برابر ۰,۲ می باشد :

$$P(P_{1,1}, ..., P_{4,4}) = \prod_{i,j=1,1}^{4,4} P(P_{i,j}) = 0.2^n \times 0.8^{16-n}$$

برای n چاله .

مشاهدات و پرس و جوها - ما واقعیات زیر را می دانیم :

$$b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$$
$$known = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$$

پرس و جو این است :

$$P(P_{1,3} \mid known, b)$$

$Unknown = P_{i,j}$ ها را متمایز از $P_{1,3}$ و Known تعریف نمایید . برای استدلال با شمارش داریم :

$$P(P_{1,3} \mid known, b) = \alpha \sum_{unknown} P(P_{1,3}, unknown, known, b)$$

با تعداد خانه ها به صورت نمایی رشد می نماید !

استفاده از استقلال شرطی - بینش پایه [1] : دیدها ، به صورت شرطی از دیگر خانه های پنهان ارایه شده توسط خانه های پنهان همسایه می باشند .

تعریف نمایید $Unknown = Fringe \cup Other$ در این صورت

$$P(b \mid P_{1,3}, Known, Unknown) = P(b \mid P_{1,3}, Known, Fringe)$$

پرس و جو را به صورتی که ما می توانیم از آن استفاده نماییم به کار ببرید !

استفاده از استقلال شرطی

$$P(P_{1,3} \mid known, b) = \alpha \sum_{unknown} P(P_{1,3}, unknown, known, b)$$

$$= \alpha \sum_{unknown} P(b \mid P_{1,3}, known, unknown) P(P_{1,3}, known, unknown)$$

$$= \alpha \sum_{fringe} \sum_{other} P(b \mid known, P_{1,3}, fringe, other) P(P_{1,3}, known, fringe, other)$$

$$= \alpha \sum_{fringe} \sum_{other} P(b \mid known, P_{1,3}, fringe) P(P_{1,3}, known, fringe, other)$$

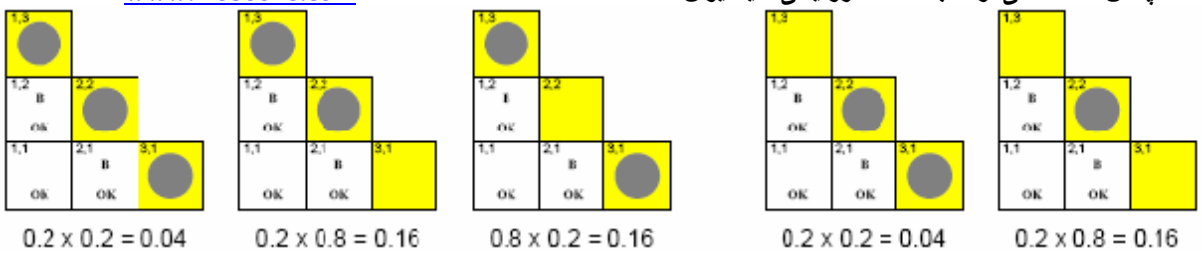
$$= \alpha \sum_{fringe} P(b \mid known, P_{1,3}, fringe) \sum_{other} P(P_{1,3}, known, fringe, other)$$

$$= \alpha \sum_{fringe} P(b \mid known, P_{1,3}, fringe) \sum_{other} P(P_{1,3}), P(known), P(fringe), P(other)$$

$$= \alpha P(known) P(P_{1,3}) \sum_{fringe} P(b \mid known, P_{1,3}, fringe) P(fringe) \sum_{other} P(other)$$

$$= \alpha' P(P_{1,3}) \sum_{fringe} P(b \mid known, P_{1,3}, fringe) P(fringe)$$

[1] basic insight



$$P(P_{1,3} \mid known, b) = \alpha' \langle 0.2(0.04+0.16+0.16), 0.8(0.04+0.16) \rangle \approx \langle 0.31, 0.69 \rangle$$

$$P(P_{2,2} \mid known, b) \approx \langle 0.86, 0.14 \rangle$$

کاربردهای استنتاج احتمالی

- آموزش بیزی – طبقه بندی [1] مثال های ندیده براساس توزیع های یک مجموعه ی آموزشی .
- شبکه های بیزی . سیستم های " برمبنای قانون " احتمالی
  - به کار گیری [2] استقلال شرطی برای نرمی [3]
  - بتواند استدلال تشخیصی یا سببی را انجام دهد
- شبکه های تصمیم گیری – پیش بینی اثرات عملکردهای نامعین .

خلاصه - احتمال یک روش بسیار خوب برای دانش نامعین می باشد . توزیع پیوسته ی احتمال ، احتمال هر رویداد خودکار را تعریف می نماید . پرس و جو ها می توانند با جمع کردن رویدادهای اتمیک پاسخ داده شوند . برای دامنه های غیر جزیی ، ما باید راهی برای کاهش اندازه ی پیوند پیدا نماییم . استقلال و استقلال شرطی ابزار ها را مهیا می نمایند .

منابع :

http://www.cs.usfca.edu
http://aima.eecs.berkeley.edu/slides-pdf/chapter13.pdf

[1] classifying
[2] exploit
[3] tractability

فصل پانزدهم

شبکه های بیزی

شبکه های بیزی [1]
فصل پانزدهم

ریوس مطالب

- **ساختار**
- **معنا**
- **توزیعات پارامتربندی شده** [2]

استنتاج احتمالی

- در گذشته ، ما در مورد سیستم های برمبنای قانونی که می توانند استنتاج های منطقی را رسم کنند صحبت کردیم .
  - $Hungry(Homer) \Rightarrow GoesTo(Homer, Quikie-mart)$
  - Hungry(Homer)
  - GoesTo(Homer,Quickie-mart)
- ما تمایل داریم این نوع از عملکرد را به جهان هایی با عدم قطعیت توسعه دهیم .
  - P(Hungry(Homer))=0.98
  - P(GoesTo(Homer,Quickie-mart)|Hungry(Homer))=0.5
  - P(GoesTo(Homer,Quickie-mart))=?
- در کل ، ما می توانیم از قانون بیز برای این کار استفاده نماییم
- مشکل در کار کردن با توزیع پیوسته ی احتمال می باشد
  - دارای جدول بزرگ به صورت نمایی می باشد .
- ما به ساختمان داده ای که واقعیتی را که در آن تعدادی متغیر بر هم تاثیر نمی گذارند را نگهداری می کند نیازمندیم .
- به عنوان مثال ، رنگ کلاه بارت بر این که Homer گرسنه می باشد تاثیری نمی گذارد .
- ما این ساختار را یک *شبکه ی بیزی* می نامیم .

شبکه های بیزی

---

[1] Bayesian networks
[2] parametrized distributions

- یک شبکه ی بیزی یک گراف مستقیم است که در آن هر گره با اطلاعات احتمالی ، تفسیر می شود . یک شبکه دارای :
  - یک مجموعه از متغیرهای تصادفی که با گره های درون شبکه برابر است
  - یک مجموعه از یال ها (لبه ها) یا پیکان ها که گره ها را متصل می کنند . این ها ، تاثیرات را نمایش می دهند . اگر پیکانی از X به طرف Y وجود داشته باشد ، آن گاه X ، پدر Y می باشد .
  - هر گره یک توزیع شرطی احتمال را که نشان دهنده ی احتمال هر مقداری که گره می تواند بگیرد و شرطی که مقادیر والدهایش می توانند بگیرند را نگهداری می کند .
  - هیچ حلقه ای وجود ندارد . (این یک گراف مستقیم بدون دور می باشد)
- توپولوژی (مکان شناسی) [1] شبکه مشخص می کند که متغیرها به طور مستقیم بر متغیرهای دیگر اثر می گذارند . (ارتباطات مستقل شرطی [2]) .

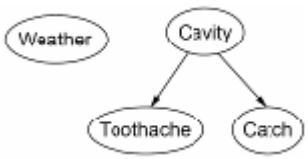
پس ، یک شبکه ی بیزی ، یک نماد [3] ساده و گرافیکی برای ادعاهای [4] مستقل شرطی و بنابراین برای توزیعات پیوسته ی کامل خصوصیات فشرده می باشد .

ساختار :

یک مجموعه از گره ها ، از هر متغیر، یکی

یک گراف غیرحلقه ای مستقیم

یک توزیع شرطی برای هر گره ی ارایه شده توسط والدهای خودش به صورت $P(X_i \mid Parents(X_i))$ می باشد . در ساده ترین حالت ، توزیع شرطی ارایه شده به صورت یک جدول احتمال شرطی [5] ، توزیع را برمبنای X_i ، برای هر ترکیب از مقادیر والد ارایه می دهد .



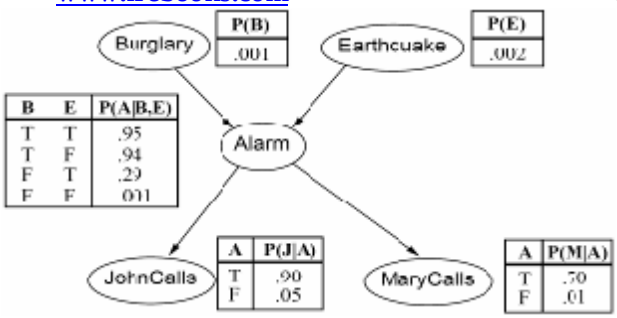
مثال - هوا مستقل از دیگر متغیرها می باشد . دندان درد و دچار شدن به آن به صورت شرطی مستقل از حفره ی دندان می باشند .

مثال دزدی [6] - من سرکار هستم و همسایه ام ، جان با من تماس می گیرد و می گوید آژیر من فعال است ، اما همسایه ی دیگرم ، مری [7] این کار را انجام نمی دهد . بعضی از وقت ها آژیر با کم ترین لرزه ای فعال می شود . آیا در آن جا واقعا دزد است ؟ متغیرها : دزد ، لرزه [8] ، آژیر [9] ، تماس های جان [10] ، تماس های مری [11] . توپولوژی شبکه آن چه که اتفاق افتاده است را برگشت می دهد :

- یک دزد می تواند آژیر را در وضعیت خاموش قرار دهد
- یک لرزه ی زمین می تواند آژیر را خاموش نماید
- آژیر می تواند سبب تماس مری شود
- آژیر می تواند سبب تماس جان شود

---

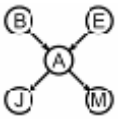
[1] topology
[2] conditional independence relationships
[3] notation
[4] assertions
[5] conditional probability table (CPT)
[6] burglar
[7] Mary
[8] earthquake
[9] alarm
[10] JohnCalls
[11] MaryCalls



تراکم [1]

- شبکه های بیزی ، به ارایه ی تراکم بیش تر دامنه اجازه می دهند
  - o اطلاعات اضافی (تکراری) [2] برداشته می شود .
- تصور کنید ما سی (۳۰) گره داریم ، که هر کدام دارای پنج (۵) پدر هستند .
- هر CPT دارای $2^5 = 32$ عدد می باشد ، در مجموع : ۹۶۰
- اتصال : 2^{30} موجودیت ، تقریبا همه تکراری هستند .

مطلب دیگری در مورد تراکم



یک CPT برای X_i با والدهای بولین k دارای 2^k سطر برای ترکیب متغیرهای والد می باشد . هر سطر یک عدد p را برای $X_i = true$ نیاز دارد (عدد مورد نیاز برای $X_i = false$ فقط 1-p می باشد) . در صورتی که هر متغیر دارای بیش از k والد نباشد ، شبکه ی کامل به تعداد $O(n.2^k)$ نیاز دارد . توجه کنید که [تراکم] به صورت خطی ، با n رشد می کند و دارای مرتبه ی $O(2^n)$ برای توزیع با اتصال کامل می باشد . برای شبکه ی دزدی تعداد ۱+۱+۴+۲+۲=۱۰ ($2^5 - 1 = 31$) .

معانی عمومی

معانی عمومی توزیع اتصال ، معانی کلی را به صورت ضرب توزیعات شرطی ، تعریف می نماید .

$$P(x_1,...,x_n) = \prod_{i=1}^{n} P(x_i \mid parents(X_i))$$

مثال :

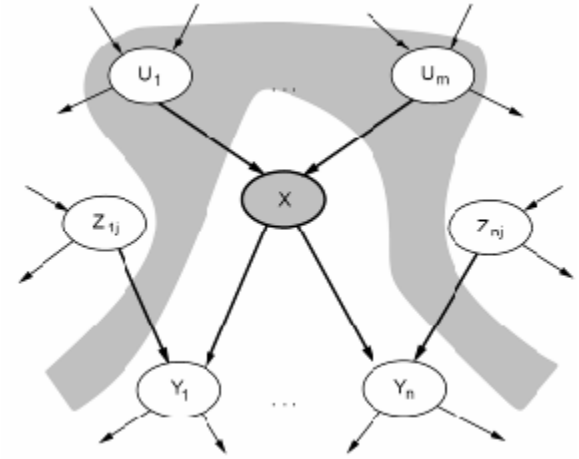
$$P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) = P(j \mid a)P(m \mid a)P(a \mid \neg b, \neg e)P(\neg b)P(\neg e)$$
$$= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \approx 0.00063$$

معانی محلی

معانی محلی هر گره به صورت شرطی ، وابسته به گره ی غیروالدی که والدهای آن را ارایه می نماید ، می باشد .

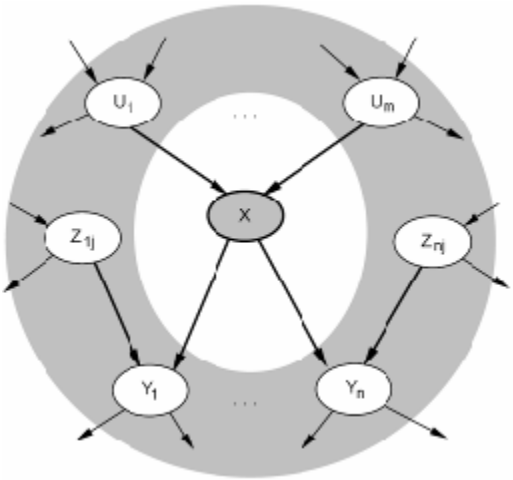
---

[1] compactness
[2] redundant



قضیه : معانی محلی ↔ معانی عمومی

پوسته ی مارکف [1] - هر گره به صورت شرطی ، مستقل از همه ی گره هایی است که آن را ارایه می نمایند .



ساختار شبکه

- هر گره دارای یک جدول احتما ل شرطی است .
- این جدول ، احتمال هر مقدار آن گره و مقادیر والدهای آن را ارایه می کند .
- گره های با هیچ والد فقط شامل قبلی ها هستند .

بیان عدم قطعیت به طورخلاصه

- توجه نمایید که ما نیازی به داشتن گره هایی برای همه ی مواردی که مری ممکن است تماس نگیرد نداریم .
  - یک نگرش احتمالی به ما اجازه می دهد که این اطلاعات را به صورت $\neg M$ خلاصه نماییم .
- این به یک عامل ساده اجازه می دهد که به جهان های بزرگی که دارای تعداد زیادی از نتایج عدم قطعیت ممکن هستند رسیدگی کند .
- چگونه ما می توانیم از این مطلب ، با منطق استفاده نماییم ؟

ارایه ی ضمنی توزیع پیوسته ی کامل

[1] Markov blanket

- به یاد بیاورید که توزیع پیوسته ی کامل به ما اجازه می داد که احتمال هر متغیر و همه ی آن هایی که ارایه شده اند را محاسبه نماییم .
- رویدادهای مستقل می توانند به جدول های مستقل تقسیم شوند .
- این ها CPT هایی هستند که در شبکه ی بیزی دیده می شوند .
- بنابراین ما می توانیم از این اطلاعات برای انجام محاسبات استفاده نماییم .
- $P(x_1, x_2, ..., x_n) = \prod P(x_i \mid parents(x_i))$
-

$$P(A \wedge \neg E \wedge \neg B \wedge J \wedge M) = P(J \mid A)P(M \mid A)P(A \mid \neg B \wedge \neg E)P(\neg B)P(\neg E) =$$
$$0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.00062$$

ساخت شبکه های بیزی

- اغلب ، چند روش برای ساخت یک شبکه ی بیزی وجود دارد .
- مهندسی دانش به کشف ارتباطات *مستقل شرطی* نیازمند است .
- والدهای یک گره باید متغیرهای آن هایی باشند که به طور مستقیم بر مقدار آن برتری دارند .
  - JohnCalls توسط لرزه دارای برتری می شود ، اما نه به طور مستقیم .
  - تماس جان و ماری دارای برتری بر یکدیگر نمی باشند
- صریحا ما تصور می کنیم :
- P(MaryCalls|JohnCalls,Alarm,Earthquake,Burglary)=P(MaryCalls|Alarm)

مطلب دیگر :

- به یک روش برای اثبات یک سری از ضمانت های مستقل محلی برای معانی عمومی نیازمندیم ،

۱) یک مجموعه ی منظم از متغیرهای $X_1,...,X_n$ را انتخاب نمایید .

۲) برای i=1 تا n کارهای زیر را انجام بده X_i را به شبکه اضافه کن . والدهای $X_1,...,X_{i-1}$ را انتخاب کن که

$$P(X_i \mid Parents(X_i)) = P(X_i \mid X_1,..., X_{i-1})$$

این نوع از انتخاب والد ها معانی عمومی زیر را به وجود می آورد :

$$P(X_1,..., X_n) = \prod_{i=1}^{n} P(X_i \mid X_1,..., X_{i-1})$$

(قانون زنجیری [1])

$$= \prod_{i=1}^{n} P(X_i \mid Parents(X_i))$$

(با ساخت [2])

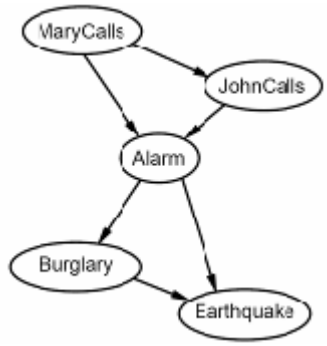
مثال - تصور کنید که ما مجموعه ی M,J,A,B,E را به صورت منظم انتخاب می نماییم

P(J|M)=P(J)? **نه**
P(A|J,M)=P(A|J)? P(A|J,M)=P(A) ? نه

---

[1] chain rule
[2] by construction

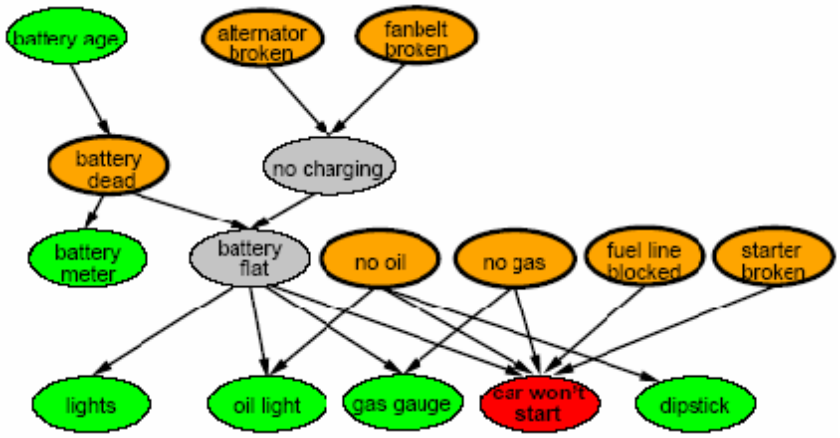
P(B|A,J,M)=P(B|A) ?بله
P(B|A,J,M)=P(B) ?نه
P(E|B,A,J,M)=P(E|A) ? نه
P(E|B,A,J,M)=P(E|A,B) ? بله



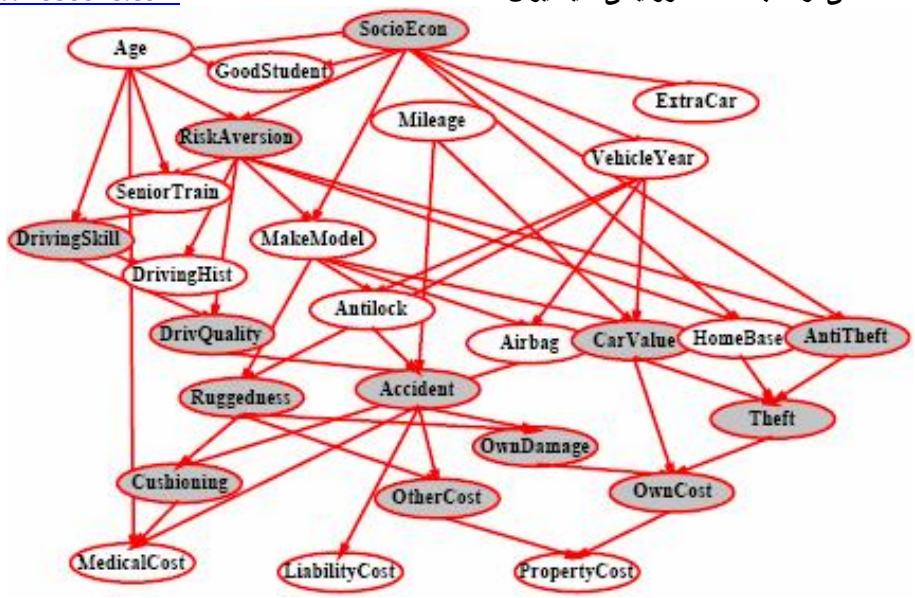
تصمیم گیری مستقل شرطی در موارد یا جهت های غیر شرطی مشکل است . (مدل های شرطی و مستقل شرطی برای انسان ها پیچیده به نظر می رسند !) . تشخیص احتمالات شرطی در موارد یا جهت های غیر سببی مشکل می باشد . شبکه دارای پیچیدگی کم تری می باشد : ۱+۲+۴+۲+۴= ۱۳ عدد لازم می باشد .

مثال : تشخیص اتومبیل

- دلیل اولیه : ماشین روشن نمی شود .
- متغیر های قابل آزمایش ، " در صورتی که شکسته باشند ، تعمیر نماییم " با رنگ نارنجی در شکل زیر نمایش داده شده اند .
- متغیر های مخفی (خاکستری) ، قطعات وابسته را چک نمایید و به این طریق پارامترها را کاهش دهید .



مثال : بیمه ی اتومبیل



ساخت یک شبکه

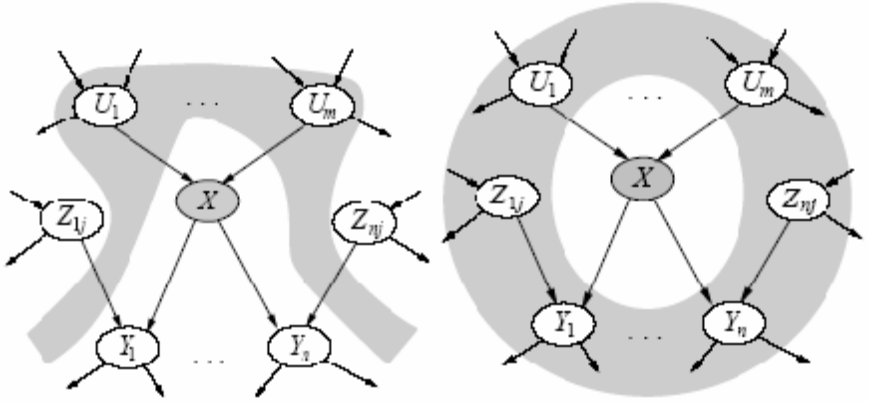
- با به وجود آوردن ریشه ها شروع نمایید
- سپس متغیر های آن هایی که به طور مستقیم هستند را اضافه نمایید .
- سپس فرزندان مستقیم آن ها را اضافه نمایید .
- این یک *مدل سببی* [1] نام دارد
  - استدلال در واژگان سبب و اثر می باشد
- تخمین ها با این راه به مراتب آسان تر به دست می آیند .
- ما ، برای ساخت می توانیم از اثر به سبب تلاش نماییم ، اما شبکه به مراتب پیچیده تر خواهد بود و تخمین CPT ها دشوار خواهد بود .

$$P(E|B)=?$$

استقلال شرطی در شبکه های بیزی

- به یاد بیاورید که استقلال شرطی دارای این معنی بود که دو متغیر ، مستقل از هم هستند
- $P(a \wedge b \mid c) = P(a \mid c)P(b \mid c)$
- یک گره به صورت شرطی مستقل از غیروالدها و والدهای ارایه اش می باشد .
  - در مثال آژیر ، JohnCalls مستقل از دزدی و لرزش زمین می باشد
- یک گره به صورت شرطی مستقل از همه ی گره های دیگر ، والدهایش ، فرزندانش و خواهر یا برادرها (والدهای دیگر فرزندان). است .

[1] causal model



استنتاج در شبکه های بیزی

- در بیش تر موارد ، ما می خواهیم از یک شبکه ی بیزی برای این که به ما احتمال های قبلی را بگوید استفاده نماییم .
- ما مشاهده می کنیم که یک متغیر چگونه متغیرهای دیگر را تغییر می دهد ؟
- ما می توانیم *متغیرهای پرس و جو* [1] (چیزهایی که می خواهیم در مورد آن ها بدانیم) و *متغیرهای شاهد* [2] (چیزهایی که می توانیم مشاهده نماییم) را تشخیص دهیم .
- همچنین متغیرهای مخفی هم وجود دارند که بر متغیرهای پرس و جو برتری دارند ، اما به طور مستقیم قابل مشاهده نمی باشند .
- JohnCalls و MaryCalls شواهدی هستند ، لرزش زمین و دزدی پرس و جو ها هستند و آژیر ، متغیر پنهان می باشد .

استنتاج در شبکه های بیزی به چهار روش انجام می شود :

- **استنتاج دقیق با تایین شماره** [3]
- **استنتاج دقیق با حذف متغیر** [4]
- **استنتاج تقریبی با با شبیه سازی اتفاقی** [5]
- **استنتاج تقریبی با زنجیره ی مارکف مونت کارلو** [6]

کارهای استنتاجی [7]

- پرس و جو های ساده :

$$P(X_i \mid E = e)$$

مثال ،

P(NoGas|Gauge=empty,Lights=on,Status=false)

- پرس و جوهای ربط دهنده :

$$P(X_i, X_j \mid E = e) = P(X_i \mid E = e)P(X_j \mid X_i, E = e)$$

---

[1] query variables
[2] evidence variables
[3] exact inference by enumeration
[4] exact inference by variable elimination
[5] approximate inference by stochastic simulation
[6] approximate inference by Markov chain Monte Carlo
[7] inference tasks

تصمیم گیری های بهینه [1] : شبکه های تصمیم گیری شامل اطلاعات مفیدی می باشند ؛ استنتاج احتمالاتی برای

P(outcome|action,evidence)

لازم می باشد .

ارزش اطلاعات [2] : کدام مدرک [3] برای جستجوی بعدی می باشد ؟

تحلیل حساسیت [4] : کدام مقادیر احتمال مهم تر می باشد ؟

توزیعات شرطی فشرده

- CPT ، با شماره ی والدها به صورت نمایی رشد می کند .
- CPT با مقادیر پیوسته ی والد یا فرزند ، نامحدود می شود .

راه حل : استفاده از توزیعات استاندارد که به صورت فشرده تعریف می شوند . گره های قطعی ساده ترین مورد می باشند :

X=f(Parents(X)) برای برخی از توابع f . مثلا ، توابع بولین

$$NorthAmerica \Leftrightarrow Canadian \vee US \vee Mexican$$

به ارتباطات میان متغیرهای پیوسته توجه نمایید

$$\frac{\partial Level}{\partial t} = \inf\ low + precipitation - outflow - evaporation$$

مطلب دیگری در مورد توزیعات شرطی فشرده - مدل توزیعات چند OR ممکن است منجر به این شود که :

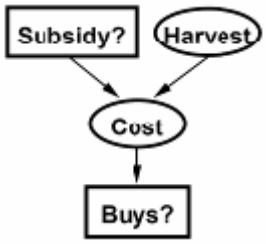
۱) والدهای $U_1 ... U_k$ شامل همه ی موارد می شوند

۲) احتمال رد شدن مستقل q_i برای هر مورد تنها

$$\Rightarrow P(X \mid U_1 ... U_j, \neg U_{j+1} ... \neg U_k) = 1 - \prod_{i=1}^{j} q_i$$

سرماخوردگی	آمفلوانزا	مالاریا	P(تب)	P(¬ تب)
F	F	F	۰,۰	۱,۰
F	F	T	۰,۹	۰,۱
F	T	F	۰,۸	۰,۲
F	T	T	۰,۹۸	۰,۰۲=۰,۲*۰,۱
T	F	F	۰,۴	۰,۶
T	F	T	۰,۹۴	۰,۰۶=۰,۶*۰,۱
T	T	F	۰,۸۸	۰,۱۲=۰,۶*۰,۲
T	T	T	۰,۹۸۸	۰,۰۱۲=۰,۶*۰,۲*۰,۱

[1] optimal decisions
[2] Value of information
[3] evidence
[4] Sensitivity analysis



شبکه های دورگه (گسسته + پیوسته) - تصور نمایید که Subsidy و Buys شبکه های گسسته و Harvest و Cost شبکه های پیوسته باشند .

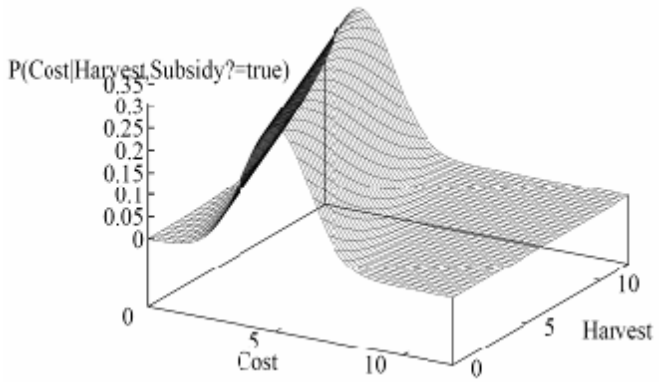
انتخاب ۱ : گسسته – ممکن است دارای خطاهای زیاد و CPT های طولانی باشد .
انتخاب ۲ : خانواده های استاندارد پارامتر بندی شده به صورت محدود .

متغیرهای فرزندی پیوسته - به یک تابع شرطی برای متغیر فرزند ارایه شده توسط والدهای پیوسته ، برای هر انتساب ممکن والدهای گسسته نیازمندیم ، معمول ترین آن ، مدل خطی گاوس [1] می باشد ، به عنوان مثال ؛

$$P(Cost=c|Harvest=h,Subsidy?=true) = N(a_t h + b_t, \sigma_t)(c)$$

$$= \frac{1}{\sigma_t \sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{c-(a_t h + b_t)}{\sigma_t} \right)^2 \right)$$

متوسط نرخ تغییر خطی ، با Harvest و واریانس [2] ثابت می شود . تغییر خطی در مورد دامنه ی کامل ، غیرمعقول می باشد اما در صورتی که محدوده ی Harvest کم باشد ، بسیار خوب کار می کند .

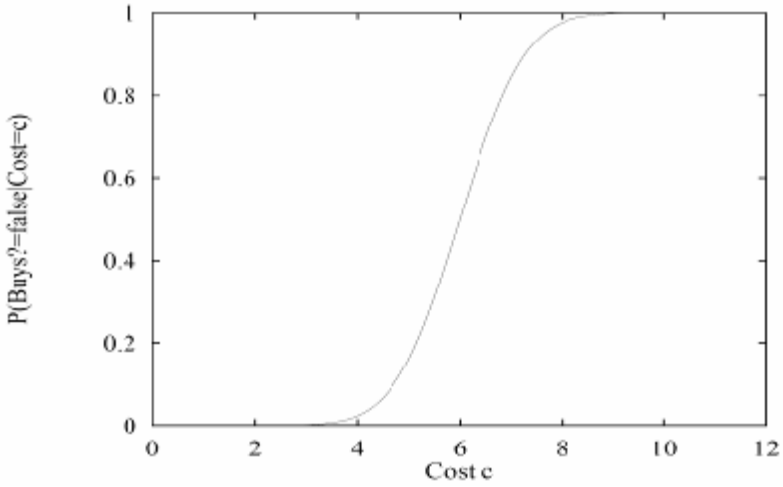


همه ی شبکه های پیوسته با توزیع های LG می باشند ← توزیع کاملا پیوسته ، یک توزیع چند متغیره ی گاوسی می باشد .
شبکه ی LG پیوسته + گسسته ، یک شبکه ی گاوسی شرطی [3] می باشد ، توجه کنید یک شبکه ی چند متغیره ی گاوسی در تمام متغیرهای پیوسته ، برای هر ترکیب گسسته ی مقادیر متغیرها می باشد .

---

[1] linear Gaussian model
[2] variance
[3] conditional Gaussian network

متغیر گسسته با والدهای پیوسته ـ احتمال Buys ارایه شده توسط Cost باید دارای یک آستانه ی [1] " ملایم یا نرم " باشد :



توزیع قیاس [2]

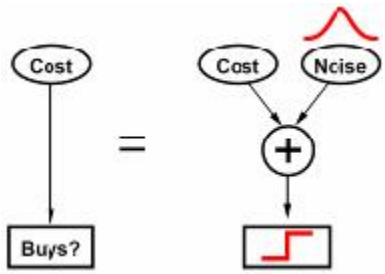
از انتگرال گاوسی استفاده می نماید :

$$\Phi(x) = \int_{-\infty}^{x} N(0,1)(x)dx$$

$$P(Buys?=true|Cost=c)=\Phi((-c+\mu)/\sigma)$$

چرا قیاس ؟

ـ آن از طرف راست مرتب می باشد و در جایی که منجر به اغتشاش می شود می تواند با آستانه ی سخت یا تند دیده شود .



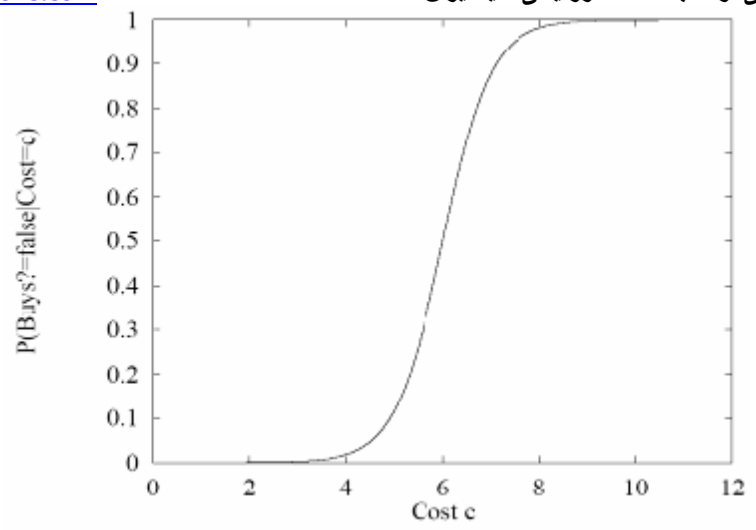
متغیر گسسته ـ توزیع مانند S ، در شبکه های عصبی هم استفاده می شود :

$$P(Buys?=true|Cost=c)=\frac{1}{1+\exp(-2\frac{-c+\mu}{\sigma})}$$

Sigmoid دارای شکلی شبیه به انحراف میانگین می باشد اما دارای دم به مراتب بلندتری می باشد :

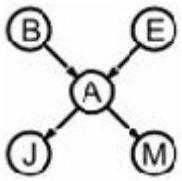
---

[1] threshold
[2] probit



استنتاج با شمارش - روشی برای جمع کردن متغیرهای پیوسته ، بدون تولید واقعی نمایشی صریح از آن ها می باشد و تا حدودی هوشمندانه می باشد . پرس و جوی ساده ای برای شبکه ی دزدی :

- احتمال $P(X \mid e) = \alpha P(X,e) = \alpha \sum P(X,e,y)$ را به یاد بیاورید ، در جایی که y جزو متغیرهای پنهان می باشد .
- این برابر است با جمع در توزیع پیوسته ی احتمال .
- توجه کنید
$P(Burglary \mid JohnCalls, MaryCalls) = \alpha P(Burglary \wedge JohnCalls \wedge MaryCalls)$
- این $\alpha \sum_E \sum_A P(B,E,A,J,M)$ برای B=true می باشد ، ما باید مقدار عبارت زیر را محاسبه نماییم :



$$P(B \mid J,M) = \alpha \sum_E \sum_A P(B)P(E)P(A \mid B,E)P(J \mid A)P(M \mid A)$$

- مساله – یک شبکه با n گره به $O(n2^n)$ محاسبه نیازمند است .
- ما می توانیم این مقدار را با خارج کردن قبلی ها از جمع ، ساده نماییم .
- $\alpha P(B) \sum_E P(E) \sum_A P(A \mid B,E)P(J \mid A)P(M \mid A)$
- هنوز به $O(2^n)$ محاسبه نیازمندیم

الگوریتم شمارش

تابع Enumeration-Ask(X,e,bn) یک توزیع را براساس X برمی گرداند

ورودی ها : X ، متغیر پرس و جو

e ، مقادیر مشاهده شده برای متغیر E

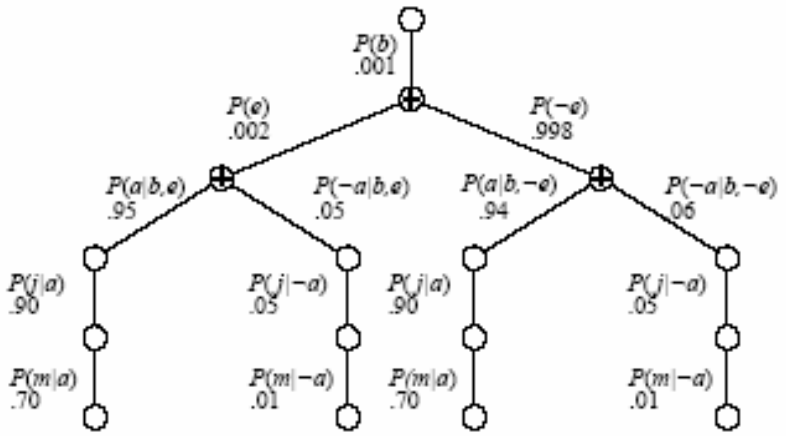
bn ، یک شبکه ی بیزی با متغیرهای $\{X\} \cup E \cup Y$

یک توزیع براساس X ، که به صورت اولیه خالی می باشد ← (منتسب کن به)Q(X)

برای هر مقدار x_i از X ، کارهای زیر را انجام بده
e را با مقدار x_i برای X توسعه بده
$Q(x_i) \leftarrow$ Enumerate-All(Vars[bn],e)
Normalize(Q(X)) را برگردان

تابع Enumerate-All(vars,e) یک عدد حقیقی را برمی گرداند
در صورتی که Empty(vars) درست است عدد ۱,۰ را برگردان
Y ← First(vars)
در صورتی که Y دارای مقدار y که در e می باشد هست P(y|Pa(Y))*Enumerate-All(Rest(vars),e) را برگردان
در غیر این صورت $\sum_y P(y \mid Pa(Y)) \times Enumerate - All(\text{Re}\,st(\text{var}\,s), e_y)$ را در جایی که e_y ، e توسعه داده شده توسط Y=y می باشد برگردان

درخت ارزیابی



- این درخت ، محاسبات لازم برای تصمیم در مورد P(B|J,M) را نشان می دهد
- توجه کنید که تعداد زیادی محاسبات تکراری در آن وجود دارد .
- با حذف محاسبات تکراری ، ما می توانیم سرعت را بالا ببریم .

پس ، شمارش ، ناکارامد است ، چون دارای محاسبات تکرار شده می باشد . مثلا ، P(j|a)P(m|a) را برای هر مقدار e محاسبه می نماید .

استنتاج با حذف متغیر

- ما می توانیم تعداد محاسبات را با نگهداری نتایج و استفاده ی مجدد از آن ها کاهش دهیم .
- عبارت را از راست به چپ ارزیابی نمایید .
- مجموع یابی ها فقط برای گره های تاثیر داده شده توسط متغیری که جمع می شود انجام می شود

مجموع یابی را از راست به چپ انجام دهید و نتایج (فاکتورهای) میانی را برای جلوگیری از محاسبه ی مجدد ذخیره نمایید .

- توجه کنید

$$P(B \mid j,m) = \alpha \underbrace{P(B)}_{B} \sum_e \underbrace{P(e)}_{E} \sum_a \underbrace{P(a \mid B,e)}_{A} \underbrace{P(j \mid a)}_{J} \underbrace{P(m \mid a)}_{M}$$

$$= \alpha P(B)\sum_e P(e)\sum_a P(a \mid B,e)P(j \mid a) f_M(a)$$
$$= \alpha P(B)\sum_e P(e)\sum_a P(a \mid B,e) f_J(a) f_M(a)$$
$$= \alpha P(B)\sum_e P(e)\sum_a f_A(a,b,e) f_J(a) f_M(a)$$
$$= \alpha P(B)\sum_e P(e) f_{\bar{A}JM}(b,e) \text{ (مجموع A)}$$
$$= \alpha P(B) f_{\bar{E}\bar{A}JM}(b) \text{ (مجموع E)}$$
$$= \alpha f_B(b) \times f_{\bar{E}\bar{A}JM}(b)$$

- قبلا ، ما P(M|a) و $P(M \mid \neg a)$ را محاسبه کردیم و نتایج آن را در ماتریس f_M نگهداری نمودیم .
- به طور مشابه برای P(J|a) و $P(J \mid \neg a)$ در f_J نگهداری می شوند
- نتیجه ی P(A|B,E) به صورت یک ماتریس $2 \times 2 \times 2$ در F_A نگهداری می شود .
- سپس ما به محاسبه ی جمع برای مقادیر مختلف ممکن برای A نیازمندیم
- مجموع برابر است با $\sum_a f_A(A,B,E) f_J(A) f_M(A)$
- ما می توانیم E را هم به روشی مشابه ، محاسبه نماییم .
- ما برنامه نویسی پویا را دراین جا انجام می دهیم .
- پیچیدگی
  - چند درختی ها : (یک مسیر غیرمستقیم میان هر دو گره) – خطی در میان تعدادی از گره ها
  - گره های ضرب شده ی متصل : به صورت نمایی می باشد

پس ، عملیات اساسی برای حذف متغیر - یک متغیر را از یک ضرب از فاکتورها جمع زنی نمایید : هر فاکتور ثابت خارج از جمع زنی را خارج نمایید . زیر ماتریس ها را در ضرب نقطه ای فاکتورهای باقی مانده اضافه نمایید .

فرض کنید که $f_1,...,f_i$ ، به X وابسته نمی باشند . ضرب نقطه ای فاکتورهای f_1 و f_2 :

$$f_1(x_1,...,x_j,y_1,...,y_k) \times f_2(y_1,...,y_k,z_1,...,z_l)$$
$$= f(x_1,...,x_j,y_1,...,y_k,z_1,...,z_l)$$
$$f_1(a,b) \times f_2(b,c) = f(a,b,c)$$

الگوریتم حذف متغیر

تابع Elimination-Ask(X,e,bn) ، این تابع توزیعی را براساس X برمی گرداند

ورودی ها : X که یک متغیر جستجو می باشد ، e ، ثابتی تعریف شده به صورت یک رویداد می باشد ، bn ، یک شبکه ی معین کننده ی توزیع پیوسته ی $P(X_1,...,X_n)$ می باشد

factors ← [ ] ؛ vars ← Reverse(Vars[bn])

برای هر var در vars کارهای زیر را انجام بده

factors ← [Make-Factor(var,e)|factors]

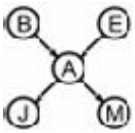
در صورتی که var یک متغیر پنهان است Sum-Out(var,factors) ← factors

Normalize(Pointwise-Product(factors)) را برگردان .

متغیرهای نامربوط [1] - به پرس و جوی زیر توجه کنید :

P(JohnCalls|Burglary=true)

$$P(J \mid b) = \alpha P(b)\sum_e P(e)\sum_a P(a \mid b,e)P(J \mid a)\sum_m P(m \mid a)$$



جمع بر روی m به صورت یکسان ۱ می باشد ؛ M به پرس وجو نامربوط می باشد

قضیه ی ۱ : Y نامربوط است مگر این که $Y \in Ancestors(\{X\} \cup E)$ باشد . در این جا ، {X=JohnCalls,E={Burglary} و $Ancestors(\{X\} \cup E) = \{Alarm, Earthquake\}$ می باشد ، بنابراین MaryCalls نامربوط می باشد . (این مطلب را با زنجیره ی معکوس از پرس و جو در KB های عبارت شیپوری مقایسه نمایید)

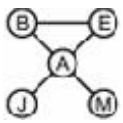
گراف تشخیص [2] **شبکه ی بیز :** تمام والدان ازدواج می کنند (جفت می شوند) و پیکان ها برداشته می شوند .

تعریف : A ، m – مشتق شده [3] از B توسط C می باشد در صورتی که توسط C در گراف تشخیص مشتق شده باشد .

قضیه ی ۲ : Y نامربوط است در صورتی که m – مشتق شده از X توسط E باشد . برای P(JohnCalls|Alarm=true) ، هم Burglary و هم Earthquake نامربوطند .

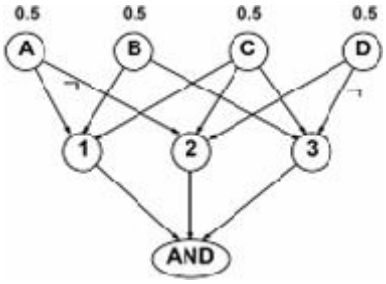
پیچیدگی استنتاج دقیق

- شبکه های متصل شده به صورت یک به یک [4] (یا چند درختی) :



- هر دو گره حداکثر با یک مسیر به هم متصل شده اند
- هزینه ی حذف زمان و فضای متغیر ، برابر است با $O(d^k n)$

- شبکه های متصل شده به صورت ضربدری [5] :
  - سه جایگزین را می توانند برای استنتاج دقیق کاهش دهند ← NP-hard می باشد
  - با شمارش مدل های با سه جایگزینی برابرند ← #P – کامل می باشد .



$1.A \vee B \vee C$, $2.C \vee D \vee \neg A$, $3.B \vee C \vee \neg D$

استنتاج با شبیه سازی اتفاقی (تصادفی)

---

[1] irrelevant variables
[2] moral graph
[3] m-separated
[4] Singly connected networks
[5] Multiply connected networks

- ایده ی اساسی :

۱) نمونه های N تایی را از یک توزیع نمونه ی S استخراج نمایید

۲) یک احتمال تقریبی قبلی $\hat{P}$ را محاسبه نمایید

۳) حاصل را با مقدار درست P مقایسه نمایید

خلاصه : از یک شبکه ی خالی ، نمونه گیری نمایید .

رد نمونه گیری [1] : نمونه های ناموافق با ثابت را رد کنید .

توزیع احتمال [2] : از ثابت برای توزیع های احتمال استفاده کنید .

زنجیره ی مارکوف مونت کارلو [3] : از یک پردازش تصادفی که توزیع ثابت قبلی آن درست بوده ، نمونه برداری کنید .

نمونه گیری از یک شبکه ی خالی –

تابع Prior-Sample(bn) یک رویداد نمونه برداری شده از bn را برمی گرداند .

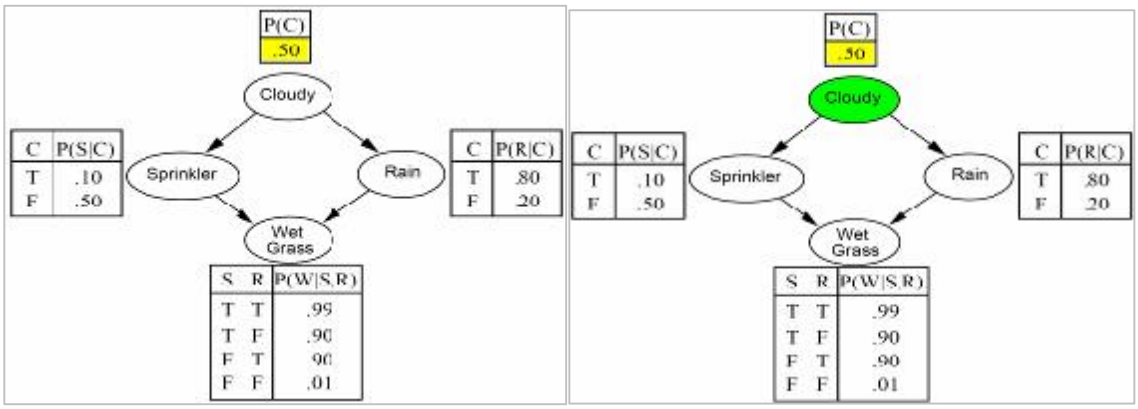
ورودی ها : bn ، یک شبکه با توزیع معین $P(X_1,...,X_n)$.

یک رویداد با n عنصر ← x

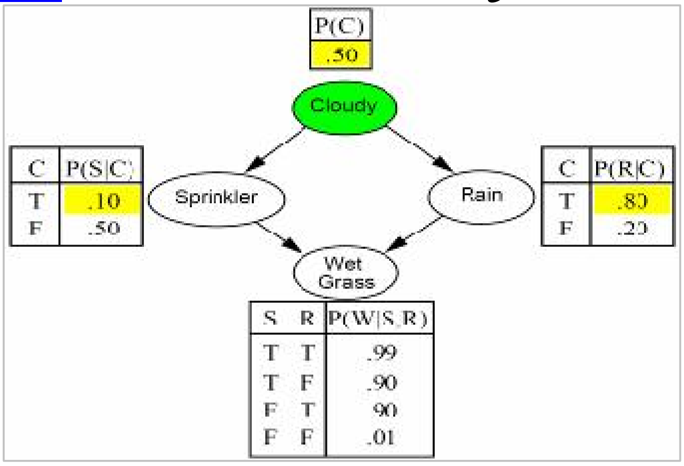
برای i=1 تا n کارهای زیر را انجام بده

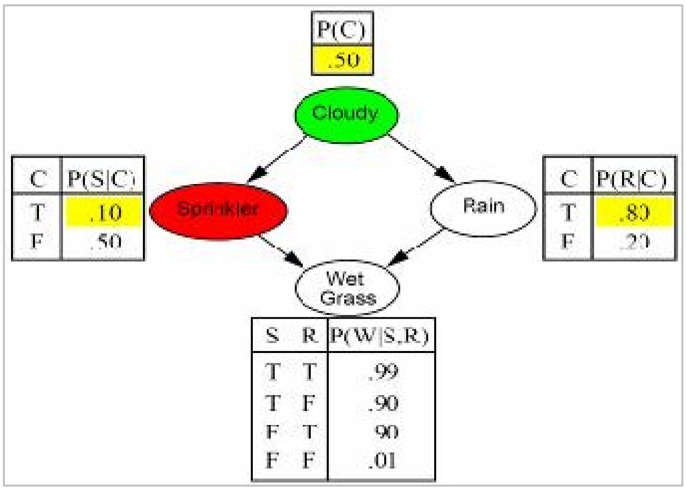
یک نونه ی تصادفی از $x_i \leftarrow P(X_i \mid parents(X_i))$

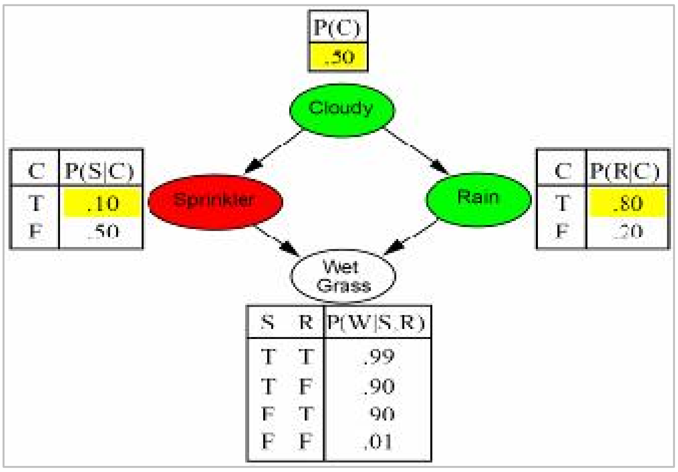
مقادیر $Parents(X_i)$ موجود در x را به دست آورید . x را برگردان

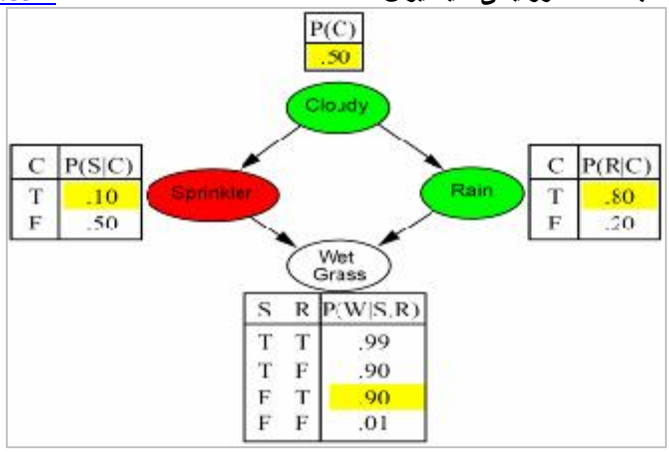


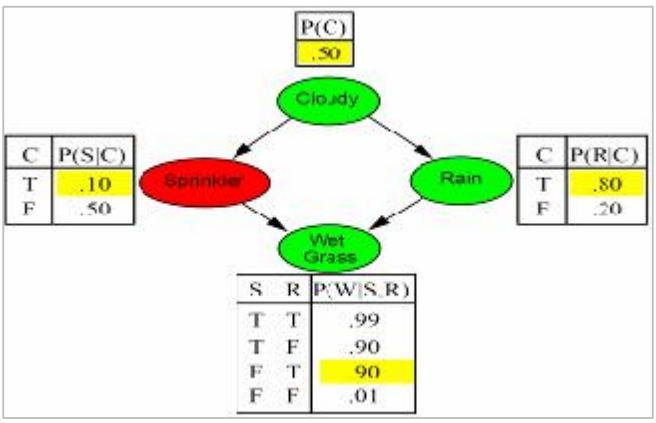
[1] rejection sampling
[2] Likelihood weighting
[3] Markov chain Monte Carlo (MCMC)


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T 90
F F .01


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T 90
F F .01


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T 90
F F .01





احتمال این که PriorSample یک رویداد ویژه را به وجود بیاورد

$$S_{PS}(x_1..x_n) = \prod_{i=1}^{n} P(x_i \mid parents(X_i)) = P(x_1..x_n)$$

به احتمال درست قبلی توجه کنید

$$S_{PS}(t, f, t, t) = 0.5 * 0.9 * 0.8 * 0.9 = 0.324 = P(t, f, t, t)$$

تصور نمایید $N_{PS}(x_1...x_n)$ تعداد نمونه های تولید شده برای رویداد $x_1,...,x_n$ باشد ، در این صورت داریم :

$$\lim_{N\to\infty} \hat{P}(x_1,...,x_n) = \lim_{N\to\infty} N_{PS}(x_1,...,x_n)/N$$

$$S_{PS}(x_1,...,x_n) = P(x_1...x_n)$$

این تخمین ها که مشتق شده از PriorSample می باشند ، سازگار می باشند
به اختصار :

$$\hat{P}(x_1,...,x_n) \approx P(x_1...x_n)$$

رد نمونه گیری

$\hat{P}(X \mid e)$ با استفاده از نمونه های سازگار با e تخمین زده می شود .
تابع Rejection-Sampling(X,e,bn,N) تخمینی از P(X|e) را برمی گرداند
متغیرهای محلی : N ، یک بردار از شمارش های در مورد X ، که به صورت اولیه دارای مقدار صفر می باشد

برای j=1 تا N کارهای زیر را انجام بده
x ← Prior-Sample(bn)
در صورتی که x با e سازگار است
در جایی که x مقداری از X درون x می باشد N[x] ← N[x]+1
Normalize(N[X]) را برگردان

مثلا ، تخمین P(Rain|Sprinkler=true) با استفاده از ۱۰۰ نمونه ، ۲۷ نمونه دارای Sprinkler=true می باشد ، از این ها ۸ تا دارای Rain=true و ۱۹ تا دارای Rain=false می باشد .

$$\hat{P}(Rain \mid Sprinkler = true) = Normalize(\langle 8,19 \rangle) = \langle 0.296,0.704 \rangle$$

شبیه روال تخمین یک دنیای واقعی تجربی می باشد .

تحلیل رد نمونه گیری

$\hat{P}(X \mid e) = \alpha N_{PS}(X,e)$ (تعریف الگوریتم)

$= N_{PS}(X,e) / N_{PS}(e)$ (نرمال سازی شده توسط $N_{PS}(e)$)

$\approx P(X,e) / P(e)$ (خصوصیت PriorSample)

P(X|e) (تعریف احتمال شرطی)

بنابراین رد نمونه گیری ، تخمین های سازگاری قبلی را برمی گرداند .

مساله : ناچارا ، در صورتی که P(e) کوچک باشد پرهزینه می باشد .

P(e) به صورت نمایی با تعداد متغیرهای گواه ، روند نزولی را طی می کند یا میرا می شود !

توزیع احتمال

ایده : متغیرهای ثابت ، متغیرهای با نمونه های فقط غیرثابت و توزیع هر نمونه توسط احتمال ثابت را وفق می دهند (سازگار می کند) .

الگوریتم :

تابع Likelihood-Weighting(X,e,bn,N) تخمینی از P(X|e) را برمی گرداند

متغیرهای محلی : W ، برداری از شمارش های توزیع شده حول X ، با مقدار اولیه ی صفر

برای j=1 تا N کارهای زیر را انجام بده

$x, w \leftarrow Weighted - Sample(bn)$

در جایی که x دارای مقداری از X موجود در x می باشد $W[x] \leftarrow W[x] + w$

Normalize(W[X]) را برگردان

تابع Weighted-Sample(bn,e) یک رویداد و یک توزیع را برمی گرداند

یک رویداد با n عنصر ← x ؛ ۱ ← w

برای i=1 تا n کارهای زیر را انجام بده

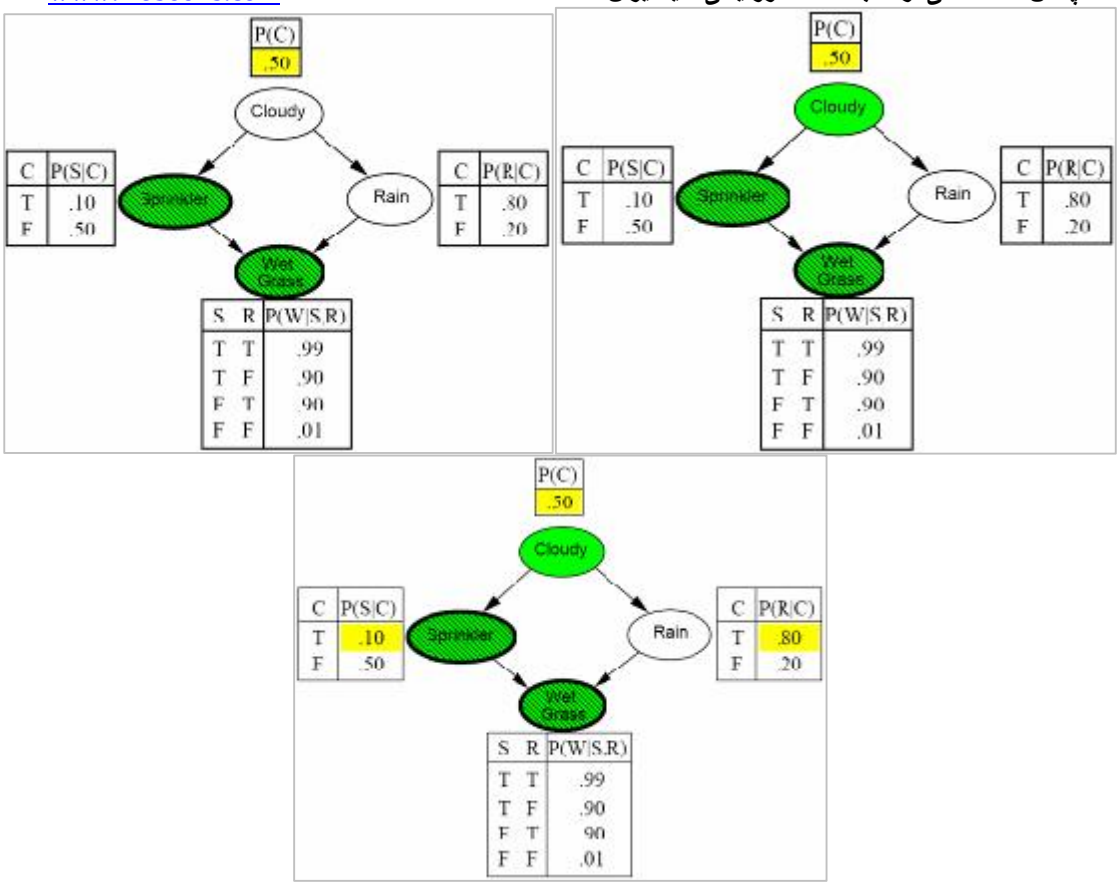
در صورتی که X_i دارای یک مقدار x_i موجود در e می باشد

$w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$

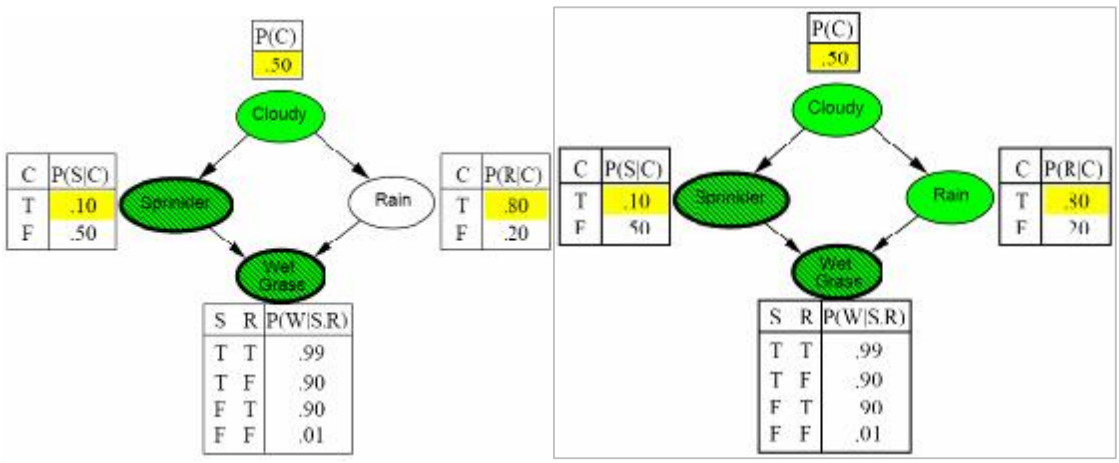
در غیر این صورت یک نمونه ی تصادفی از $x_i \leftarrow P(X_i \mid parents(X_i))$

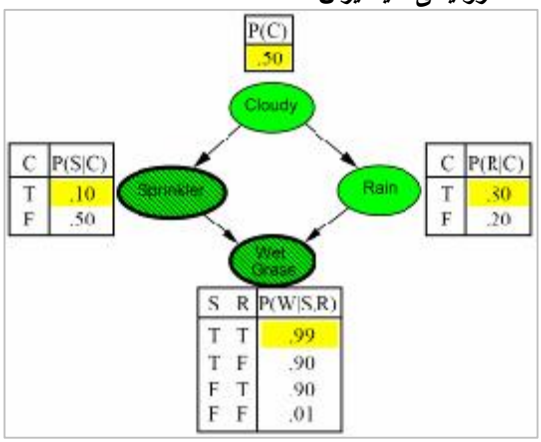
x,w را برگردان

مثال توزیع احتمال

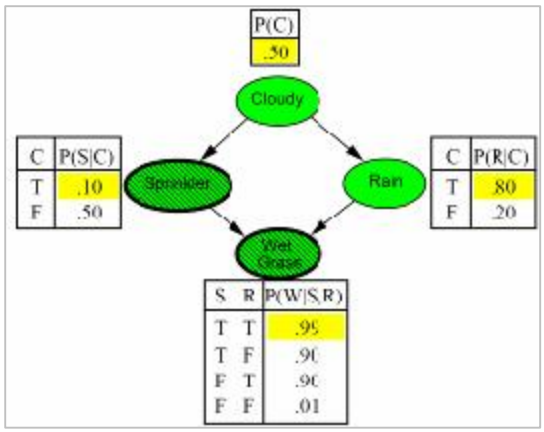

P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

w=1.0


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01



w=1.0*0.1



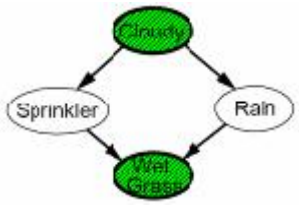
w=1.0*0.1*0.99=0.099

تحلیل توزیع احتمال

توزیع احتمال برای WeightedSample به صورت زیر است :

$$S_{WS}(z,e) = \prod_{i=1}^{l} P(z_i \mid parents(Z_i))$$

نکته : توجه کنید که ثابت فقط در پیشینیان ها (اجداد) می باشد در نتیجه در مکانی " در میان " قبلی و قبلی تر ها ثابت وجود دارد .



توزیع برای یک نمونه ی ارایه شده ی z,e به صورت زیر می باشد :

$$w(z,e) = \prod_{i=1}^{m} P(e_i \mid parents(E_i))$$

نمونه ی توزیع شده ی احتمال به صورت زیر می باشد :

$$S_{WS}(z,e)w(z,e) = \prod_{i=1}^{l} P(z_i \mid parents(Z_i)) \prod_{i=1}^{m} P(e_i \mid parents(E_i)) = P(z,e)$$

(به وسیله ی معانی عمومی شبکه)

در این صورت ، توزیع احتمال ، تخمین های سازگار را برمی گرداند ، اما عملکرد هنوز با تعداد زیادی از متغیرهای ثابت کاهش می یابد ، زیرا تعداد کمی از نمونه ها تقریبا همه ی توزیع ها را دارند .

استنتاج تقریبی با استفاده از MCMC

" وضعیت " شبکه = انتساب جاری به همه ی متغیرها .

حالت بعدی را به وسیله ی نمونه برداری از یک متغیر ارایه شده ی لایه ی مارکوف تولید می نماید .

نمونه ی هر متغیر در برگشت ، ثابت ثابت شده را نگهداری می نماید .

الگوریتم:

تابع MCMC-Ask(X,e,bn,N) یک تخمین از P(X|e) را برمی گرداند

متغیرهای محلی : N[X] ، یک بردار از شمارش های حول X ، که در ابتدا دارای مقدار صفر می باشد

Z ، متغیرهای غیر ثابت در bn

x ، وضعیت جاری شبکه ، در ابتدا از e کپی (گرفته) شده

x را با مقادیر تصادفی متغیرهای درون Y مقداردهی اولیه نمایید

برای j=1 تا N کارهای زیر را انجام بده

برای هر Z_i در Z کارهای زیر را انجام بده

نمونه ی مقدار Z_i در x از $P(Z_i \mid mb(Z_i))$

مقادیر $MB(Z_i)$ درون x ارایه شده

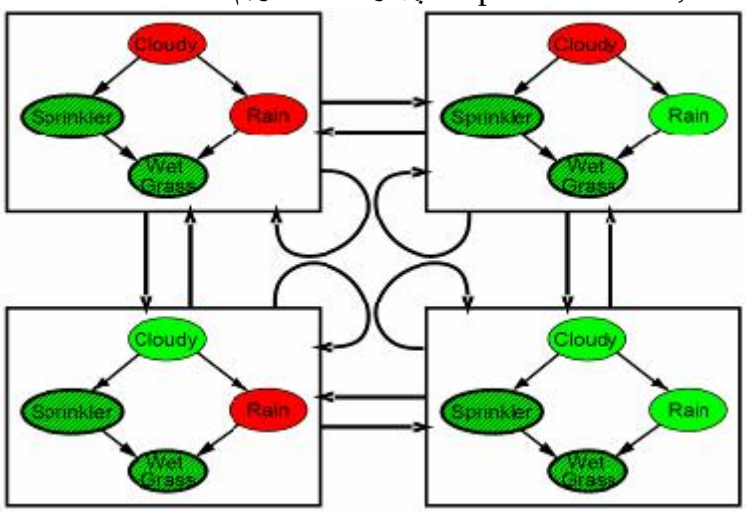
در جایی که مقدار X درون x می باشد ، N[x] ← N[x]+1

Normalize(N[X]) را برگردان

همچنین می توانیم یک متغیر را برای نمونه به صورت تصادفی در هر زمان انتخاب نماییم .

زنجیره ی مارکوف

با Sprinkler=true , WetGrass=true ، چهار حالت داریم :



مثال MCMC

P(Rain|Sprinkler=true,WetGrass=true) را تخمین بزنید .

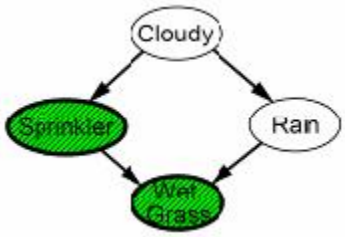
از Cloudy یا Rain ارایه شده توسط لایه ی مارکوف ، نمونه برداری نمایید و این کار را تکرار نمایید .
تعداد دفعاتی را که Rain درست و غلط است را در نمونه ها شمارش نمایید .
مثلا دارای ۱۰۰ وضعیت هستیم ، در ۳۱ وضعیت داریم Rain=true و در ۶۹ وضعیت Rain=false می باشد .

$$\hat{P}(Rain \mid Sprinkler = true, WetGrass = true) = Normalize(\langle 31,69 \rangle) = \langle 0.31,0.69 \rangle$$

قضیه : در زنجیره با توزیع ثابت [1] : اجرای طولانی [2] مدت در هر وضعیت ، دقیقا متناسب است با احتمال قبلی آن .

نمونه گیری از لایه ی مارکوف

لایه ی مارکوف Cloudy ، Sprinkler و Rain می باشد .
لایه ی مارکوف Rain ، Cloudy ، Sprinkler و WetGrass می باشد .



احتمال ارایه شده توسط لایه ی مارکوف به صورت زیر محاسبه می شود :

$$P(x_i' \mid mb(X_i)) = P(x_i' \mid parents(X_i))$$
$$\prod_{Z_j \in Children(X_i)} P(z_j \mid parents(Z_j))$$

به سادگی در سیستم های اجراکننده ی پیام [3] و مغزها به کار گرفته می شود .
مسایل محاسباتی اصلی :

۱) مشکل است که بگوییم آیا به همگرایی رسیده ایم
۲) در صورتی که لایه ی مارکوف طولانی باشد ، ممکن است بی فایده باشد : $P(X_i \mid mb(X_i))$ نمی تواند به صورت زیاد تغییر کند

افزایش به نسبت ثابت [4]

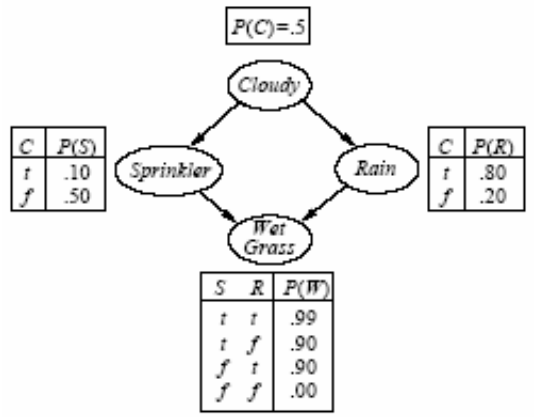
- تعدادی روش برای افزایش شبکه های بیزی به صد ها گره توسعه داده شده است .
- دسته بندی [5]
  - گره ها برای ساخت شبکه به صورت یک درخت چندگانه به هم متصل می شوند .
  - CPT در گره رشد می کند ، اما ساختار شبکه ی ما ساده شده است .
- تخمین استنتاج
  - نمونه برداری [1] مونت کارلو [2] برای تخمین زدن احتمال های شرطی استفاده می شود .

---

[1] stationary distribution
[2] long-run
[3] message-passing
[4] scaling up
[5] clustering

مثال نمونه برداری مونت کارلو

- از بالای شبکه شروع نمایید و یک نمونه ی تصادفی را انتخاب کنید . (تصور کنید که آن درست می باشد)
- یک نمونه ی تصادفی را از فرزندانش بکشید . مشروط بر درست بودن .
  - P(Sprinkler|Cloudy=true)=<0.1,0.9> تصور نمایید که ما غلط را انتخاب نمودیم .
  - P(Rain|Cloudy=true)=<0.8,0.2> . تصور کنید که ما درست را انتخاب نمودیم
  - P(WetGrass|Sprinkler=false,Rain=true)=<0.9,0.1> . تصور نمایید که ما درست را انتخاب کردیم .
- این به ما یک نمونه برای $< Cloudy, \neg Sprinkler, Rain, WetGrass >$ ارایه می دهد .
- در صورتی که ما تعداد نمونه گیری ها را افزایش دهیم ، این تخمینی از $< Cloudy, \neg Sprinkler, Rain, WetGrass >$ را ارایه می دهد .



نگرش های دیگر به عدم قطعیت

- استدلال پیش فرض
  - منطقی
  - " وگرنه ، ما می دانیم A از B تبعیت می کند "
- دمپستر – شیفر
  - عدم قطعیت را با احتمال ، یکی می کند
- منطق فازی
  - عبارت هایی را که تاحدی درست هستند مجاز می داند .

کاربردهای شبکه های بیزی

- تشخیص (به طور گسترده ای در محصولات میکروسافت استفاده می شود)
- تشخیص طبی
- فیلتر نمودن نامه های الکترونیکی که ناشناس می باشند [۳]
- کاربرد در سیستم های خبره (کنترل وسایل ،مانیتورینگ)

---

[1] sampling
[2] Monte Carlo
[3] spam filtering

- کنترل روبوتیک

خلاصه

- شبکه های بیزی یک ارایه ی طبیعی را برای استقلال شرطی مهیا می نمایند .
- توپولوژی + CPT ها = ارایه ی فشرده ی توزیع پیوسته .
- معمولا تولیدش برای خبره ها (غیر خبره ها) آسان می باشد .
- توزیعات استاندارد (مانند OR اغتشاش) = ارایه ی فشرده ی CPT ها .
- متغیر های پیوسته ، سبب توزیعات پارامتربندی شده (گاوسی خطی) می شوند .
- استنتاج دقیق با حذف متغیر :
  - اگر آزمایش را در چند بار انجام دهیم (چند زمانی) در درخت های چندگانه ، NP-hard را در گراف های عمومی سبب می شود .
  - فضا = زمان ، برای توپولوژی ، خیلی قابل حس است

منابع :

http://www.cs.usfca.edu
http://aima.eecs.berkeley.edu/slides-pdf/chapter14a.pdf
http://aima.eecs.berkeley.edu/slides-pdf/chapter14b.pdf

فصل شانزدهم

مدل های زمانی احتمال

مدل های زمانی احتمال [1]
فصل شانزدهم

ریوس مطالب

- **زمان و عدم قطعیت**
- **استنتاج : فیلترینگ [2] ، پیش بینی [3] ، هموارسازی [4]**
- **مدل های مخفی مارکوف [5]**
- **فیلترهای کالمن [6] (یک اشاره ی مختصر)**
- **شبکه های پویای بیزی [7]**
- **فیلترینگ ذره ای [8]**

زمان و عدم قطعیت - جهان تغییر می کند ؛ ما نیاز داریم که زمان را اندازه گیری و پیش گویی نماییم .

مدیریت دیابت (مرض قند) و مدیریت ابزار

- ایده ی اساسی : کپی کردن حالت و متغیرهای ثابت برای هر مرحله ی زمان

X_t = مجموعه ای از متغیرهای وضعیت غیر قابل مشاهده در زمان t . به عنوان مثال ، قند خون در لحظه ی t ($BloodSugar_t$) ، محتویات معده در زمان t ($StomachContents_t$) و .... .

E_t = مجموعه ای از متغیرهای ثابت قابل مشاهده در زمان t ، مثلا ، اندازه ی قند خون در زمان t ($MeasuredBloodSugar_t$) ، اندازه ی ضربان در لحظه ی t ($PulseRate_t$) ،

---

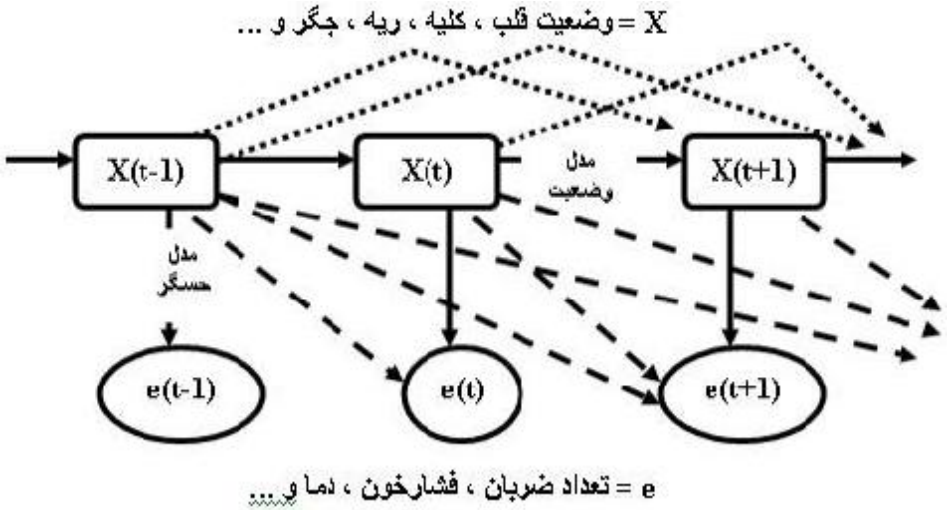
[1] temporal probability models
[2] filtering
[3] prediction
[4] smoothing
[5] Hidden Markov models
[6] Kalman filters
[7] Dynamic Bayesian networks
[8] Particle filtering

غذای خورده شده در زمان t ($FoodEaten_t$) . موارد فوق ، مفروضات زمانی مستقل می باشند ؛ اندازه ی مرحله ، وابسته به زمان می باشد .

یادداشت : $X_{a:b} = X_a, X_{a+1}, ..., X_{b-1}, X_b$

کاهش دادن وابستگی های شرطی

- فقط ارتباط های شرطی اساسی باید در مدل های ما باشند . در غیر این صورت ، خیلی پیچیده خواهد بود .
- نکاتی برای کاهش تعداد وابستگی های متغیر :
  - مدل حسگر - دلیل در مجموع وابسته به وضعیت جاری می باشد . اتصال های به رنگ قرمز (خط چین های بزرگ تر) را برمی دارد .
  - فرض مارکوف [1] - وضعیت جاری = (تاریخچه ی محدود وضعیت های قبلی) f . اتصال های به رنگ آبی (خط چین های کوچک تر) را برمی دارد .
  - فرض پردازه ی ثابت – مش های [2] شرطی یکسان در هر مرحله .



پردازش های مارکوف (زنجیره های مارکوف)

- تولید یک شبکه ی مارکوف از متغیر ها :

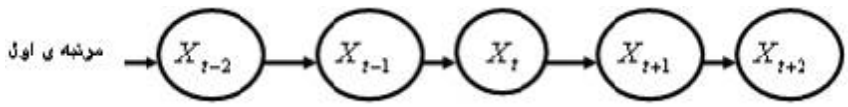
فرض مارکوف : X_t وابسته به زیر مجموعه ای کراندار از $X_{0:t-1}$ باشد .

پردازش مرتبه ی اول مارکوف :

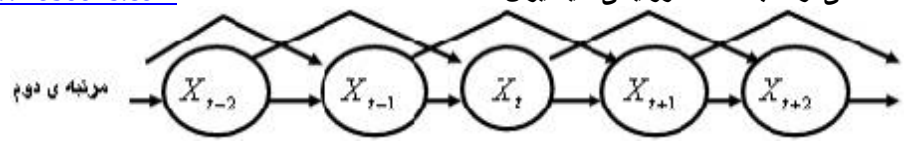
$$P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-1})$$

پردازش مرتبه ی دوم مارکوف :

$$P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-2}, X_{t-1})$$



[1] Markov Assump
[2] mechs

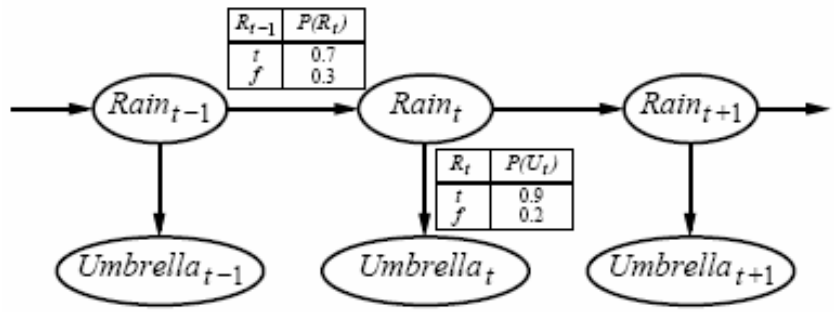


فرض حسگر مارکوف :

$$P(E_t \mid X_{0:t}, E_{0:t-1}) = P(E_t \mid X_t)$$

پردازش ثابت : مدل وضعیت $P(X_t \mid X_{t-1})$ و مدل حسگر $P(E_t \mid X_t)$ برای همه ی مقادیر t ثابت می باشد .

مثال



فرض مرتبه ی اول مارکوف ، دقیقا در دنیای واقعی صحیح نمی باشد ! .

موارد ممکن :

۱. افزایش مرتبه [1] ی پردازش مارکوف

۲. تکمیل وضعیت [2] ، مثال ، اضافه نمودن دما در لحظه ی t ($Temp_t$) و فشار در لحظه ی t ($\Pr essure_t$)

مثال : حرکت روبات .

تکمیل وضعیت و سرعت با زمان باتری ($Battery_t$)

کارهای استنتاجی

- فیلترینگ : $P(X_t \mid e_{1:t})$.

وضعیت – ورودی ، برای پردازش تصمیم گیری یک مولفه ی مستدل می باشد .

پیش بینی : $P(X_{t+k} \mid e_{1:t})$ برای k>0 : ارزیابی کارهای پشت سر هم ممکن ؛ شبیه فیلترینگ بدون ثابت می باشد .

هموارسازی : $P(X_k \mid e_{1:t})$ برای $0 \le k < t$ تخمین بهتر وضعیت های گذشته ، برای یادگیری ضروری می باشد .

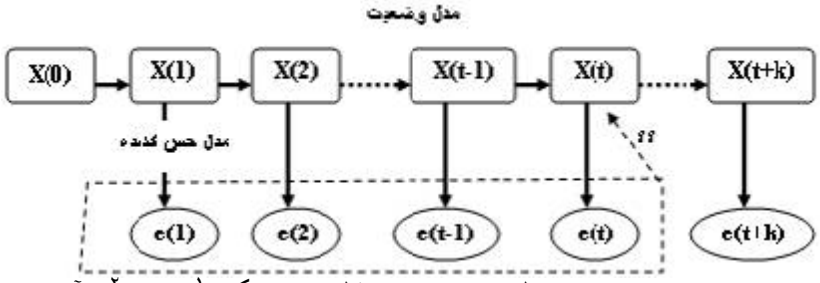
شبیه ترین توضیح : $\arg\max_{x_{1:t}} P(x_{1:t} \mid e_{1:t})$

شناخت سخن و رمز گشایی با یک کانال دارای اغتشاش .

مرورکلی بر فیلترینگ

---

[1] Increase order
[2] Augment state



داده : شاهدی از زمان صفر به زمان حال (به عنوان مثال ، خستگی [1] ، تب [2] ، آبریزش بینی [3])
محاسبه : احتمال یک وضعیت داده شده ی جاری (به عنوان مثال ، آمفلوانزا [4])
فیلترینگ ـ هدف : کشف یک الگوریتم تخمین بازگشتی حالت :

$$P(X_{t+1} \mid e_{1:t+1}) = f(e_{t+1}, P(X_t \mid e_{1:t}))$$

$$P(X_{t+1} \mid e_{1:t+1}) = P(X_{t+1} \mid e_{1:t}, e_{t+1})$$
$$= \alpha P(e_{t+1} \mid X_{t+1}, e_{1:t}) P(X_{t+1} \mid e_{1:t})$$
$$= \alpha P(e_{t+1} \mid X_{t+1}) P(X_{t+1} \mid e_{1:t})$$

توجه کنید ، پیش بینی + تخمین . پیش بینی با جمع کردن X_t :

$$P(X_{t+1} \mid e_{1:t+1}) = \alpha P(e_{t+1} \mid X_{t+1}) \sum_{x_t} P(X_{t+1} \mid x_t, e_{1:t}) P(x_t \mid e_{1:t})$$
$$= \alpha P(e_{t+1} \mid X_{t+1}) \sum_{x_t} P(X_{t+1} \mid x_t) P(X_t \mid e_{1:t})$$

در جایی که $f_{1:t} = P(X_t \mid e_{1:t})$ ، $f_{1:t+1} = Forward(f_{1:t}, e_{t+1})$ می باشد

.

زمان و فضا ثابت هستند و مستقل از t می باشند .

تعمیم یافته ی قانون بیز

$$P(Y \mid X, e) = \frac{P(X \mid Y, e) P(Y \mid e)}{P(X \mid e)}$$

اثبات :

$$P(Y \mid X, e) = \frac{P(X, Y, e)}{P(X, e)} = \frac{P(X \mid Y, e) P(Y, e)}{P(X, e)} = \frac{P(X \mid Y, e) P(Y \mid e) P(e)}{P(X \mid e) P(e)} = \frac{P(X \mid Y, e) P(Y \mid e)}{P(X \mid e)}$$

تعمیم یافته ی قانون بیز در فیلترینگ

---

[1] tiredness
[2] fever
[3] runny nose
[4] influenza

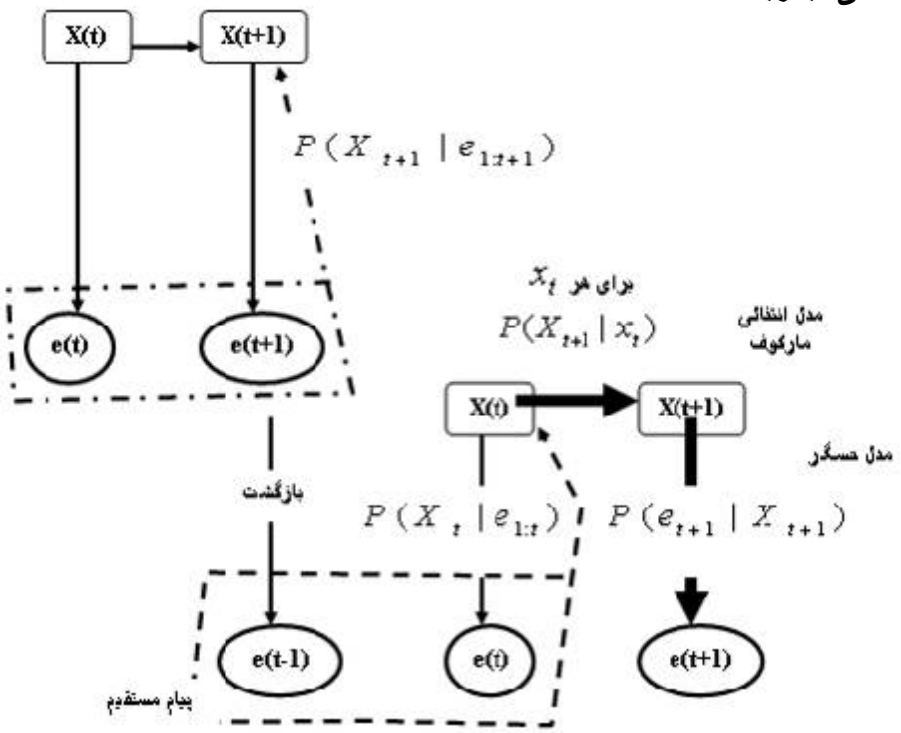
$$P(X_{t+1} \mid e_{1:t+1}, e_t) = \frac{P(e_{t+1} \mid X_{t+1}, e_{1:t}) P(X_{t+1} \mid e_{1:t})}{P(e_{t+1} \mid e_{1:t})} = \alpha P(e_{t+1} \mid X_{t+1}, e_{1:t}) P(X_{t+1} \mid e_{1:t})$$

$P(e_{t+1} \mid e_{1:t})$ ثابت نرمال سازی صحیح است زیرا :

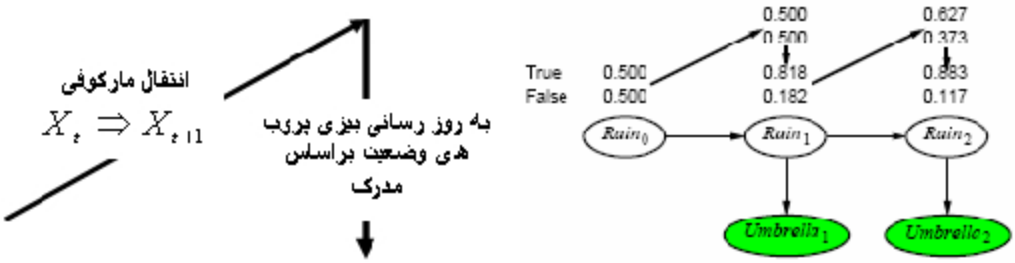
- در زمینه ی تولید توزیع $\forall x_{t+1} \in X_{t+1}$ و
- ما در میان همه ی مسیرهای ممکن از $e_{1:t}$ تا e_{t+1} از طریق استفاده از همه ی x_{t+1} ممکن در صورت کسر قانون بیز کلی حرکت می کنیم .
- بنابراین مجموع این صورت کسرها برای همه ی x_{t+1} ممکن برابر است با : $P(e_{t+1} \mid e_{1:t})$.

بردن بزرگ : هم اکنون ما در حال استفاده از مدل شرطی (حسگر) برای e_{t+1} (از طریق $P(e_{t+1} \mid X_{t+1}, e_{1:t})$) که به $P(e_{t+1} \mid X_{t+1})$ کاهش می یابد می باشیم .
رسیدگی کردن به $P(X_{t+1} \mid e_{1:t})$ آسان می باشد ، چون که به وضعیت های مارکوف کاهش می یابد و یک فراخوان بازگشتی می باشد .

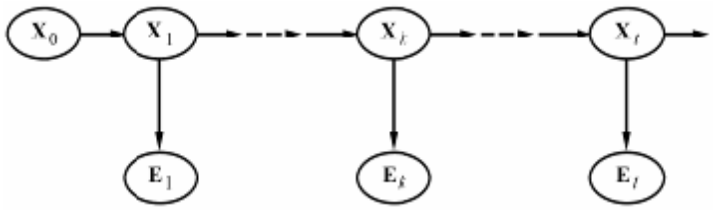
طبیعت بازگشتی فیلترینگ



مثال فیلترینگ



هموارسازی



$e_{1:t}$ را به $e_{1:k}, e_{k+1:t}$ تقسیم کنید :

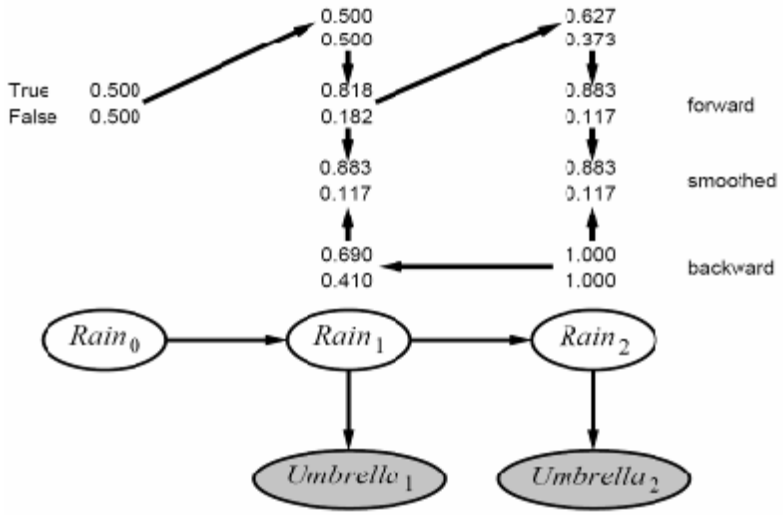
$$P(X_k \mid e_{1:t}) = P(X_k \mid e_{1:k}, e_{k+1:t})$$
$$= \alpha P(X_k \mid e_{1:k}) P(e_{k+1:t} \mid X_k, e_{1:k})$$
$$= \alpha P(X_k \mid e_{1:k}) P(e_{k+1:t} \mid X_k)$$
$$= \alpha f_{1:k} b_{k+1:t}$$

پیام معکوس با بازگشت های معکوس محاسبه می شود :

$$P(e_{k+1:t} \mid X_k) = \sum_{x_{k+1}} P(e_{k+1:t} \mid X_k, x_{k+1}) P(x_{k+1} \mid X_k)$$
$$= \sum_{x_{k+1}} P(e_{k+1:t} \mid x_{k+1}) P(x_{k+1} \mid X_k)$$
$$= \sum_{x_{k+1}} P(e_{k+1} \mid x_{k+1}) P(e_{k+2:t} \mid x_{k+1}) P(x_{k+1} \mid X_k)$$

سه قسمت از جمع نهایی : مدل حسگر ، مرحله ی بازگشتی ، مدل انتقال

مثال هموارسازی



الگوریتم مستقیم – معکوس : پیام های در طول مسیر مستقیم را ذخیره نمایید .
زمان در t (استنتاج چند درختی) و فضا (O(t|f|)) به صورت خطی می باشد .

شبیه ترین توضیح

شبیه ترین مسیر برای هر x_{t+1} = شبیه ترین مسیر برای برخی از x_t ها به اضافه ی یک مرحله بیش تر .

$$\max_{x_1 \dots x_t} P(x_1, \dots, x_t, X_{t+1} \mid e_{1:t+1}) =$$

$$P(e_{t+1} \mid X_{t+1}) \max_{x_t} (P(X_{t+1} \mid x_t) \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, x_t \mid e_{1:t}))$$

برای فیلترینگ یکسان است به غیر از موقعی که $f_{1:t}$ جایگزین می شود با

$$m_{1:t} = \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, X_t \mid e_{1:t})$$

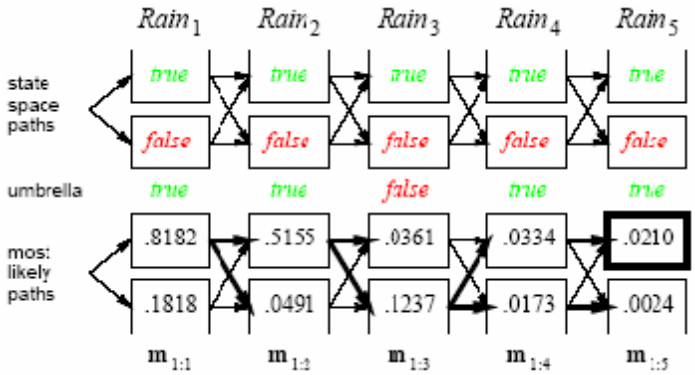
توجه کنید که $m_{1:t}(i)$ ، احتمال بهترین مسیر را برای وضعیت i برمی گرداند . به روز کردن دارای جمعی جایگزین شده با max می باشد و ارایه کننده ی الگوریتم ویتربی [1] می باشد :

$$m_{1:t+1} = P(e_{t+1} \mid X_{t+1}) \max_{x_t} (P(X_{t+1} \mid x_t) m_{1:t})$$

الگوریتم ویتربی

برای یک مدل مخفی مارکوف مشخص ، الگوریتم ویتربی برای پیداکردن محتمل ترین رشته از وضعیت های پنهان یک رشته از وضعیت های مشاهده شده ، به کار می رود . [2]

مثال ویتربی



مدل های مخفی مارکوف ـ X_t از یک متغیر منفرد و گسسته استفاده می کند (معمولا E_t هم همین جور است) . دامنه ی X_t ، $\{1,..,S\}$ می باشد . ماتریس تبدیل به صورت $T_{ij} = P(X_t = j \mid X_{t-1} = i)$ می باشد ، به عنوان مثال ، $\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$

[1] Viterbi
[2] http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/viterbi_algorithm/s4_pg1.html

ماتریس حسگر [1] O_t برای هر گام زمانی ، با عناصر قطری برابر $P(e_t \mid X_t = i)$ می باشد ، برای مثال ، با $U_1 = true$ داریم : $O_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$

پیام های مستقیم و معکوس به صورت بردارهای ستونی می باشند :

$$f_{1:t+1} = \alpha O_{t+1} T^{\perp} f_{1:t} , \quad b_{k+1:t} = T O_{k+1} b_{k+2:t}$$

الگوریتم مستقیم – معکوس به زمان $O(S^2 t)$ و فضای O(St) نیازمند می باشد .

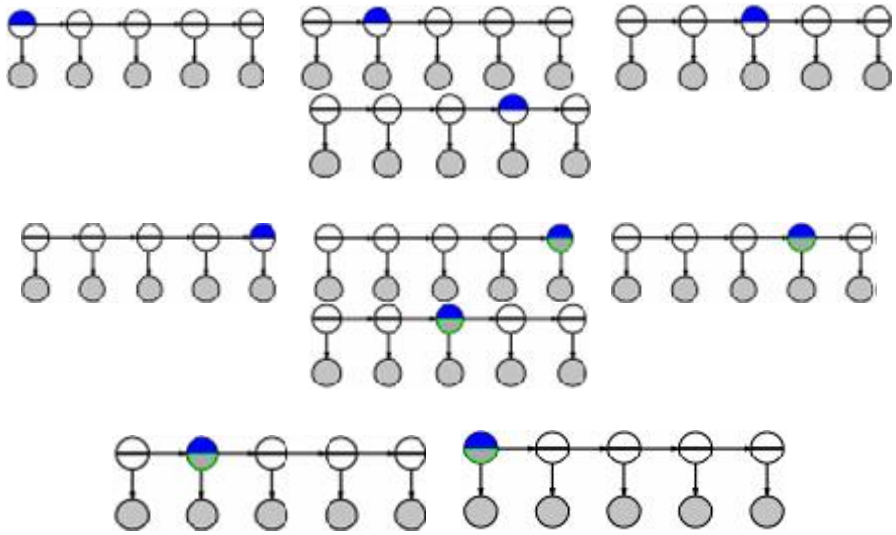
الگوریتم رقص کشور (مکان ها) - می توانیم از نگهداری همه ی پیام های مستقیم ، در هموارسازی ، با اجرای معکوس الگوریتم های مستقیم جلوگیری نماییم :

$$f_{1:t+1} = \alpha O_{t+1} T^{\perp} f_{1:t}$$

$$O_{t+1}^{-1} f_{1:t+1} = \alpha T^{\perp} f_{1:t}$$

$$\alpha'(T^{\perp})^{-1} O_{t+1}^{-1} f_{1:t+1} = f_{1:t}$$

الگوریتم : اجرای مستقیم f_t را محاسبه می کند و اجرای معکوس f_i , b_i را انجام می دهد .



شبکه های بیزی پویا [2]

X_t و E_t به طور قراردادی شامل تعدادی متغیر در یک شبکه ی بیزی تکرار شده می باشد .

احتیاجات شبکه ی بیزی پویا

برای ساختن یک شبکه ی بیزی پویا به موارد زیر نیاز داریم :

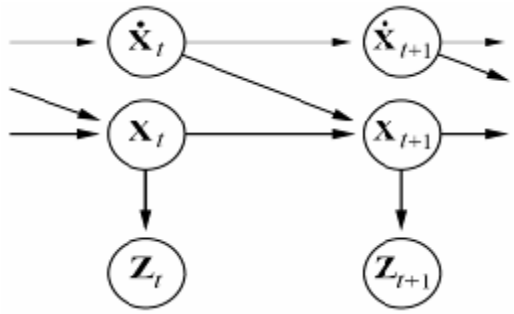
- احتمال های قبلی برای متغیرهای وضعیت ورودی : مقادیر X_0 .
- مدل انتقال : $P(X_{t+1} \mid X_t)$.

[1] sensor matrix
[2] Dynamic Bayesian Networks (DBNs)

- مدل حسگر : $P(E_t \mid X_t)$.

با داشتن موارد فوق ، ما می توانیم فیلترینگ ، پیش گویی و هموارسازی و بهترین کارهای توصیفی را انجام دهیم . اما این کار می تواند پر هزینه باشد . هر پرس و جوی پروب شرطی به استدلال کننده ی بیزی استناد می کند ، که می تواند :

- یک جواب دقیق را از راه شمارش تولید نماید .
- یک جواب تقریبی را از راه نمونه برداری تولید نماید .



فیلترهای کالمن - سیستم های مدل بندی توصیف شده با یک مجموعه از متغیرهای پیوسته می باشند ، به عنوان مثال ، ضبط یک پرنده ی در حال پرواز ___ $X_t = X, Y, Z, \dot{X}, \dot{Y}, \dot{Z}$ ، هواپیماها ، روبات ها ، اکوسیستم ها (زیستگاه ها) [1] ، اقتصاد ، کشت گیاهان شیمیایی از فیلترهای کالمن استفاده می کنند .

گاوس ابتدایی ، مدل وضعیت خطی گاوس و مدلی از حسگر از فیلتر کالمن استفاده می نمایند .

به روزرسانی توزیعات گاوس

- مرحله ی پیش بینی : اگر $P(X_t \mid e_{1:t})$ گاوسی باشد ، آن گاه پیش بینی به صورت $P(X_{t+1} \mid e_{1:t}) = \int_{x_t} P(X_{t+1} \mid x_t) P(x_t \mid e_{1:t}) dx_t$ گاوسی می باشد . در صورتی که $P(X_{t+1} \mid e_{1:t})$ گاوسی باشد ، در این صورت توزیع به روز رسانی شده ی $P(X_{t+1} \mid e_{1:t+1}) = \alpha P(e_{t+1} \mid X_{t+1}) P(X_{t+1} \mid e_{1:t})$ گاوسی می باشد . با این وجود ، $P(X_t \mid e_{1:t})$ گاوسی چند متغیری $N(\mu_t, \sum_t)$ برای همه ی مقادیر t می باشد .

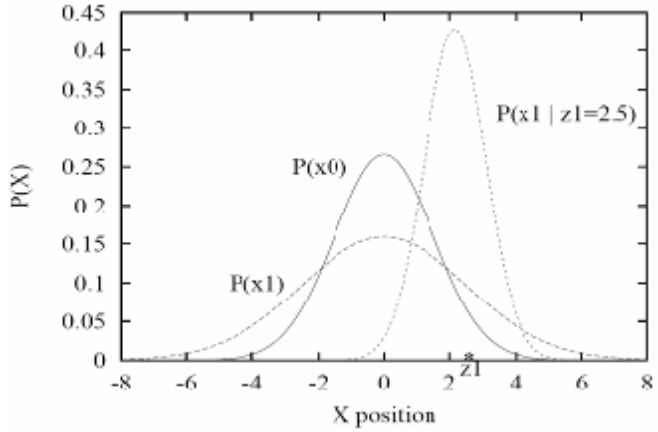
در پردازش عمومی (غیر خطی ، غیر گاوسی) : توصیف قبلی به صورت نامحدود به صورت $t \to \infty$ رشد می کند .

مثال ساده ی یک بعدی - گاوس تصادفی بر روی محور X ها حرکت می کند :

$$\mu_{t+1} = \frac{(\sigma_t^2 + \sigma_x^2) z_{t+1} + \sigma_z^2 \mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}$$

$$\sigma_{t+1}^2 = \frac{(\sigma_t^2 + \sigma_x^2) \sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}$$

[1] ecosystems



به روز رسانی عمومی کالمن

- وضعیت و مدل های حسگر :

$$P(x_{t+1} \mid x_t) = N(Fx_t, \sum_x)(x_{t+1})$$

$$P(z_t \mid x_t) = N(Hx_t, \sum_z)(z_t)$$

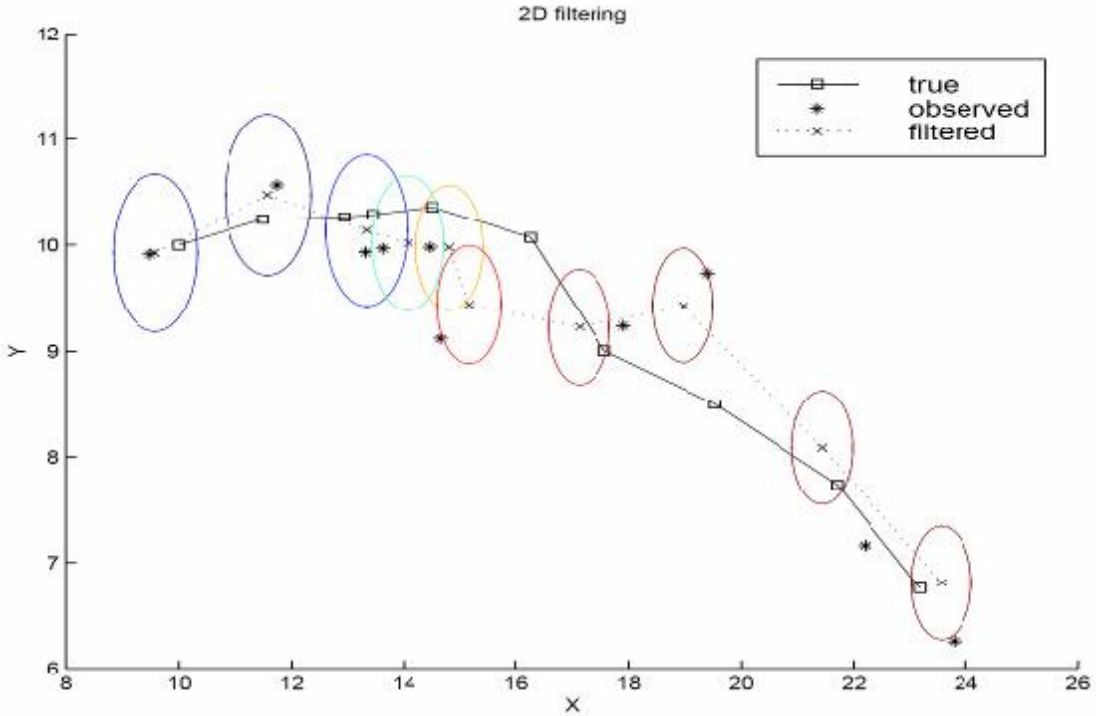
F ماتریس انتقال است ؛ $\sum_x$ وضعیت اغتشاش کوواریانس می باشد . H ماتریسی برای حسگرها است ؛ $\sum_z$ حسگر اغتشاش کوواریانس می باشد . فیلتر ، به روز رسانی زیر را محاسبه می کند :

$$\mu_{t+1} = F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t)$$

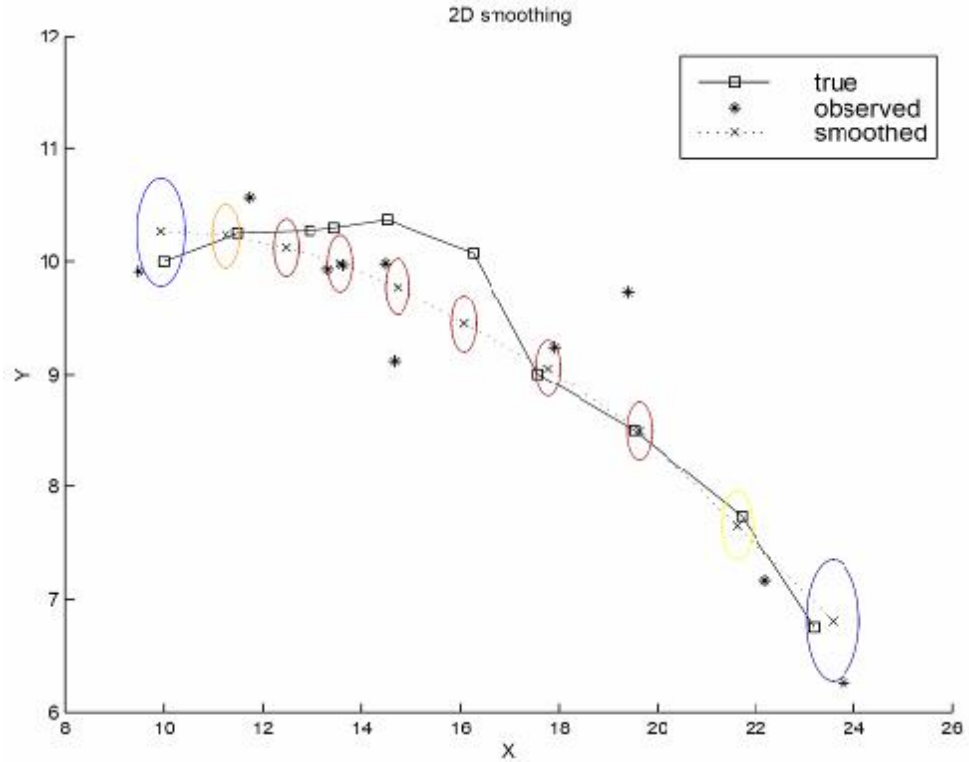
$$\sum_{t+1} = (I - K_{t+1})(F\sum_t F^{\perp} + \sum_x)$$

در جایی که $K_{t+1} = (F\sum_t F^{\perp} + \sum_x)H^{\perp}(H(F\sum_t F^{\perp} + \sum_x)H^{\perp} + \sum_z)^{-1}$ ماتریس بهره ی کالمن می باشد . و K_t مستقل از ترتیب مشاهده هستند ، بنابراین به صورت برون خطی محاسبه می شوند .

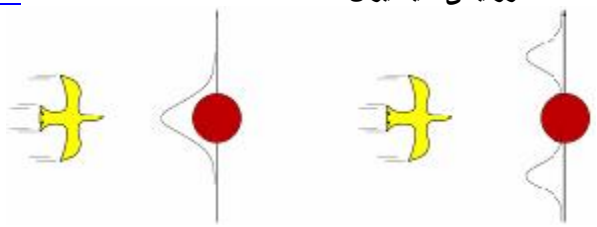
مثال جریان دو بعدی : فیلترینگ



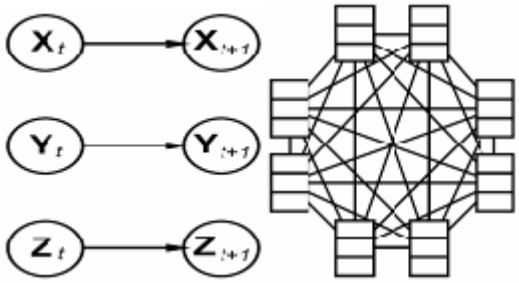
مثال جریان دوبعدی : هموارسازی



کجا پاره می شود ؟ - در صورتی که مدل وضعیت غیر خطی باشد نمی تواند به کارگرفته شود . مدل های وضعیت فیلتر توسعه داده شده ی کالمن به صورت محلی ، خطی در مورد $x_t = \mu_t$ می باشند . در صورتی که سیستم ها به صورت محلی ناهمسان باشند رد می شوند .

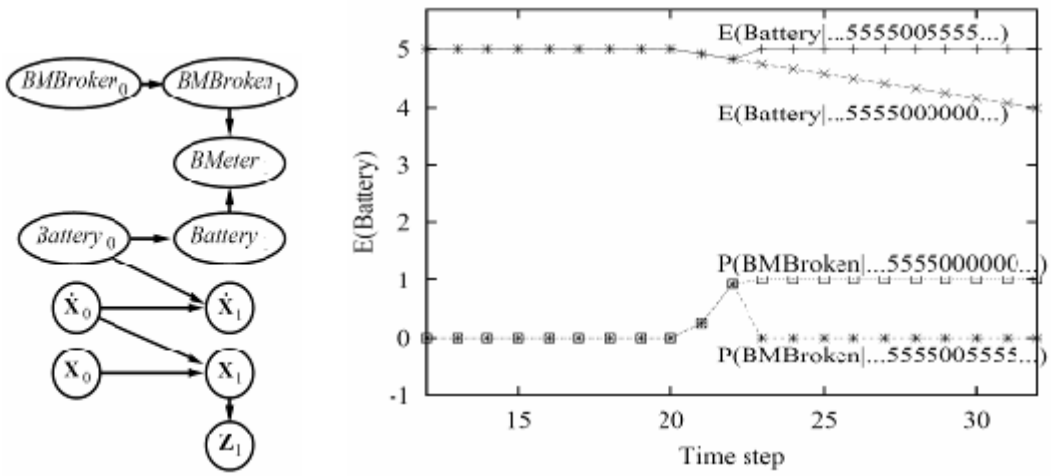


DBN ها و HMM ها - هر HMM یک DBN تک متغیری می باشد ؛ هر DBN گسسته یک HMM می باشد .



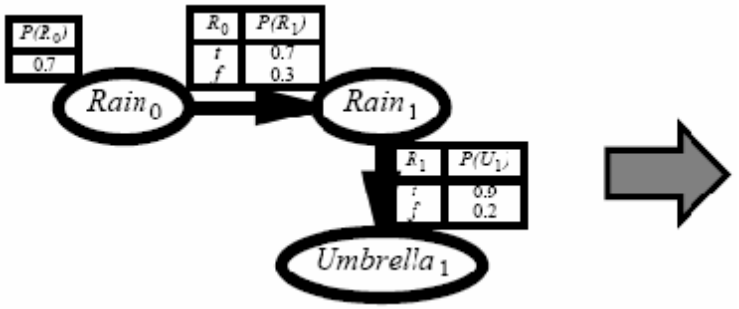
وابستگی اسپارس است ← به صورت نمایی دارای پارامترهای کم تری هستیم ؛ به عنوان مثال ، با متغیرهای بیست حالت ، سه والد در هر DBN دارای $20 \times 2^3 = 160$ پارامتر می باشد و HMM دارای $2^{20} \times 2^{20} \approx 10^{12}$ می باشد .

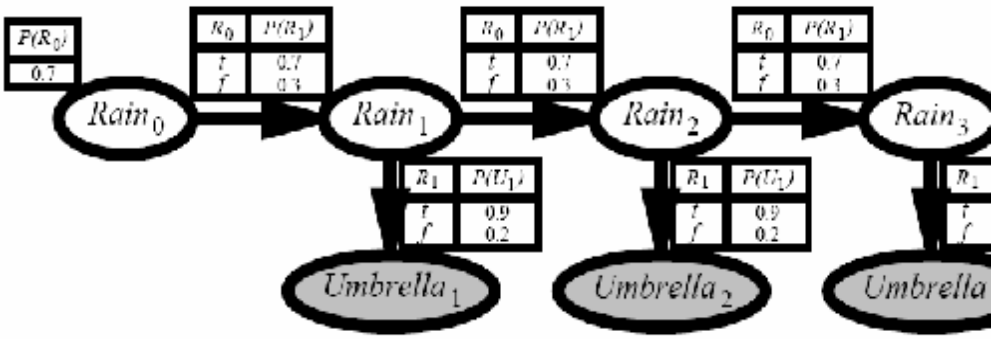
DBN ها و فیلترهای کالمن - هر فیلتر کالمن ، یک DBN می باشد ولی تعداد کمی از DBN ها فیلترهای کالمن (KFs) می باشند ؛ دنیای واقعی به غیرگاوسی های قبلی نیازمند است . مثلا سطل پر و کلیدهای من کجا هستند ؟ شارژ باتری چیست ؟



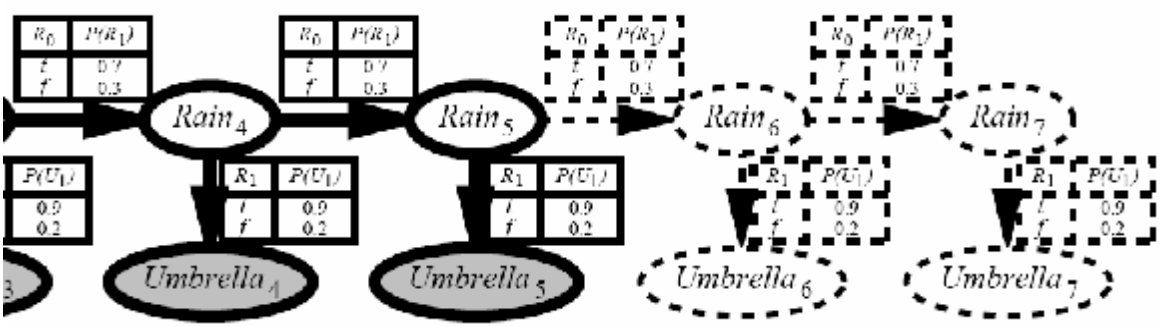
استنتاج دقیق در DBN ها - متد ساده [1] : بازکردن شبکه و اجرای هر الگوریتم دقیق

[1] Naïve method





بقیه ی شکل در ادامه

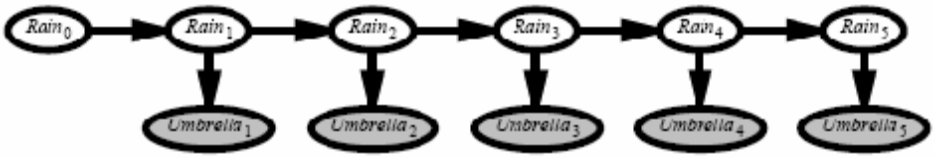


هزینه ی استنتاج برای هر به روز رسانی با t رشد می کند .

فیلترینگ جمع کننده [1] : تکه ی t+1 را اضافه کن ، تکه های t را با استفاده از حذف متغیر جمع کن . بزرگ ترین فاکتور $O(d^{n+1})$ می باشد و هزینه ی به روز رسانی $O(d^{n+2})$ می باشد .

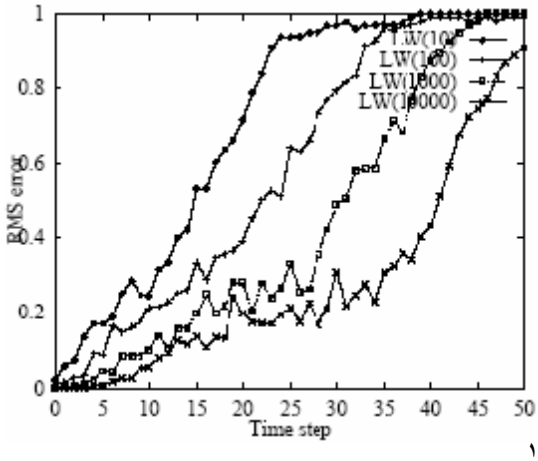
(هزینه ی به روز رسانی HMM برابر است با $O(d^{2n})$)

توزیع احتمال برای DBN ها - مجموعه ی نمونه های توزیع شده وضعیت مورد تصور را تخمین می زند .



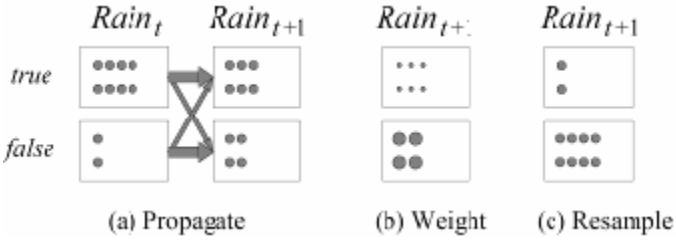
نمونه های LW هیچ توجهی به ثابت (شاهد) ندارند ! ← شکاف " توافق " به صورت نمایی با t افت می نماید ← تعداد نمونه های لازم به صورت نمایی همراه با t رشد می کند .

[1] Rollup filtering



فیلترینگ ذره ای (جزیی) [1]

- ایده ی اساسی : مطمین شوید که تعداد نمونه ها (" ذره ها ") در مناطقی با احتمال بالای فضای حالت جریان می یابد . تکرار ذره ها را متناسب با احتمال ، برای e_t انجام دهید .



به طور گسترده ای برای سیستم های غیرخطی جریان و .esp در آینده استفاده می شوند . همچنین برای شبیه سازی های تایین جا و نگاشت در روبوت های متحرک با ۱۰۰۰۰۰ بعد فضای حالت استفاده می شوند .

الگوریتم :

۱. N رویداد نمونه را برای همه ی متغیرهای وضعیت با استفاده از پروب های قبلی و ابتدایا با X_0 تولید نمایید .
۲. t=0
۳. برای هر نمونه ی x_t ، آن را با x_{t+1} با استفاده از مدل وضعیت و $P(X_{t+1} \mid X_t)$ منتشر نمایید .
۴. $weight(x_{t+1}) = P(e_{t+1} \mid x_{t+1})$.
۵. نمونه گیری مجدد جمعیت براساس وزن ها ی نمونه ها در t+1 :
   a. N رویداد نمونه ی جدید را از دوره ی جدید N نمونه انتخاب نمایید .
   b. نمونه های خیلی توزین شده چند بار انتخاب خواهند شد .
   c. نمونه های کم توزین شده کم انتخاب خواهند شد
۶. اگر در مرحله ی پایان بودیم : براساس پروب های وضعیت در تکه های کوچک متوقف شوید .
۷. در غیر این صورت t=t+1 و به مرحله ی ۳ بروید .

فیلترینگ ذره - سازگاری را در زمان t در نظر بگیرید :

$$N(x_t \mid e_{1:t}) / N = P(x_t \mid e_{1:t})$$

[1] particle filtering

انتشار مستقیم [1] : برای تعداد x_{t+1} برابر است با :

$$N(x_{t+1} \mid e_{1:t}) = \sum_{x_t} P(x_{t+1} \mid x_t) N(x_t \mid e_{1:t})$$

نمونه های توزیع با احتمال آن ها برای e_{t+1} برابر است با :

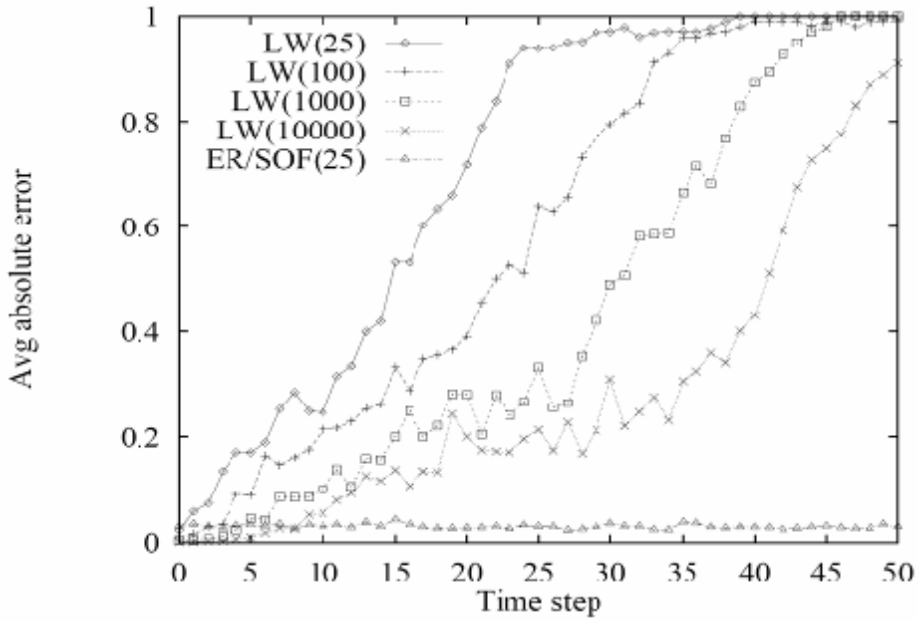
$$W(x_{t+1} \mid e_{1:t+1}) = P(e_{t+1} \mid x_{t+1}) N(x_{t+1} \mid e_{1:t})$$

نمونه گیری مجدد برای گرفتن تعداد مناسب برای W :

$$N(x_{t+1} \mid e_{1:t+1}) / N = \alpha W(x_{t+1} \mid e_{1:t+1}) = \alpha P(e_{t+1} \mid e_{1:t})$$
$$= \alpha P(e_{t+1} \mid x_{t+1}) \sum_{x_t} P(x_{t+1} \mid x_t) N(x_t \mid e_{1:t})$$
$$= \alpha' P(e_{t+1} \mid x_{t+1}) \sum_{x_t} P(x_{t+1} \mid x_t) P(x_t \mid e_{1:t})$$
$$= P(x_{t+1} \mid e_{1:t+1})$$

عملکرد فیلترینگ ذره

خطای تقریب فیلترینگ ذره محدود به زمان باقی مانده و در کم ترین مشاهده می باشد --- تحلیل تیوری مشکل می باشد .



خلاصه - مدل های زمانی ، وضعیت و متغیرهای حسگر را به صورت تکرار شده حول زمان استفاده می کند .

مفروضات مارکوف و فرض آن ثابت می باشد و بنابراین ما لازم داریم :

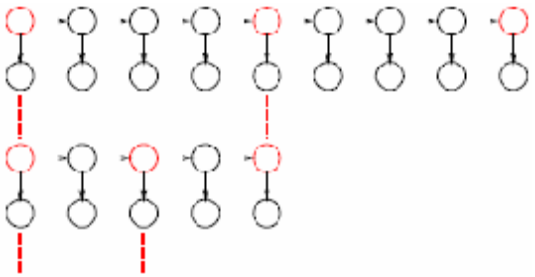
[1] propagate forward

- مدل وضعیت $P(X_t \mid X_{t-1})$
- مدل حسگر $P(E_t \mid X_t)$

کارها ، فیلترینگ و پیش بینی و همسان سازی که بیش تر به صورت متوالی می باشند ، هستند ؛ همه ی این موارد به صورت بازگشتی با هزینه ی ثابت در هر مرحله ی زمانی ، انجام می شوند . مدل های مخفی مارکوف دارای یک متغیر وضعیت گسسته ی منفرد می باشند ؛ برای شناخت سخن از آن ها استفاده می شود .فیلترهای کالمن به n متغیر حالت و گاوسی خطی به $O(n^3)$ به روز رسانی اجازه می دهند . شبکه های بیزی پویا ، HMM ها و فیلترهای کالمن را رده بندی می کنند ؛ به روز رسانی دقیق به صورت تعاملی می باشد .فیلترینگ ذره ، تقریب خوبی از الگوریتم فیلترینگ برای DBN ها می باشد .

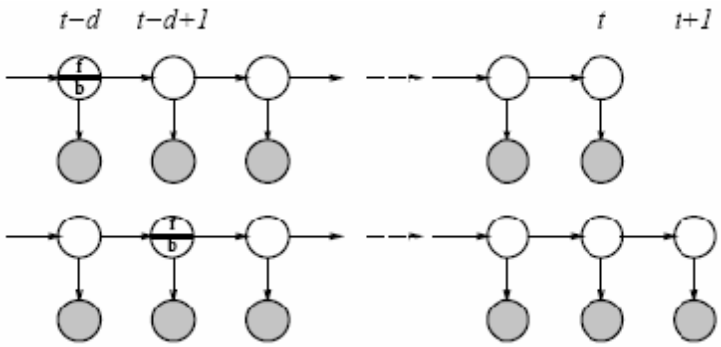
الگوریتم جزیره

ایده : نگهداری کننده ی مستقیم – معکوس f_t و b_t را فقط در k-1 نقطه اجرا نمایید . در زیر عملیات k به صورت بازگشتی اول – عمق را فراخوانی نمایید .



به فضای $O(k \mid f \mid \log_k t)$ و زمان بیش تر از $O(k \log_k t)$ نیازمندیم .

تاخیر زمانی هموارسازی ثابت برخط



روش آشکار مستقیم – معکوس را برای مراحل d در هر لحظه اجرا می نماید
به صورت بازگشتی $f_{1:t-d+1}$ و $b_{t-d+2:t+1}$ را از $f_{1:t-d}$ و $b_{t-d+2:t}$ محاسبه می نماید ؟
پیام مستقیم ، مناسب می باشد ، پیام معکوس به صورت مستقیم قابل دسترسی نمی باشد .

$b_{t-d+2:t+1} = B_{t-d+2:t+1}1$ و $b_{t-d+1:t} = B_{t-d+1:t}1$ تعریف نمایید ، بنابراین $B_{j:k} = \prod_{i=j}^{k} TO_i$

حالا ما می توانیم یک به روز رسانی بازگشتی را برای B داشته باشیم :

$$B_{t-d+2:t+1} = O_{t-d+1}^{-1} T^{-1} B_{t-d+1:t} TO_{t+1}$$

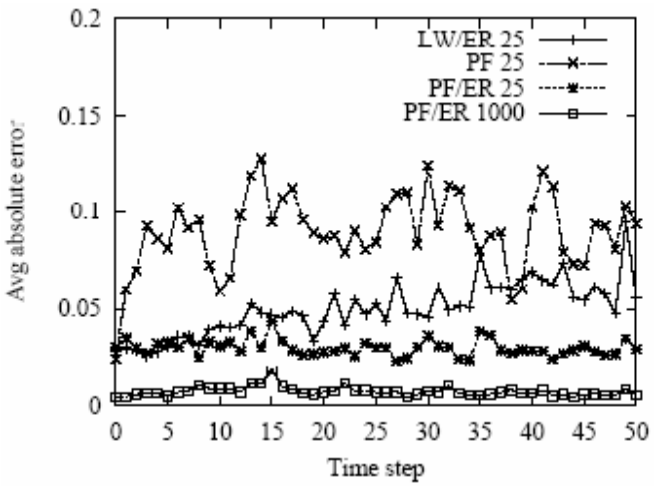
بنابراین هزینه ی به روز رسانی ثابت می باشد و مستقل از تاخیر زمانی d می باشد

استنتاج تقریبی در شبکه های پویای بیزی

فیلترینگ ذره (گوردون [1] ، ۱۹۹۴ ؛ کانازاوا [2] ، کولر [3] و راسل [4] ، ۱۹۹۵ ؛ بلیک [5] و ایسارد [6] ، ۱۹۹۶)
تقریب ضریب [7] (بوین [8] و کولر ، ۱۹۹۹)
انتشار لوپی [9] (پیرل [10] ، ۱۹۸۸ ؛ یدیدیا [11] ، فریمن [12] و ویس [13] ، ۲۰۰۰)
تقریب تنوعی [14] (قهرمانی [15] و جوردن [16] ، ۱۹۹۷)
MCMC محوشده [17] (هنوز منتشر نشده)

نقض مدرک [18]

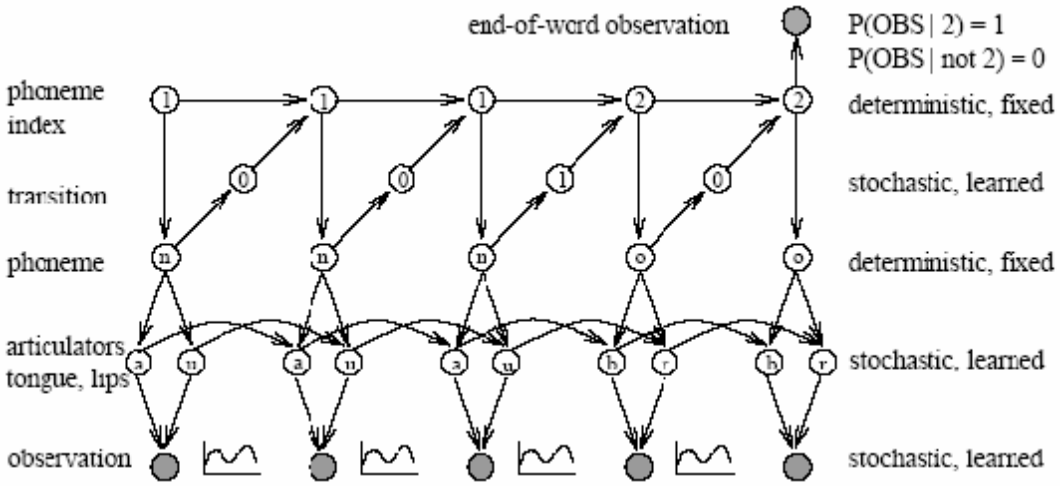
بهتر است که نمونه های شرطی جدیدی را در مدارک جدید تصور نماییم
واریانس تخمین های اولیه را کمینه می کند (کونگ [19] و لیو [20] ، ۱۹۹۶)



---

[1] Gordon
[2] Kanazawa
[3] Koller
[4] Russell
[5] Blake
[6] Isard
[7] factored approximation
[8] Boyen
[9] Loopy propagation
[10] Pearl
[11] Yedidia
[12] Freeman
[13] Weiss
[14] Variational approximation
[15] Ghahramani
[16] Jordan
[17] Decayed MCMC
[18] evidence reversal
[19] Kong
[20] Liu

مثال : شبکه ی پویای بیزی برای سخن شناسی



اضافه کردن متغیرها برای مثال جنسیت [1] ، لهجه [2] و سرعت هم آسان مـی باشد . زوایـگ [3] و راسل ، تا ۴۰% کاهش خطا را برای HMM ها نشان دادند.

منابع :

http://www.idi.ntnu.no/~keithd/classes/advai/lectures/chapter15.pdf
http://www.cs.berkeley.edu/~russell/classes/cs188/f05/slides/chapter15a.pdf
http://www.cs.sfu.ca/~mori/courses/cmpt310/slides/chapter15a-6pp.pdf

[1] gender
[2] accent
[3] Zweig

فصل هفدهم

شناخت سخن یا سخن شناسی (به طور خلاصه)

شناخت سخن یا سخن شناسی [۱] (به طور خلاصه)
فصل هفدهم

ریوس مطالب

- ☒ **سخن به صورت استدلال احتمالی** [۲]
- ☒ **صداهای سخن** [۳]
- ☒ **تلفظ کلمه** [۴]
- ☒ **ترتیب کلمات** [۵]

سخن شناسی پردازه ی توانمندسازی یک کامپیوتر برای شناسایی و واکنش دادن به صداهای به وجود آمده در سخن انسان می باشد [۶] . با تعریفی دیگر ، سخن شناسی یا تشخیص صدا [۷] ، توانایی سیستم های کامپیوتری برای دریافت سخن ورودی و عمل بر روی آن یا بیان آن به صورت نوشته می باشد . تلاش های تحقیقاتی جدید در مورد سخن شناسی اتوماتیک [۸] می باشند ، که هدف آن تغییر محتوای سخن به دانش است که پایه ای برای زبان شناسی [۹] یا کارهای شناختی می باشد ، نظیر ترجمه به زبان دیگر . کاربردهای عملی شامل سیستم های پایگاه داده – پرس و جو [۱۰] و سیستم های بازیابی اطلاعات [۱۱] می باشد . سخن شناسی دارای کاربرد در روبوتیک و مخصوصا توسعه ی روبوت هایی که می توانند "بشنوند" می باشد . [۱۲]

---

[1] Speech recognition
[2] speech as probabilistic inference
[3] speech sounds
[4] word pronunciation
[5] word sequences
[6] فرهنگ آکسفورد
[7] voice recognition
[8] Automatic Speech Recognition (ASR)
[9] linguistic
[10] database-query systems
[11] information retrieval systems
[12] دایره المعارف بریتانیکا

سخن به صورت استدلال احتمالی ـ سیگنال های سخن ، اغتشاش دار [1] ، متغیر و مبهم [2] می باشند . شبیه ترین ترتیب کلمات و سیگنال سخن ارایه شده چیست ؟ توجه کنید کلمات دارای بیش ترین P(Words|signal) را انتخاب نمایید . از قانون بیز استفاده نمایید :

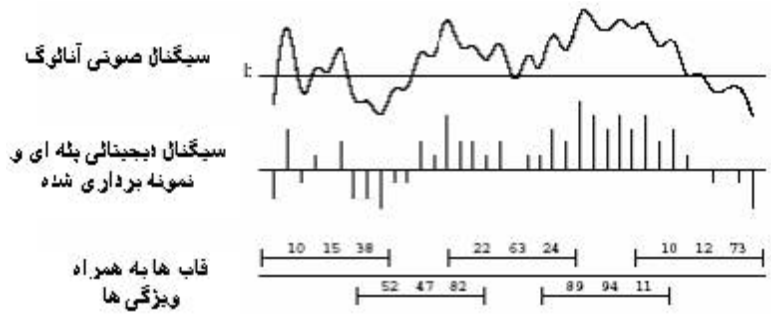
$$P(Words|signal)=\alpha P(signal|Words)P(Words)$$

توجه کنید که ، تجزیه به مدل صوتی [3] + مدل زبانی [4] انجام می شود . **کلمات** [5] ، ترتیب وضعیت های پنهان می باشند و **سیگنال** [6] ، ترتیب مشاهده می باشد . در مورد **اصوات** [7] ، باید بدانیم که ، تمام سخنان بشر ترکیبی از ۴۰ الی ۵۰ صوت می باشد . برای یک سطح میانی وضعیت های مخفی شده ی میان کلمات و سیگنال ها داریم ، مدل صوتی [8] = مدل تلفظ [9] + مدل صوت [10] . آرپابت [11] تایین شده برای انگلیسی آمریکایی به صورت زیر می باشد :

[iy]	beat	[b]	bet	[p]	pet
[ih]	bit	[ch]	Chet	[r]	rat
[ey]	bet	[d]	debt	[s]	set
[ao]	bought	[hh]	hat	[th]	thick
[ow]	boat	[hv]	high	[dh]	that
[er]	Bert	[l]	let	[w]	wet
[ix]	roses	[ng]	sing	[en]	button
٨	٨	٨	٨	٨	٨

برای مثال ، برای کلمه ی " ceiling " داریم : [s iy l en] / [s iy l ix ng] / [s iy l ih ng]

صداهای سخن ـ سیگنال خام میکروفون به صورت یک تابع یا عملکرد از زمان می باشد ؛ در پردازش ، قاب های ۳۰ میلی ثانیه ای روی هم می افتند و همگی به وسیله ی پستی و بلندی اشان توصیف می شوند .



---

[1] noisy
[2] ambiguous
[3] acoustic model
[4] language model
[5] words
[6] signal
[7] phones
[8] acoustic model
[9] pronunciation model
[10] phone model
[11] ARPAbet

پستی و بلندی های قاب با فرمت های معمولی می باشند – قله [1] ها دارای طیف [2] قوی می باشند .

مدل های صوت - پستی و بلندی های قاب در P(feautures|phone) به صورت زیر خلاصه می شوند :

- یک عدد صحیح در بازه ی [۰..۲۵۵] (با استفاده از تبدیل برداری) ؛ یا
- پارامترهای مخلوط گاوسی

اصوات سه حالته [3] **:** هر صوت دارای سه وجه می باشد (آغاز [4] ، وسط [5] ، پایان [6]) ، به عنوان مثال ، حرف [t] دارای ابتدای آرام [7] ، وسط قوی [8] ، انتهای خشن [9] می باشد در نتیجه P(features|phine,phase)

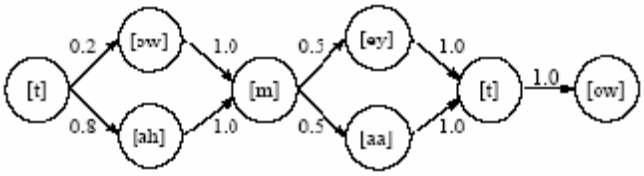
متن سه صوتی [10] **:** هر صوت دارای n^2 صوت متمایز می باشد و وابسته به صوت های چپ و راست خود می باشد ، به عنوان مثال ، [t] در کلمه ی "star" نوشته می شود [t(s,aa)] (متمایز از "tar" !) . سه صوتی ها برای استفاده در هم تلفظی ها [11] مفیدند : تلفظ کننده ها [12] دارای ثبات (ماند یا اینرسی) [13] هستند و نمی توانند به سرعت [14] میان وضعیت ها تغییر (سوییچ) کنند ، به عنوان مثال ، [t] در کلمه ی "eighth" (به معنی یک هشتم) دارای تلفظی مقابل جلوی دندان می باشد .

مثالی برای مدل صوتی - صوت HMM برای [m] :

احتمالات خروجی برای صوت HMM :

Onset:	Mid:	End:
C1: 0.5	C3: 0.2	C4: 0.1
C2: 0.2	C4: 0.7	C6: 0.5
C3: 0.3	C5: 0.1	C7: 0.4

مدل های تلفظ کلمه - هر کلمه به صورت یک توزیع براساس ترتیب صوت ، توصیف می شود .
توزیع ارایه شده به صورت یک مدل وضعیت HMM به صورت زیر می باشد :



---

[1] peak
[2] spectrum
[3] Three-state phones
[4] Onset
[5] Mid
[6] End
[7] silent Onset
[8] explosive Mid
[9] hissing End
[10] Triphone context
[11] coarticulation
[12] articulators
[13] inertia
[14] instantaneously

P([towmeytow]|"tomato")=P([tomaatow]|"tomato")=0.1
P([tahmeytow]|"tomato")=P([tahmaatow]|"tomato")=0.4

ساختار به صورت دستی ساخته می شود و احتمال وضعیت ها از داده ها به دست می آیند .

کلمات گسسته [1] - مدل های صوتی + مدل های کلمه با احتمال ثابت $P(e_{1:t} \mid word)$ برای کلمه ی مستقل می باشد .

$$P(word \mid e_{1:t}) = \alpha P(e_{1:t} \mid word) P(word)$$

احتمال قبلی P(word) به سادگی با شمارش فرکانس های کلمه به دست می آید . $P(e_{1:t} \mid word)$ می تواند به صورت بازگشتی محاسبه شود : تعریف می کنیم ، $\lambda_{1:t} = P(X_t, e_{1:t})$ و از به روز رسانی بازگشتی $\lambda_{1:t+1} = Forward\ (\lambda_{1:t}, e_{t+1})$ استفاده می کنیم و سپس $P(e_{1:t} \mid word\ ) = \sum_{x_t} \lambda_{1:t}(x_t)$

سیستم های دیکته [2] ی کلمات مجزا با تلاش به دقت [3] ۹۵ – ۹۹% رسیدند .

سخن پیوسته - فقط یک رشته کلمات مجزا مسایل را به وجود نمی آورند !

- کلمات کنار هم خیلی زیاد به هم وابسته اند
- توالی شبیه ترین کلمات ≠ شبیه ترین توالی کلمات
- قطعه بندی [4] : مقدار کمی گپ (فاصله) در سخن وجود دارد .
- تلفظ به همراه هم کلمات – مثل "next thing"

سیستم های پیوسته ی سخن ۶۰ – ۸۰% دقت را در یک موقعیت خوب دارند .

مدل زبان - احتمال یک توالی کلمه با قانون زنجیری ارایه می شود :

$$P(w_1 \ldots w_n) = \prod_{i=1}^{n} P(w_i \mid w_1 \ldots w_{i-1})$$

مدل بیگرام [5] :

$$P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

شمارش همه ی جفت کلمات درون یک رشته ی متنی طولانی را انجام دهید . مدل های بهتر (گرامرها و ...) مقداری کم تر کمک می کنند .

HMM ترکیب شده - وضعیت های مدل زبان + کلمه + صوت ترکیب شده ، توسط کلمه ای که در حال ادای آن هستیم + صوتی که در آن کلمه هست + وضعیت صوت درون آن صوت برچسب زده می شود . الگوریتم ویتربی شبیه ترین توالی (رشته ی) وضعیت صوت را پیدا می کند . قطعه بندی با توجه به همه ی رشته های کلمه و محدوده ها انجام می شود ؟ همیشه بهترین ترتیب کلمه را ارایه نمی کند زیرا هر توالی کلمه حول همه ی ترتیب های وضعیت جمع می شود . جلینک [6] در سال ۱۹۶۰ A^* را برای پیداکردن بهترین ترتیب کلمه در جایی که " هزینه ی مرحله ی " آن $-\log P(w_i \mid w_{i-1})$ می باشد را ارایه نمود .

---

[1] Isolated words
[2] Dictation systems
[3] accuracy
[4] segmentation
[5] Bigram model
[6] Jelinek

خلاصه - از اواسط دهه ی ۱۹۷۰ ، شناخت سخن به صورت استدلال احتمالی فرمول بندی شده است . شواهد = سیگنال سخن و متغیرهای مخفی = کلمه و ترتیب های کلمه . جلوه های متن (تلفظ با هم) با تکمیل وضعیت به کار گرفته می شوند . تنوع درون سخن انسان (سرعت ، طنین و غیره و غیره) و اغتشاش در زمینه شناسایی سخن پیوسته را در دنیای واقعی به یک مساله ی باز تبدیل می کند .

منابع :

http://aima.eecs.berkeley.edu/slides-pdf/chapter15b.pdf

فصل هیجدهم

تصمیم گیری های عاقلانه

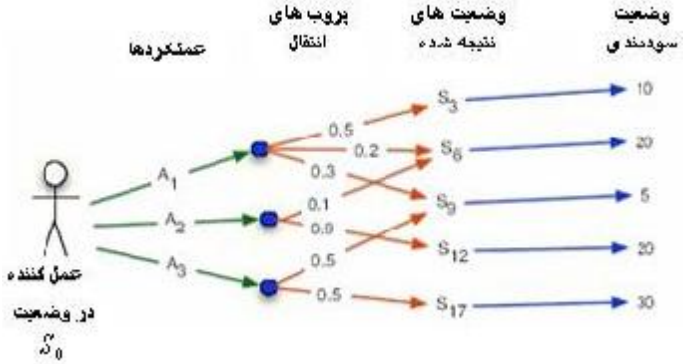
تصمیم گیری های عاقلانه [1]
فصل هیجدهم

ریوس مطالب

- **تقدم های عاقلانه** [2]
- **تسهیلات** [3]
- **پول** [4]
- **تسهیلات چند خصوصیتی** [5]
- **شبکه های تصمیم گیری** [6]
- **ارزش اطلاعات** [7]

تصمیم گیری عاقلانه

کار درست را انجام دهید = عملی که سود مورد انتظار را بیشینه می نماید را انتخاب نمایید



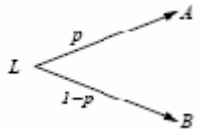
$$EU(A_1) = (0.5)(10) + (0.2)(20) + (0.3)(5) = 10.5$$

$$EU(A_2) = (0.1)(20) + (0.9)(20) = 20\text{،}\quad EU(A_3) = (0.5)(5) + (0.5)(30) = 17.5$$

تقدم ها - یک عامل از میان بهترین ها $(A,B,\ldots)$ و به صورت تصادفی انتخاب می نماید .

---

[1] Rational decisions
[2] Rational preferences
[3] utilities
[4] Money
[5] Multiattribute utilities
[6] Decision networks
[7] Value of information



احتمال L=[p,A; (1-p),B] .
یادداشت :
$A \phi B$:A بـر B مقـدم اسـت . $A \sim B$:بـی تفـاوتی میـان A و B .
$A \underset{\sim}{\phi} B$:B بر A مقدم نمی باشد .

اولویت های عاقلانه

ـ ایده : اولویت های عامل عاقلانه تابع محدودیت ها می باشد . با توجه به اولویت های عاقلانه داریم ، رفتار قابل توضیح به صورت بیشینه سازی سودمندی مورد انتظار می باشد .
محدودیت ها :

توانایی نظم دهی [۱] $(A \phi B) \vee (B \phi A) \vee (A \sim B)$

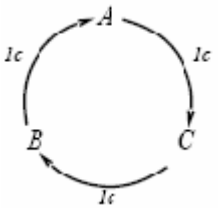
انتقال پذیری [۲] $(A \phi B) \wedge (B \phi C) \Rightarrow (A \phi C)$

پیوستگی [۳] $A \phi B \phi C \Rightarrow \exists p[p, A; 1-p, C] \sim B$

تعویض پذیری [۴] $\Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$ A~B

یکنواختی [۵] $A \phi B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1-p, B] \underset{\sim}{\phi} [q, A; 1-q, B])$

اولویت های عاقلانه ـ تخلف از محدودیت ها ، منجر به نمودها یا آشکارسازی نامعقولانه می شود ، برای مثال : یک عامل با اولویت های لازم می تواند منجر به از دست دادن همه ی پول شود . در صورتی که $B \phi C$ باشد ، آن گاه یک عامل دارای C می باشد باید یک سنت [۶] برای دریافت B پرداخت نماید . در صورتی که $A \phi B$ باشد ، آن گاه یک عامل که دارای B می باشد باید یک سنت برای دریافت A بپردازد . در صورتی که $C \phi A$ باشد ، آن گاه یک عامل که دارای A می باشد باید یک سنت برای دریافت C بپردازد .



بیشینه کردن سودمندی (تسهیلات) مورد انتظار
ـ قضیه (رمزی [۷] ، ۱۹۳۱ ؛ ون نیومن [۸] و مورگنسترن [۹] ، ۱۹۴۴) :
اولویت های ارایه شده ، محدودیت ها یی که در یک تابع مقداردهی شده U وجود دارند را برطرف می کنند تا این که

$$U(A) \geq U(B) \Leftrightarrow A \underset{\sim}{\phi} B$$

$$U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i U(S_i)$$

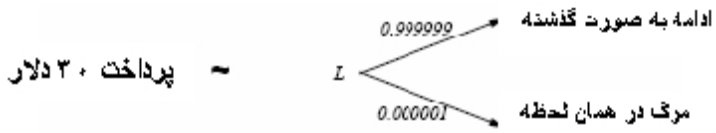
قانون MEU : عملکردی را که سود مورد نیاز را بیشینه می کند را انتخاب نمایید .

---

[1] Orderability
[2] Transitivity
[3] Continuity
[4] Substitutability
[5] Monotonicity
[6] cent که معادل یک صدم دلار آمریکایی می باشد
[7] Ramsey
[8] von Neumann
[9] Morgenstern

نکته : یک عامل می تواند کاملا عاقلانه باشد (سازگار با قانون MEU باشد) بدون حتی ارایه یا به کارگیری سود و احتمالات . مثلا ، یک جدول مراجعه برای تیکتاکتوی کامل .

استاندارد نزدیک به صرفه های بشر : یک وضعیت ارایه شده ی A را با یک احتمال استاندارد L_p که دارای " بهترین ارزش ممکن است " مقایسه کنید u_T دارای احتمال p می باشد . در " بدترین رخداد ممکن " $u_\perp$ دارای احتمال (1-p) که نزدیک به احتمال p است تا زمانی که $A \sim L_p$ است ، می باشد .



مقیاس های تسهیلات – تسهیلات نرمال شده : $u_\top=1.0, u_\perp=0.0$.

موتورهای کوچک [1] **:** شانس یک میلیون کشته برای قمار روسی [2] کافی است ، برای کاهش ریسک تولیدات و ... تلاش کنید . قمار روسی : یک بازی خطرناک شانسی که در آن یک نفر گلوله ای را با استفاده از یک هفت تیر (که به سوی سر یک شخص نشانه رفته است) شلیک می نماید [3] .

QALY ها : کیفیت منطبق شده با سال های زندگی برای تصمیم گیری های پزشکی شامل ریسک ذاتی مفید می باشد .

نکته : تصور ، تغییرناپذیری [4] + تغییر شکل [5] خطی است .

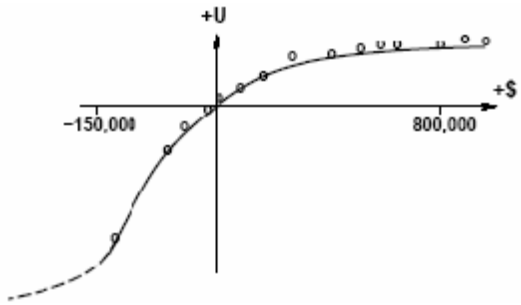
$U'(x) = k_1 U(x) + k_2$ به شرط این که $k_1 > 0$ باشد

در حالت قطعی (بدون انتخاب های شانسی [6]) ، فقط با سود ترتیبی [7] می توانند ادامه یابد ، به نظم موجود توجه نمایید .

پول - پول به صورت یک تابع سودمند رفتار نمی کند . با یک L که به صورت شانسی ارایه شده ، با مقدار مالی مورد انتظار EMV(L) معمولا (U(L)<U(EMV(L)) می باشد ، توجه کنید که افراد مخالف ریسک هستند .

منحنی سودمند [8] **:** با چه مقدار احتمال p من در بی تفاوتی میان یک پاداش x و یک [p,$M;(1-p),$0] شانسی برای M بزرگ هستم ؟

داده های تجربی [9] معمولی و قیاسی با ریسک متعادل به صورت زیر رفتار می کنند :



اهمیت پول

- ارزش (سودمندی) پول وابسته به اندازه ای است که شما از پول دارید .

---

[1] Micromotors
[2] Russian roulette
[3] فرهنگ آکسفورد
[4] invariant
[5] transformation
[6] lottery
[7] ordinal utility
[8] utility curve
[9] empirical

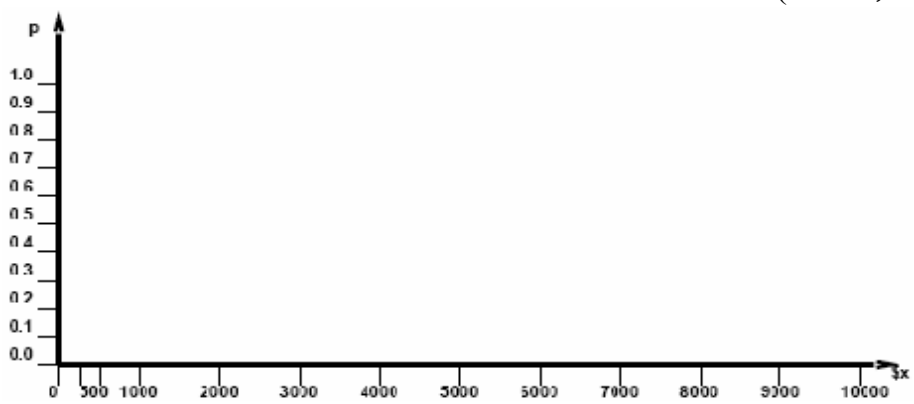
- هزار دلار برای یک فرد فقیر ارزشی بیش تر از ارزش همان مقدار پول برای یک فرد ثروتمند دارد .
- اگر S اندازه ی ارزش پول برای یک فرد باشد ، U(S+$1000)-U(S) برای فرد فقیر ارزشی به مراتب بزرگ تر از ارزشی که برای یک فرد پولدار دارد خواهد داشت .
- توجه کنید که U(Money) یک تابع غیرخطی می باشد : شیب آن برای همه ی مقادیر S یکسان نمی باشد .
- برنولی [1] (۱۷۳۸) : U(Money) تقریبا برابر است با log(Money)

بیزاری از خطر (ریسک) و بیمه [2]

- یک شانس بخت آزمایی L=[p,$M;(1-p),$0] داده شده است ، چه مقدار پول شما خواهید گرفت تا در این بخت آزمایی شرکت نکنید ؟
- در صورتی که X<EMV(L) باشد ، آن گاه شما از ریسک ، بیزار خواهید بود : شما یک مقدار کم تر از بخت آزمایی را دریافت خواهید کرد تا از ریسک بخت آزمایی اجتناب نمایید . X ، **برابری حتمی** [3] بخت آزمایی نام دارد .
- اجرت بیمه [4] = EMV(L)-X .

فرض نمایید که شما دارای یک خانه به ارزش یک میلیون دلار هستید ، فرض کنید که احتمال از دست دادن خانه براثر آتش سوزی برابر یک هزارم باشد . بنابراین شانس شامل مقادیر ممکن که شما باید بپردازید می باشد . $L_h = [p = .001,\$M;(1-p) = .999,\$0]$ و $EMV(L_h) = \$1{,}000$. شما چه مقدار X حاضرید به بیمه بپردازید تا از این شانس آتش سوزی جلوگیری نمایید ؟ باقی مانده ی $EMV(L_h) - X$ مقداری است که شما به بیمه خواهید پرداخت . اگر شما خیلی از ریسک متنفر باشید ، آن گاه X کم خواهد بود و شما هزینه ی زیادی برای اجرت بیمه جهت جلوگیری از عواقب اقتصادی فردی خواهید پرداخت .

سودمندی گروه دانشجویان - برای هر x ، p تا نصف رای کلاس به صورت شانسی تنظیم می شود . (M=10,000) .



یک مثال واقعی بخت آزمایی

- L=[p=.02,$10;q=.000005,$1,000,000;(1-p-q)=.9799995,$0]
- پول بلیط بخت آزمایی برابر یک دلار می باشد .

---

[1] Bernoulli
[2] insurance
[3] certainty equalent
[4] insurance premium

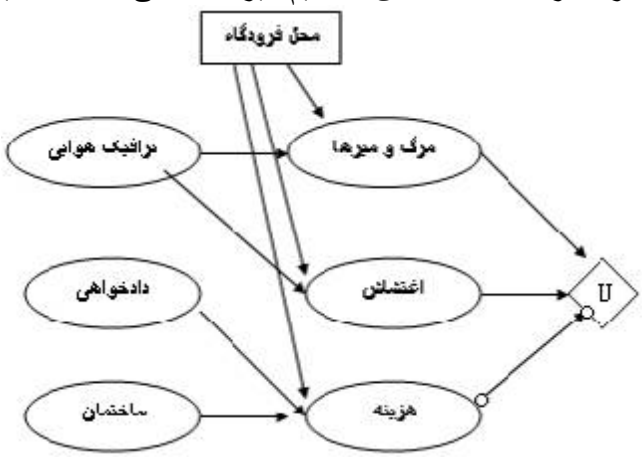
- EMV(L)=?
- چه هنگام معقولانه است که یک بلیط بخت آزمایی را بخریم ؟

جواب :

- در ابتدا ، هزینه ی بلیط را در عواقب بخت آزمایی به کار می بریم .
- L=[p=.02;$9;q=.0000005,$999,999;(1-p-q)=.9799995,$-1]
- EMV(L)=(.02)(9)+(.0000005)(999,999)+(.9799995)(-1)=$-0.3
- در صورتی که ما از هزینه ی بلیط ، چشم پوشی نماییم EMV(L)=$+0.7

شبکه های تصمیم گیری

گره های عملکرد (action nodes) و گره های سودمند (utility nodes) را به شبکه های رفتاری برای فعال ساختن تصمیم گیری منطقی اضافه نمایید

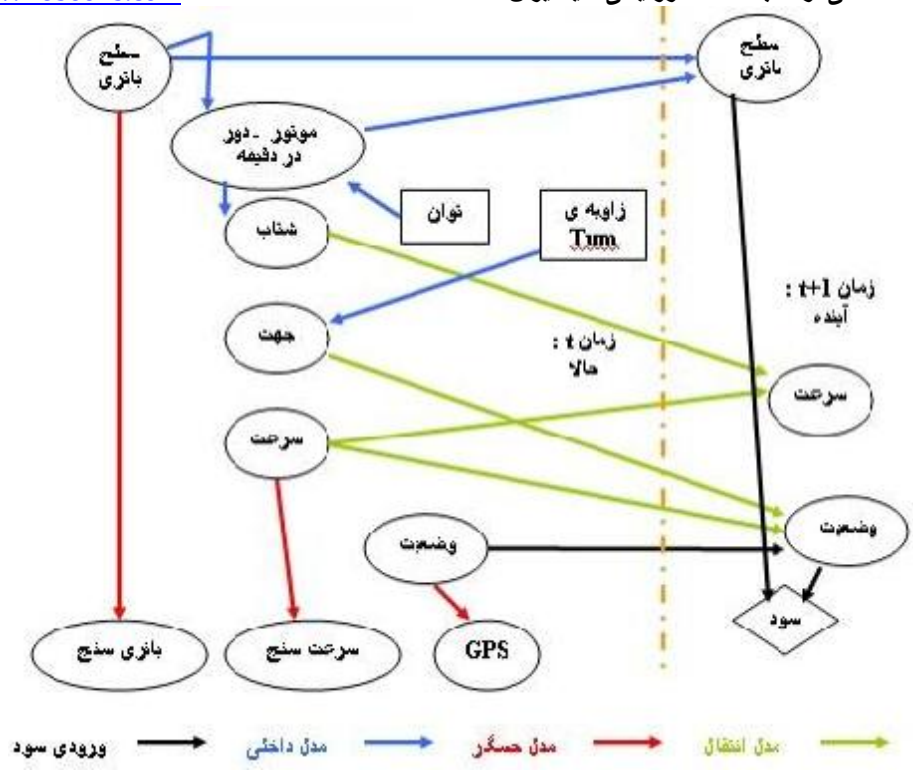


الگوریتم :

برای هر مقدار گره ی عملکرد ، مقدار مورد انتظار سودمندی عملکرد و ثبات (پایداری) گره ی ارایه شده را محاسبه نمایید

عملکرد MEU را برگردان

شبکه ی تصمیم گیری روبوت



ارزیابی شبکه ی تصمیم گیری

۱. متغیرهای ثابت را برای وضعیت جاری سیستم قرار دهید .
۲. برای هر بردار انتساب های مقدار برای متغیرهای تصمیم گیری (به عنوان مثال ، فرمان یا توان) :
   a. گره های تصمیم گیری را به مقادیر آن ها انتساب دهید .
   b. از استنتاج بیزی برای محاسبه ی احتمال های قبلی همه ی والدهای گره سود استفاده نمایید (دقیق یا تخمینی) .
   c. سود را براساس توزیع احتمال روی مقادیر والدها محاسبه نمایید .
۳. بردار تصمیم گیری دارای بیش ترین سود را برگردان

تصمیم گیری روبوت

۱. مقادیر شناخته شده را به متغیرهای ثابت نسبت دهید :
   باتری سنج ، سرعت سنج و GPS (همه در زمان t)
۲. برای هر ترکیب ممکن از توان و زاویه ی tum :
   a. از استنتاج بیزی برای محاسبه ی توزیع احتمال برای سطح باتری و وضعیت در زمان t+1 استفاده نمایید .
   b. سپس آن توزیع ها را برای محاسبه ی سود به کار ببرید .
۳. تبدیل توان کمبو [1] را با بالاترین سود انجام دهید .

نکته : از سود مقادیر **آینده ی طرح** [2] سطح باتری و وضعیت به صورت پایه ای برای انتخاب عملکردهای جاری ، استفاده نمایید .

عقلانیت و سود بخت آزمایی

---

[1] combo
[2] PROJECTED FUTURE

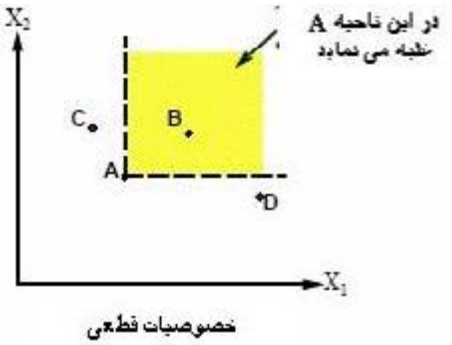
S_k را وضعیت شامل دارایی های جاری افراد قرار دهید . در این صورت سود بخت آزمایی براساس $U(S_{k+n})$ برای مقادیر مختلف n می باشد . با صرف نظر از هزینه ی بلیط : $U(L) = (.02)U(S_{k+10)} + (.0000005)U(S_{k+1,000,000})$. حالا ما می توانیم بپرسیم که آیا $U(L) > U(S_{k+1})$ (سود پس انداز هزینه ی بلیط) و این وابسته به U و K می باشد . در صورتی که $U(L) > U(S_{k+1})$ باشد ، خرید بلیط کاری معقولانه می باشد .

مزیت چند خصوصیتی بودن [1] - چگونه می توانیم توابعی سودمند با تعدادی متغیر $X_1 ... X_n$ را به کار بگیریم ؟ برای مثال U(Deaths,Noise,Cost) چیست ؟ چگونه می توانیم توابع سودمندی که دارای تقدم در رفتار هستند را ترکیب کنیم ؟

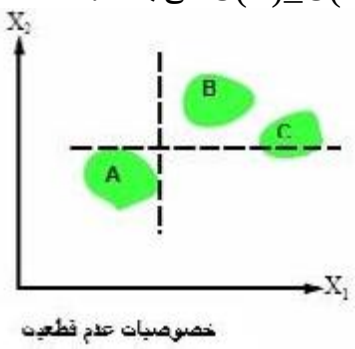
ایده ی ۱ : وضعیت هایی را تحت تصمیم گیری هایی که می توانند بدون شناسنامه ی کامل $U(x_1,...,x_n)$ ساخته شوند را تایین نماییم .

ایده ی ۲ : انواع مختلف مستقل در اولویت ها و مشتق از فرم های استاندارد نتیجه را برای $U(x_1,...,x_n)$ شناسایی نماییم .

تسلط (نفوذ) سخت [2] - معمولا خصوصیاتی را برای U که در هر نفوذ سخت یکنواخت است ، تعریف می کنیم : انتخاب B به صورتی شدید نفوذ بر انتخاب A دارد ، اگر : $\forall i X_i(B) \geq X_i(A)$ و بنابراین U(B)≥U(A) می باشد .



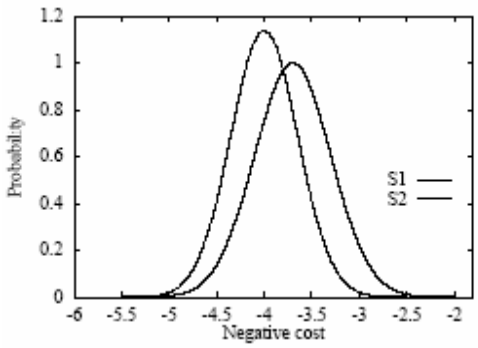
خصوصیات قطعی

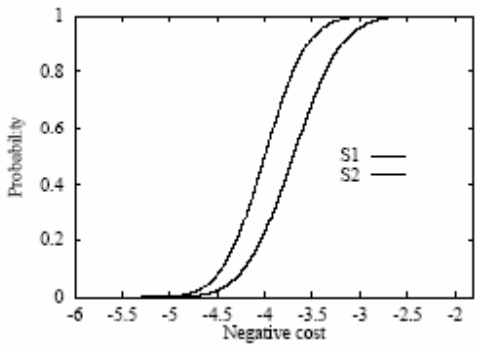


خصوصیات عدم قطعیت

نفوذ سخت در عمل به ندرت اتفاق می افتد

نفوذ اتفاقی





[1] Multiattribute utility
[2] strict dominance

توزیع p_1 در صورتی بر توزیع p_2 اثر اتفاقی می گذارد که :

$$\forall\ t : \int_{-\infty}^{t} p_1(x)\,dx \ \le\ \int_{-\infty}^{t} p_2(t)\,dt$$

در صورتی که U دارای اثر یکنواخت بر x باشد ، در این صورت A_1 با توزیع منتج (به دست آمده) از p_1 ، بر A_2 ، با توزیع به دست آمده از p_2 اثر اتفاقی می گذارد اگر :

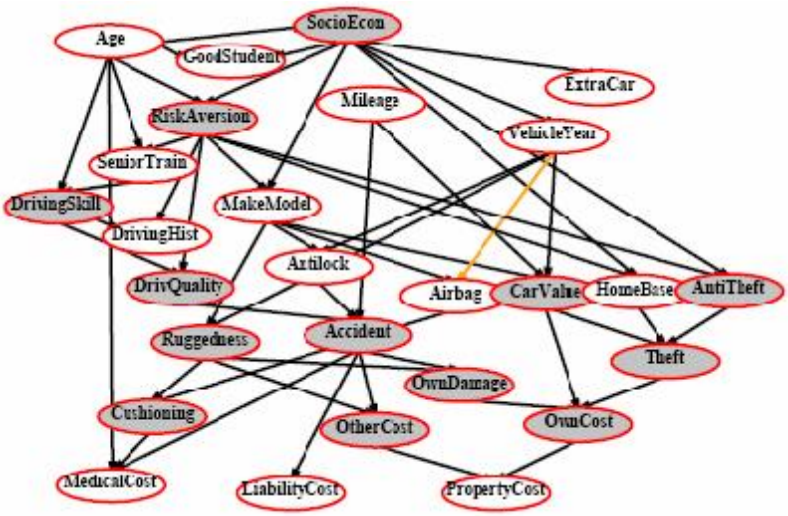
$$\int_{-\infty}^{\infty} p_1(x)U(x)\,dx \ \ge\ \int_{-\infty}^{\infty} p_2(x)U(x)\,dx$$

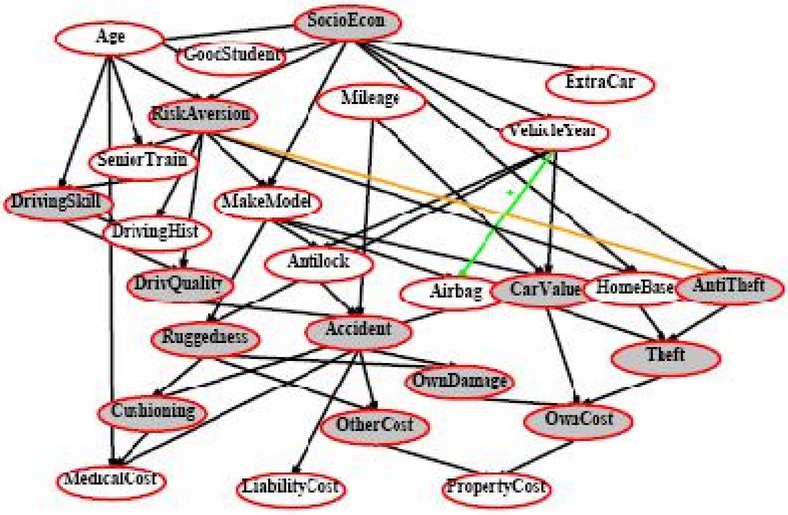
کاربرد چند خصوصیتی : در صورت وجود اثر تصادفی در همه ی خصوصیات ، چند خصوصیتی ، بهینه خواهد بود .

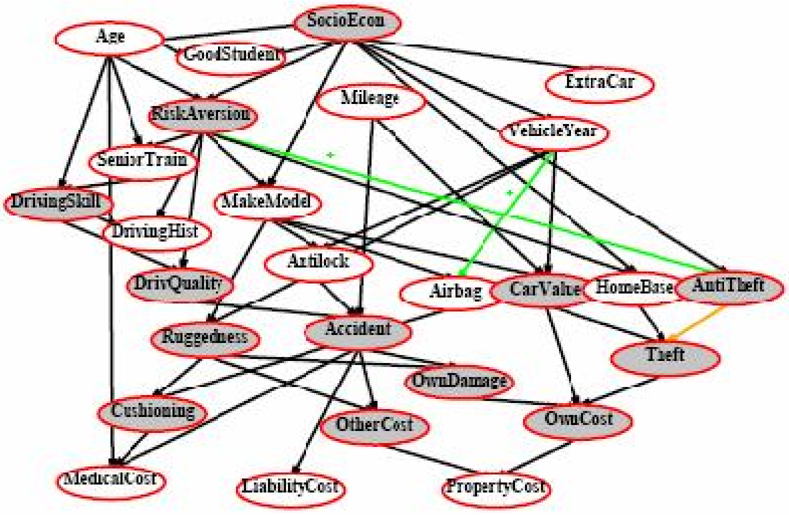
اثر اتفاقی ـ اثر اتفاقی ، اغلب می تواند بدون توزیعات دقیق و با استفاده از چگونگی منطق ، دنبال شود . به عنوان مثال ، هزینه با افزایش فاصله شهر S_1 از شهر S_2 افزایش می یابد ، در نتیجه ، S_1 دارای اثر تصادفی بر S_2 از نظر هزینه می باشد . به طور مثال ، اسیب دیدگی (صدمه) با افزایش سرعت در لحظه ی تصادف افزایش می یابد . می توان رفتار شبکه ها را با اطلاعات اثر تصادفی تفسیر نمود : $X \xrightarrow{+} Y$ (X دارای اثرات مثبت بر Y می باشد) دارای این معنی است که : برای هر مقدار z از سایر والدهای Z متعلق به Y

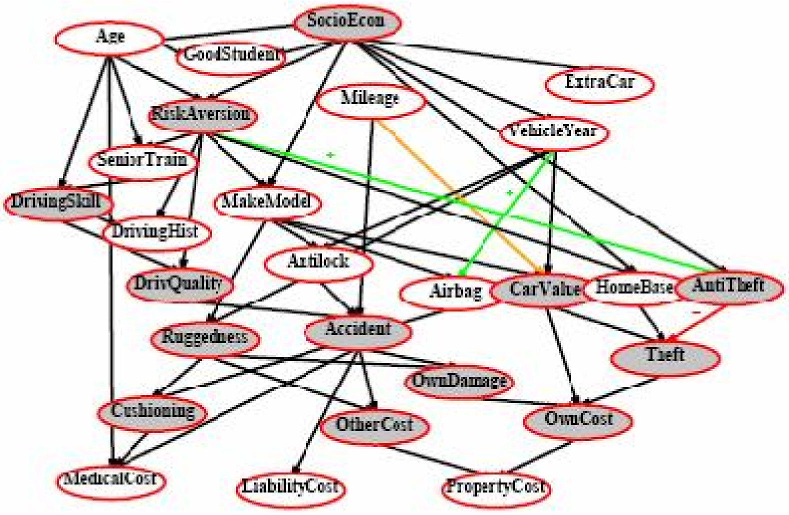
$\forall x_1, x_2 : x_1 \ge x_2 \Rightarrow P(Y \mid x_1, z)$ دارای اثر اتفاقی بر $P(Y \mid x_2, z)$ می باشد .

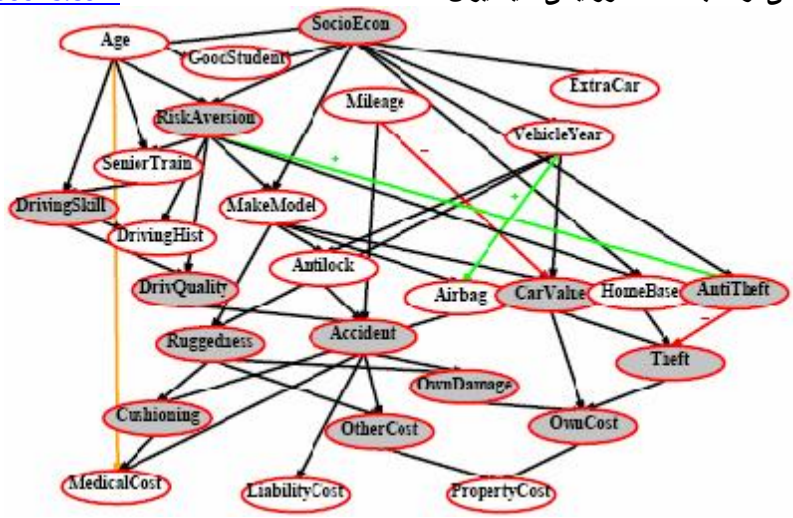
برچسب قوس های + یا –

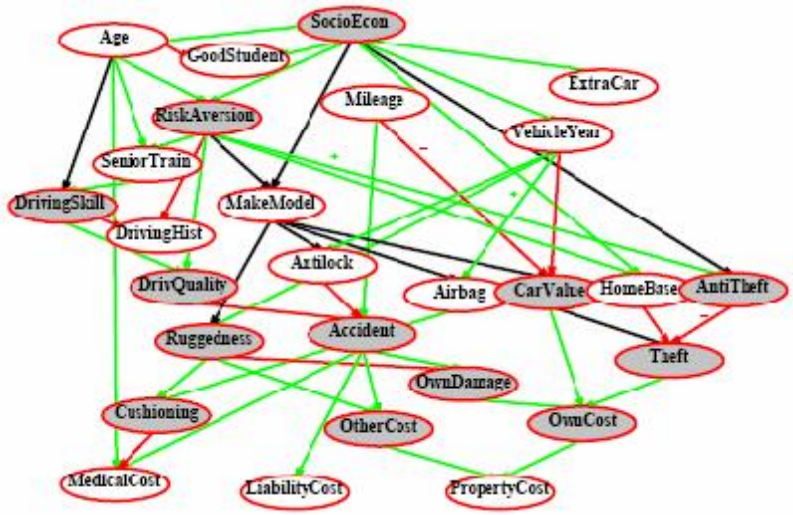



SocioEcon
Age
GoodStudent
ExtraCar
Mileage
RiskAversion
VehicleYear
SeniorTrain
DrivingSkill
MakeModel
DrivingHist
Antilock
DrivQuality
Airbag
CarValue
HomeBase
AntiTheft
Ruggedness
Accident
Theft
OwnDamage
Cushioning
OtherCost
OwnCost
MedicalCost
LiabilityCost
PropertyCost


SocioEcon
Age
GoodStudent
ExtraCar
Mileage
RiskAversion
VehicleYear
SeniorTrain
DrivingSkill
MakeModel
DrivingHist
Antilock
DrivQuality
Airbag
CarValue
HomeBase
AntiTheft
Ruggedness
Accident
Theft
OwnDamage
Cushioning
OtherCost
OwnCost
MedicalCost
LiabilityCost
PropertyCost


SocioEcon
Age
GoodStudent
ExtraCar
Mileage
RiskAversion
VehicleYear
SeniorTrain
DrivingSkill
MakeModel
DrivingHist
Antilock
DrivQuality
Airbag
CarValue
HomeBase
AntiTheft
Ruggedness
Accident
Theft
OwnDamage
Cushioning
OtherCost
OwnCost
MedicalCost
LiabilityCost
PropertyCost





اولویت ساختار : قطعی

- X_1 و X_2 مستقل از X_3 هستند درصورتی که : اولویت میان $\langle x_1, x_2, x_3 \rangle$ و $\langle x_1', x_2', x_3 \rangle$ وابسته به X_3 نباشد . مثلا در <Noise,Cost,Safety> : <۰,۰۶ و ۴,۶ بیلیون دلار و ۲۰۰۰۰> و <۰,۰۶ و ۴,۲ بیلیون دلار و ۷۰۰۰۰> .

قضیه (لیون تیف [1] ،۱۹۴۷) : اگر هر جفت از خصوصیات .P.I متمم خود می باشد ، بنابراین هر زیر مجموعه از خصوصیات .P.I متمم خود می باشد : .P.I دوسره .

قضیه (دبرو [2] ، ۱۹۶۰) : در صورتی که .P.I دو سره باشد ، در نتیجه تابع دارای مقدار افزودنی است :

$$V(S) = \sum_i V_i(X_i(S))$$

بنابراین به دست آوردن توابع تک خصوصیتی [3] n تایی ؛ اغلب با تقریب خوبی انجام می شود .

---

[1] Leontief
[2] Debreu
[3] single-attribute

اولویت ساختار : اتفاقی - لازم است که اولویت های شانس را بدانیم : X دارای استقلال سودمندی Y [1] می باشد ، در صورتی که اولویت های شانس در X وابسته به y نباشند .

U.I. دوسره : هر زیر مجموعه U.I. ، مکمل خود می باشد ، در نتیجه ، افزایش دهنده ی سودمندی تابع می باشد :

$$U = k_1U_1 + k_2U_2 + k_3U_3 + k_1k_2U_1U_2 + k_2k_3U_2U_3 + k_3k_1U_3U_1 + k_1k_2k_3U_1U_2U_3$$

روال زیر برنامه ها و بسته های نرم افزاری برای تولید تست های اولویت ، خانواده هایی با استاندارد های متفاوت توابع سودمند را معرفی می نماید .

ارزش اطلاعات

- ایده : مقدار به دست آمده برای هر جزء ثابت ممکن را محاسبه نمایید . این کار به طور مستقیم از شبکه ی تصمیم گیری می تواند انجام شود .

مثال : تمرین خرید صحیح روغن . از دو قوطی A و B ، دقیقا یکی محتوی روغن می باشد ، قیمت را k در نظر بگیرید . احتمال هر کدام را هم ۰,۵ در نظر بگیرید . قیمت فعلی هر قوطی k/2 می باشد . " متخصص " پیشنهاد بررسی A را می دهد . آیا قیمت مناسب است ؟

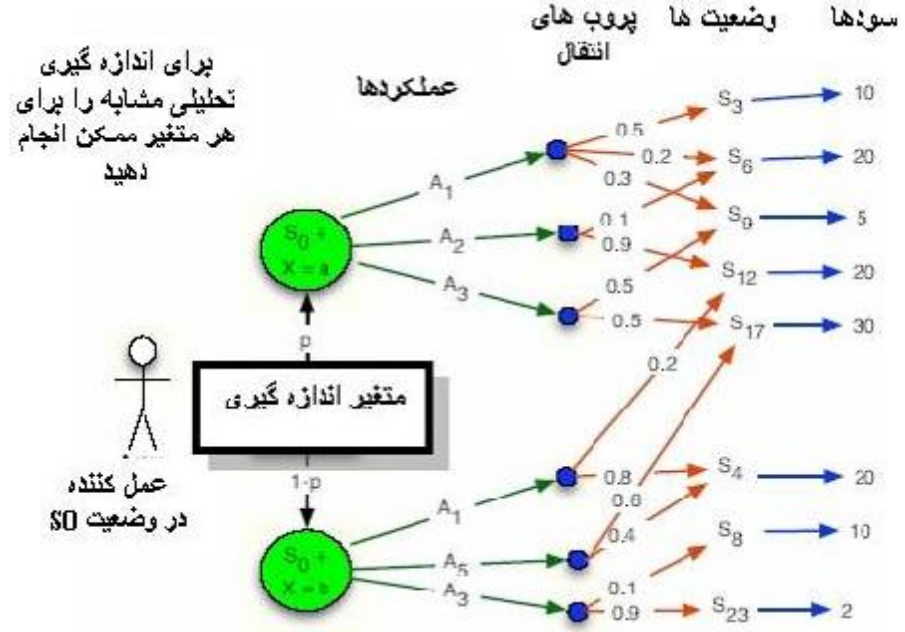
راه حل : مقدار مورد انتظار اطلاعات را محاسبه نمایید = مقدار مورد انتظار بهترین عملکرد اطلاعات ارایه شده منهای مقدار مورد انتظار بهترین عملکرد بدون اطلاعات . بررسی ممکن است بگوید که " روغن در A است " یا " روغن در A نمی باشد " و احتمال ۰,۵ برای هر کدام داده شده است ! = [۰,۵ * ارزش داده شده ی " خرید A " و " روغن درون A می باشد " + ۰,۵ ارزش " خرید B " و " روغن درون A نمی باشد "] – ۰

$$=(0.5*k/2)+(0.5*k/2)-0=k/2$$

انتخاب اندازه گیری [2]

- اطلاعات معمولا مستقل نمی باشند .
- به دست آوردن اندازه گیری ها می تواند پرهزینه باشد : آزمایش های طبی ، تحلیل های شیمیایی اعماق دریا و ...
- احتیاج به اولویت بندی اندازه ها یا آزمایش ها داریم .
- آن هایی که منجر به وضعیت های با بالاترین سود مورد انتظار می شوند را انتخاب نمایید .
- اندازه گیری ها ← اطلاعات وضعیت ← انتخاب عملکردها ← وضعیت های کلی جدید (info+physical) ← ارزشیابی سود [3]
- از آنجایی که تعدادی از این اتصالات به صورت اتفاقی می باشند ، ما به توزیع احتمالی که هم نتایج اندازه گیری و هم عملکردهای منظم را دارا هستند رسیدگی می نماییم .

[1] utility-independent
[2] Measurement Selection
[3] Utility Assessment



فرمول کل - E را ثابت در نظر بگیرید ، بهترین عملکرد جاری ، α می باشد . نتیجه ی عملکرد ممکن S_i و توان بالقوه ی ثابت جدید ، E_j باشد .

$$EU\ (\alpha\ |\ E) = \max_a \sum_i U\ (S_i) P\ (S_i\ |\ E, a)$$

تصور نمایید که ما می دانیم $E_j = e_{jk}$ ، بنابراین ما $\alpha_{e_{jk}}$ را انتخاب می کنیم .

$$EU\ (\alpha_{e_{jk}}\ |\ E, E_j = e_{jk}) = \max_a \sum_i U\ (S_i) P\ (S_i\ |\ E, a, E_j = e_{jk})$$

E_j یک متغیر تصادفی است که مقدار فعلی آن ناشناخته می باشد ، در نتیجه ، باید مقادیر ممکن آن را حول تمام مقادیر ممکن آن محاسبه نماییم :

$$VPI_E(E_j) = (\sum_k P(E_j = e_{jk}\ |\ E) EU\ (\alpha_{e_{jk}}\ |\ E, E_j = e_{jk})) - EU\ (\alpha\ |\ E)$$

(VPI مخفف Value of Perfect Information می باشد ، به معنی ارزش اطلاعات کامل می باشد)

خصوصیات VPI

- **نا منفی** [1] می باشد : $\forall j, E : VPI_E(E_j) \geq 0$

- **جمع ناپذیری** [2] ، در مثال، توجه کنید که E_j دوبار آمده است :

$$VPI_E(E_j, E_k) \neq VPI_E(E_j) + VPI_E(E_k)$$

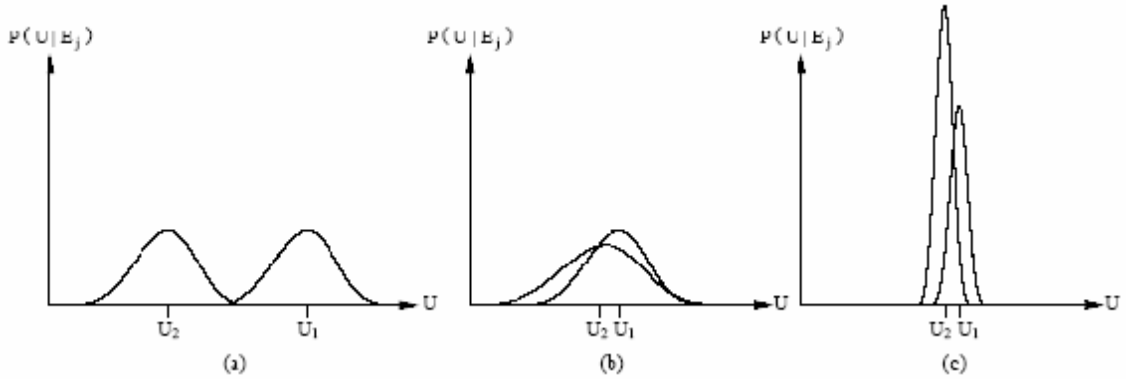
مستقل از نظم بودن [3] :

$$VPI_E(E_j, E_k) = VPI_E(E_j) + VPI_{E,E_j}(E_k) = VPI_E(E_k) + VPI_{E,E_k}(E_j)$$

[1] nonnegative
[2] nonadditive
[3] order-independent

توجه : در زمانی که یک بخش ثابت می تواند جمع آوری شود ، بیشینه کردن VPI برای هر قسمت ، برای انتخاب یکی از آن ها که همیشه هم بهینه نمی باشد منجر به این می شود که جمع کردن ثابت ها تبدیل به یک مساله ی تصمیم گیری ترتیبی شود .

کیفیت (چگونگی) رفتارها - اگر انتخاب ، قابل مشاهده می باشد ، اطلاعات دارای ارزش کمی می باشند . اگر انتخاب ، غیرقابل مشاهده می باشد ، اطلاعات دارای ارزش زیادی می باشند . اگر انتخاب غیرقابل مشاهده می باشد ، اطلاعات دارای ارزش کمی می باشند .



منابع :

http://aima.eecs.berkeley.edu/slides-pdf/chapter16.pdf
http://www.idi.ntnu.no

فصل نوزدهم

شبکه های عصبی

شبکه های عصبی [1]
فصل نوزدهم

ریوس مطالب

- **مغز** [2]
- **شبکه های عصبی**
- **پرسپترون** [3] **ها**
- **پرسپترون های چند سطحی**
- **کاربردهای شبکه های عصبی**

شبکه های عصبی

- مقداری از آنچه که تا کنون ما مطالعه کرده ایم می تواند به صورت **هوش مصنوعی نمادین** [4] طبقه بندی شود .
- تاکید بر سمبل ها و ارتباطات میان آن ها می باشد .
  - جستجو ، منطق ، درخت های تصمیم گیری و برنامه ریزی
- فرض اصلی این است که کاربرد سمبل ها نیاز کلیدی برای رفتار هوشمند می باشد
- شبکه های عصبی بر رفتار *نمادین* تمرکز می نماید .
- رفتار هوشمند از تعامل اجزای ساده به وجود می آید .

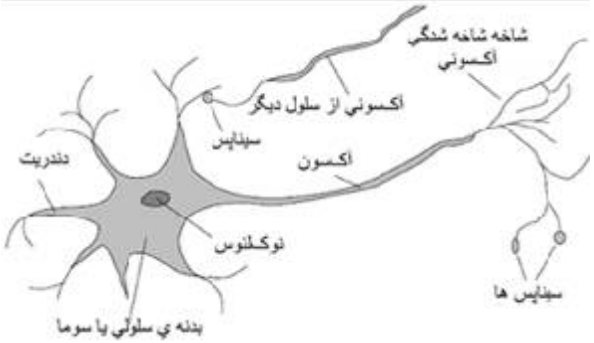
زیست شناسی و علم کامپیوتر

- در زیست شناسی ، نورون ها [5] سیگنال هایی دریافت شده توسط دندریت ها [6] و منتشر شده به دیگر نورون ها از راه آکسون [7] می باشند .
- علامت دهی [8] و شلیک [9] خیلی پیچیده می باشند .
- فکر و رفتار کاملا در تعامل هزاران نورون به وجود می آیند .

---

[1] neural networks
[2] brains
[3] Perceptron
[4] symbolic AI
[5] neurons
[6] dendrites
[7] axon
[8] signaling
[9] firing

- شبکه های عصبی محاسباتی وابسته به شبکه های عصبی زیست شناسی عمدتا با مقایسه (قیاس) هستند
  - علم اعصاب محاسباتی ، مدلسازی زیستی نورون ها را مطالعه می نمایند .
- محققان هوش مصنوعی اغلب جذب توسعه ی الگوریتم های موثر می باشند .
  - مانند الگوریتم های ژنتیکی ، ما برروی ایده هایی که کاملا طبیعی هستند کار می کنیم و آن هایی که مفید هستند را به کار می بندیم .



شبکه های عصبی محاسباتی

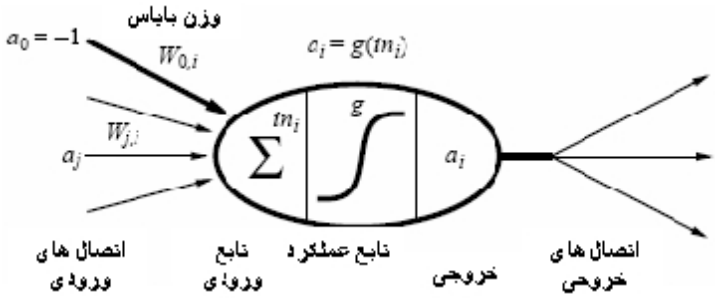
- شبکه های عصبی از نورون ها تشکیل شده اند .
- این گره ها توسط اتصال هایی به هم متصل می باشند .
  - جدا از آکسون ها
- هر اتصال دارای یک فشار است که قدرت سیگنال (علامت) را مشخص می کند .
- هر گره دارای یک تابع عملکرد غیرخطی است
  - خروجی گره به صورت یک تابع توزیع شده ، جمع ورودی ها را مدیریت می نماید .

فعالیت های مناسب برای آموزش عصبی

- تعدادی جفت خصوصیت – مقدار
- ورودی های مقداردهی شده ی حقیقی
- مقدار مقصد واقعی یا گسسته
- داده های پراغتشاش یا با خطا
- زمان آموزش طولانی خیلی خوب می باشد
- ارزیابی سریع موارد لازم شده ی تست
- توانایی بشر برای فهمیدن فرضیه ی آموزش داده شده ، مهم نمی باشد .

واحد McCulloch-Pitts – خروجی ، یک تابع خطی له شده [1] از ورودی ها می باشد :

$$a_i \leftarrow g(in_i) = g(\sum_j W_{j,i} a_j)$$



یک تعداد زیاد ساده شده از نورون های واقعی می باشد ، اما هدف این است که بفهمیم واحدهای ساده ی شبکه می توانند چه کاری را انجام دهند

[1] squashed

- واحد بایاس برای کنترل *مقدار آستانه* [1] استفاده می شود
  - سیگنال ورودی توزیع شده چه مقدار باید باشد که گره شلیک کند .

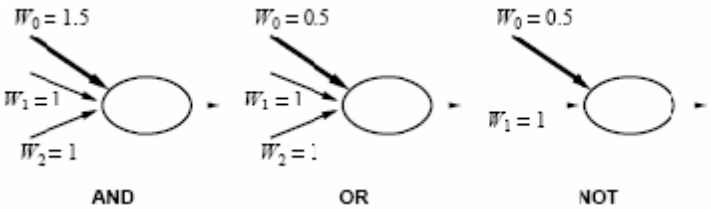
توابع عملکرد

- در حالت کلی از هر تابع غیرخطی می توان استفاده نمود .
- عمومی ترین توابع به صورت زیر می باشند :
  - تابع مرحله (تابع آستانه) – خروجی ها ، اگر ورودی مثبت باشد ، یک می باشد و در غیر این صورت صفر می باشد .
  - تابع S شکل (سیگموید) [2] یا نقلی : $1/(1+e^{-x})$
    - به صورت پیوسته قابل تشخیص می باشد
    - تغییر سریع در نزدیکی آستانه می باشد و به صورت تدریجی به سمت بی نهایت می رود .

مثال ها :

تکمیل توابع منطقی

مک کلاچ و پیتز [3] : هر تابع بولین می تواند تکمیل شود



- شبکه های عصبی می توانند به سادگی برای انجام برخی از عملیات استاندارد منطقی با استفاده از تابع عملکرد آستانه ای ساخته شوند .

انواع گره ها

- ما می توانیم سه نوع گره را تشخیص دهیم :
  - گره های ورودی
  - گره های خروجی
  - گره های پنهان
- ما همچنین می توانیم انواع شبکه ها را تشخیص دهیم
  - شبکه های تغذیه ی مستقیم [4] : علامت ها در یک جهت حرکت می کنند و بدون دور می باشند . شبکه های تغذیه ی مستقیم ، توابع را تکمیل می کنند و وضعیت داخلی ندارند .
  - شبکه های بازگشت کننده [5] : در انتشار علامت ، دور وجود دارد .

شبکه های Hopfield دارای اوزان هم اندازه می باشند $(W_{i,j} = W_{j,i})$

- ما در ابتدا روی شبکه های تغذیه ی مستقیم تمرکز می نماییم .

---

[1] threshold value
[2] sigmoid
[3] McCulloch & Pitts
[4] feed-forward networks
[5] recurrent networks

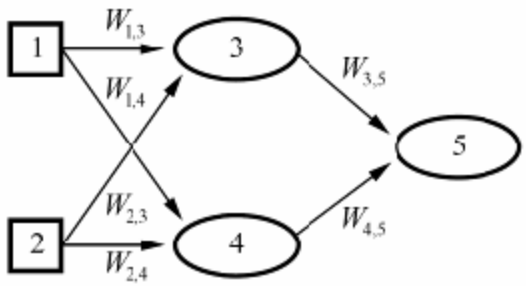
شبکه های تغذیه ی مستقیم به شکل تخمین زننده ی توابع

- شبکه های عصبی با تغذیه ی مستقیم در یک خانواده از الگوریتم ها به نام تخمین زننده های تابع غیرخطی قرار می گیرند
- خروجی یک شبکه ی عصبی یک تابع از ورودی هایش می باشد
- تابع فعال سازی غیرخطی ارایه ی توابع پیچیده را اجازه می دهد
- با تطبیق دادن وزن ها ، ما تابعی که ارایه می شود را تغییر می دهیم
- شبکه های عصبی اغلب برای تخمین توابع پیچیده از داده ها استفاده می شوند .

طبقه بندی با شبکه های عصبی

- شبکه های عصبی ، طبقه بندی را هم خیلی خوب انجام می دهند .
- ورودی ها را به صورت یک یا بیش تر خروجی نگاشت می نماید .
- دامنه ی خروجی به دو کلاس مجزا تقسیم می شود
- برای کارهای آموزشی در جایی که " در جستجوی چه هستیم " معلوم نمی باشد خیلی مفید می باشد
  - صورت شناسی [1] ، دست خط شناسی [2] و رانندگی یک اتومبیل

مثال تغذیه ی مستقیم



شبکه ی تغذیه ی مستقیم = یک خانواده ی پارامتربندی شده از توابع غیرخطی :

$$a_5 = g(W_{3,5}.a_3 + W_{4,5}.a_4)$$
$$= g(W_{3,5}.g(W_{1,3}.a_1 + W_{2,3}.a_2) + W_{4,5}.g(W_{1,4}.a_1 + W_{2,4}.a_2))$$

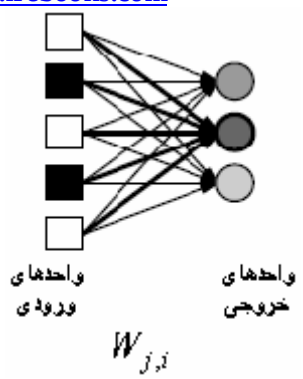
میزان نمودن وزن ها عملکرد را تغییر می دهد : این روش را یاد بگیرید !

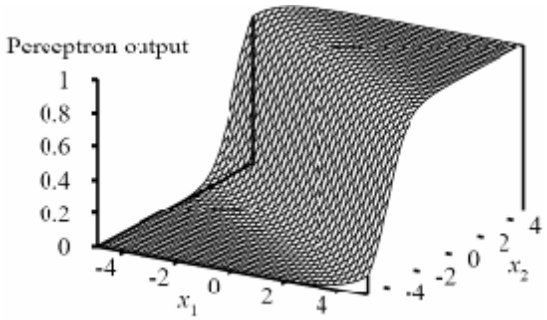
پرسپترون ها

پرسپترون ، یک نورون با یک تابع انتقال بامحدودیت سخت و یک مکانیزم تطبیق وزن با مقایسه ی جواب های واقعی و خروجی های مورد انتظار برای هر ورودی داده شده می باشد .

پرسپترون های تک لایه ای

[1] face recognition
[2] handwriting recognition





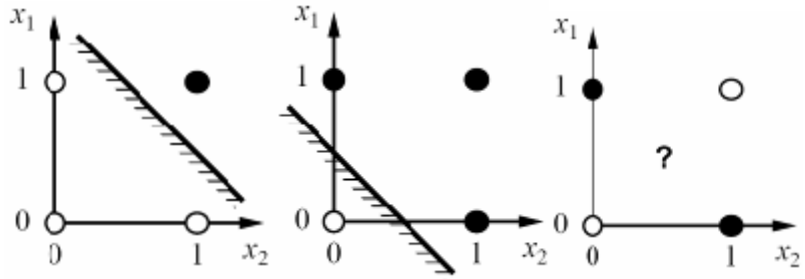
- ساده ترین شکل شبکه ی عصبی می باشند .
- برای فهم ، ساده می باشند ولی برای محاسبه ، محدود می باشند .
- هر واحد ورودی به صورت مستقیم به یک یا بیش تر واحد خروجی متصل شده است .
- خروجی آستانه ی توزیع شده مجموع ورودی ها است .
- تابع شلیک آستانه در این جا استفاده می شود .

اگر $w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n > 0$ باشد ، آن گاه $o(x_1,...,x_n) = 1$

- همه ی واحدهای خروجی به طور مستقل عمل می نمایند – نه به صورت اشتراکی .

توجه نمایید که یک پرسپترون با g = تابع مرحله ای ، می تواند AND ، OR ، NOT و ... را ارایه نماید به جز XOR . در فضای ورودی ، یک مجزا کننده ی خطی [1] را ارایه می نماید :

$\sum_j W_j x_j > 0$ یا W.x>0



توان نمایش پرسپترون ها

- خروجی شبکه : $\sum_{j=0}^{n} W_j i_j > 0$
- پرسپترون ها قادر به ارایه ی هر تابع خطی جداشدنی هستند .
- متاسفانه ، تعدادی از توابع (XOR و parity) به صورت جداشدنی خطی نمی باشند .
- در روزهای ابتدایی هوش مصنوعی ، پرسپترون ها عمومی بودند ، به علت این واقعیت که وزن آن ها می توانست به سادگی پیدا شود

الگوریتم آموزشی پرسپترون

inputs = in1, in2, ..., inj
weights = w1, w2, ..., wn - initialize each to rand(-0.5, 0.5)
output = o
training examples: t1 = (tin1, to1), t2 = (tin2, to2) , ...

[1] linear separator

```
do
        for t in training examples
        inputs = tin
        o = compute net's output with current weights
        E = to - o
        for w in weight
                w = w + alpha * tin[i] * E
while notConverged
```

آموزش پرسپترون

- بینش : اگر علامت خروجی ، خیلی بالا باشد ، اوزان باید کاهش یابند .
  - اوزان "برگشته [1] " در خروجی شرکت دارند .
  - اوزان با ورودی صفر ، موثر نمی باشند .
- اگر خروجی خیلی پایین باشد ، اوزان باید افزایش یابند .
  - اوزان "ظاهر شده [2] " که در خروجی شرکت دارند .
  - دوباره ، اوزان ورودی صفر دارای اثر نمی باشند .
- آموزش ، واقعا یک بهینه سازی وزن ها است .
- متناوبا ، ما در حال انجام یک جستجوی تپه نوردی در میان فضای وزن هستیم .

با میزان کردن توزیع ها برای کاهش خطا در مجموعه ی متوالی ، این مطلب را یاد بگیرید .
خطای مربعی برای یک مثال با ورودی x و خروجی صحیح y :

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_w(x))^2$$

جستجوی بهینه را با گرادیان نزولی انجام دهید :

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}(y - g(\sum_{j=0}^{n} W_j x_j)) = -Err \times g'(in) \times x_j$$

قانون به روز ساده ی وزن :

$$W_j \leftarrow W_j + \alpha \times g'(in) \times x_j$$

مثال ، خطای +ve ← خروجی شبکه افزایش می یابد ← توزیع ها در ورودی های +ve افزایش می یابد و در ورودی های –ve کاهش می یابد .

مثال آموزشی AND

Step function - step at 0
Two inputs, plus a bias input set at -0.2
Learning rate = 0.1
Initial weights: 0.3, -0.1

---

[1] turn down
[2] turn up

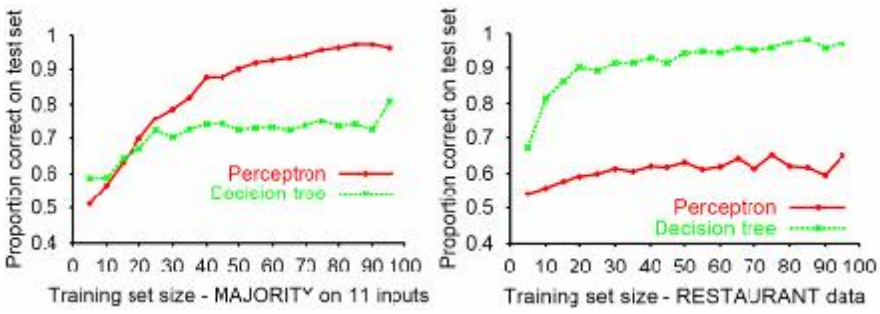
گرادیان وراثت و قانون دلتا

- مواردی که ما نمی توانیم تابع را به صورت مستقیم یاد بگیریم چگونه هستند ؟
  - تابع به صورت خطی جداشدنی نباشد
- در این مورد ، ما می خواهیم تا جای ممکن ، خوب عمل کنیم .
- ما این را می خواهیم به صورت کمینه سازی مجموع خطای مربع شده تفسیر نماییم .
- $E = 1/2 \sum (t_d - o_d)^2$ برای d در مجموعه ی آموزشی .
- ما می توانیم این را به صورت یک جستجو در میان یک فضای اوزان ببینیم
- تعریف E در این روش یک فضای سهمی شکل را با یک کمینه ی عمومی ارایه می کند (برای واحدهای خطی) .
- با دنبال کردن گرادیان در این فضا ، ما ترکیب وزن هایی که خطا را کمینه می کنند را می فهمیم .
- از خروجی غیرآستانه ای استفاده نمایید – شماره ی حقیقی محاسبه شده توسط جمع توزین شده ی ورودی ها چیست ؟
- در گرادیان وراثت ما سرزیرترین سطح خطا را دنبال می کنیم .
- ما به مشتق E با رعایت هر وزن توجه می کنیم
- بعد از مشتق ، ما ازقانون به روزرسانی (به نام قانون دلتا) استفاده می نماییم :
- $\Delta w_i = \alpha \sum_{d \in D} (t_d - o_d) x_{id}$
- در جایی که D مجموعه ی آموزشی می باشد ، t_d خروجی مورد انتظار است و o_d خروجی واقعی می باشد .

آموزش اضافی (نهایی) [1]

- اغلب عملی نیست که یک تغییر وزن عمومی را برای تمام مجموعه ی آموزشی محاسبه نماییم .
- در عوض ، ما می خواهیم وزن ها را به صورت اضافی به روز نماییم .
  - به یک قسمت از داده ها توجه نمایید و سپس آن را به روز نمایید
- بنابراین ، قانون به روز رسانی ما ، $w_i = \alpha(t - o)x_i$ خواهد بود
  - شبیه قانون آموزش پرسپترون ، انتظار داریم که از خروجی غیرآستانه ای استفاده شود .
- اندازه ی کوچک تر (α) معمولا استفاده می شود .

آموزش پرسپترون- به یک تابع مرکب ، برای هر مجموعه ی داده ای قابل تجرد خطی نزدیک می شود

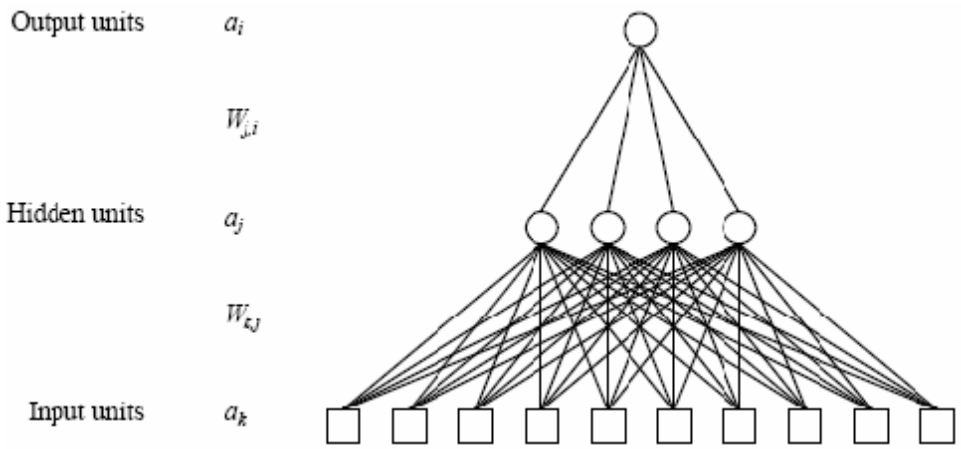


[1] incremental learning

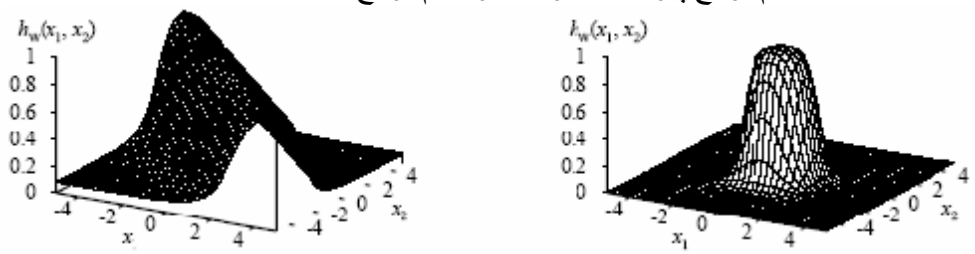
پرسپترون ، تابع اکثریت را به سادگی یاد می دهد ولی DTL این کار را انجام نمی دهد . DTL توابع رستوران را به سادگی یاد می دهد و این در حالی است که پرسپترون نمی تواند این کار را انجام بدهد

پرسپترون های چند لایه ای - لایه ها معمولا به صورت کامل متصل می شوند ؛ تعدادی از واحدهای مخفی به طور معمول به صورت دستی انتخاب می شوند .

- پرسپترون ها دارای این مزیت هستند که یک الگوریتم آموزشی ساده دارند ولی محدودیت های محاسباتی آن ها یک مشکل است .
- در صورتی که ما یک لایه ی مخفی دیگر اضافه نماییم چه اتفاقی می افتد ؟
- توان محاسباتی افزایش می یابد
  - با یک لایه ی پنهان ، که می تواند هر تابع پیوسته را نمایش دهد
  - با دو لایه ی پنهان ، که می تواند هر تابعی را نمایش دهد
- مشکل : چگونه وزن های صحیح را برای گره های پنهان ، پیدا نماییم ؟



بیانگری MLP ها - تمام توابع پیوسته با دو لایه و تمام توابع با سه لایه



دو تابع آستانه ی در جهت متفاوت را برای ساختن یک برآمدگی با هم ترکیب نمایید . دو برآمدگی عمدی را برای ساختن یک گودی ، با هم ترکیب نمایید . گودی های با اندازه ها و محل های متفاوت را برای مناسب نمودن هر سطحی استفاده نمایید . به صورت تعریفی ، ثابت نمایید تعدادی واحدهای مخفی مورد نیاز می باشد (اثبات cf DTL) .

آموزش انتشار معکوس - لایه ی خروجی : شبیه پرسپترون تک لایه ای می باشد ، $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$ در جایی که ، $\Delta_i = Err_i \times g'(in_i)$.

لایه ی مخفی : انتشار معکوس خطا از لایه ی خارجی :

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

قانون را برای وزن های درون لایه ی مخفی به روز نمایید :

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

بیش تر متخصصان اعصاب بی توجهی می کنند به این که انتشار معکوس در مغز رخ داده است)

اشتقاق انتشار معکوس

خطای مجذور در یک مثال منفرد به صورت زیر تعریف می شود :

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2$$

و در جایی که مجموع بیش از گره های درون لایه ی خروجی می باشد هم به صورت فوق تعریف می شود .

$$\frac{\partial E}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}}$$

$$= -\sum_i (y_i - a_i) g'(in_i)\frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}}\left(\sum_j W_{j,i} a_j\right)$$

$$= -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{k,j}}$$

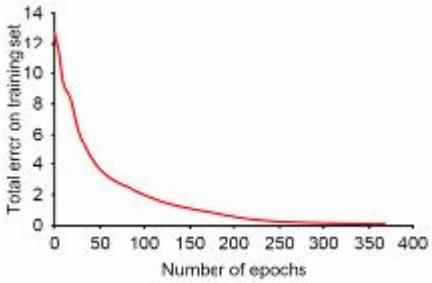
$$= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial in_j}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial}{\partial W_{k,j}}\left(\sum_k W_{k,j} a_k\right)$$

$$= -\sum_i \Delta_i W_{j,i} g'(in_j) a_k = -a_k \Delta_j$$

در هر دوره ، مجموع گرادیان برای همه ی مثال ها به روز می شود و به کار گرفته می شود .

منحنی به شکل قطار برای مثال ۱۰۰ رستوران : محل مناسب را پیدا می کند

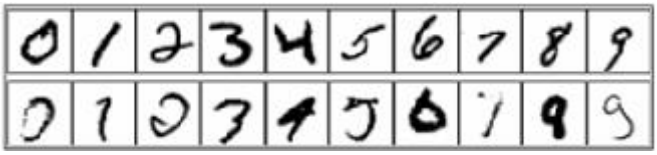


مسایل عادی : همگرایی کند و کمینگی محلی

منحنی آموزشی برای MLP با چهار واحد مخفی :

MLP ها کاملا برای کارهای شناسایی الگو مناسب می باشند ولی نتیجه به سادگی نمی تواند فهمیده شود .

شناسایی ارقام دستنوشته



نزدیک ترین همسایه به ۳ = ۲٫۴% خطا
۴۰۰-۳۰۰-۱۰ واحد MLP = ۱٫۶ % خطا
LeNet : ۷۶۸ – ۱۹۲ – ۳۰ – ۱۰ واحد MLP = ۰٫۹% خطا
بهترین حالت (ماشین های هسته ای و الگوریتم های آمال) ≈ ۰٫۶% خطا

انتشار معکوس [1]

- انتشار معکوس ، شاخه ای از الگوریتم آموزشی پرسپترون برای رسیدگی کردن به لایه های چندگانه ی گره ها .
- گره ها از تابع فعال سازی سیگموید استفاده می نمایند
  - $g(input_i) = \frac{1}{1 + e^{-input_i}}$
  - $g'(input_i) = g(input_i)(1 - input_i))$
- یادداشت :
  - $h_w(x)$ - بردار خروجی های شبکه
  - y – خروجی مطلوب برای یک مثال آموزشی
  - a_j - خروجی j امین واحد مخفی
  - o_i خروجی i امین واحد خروجی
  - t_i خروجی برای i امین مثال آموزشی
  - خطای خروجی برای گره ی خروجی i :
    $\Delta_i = (t_i - o_i) * g(input_i) * (1 - g(input_i))$
  - به روز رسانی وزن (خروجی پنهان) :
    $W_{j,i} = W_{j,i} + \alpha * a_j * \Delta_i$
- به روز رسانی وزن های ورودی پنهان :
- ایده : هر گره ی پنهان در جستجوی یک تابع خطا به صورت Δ_i می باشد .

[1] backpropagation

- Δ_i را با توجه به شدت اتصال میان گره پنهان و خروجی تقسیم نمایید .
- $Delta_j = g(input)(1-g(input))\sum_i W_{j,i}\Delta_i$
- قانون را برای وزن های پنهان ورودی به روز نمایید :
- $W_{k,j} = W_{k,j} + \alpha * input_k * \Delta_j$

الگوریتم انتشار معکوس

مادامی که انجام نشده است
برای d در مجموعه ی آموزشی
ورودی های d و انتشار مستقیم را به کار ببر .
برای گره ی i در لایه ورودی

$$Delta_i = output * (1-output) * (t_{\exp} - output)$$

برای هر گره ی پنهان

$$Delta_j = output * (1-output) * \sum W_{k,i}\Delta_i$$

برای هر وزن

$$W_{j,i} = W_{j,i} + \alpha * \Delta_i * input_j$$

متوقف کردن شرط ها

- چه موقع ، آموزش را متوقف نماییم ؟

وقتی که تعداد تکرار ها ثابت شده باشد ، مجموع خطای زیر یک مجموعه ی آستانه باشد و

- همگرایی [1] - هیچ تغییری در وزن ها به وجود نیاید

صورت شناسی با شبکه های عصبی

- یکی از مزایای شبکه های عصبی این است که آن ها به یک توضیح صریح از ارتباطات میان بعدی ها نیاز ندارند . آن ها واقعا توضیحات را یاد نمی گیرند ، برای مسایلی که یک جواب نمادین ندارند به خوبی مناسب می باشند . شما می توانید کد شبکه ی عصبی موجود را برای تشخیص صورت ها بازنگری نمایید .
  - در CMU توسعه یافته اند
  - به زبان سی نوشته شده اند .

نکاتی در مورد انتشار معکوس

- همیشه برای چند لایه ی پنهان کار می کنند . انتشار معکوس فقط برای گرایش دادن به یک کمینه ی محلی ضمانت شده است .
  - ممکن است مجموعه ی وزن های کامل را پیدا نکنیم
- وزن های اولیه ی پایین می توانند به ما کمک زیر را بکنند
  - شبکه به مراتب خطی تر کار کند
- همچنین می توانند از شروع مجدد تصادفی استفاده نمایند .

اندازه حرکت [2]

- یک گستره ی انتشار معکوس با اضافه نمودن واژه ی اندازه حرکت به وجود می آید .
  - الگوریتم را به کمینه و حالت مسطح می برد

---

[1] convergence
[2] momentum

- ایده : " جهتی " را که به آن می روید را به یاد بیاورید و به صورت پیش فرض حرکت به سمت آن را ادامه دهید .
- از لحاظ ریاضی ، این به این معنی می باشد که از مشتق مرتبه ی دوم استفاده نماییم .
- پیاده سازی اندازه حرکت معمولا به معنی به یاد آوردن این است که به روز رسانی در مرحله ی قبل انجام شده است .
- قانون به روز رسانی ما : $\Delta w_{ji}(n) = \alpha \Delta_j x_{ji} + \alpha \Delta w_{ji}(n-1)$
- برای فهمیدن چگونگی عملکرد ، تصور نمایید که دلتای جدید ما صفر می باشد (ما هیچ بهبودی را انجام نداده ایم)
- اندازه حرکت وزن های حرکت کننده در جهت یکسان را نگهداری می نمایند .
- همچنین به تدریج اندازه ی مرحله را در ناحیه ای که گرادیان تغییر نمی کند افزایش می دهد .
  - این سرعت همگرایی را افزایش می دهد

شبکه های با عملکرد ستاره ای

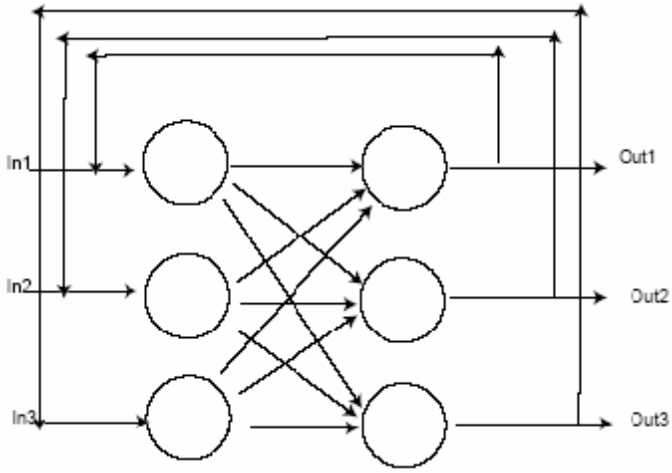
- یک مساله ی انتشار معکوس این است که هر گره با خروجی یک راه حل همکاری می نماید . این به این معنی است که همه ی وزن ها باید برای کمینه کردن خطای عمومی میزان شوند . اغتشاش در یک بخش از داده ها می تواند سبب یک ضربه در همه ی خروجی شبکه شود . همچنین رشته های زمانی طولانی می باشند . شبکه های با عملکرد ستاره ای یک راه حل را برای این مورد مهیا می نمایند .
- بینش : هر گره در شبکه یک جزء از فضای ورودی را نمایش می دهد .
- دنبال طبقه بندی مثال هایی که در اطراف آن رخ می دهند می باشد .
- نگرش وانیلی [1] : برای هر نقطه ی متوالی $< x_i, f(x_i) >$ گره ای را بسازید که در وسط ، x_i قرار دارد . خروجی این گره برای یک ورودی جدید x ، $W * \phi(|x - x_i|)$ خواهد بود . که W ، وزن می باشد و $\phi = \exp(-\frac{x^2}{2\sigma^2})$ می باشد . ϕ تابع اساسی می باشد و هر گره دارای یک " ناحیه ی تاثیر " است که می تواند نمونه های نزدیک را طبقه بندی نماید . همچنین ، شبکه ، تک لایه ای می باشد . زن ها می توانند با یک معادله ی ماتریسی به دست آیند :
  - $\Phi W = t$
  - $W = \Phi^{-1} t$
- معکوس کردن یک ماتریس به مراتب سریع تر از ساختن آن با انتشار معکوس می باشد .

شبکه های عصبی برگشت کننده [2]

- تا کنون ما فقط در مورد شبکه های تغذیه ی مستقیم صحبت کرده ایم . در این شبکه ها علامت ها (سیگنال ها) در یک جهت منتشر می شوند ، خروجی بلافاصله در دسترس می باشد و دارای الگوریتم هایی هستند که به سادگی قابل فهم می باشند . تعداد زیادی کار می تواند با شبکه های عصبی برگشت کننده انجام شود .
  - لااقل برخی از خروجی ها به عقب متصل شده برای ورودی ها هستند .
- در زیر یک شبکه عصبی برگشت کننده ی تک لایه ای را می بینید

---

[1] vanilla approach
[2] recurrent NNs(Neural Networks)



شبکه های هوپفیلد [1]

- یک شبکه ی هوپفیلد دارای هیچ گره معین ورودی یا خروجی نمی باشد .
- هر گره یک ورودی و روال های یک خروجی را دریافت می نماید
- هر گره به گره های دیگر متصل شده است .
- معمولا ، از توابع آستانه ای استفاده می شود .
- شبکه بلافاصله یک خروجی را تولید نمی نماید .
  - در عوض ، مردد می باشد .
- تحت برخی از وضعیت های به آسانی قابل دسترس ، سرانجام به حالت موازنه می رسد .
- وزن ها با استفاده از گرم و سرد کردن شبیه سازی شده به دست می آیند .
- شبکه های هوپفیلد می توانند برای ساختن یک حافظه ی *شرکت پذیر* [2] به کار روند
- یک قسمت از یک الگو برای شبکه ارایه می شود و شبکه تمام الگو را به یاد می آورد .
- برای حرف شناسی [3] به کار می رود ، همچنین برای مسایل بهینه سازی هم به کار می رود و اغلب برای مدل فعالیت مغز استفاده می شود .

خلاصه

بیش تر مغز ها دارای تعداد زیادی نورون می باشند ؛ هر نورون ≈ واحد آستانه ی خطی می باشد . ایده ی کلیدی : بخش های محاسباتی ساده با استفاده از وزن ها به هم وصل می شوند . پرسپترون (شبکه های تک لایه ای) با بیان ناکافی می باشند . شبکه های چند لایه ای دارای بیان کافی می باشند ، می توانند با گرادیان نزولی بیان شوند . تعدادی از کاربردها : سخن گفتن ، رانندگی ، دست خط ، کشف حیله یا تقلب و ... . در زمانی که یک ماشین که یک تابع را تخمین می زند لازم است مفید می باشد

- احتیاجی به فهمیدن فرضیه ی یادداده شده ندارد

منابع :

http://www.cs.usfca.edu
http://aima.eecs.berkeley.edu/slides-pdf/chapter20b.pdf

[1] Hopfield networks
[2] associative memory
[3] letter recognition

فصل بیستم

الگوریتم های ژنتیکی

الگوریتم های ژنتیکی
فصل بیستم

تعریف : الگوریتم های ژنتیکی ، توسط جان هلند [1] و دانشجویانش در سال ۱۹۷۵ براساس تئوری تکاملی داروین (باقی ماندن شایسته ترین) طراحی شدند . این الگوریتم ها ، یک روال تکراری هستند که از رشته های با طول ثابت [2] به نام جمعیت افراد تشکیل شده اند ، هر فرد با یک رشته ی محدود از سمبل ها به نام کروموزوم ها[3] و کد کننده ی یک راه حل ممکن در فضای مساله ی داده شده ، نمایش داده می شود . الگوریتم ژنتیکی استاندارد به این صورت توسعه می یابد : یک جمعیت اولیه از افراد به صورت تصادفی یا مکاشفه ای تولید می شوند ، هر مرحله ی تکاملی ، نسل نام دارد ، افراد در جمعیت فعلی رمز گشایی می شوند و با توجه به ملاک کیفیتی که از قبل به صورت تابع شایستگی تعریف شده ارزیابی می شوند . برای تشکیل یک جمعیت جدید ، افراد با توجه به شایستگی اشان انتخاب می شوند . افراد با شایستگی بالا (خوب) دارای شانش بیش تری برای تولید می باشند . الگوریتم های ژنتیکی ، برای حل یک طیف از مسایلی که با روش های دیگر به آسانی حل نمی شوند ، مناسب می باشند .

تکامل در دنیای واقعی – هر سلول یک موجود زنده دارای کروموزوم هایی می باشد – رشته هایی از DNA . هر کروموزوم شامل یک مجموعه از ژن ها می باشد – بلوک های DNA . هر ژن برخی از خواص موجود زنده (مانند رنگ چشم) را تایین می کند . یک مجموعه از ژن ها در برخی از موارد ژنوتیپ (نوع معرف و نماینده ی یک جنس) [4] نامیده می شود . یک مجموعه از خصوصیات (نظیر رنگ چشم) گاهی فنوتیپ (ظاهر یا عملکرد یک فرد که به وسیله ی انسان قابل اندازه گیری می باشد) نامیده می شود . انتشار شامل بازترکیب (ترکیب جدیدی از فنوتیپ ها در نتایج حاصل از والدین با فنوتیپی متفاوت) [5] ژن ها از والیدن و سپس مقدار کوچکی جهش (خطاها) در کپی کردن می باشد . شایستگی یک موجود زنده مقداری است که می تواند قبل از مردنش انتشار دهد . تکامل ، براساس "بقای مناسب ترین" می باشد .

یک راه حل گنگ – یک الگوریتم "تولید و تست گنگ" :

تکرار کن
    یک راه حل ممکن تصادفی را تولید کن

---

[1] John Holland
[2] fixed_size
[3] chromosomes
[4] genotype
[5] recombination

راه حل را امتحان کن و خوب بودن آن را بسنج
تا موقعی که راه حل به اندازه ی کافی خوب باشد

آیا ما می توانیم از این ایده ی گنگ استفاده نماییم ؟ برخی اوقات بله : در صورتی که تعداد راه حل های کمی وجود داشته باشد و شما به اندازه ی کافی زمان در اختیار داشته باشید ، آن گاه روش های این فرمی می توانند استفاده شوند . اما برای بیش تر مسایل ما نمی توانیم از این راه حل ها استفاده نماییم : در زمانی که راه حل های ممکن ، زیاد باشند و شما به اندازه ی کافی زمان برای امتحان همه ی آن ها نداشته باشید ، بنابراین این راه حل ها نمی توانند استفاده شوند .

ایده ای که کم تر دارای گنگ بودن می باشد (الگوریتم ژنتیکی) – یک مجموعه ی تصادفی از راه حل ها را تولید نمایید
تکرارکن
هر راه حل را در مجموعه امتحان کن و آن ها را رتبه بندی کن
برخی از راه حل های بد را از مجموعه بردار
برخی از راه حل های خوب را تکثیر کن
برخی از تغییرات کوچک را در مورد آن ها اعمال کن
تا زمانی که بهترین راه حل به اندازه ی کافی خوب شود

چگونه شما یک راه حل را کد می کنید ؟ بدیهی است که این وابسته به مساله می باشد ! الگوریتم های ژنتیکی ، اغلب راه حل ها را به صورت رشته های بیتی باطول ثابت ، کد می کنند (به عنوان مثال ، ۱۰۱۱۱۰ ، ۱۱۱۱۱۱ ، ۰۰۰۱۰۱) . هر بیت برخی از خصوصیات راه حل های ارایه شده برای مساله را ارایه می کند . برای این که الگوریتم های ژنتیکی کار کنند ، ما نیاز به این داریم که هر رشته را تست نماییم و به آن امتیازی بدهیم که نشان دهنده ی چگونگی خوب بودن آن باشد .

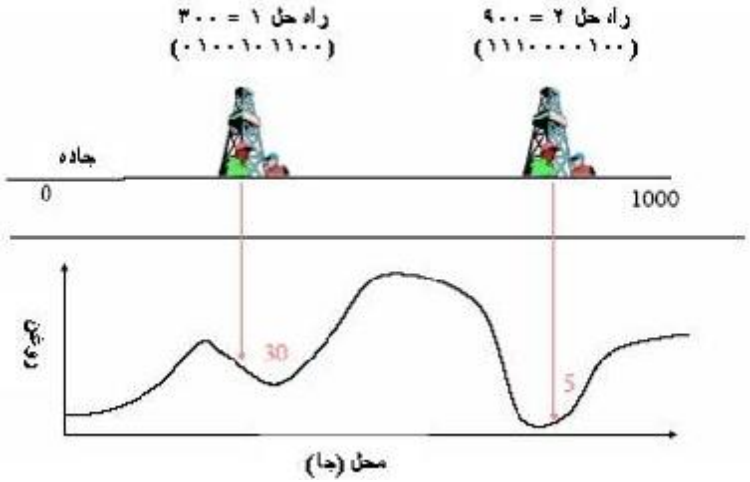
مثال نادان – حفر برای روغن – تصور کنید که شما باید برای روغن جایی را در طول یک جاده ی بیابانی یک کیلومتری ، حفر نمایید . مساله : بهترین مکان در جاده را که بیش ترین مقدار روغن را در روز تولید می کند ، انتخاب نمایید . ما می توانیم هر راه حل را به صورت یک وضعیت در جاده نمایش دهیم . تصور نمایید که تمام اعداد بین [۱۰۰۰، ۰] باشند .

کجا را برای روغن ، حفر نماییم ؟

مجموعه ی همه ی راه حل های ممکن [۱۰۰۰،۰] ، فضای جستجو یا فضای حالت نام دارد . در این مورد ، این فقط یک عدد است اما می تواند تعداد زیادی عدد یا سمبل باشد . اغلب ، الگوریتم های ژنتیکی اعداد را به صورت دودویی (باینری) کد می کنند . در مورد این مثال ، ده بیت را که برای نمایش ۰ ... ۱۰۰۰ کافی است را انتخاب می نماییم .

	۵۱۲	۲۵۶	۱۲۸	۶۴	۳۲	۱۶	۸	۴	۲	۱
۹۰۰	۱	۱	۱	۰	۰	۰	۰	۱	۰	۰
۳۰۰	۰	۱	۰	۰	۱	۰	۱	۱	۰	۰
۱۰۲۳	۱	۱	۱	۱	۱	۱	۱	۱	۱	۱

در الگوریتم های ژنتیکی، این رشته های کد شده گاهی ژنوتیپ ها یا کروموزوم ها نامیده می شوند و بیت های فرد گاهی اوقات ژن ها نامیده می شوند .



درالگوریتم های ژنتیکی ، وضعیت ها (راه حل های بالقوه) به صورت رشته های بیتی کد (رمزگذاری) می شوند . راه حل های جزیی با استفاده از تقاطع [1] ترکیب می شوند . رشته های مناسب تر احتمال دوباره استفاده کردنشان بیش تر می باشد . تاریخچه ی گذشته ی جستجوی محلی نگهداری نمی شود . از شکل احتمالی جستجوی تپه نوردی استفاده می شود

pop = makeRandomPopulation
مادامی که انجام نشده است
برای هر p درون pop
p.fitness = evaluate(p)
برای i=1 تا اندازه ی pop با ۲
والدها را برای تکثیر انتخاب کن
دو راه حل تصادفی را از pop انتخاب کن = [parent1,parent2]
[child1,child2] = crossover(parent1,parent2)
child1 و child2 را تغییر بده [2]
جمعیت قدیمی را با جمعیت جدید جایگزین نما

یک کروموزوم می تواند به صورت یک رشته ی بیتی ، مانند ۰۰۱۰۱۰۰۰۰۰ کد شود . هر کروموزوم دارای یک مقدار شایستگی می باشد .

اسکلت بندی یک الگوریتم ژنتیکی

[1] crossover
[2] mutate

روش های انتخاب : برای یک جمعیت ارایه شده ، بهترین کروموزوم ها باید برای تولید نسل های جدید ، باقی بمانند . تعدادی روش برای انتخاب بهترین کروموزوم ها عبارتند از : انتخاب رولت ، انتخاب بولتزمان ، انتخاب مسابقه ای ، انتخاب طبقه ای ، انتخاب با حالت دایمی .

کاربردهای الگوریتم های ژنتیکی : الگوریتم های ژنتیکی در بسیاری از موارد و زمینه های تحقیقاتی ، نظیر مسایل عددی ، مساله ی مسافرت شخص دوره گرد ، مدار همیلتونی ، مسیر اویلری ، پیدا کردن شکل مولکول های پروتئین ، مسایل NP سخت ، طراحی شبکه های عصبی ، توابع برای به وجود آوردن تصاویر ، مدلسازی شناختی و تئوری بازی ها به کار می روند .

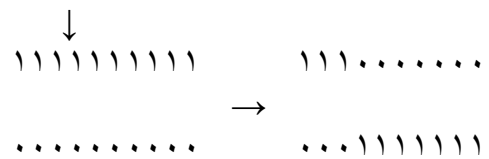
شبه کدهای الگوریتم ژنتیکی : الگوریتم با یک population(t) مقدار دهی اولیه شده شروع می شود ، این population(t) توسط برخی توابع ارزیابی می شود ، سپس الگوریتم وارد یک حلقه می شود ، در هر مرحله در حلقه population(t+1) از population(t) قبلی انتخاب می شود . سپس عملگرهای انقطاع و جهش انجام می شود و جمعیت جدید ارزیابی می شود ، حلقه تا زمانی که به یک شرایط پایانی برسیم ، اجرا می شود .

عملگرهای ژنتیکی :

انتخاب : برای هر فرد i ، یک گزاره ی احتمالی انتخاب را برای شایستگی اش تعریف نمایید :

$$s(i) = f(i)/\_\ f(j), 1 \le j \le n$$

انقطاع :



جهش : یک یا بیش تر ژن به صورت تصادفی به یک مقدار تصادفی ، جهش داده می شوند :

۱۱۱۱۱۱۱۱۱۱ → ۱۱۰۱۰۱۱۱۱۱

انتخاب رولت [1]

- چگونه والدها را برای تکثیر انتخاب نماییم ؟ انتخاب رولت ، احتمال هر فرد را با شایستگی آن توزین می کند
- الگوریتم :

[1] Roulette Selection

- تمام شایستگی ها را جمع کن
- برای هر i در جمعیت
- جمع/شایستگی i = شایستگی i وابسته

•

- جمع شایستگی های وابسته ، برابر ۱ می باشد .
- می تواند به صورت احتمال استفاده شود .

مثال : تصور نمایید که ما می خواهیم $f(x) = x^2$ را در بازه ی [۰،۳۱] بهینه نماییم . برای کد کردن راه حل از ۵ بیت استفاده می شود . جمعیت اولیه را تولید نمایید :

رشته	شایستگی	شایستگی وابسته
۰۱۱۰۱	۱۶۹	۰,۱۴۴
۱۱۰۰۰	۵۷۶	۰,۴۹۲
۰۱۰۰۰	۶۴	۰,۰۵۵
۱۰۰۱۱	۳۶۱	۰,۳۰۹
جمع	۱۱۷۰	۱,۰

- با انتخاب رولت ، والدها را انتخاب کنید و سپس در یک نقطه ی تصادفی قطع کنید :

رشته	شایستگی	شایستگی وابسته
۰۱۱۰‖۱	۱۶۹	۰,۱۴۴
۱۱۰۰‖۰	۵۷۶	۰,۴۹۲
۰۱۰۰۰	۶۴	۰,۰۵۵
۱۰۰۱۱	۳۶۱	۰,۳۰۹
جمع	۱۱۷۰	۱,۰

- فرزندان :
  - ۰۱۱۰۰ و ۱۱۰۰۱
- با انتخاب رولت ، والدها را انتخاب کن . در یک نقطه ی تصادفی قطع کنید .

رشته	شایستگی	شایستگی وابسته
۰۱۱۰۱	۱۶۹	۰,۱۴۴
۱۱‖۰۰۰	۵۷۶	۰,۴۹۲
۰۱۰۰۰	۶۴	۰,۰۵۵
۱۰‖۰۱۱	۳۶۱	۰,۳۰۹
جمع	۱۱۷۰	۱,۰

- فرزندان :
  - ۰۱۱۰۰ و ۱۱۰۰۱
  - ۱۱۰۱۱ و ۱۰۰۰۰
- جمعیت قبلی را با جمعیت جدید جایگزین نمایید . تغییر را به جمعیت جدید اعمال نمایید (با جمعیت کوچک و تغییر کم با نرخ ۰,۱% ، هیچ تغییری رخ نمی دهد) . تولید جدید :
  - ۰۱۱۰۰ ، ۱۱۰۰۱ ، ۱۱۰۱۱ ، ۱۰۰۰۰

- میانگین شایستگی افزایش یافته است (۲۹۳ به ۴۳۹) . بیشینه ی شایستگی افزایش یافته است (۵۷۶ به ۷۲۹)

چه اتفاقی در حال افتادن است ؟

- زیر راه حل های ۱۱*** و ۱۱*** برای تولید یک راه حل بهتر ، دوباره ترکیب می شوند .
- وابستگی میان رشته ها و شایستگی :

شایستگی	رشته
۱۶۹	۰۱۱۰۱
۵۷۶	۱۱۰۰۰
۶۴	۰۱۰۰۰
۳۶۱	۱۰۰۱۱

- یک ۱ در اولین وضعیت به نظر می رسد که با شایستگی وابسته باشد .
  - این موضوع نباید خیلی تعجب آور باشد . ما یک ۱ را در اولین وضعیت ، یک *بلاک ساختمانی* [1] می نامیم .

الگو یا طرح [2]

- ما به راهی برای صحبت درباره ی رشته هایی که شبیه به هم می باشند نیازمندیم . "*" را (به عنوان یک حالت بی تفاوت) برای {۱،۰} اضافه نمایید . یک *الگو* نمونه ای است که یک مجموعه از رشته ها را با استفاده از {*،۱،۰} توصیف می کند
- **۱۱۱ با ۱۱۱۰۰ ، ۱۱۱۰۱ ، ۱۱۱۱۰ و ۱۱۱۱۱ تطبیق می نماید
- *۱۱*۰ با ۰۰۱۱۰ ، ۰۰۱۱۱ ، ۰۱۱۱۰ و ۰۱۱۱۱ تطبیق می نماید
- *۰***۱ با ۰۰۰۰۱ ، ۰۰۰۱۱ ، ۰۰۱۰۱ ، ۰۰۱۱۱ ، ۰۱۰۰۱ ، ۰۱۰۱۱ ، ۰۱۱۰۱ ، ۰۱۱۱۱ تطبیق می نماید
  - قضیه ی ثابت شده : الگو ، وابسته به شایستگی می باشد
- الگوریتم های ژنتیکی در واقع الگو را پردازش می کنند ، نظیر رشته ها . متقاطع کردن ممکن است یک الگو را خراب نماید
  - *۱۱** و ۱***۰
- الگوی مناسب کوچک و زیاد مرتبه ی پایین احتمال این که بدون خرابی بماند بیش تر است
  - مرتبه : تعداد بیت های ثابت در یک الگو .
    - ۱**** – دارای مرتبه ی یک می باشد
    - ۰*۱*۱ – دارای مرتبه ی سه می باشد
- تعریف طول : فاصله ی میان اولین و آخرین بیت معین .
  - ۰۱۱*۱ – تعریف طول برابر است با ۴
  - ۱*۱** - تعریف طول برابر با ۲ می باشد
  - ۰**** – تعریف طول صفر است
- ساخت فرض بلاک : الگوریتم های ژنتیکی ، الگوی مناسب ، کوتاه و دارای مرتبه ی پایین را برای تولید تصاعدی راه حل های بیش تر با هم ترکیب مجدد می نمایند .

انتشار اسکماتا

---

[1] building block
[2] schema

- تصور نمایید m نمونه از یک الگوی H در یک جمعیت با اندازه ی n در زمان t وجود دارد m(H,t) . رشته ها با توجه به شایستگی وابسته انتخاب می شوند $p = \frac{f}{\sum f}$. در زمان t+1 داریم : $m(H,t+1) = m(H,t)n\frac{f(H)}{\sum f}$ که f(H) میانگین شایستگی رشته های نمایش دهنده ی این الگو می باشد . الگو با توجه به شایستگی وابسته به باقی مانده ی جمعیت افزایش یا کاهش می یابد .
  - میانگین اسکماتای بالا ، نمونه های بیش تری را دریافت می کند . میانگین اسکماتای پایین ، نمونه های کم تری را دریافت می نماید
- اما چه مقدار بالا/پایین میانگین است ؟ فرض نمایید که الگوی H بالای میانگین توسط یک مقدار c می ماند . ما می توانیم الگوی معادله ی دیگری را به صورت زیر بنویسیم : m(H,t+1) = (1+c)m(H,t) . با شروع در t=0 ، ما داریم : $m(H,t) = m(H,0)(1+c)^t$ ، که یک تصاعد هندسی [1] می باشد . تکثیر به صورت نمایی مقدار بیش تر (کم تر) بالای (پایین) اسکماتای میانگین را انتخاب می نماید .

اسکماتا و متقاطع کردن

- انتخاب ، فقط نیمی از ماجراست . اسکماتای با طول تعریف شده ی بیش تر انتظار خراب شدنش توسط انقطاع بیش تر می باشد .
- $P(survival) \geq 1 - \frac{defininglength}{Strlen - 1}$
- ما می توانیم این را با معادله ی قبلی ترکیب نماییم :

$$m(H,t+1) \geq M(H,t)\frac{f(H)}{\sum f}1 - \frac{defininglength}{Strlen - 1}$$

- ما می فهمیم که اسکماتای با شایستگی کوچک و بالای میانگین به صورت نسبت های افزایشی نمایی ، نمونه برداری می شوند . این مطلب *قضیه ی طرح* [2] نام دارد .

طرح در مثال ما

- در مثال قبلی ، $H_1 = 1****$ ، $H_2 = *10**$ و $H_3 = 1***0$ بود .
- والدها : ۲ (دو بار) ، ۳ و ۴
  - ۲و۴ هر دو نمونه هایی از H_1 می باشند .
- فرزندان : ۰۱۱۰۰ ، ۱۱۰۰۱ ، ۱۱۰۱۱ و ۱۰۰۰۰
  - سه مورد از H_1 می باشند
  - قضیه ی طرح $m\frac{f(H)}{\sum f}$ کپی را پیش بینی می نماید
  - $f(H_1) = 468.5$ ، میانگین pop = ۲۹۳ و عدد مورد انتظار : $\frac{468.5}{293} = 3.2$

رشته	شایستگی	شایستگی وابسته
۰۱۱۰۱	۱۶۹	۰,۱۴۴
۱۱۰۰۰	۵۷۶	۰,۴۹۲
۰۱۰۰۰	۶۴	۰,۰۵۵
۱۰۰۱۱	۳۶۱	۰,۳۰۹
جمع	۱۱۷۰	۱,۰

[1] geometric progression
[2] Schema Theorem

- فرزندان : ۰۱۱۰۰ ، ۱۱۰۰۱ ، ۱۱۰۱۱ و ۱۰۰۰۰
- $e(H_2) = 2 \times \frac{320}{293} \times 0.75 = 1.64$ ، مقدار واقعی H_2 : ۲ ، $e(H_3) = 1 \times \frac{576}{293} \times 0.2 = 0.39$ ، مقدار واقعی H_3 : ۱ ، H_2 به واسطه ی شاییستگی نسبتا خوب و طول تعریف شده ی کوتاه ، خوب می باشد . تعریف طول طولانی دارای این معنی است که H_3 معمولا خراب می شود .

تئوری و پیاده سازی - قضیه ی طرح به ما می گوید که الگوریتم های ژنتیکی ، به صورت تئوری چگونه کار می کنند . در عمل ، جزئیات پیاده سازی می توانند باعث یک اختلاف در کارایی یک الگوریتم ژنتیکی شوند .

انتخاب مسابقه

انتخاب رولت موثر است ، ولی از لحاظ محاسباتی می تواند گران باشد (هر فرد باید ارزیابی شود و دو تکرار در میان همه ی جمعیت وجود دارد) . انتخاب مسابقه ، یک مکانیزم ارزان تر می باشد . برای هر والد ، دو فرد را به صورت تصادفی انتخاب نمایید . شایستگی بالاتر برای انتشار انتخاب می شود . برای انجام انقطاع ، دو جفت را انتخاب نمایید و مناسب تر را برای هر جفت انتخاب نمایید .

نخبه سالاری [1] - در عمل ، حذف همه ی راه حل ها از یک راه حل می تواند یک الگوریتم ژنتیکی را به تاخیر بیندازد (طرح بد در RNG می تواند پردازش را خراب نماید) . نخبه سالاری ، روش نگهداری یک عملکرد (تابع) جمعیت از نسل قبلی می باشد . انتخاب رولت را برای انتخاب یک بخش از جمعیت با انجام بدون انقطاع استفاده نمایید .

اکتشاف و استخراج [2]

- نخبه سالاری یک مثال از توزیع بیش تر در آموزش می باشد . اکتشاف ، پردازه ی جمع کردن اطلاعات یا تلاش برای راه حل های جدید می باشد . استخراج ، پردازه ی انتخاب راه حلی است که در حال حاضر بهترین راه حل می باشد . توزیع : راه حل های جدید ، ممکن است بدتر از آن چیزی باشد که شما تا کنون دیده اید . اما ، بهترین راه حل شما برای داده ها ممکن است بهینه ی عمومی نباشد .
- آیا شما باید بیش تر یاد بگیرید (چه مقدار) یا متوقف شوید ؟

کد کردن مسایل

- یک چالش در زمان کار با الگوریتم های ژنتیکی در برپا کردن مساله می باشد . قضیه ی طرح به ما می گوید که پارامترهای با کدکننده های کوتاه ، خوب می باشند . پارامترهایی که وابسته به هم هستند باید در محل نزدیک به هم قرار گیرند .
- N- وزیر : تصور نمایید که هر وزیر در یک ستون قرار خواهد گرفت . مساله : برای هر وزیر ، ردیف مناسب را پیدا نمایید . N ردیف به $\log_2 N$ بیت نیاز دارد . همه ی رشته دارای طول $N \times \log_2 N$ می باشد .

تبدیل ها [3]

- برخی از مسایل دارای یک ارایه به صورت رشته ی بیتی طبیعی نمی باشند . ما ممکن است لازم باشد که مطمین شویم که انقطاع راه حل های مجاز را تولید می کند یا نه . مساله

---

[1] elitism
[2] Exploration vs Exploitation
[3] permutations

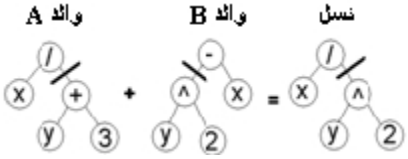
ی مسافرت شخص دوره گرد – کوتاه ترین مسیر را میان شهرهای (n ، ... ، ۲ ، ۱) به شرطی که هر شهر فقط یک بار ملاقات شود پیدا نمایید . به نوبت ، جایگشت (n ، ... ، ۲ ، ۱) را از کمینه ی طول گشت پیدا نمایید .

- نمایش : لیست شهرها : ۳ ، ۱ ، ۲ ، ۴ ، ۵

تقاطع تطبیق داده شده به صورت جزئی [1]

- چگونه در این مورد می توانیم crossover را انجام دهیم ؟ وضعیت ها را ترجیحا با زیر رشته ها عوض نمایید . مثال :
  - t1: 3 5 4 6 1 2 8 7 و t2: 1 3 6 5 8 7 2 4
- اول (مکان هندسی) نقاط را به صورت تصادفی انتخاب نمایید .
- t1: 3 5 | 4 6 1 2 | 8 7 و t2: 1 3 | 6 5 8 7 | 2 4
- تطبیق دو به دو را با تعویض شهرهای دارای رابطه در هر گشت انجام دهید .
  - در هر رشته ، ۴ جایش را با ۶ عوض می کند ، ۶ با ۵ عوض می کند ۱ با ۸ و ۲ با ۷ .
  - t1: 3 6 5 4 8 7 1 2 و t2: 8 3 4 6 1 2 7 5
- بینش : ساخت بلوک ها در این جا بخش هایی از یک گشت هستند که باید با هم باقی بمانند .

برنامه نویسی ژنتیکی – وقتی که کروموزوم ها یک مساله ی کامل یا تابع را کد می کنند ، این کار برنامه نویسی ژنتیکی نام دارد . برای انجام این کار ، کدکردن اغلب به صورت یک ارایه ی درختی انجام می شود . انقطاع ، عوض کردن زیر درخت های میان والدها را دنبال می کند .



این کار برای برنامه های کوچک ، مفید است . برای برنامه های بزرگ با توابع پیچیده ، ناکارآمد می باشد .

توابع شایستگی نا آشکار (ضمنی) – اغلب الگوریتم های ژنتیکی از توابع شایستگی ضمنی استفاده می کنند (همانند آنچه که در مورد مثال روغن دیدید) . برخی از الگوریتم های ژنتیکی (نظیر زندگی مصنوعی یا روبوتیک تکاملی) از توابع شایستگی ضمنی و پویا استفاده می کنند .

مساله – در مساله ی مسافرت شخص دوره گرد یک دوره گرد باید کوتاه ترین مسیر را که یک مجموعه از شهرها را طی می کند ، پیدا نماید . فرض کنید که ما فاصله ی میان هر شهر را می دانیم . این مساله ای مشکل می باشد ، زیرا تعداد مسیرهای ممکن برابر N! می باشد ، که N = تعداد شهرها . الگوریتم ساده ای که در این مورد بهترین جواب را به سرعت ارایه نماید وجود ندارد . یک کد کننده ی کروموزوم ، یک عملکرد جهش و یک تابع انقطاع را برای مساله ی مسافرت شخص دوره گرد پیدا نمایید . فرض کنید تعداد شهرها (N) برابر ۱۰ می باشد . بعد از همه ی عملیات ، کروموزوم های تولید شده باید همیشه مسافرت های ممکن مجاز را ارایه نمایند (هر شهر فقط باید یک بار ملاقات شود) . برای این مساله ، یک راه حل وجود ندارد و تعداد زیادی الگوهای مختلف ، قبلا استفاده شده اند .

منابع :

http://www.cs.usfca.edu
http://barbara.inrialpes.fr/exmo/people/alkhateeb/papers/masterjordan.doc

[1] Partially Matched Crossover (PMX)

http://www.crim.ca/files/documents/evenements/duCRIM/tiDinners/tidinner010501.pdf

http://www.cs.unibo.it/~babaoglu/courses/cas/slides/ga.pdf

فصل بیست و یکم

سیستم های خبره

سیستم های خبره
فصل بیست و یکم

سیستم های خبره

- سیستم های خبره برنامه هایی طراحی شده برای مدل سازی و استدلال در مورد دانش انسان می باشند و معمولا حول یک دامنه تمرکز می کنند
  - تشخیص بیماری
  - پرواز فضایی
  - لجستیک (اقدامات مربوط به تهیه و توزیع)
  - عیب یابی نرم افزار
- برنامه ریزی شده به صورت اظهاری (اعلانی) می باشند
  - برنامه نویسان *واقعیات* و *قوانین* را اضافه می کنند
  - سیستم پردازه ی استنتاج را به صورت خودکار در می آورد .
- می توانند یا از زنجیره ی مستقیم (سیستم های تولید) یا از زنجیره ی معکوس (ثابت کننده های قضیه) استفاده نمایند .

یک سیستم خبره چیست ؟

خبره - در صورتی که ما بگوییم که کسی در رشته ای ، یک خبره می باشد منظور ما این است که ، در آن تخصص محدود ، شایستگی بالایی را به نمایش می گذارد . اگر ما یک عامل تنها را بسازیم و آن را یک سیستم خبره بنامیم ، سیستم همیشه اطلاعات کلی تر را به نمایش نمی گذارد ؛ در یک محدوده ی نسبی و با تخصص نسبی کار می کند . اما در آن زمینه عملکردش در سطح خبره می باشد . این سیستم ها ممکن است برای طبقه بندی ، تشخیص عیب ، پیداکردن خرابی ، تفسیر اطلاعات ، طراحی ، پیکربندی یا پیش بینی به کار روند . آن ها شاید برای آموزش دادن ، نمایش دادن ، تحلیل ، مشاوره ، تجدید نظر یا کنترل به کار روند . استفاده ی تجاری سیستم های خبره شامل این موارد می باشند : یک سیستم استفاده شده برای ارایه ی نصیحت در مورد وام خانه ؛ یک سیستم استفاده شده توسط یک تولید کننده ی کامپیوتر برای بررسی اجرای کامل دستورات مشتریان ؛ یک سیستم استفاده شده در بیمارستان برای تفسیر اندازه گیری های ریوی برای مشاهده ی علایم ناخوشی ریه ؛ یک سیستم استفاده شده توسط شیمیدان ها برای تفسیر توده ی داده های

طیف سنج برای کمک به پی بردن به ساختار مولکولی ترکیبات آلی (زیستی) [1] ؛ و یک سیستم برای کمک به زمین شناسان برای ارزیابی مکان های معدنی برای ذخایر .

چرا یک سیستم خبره را می سازیم ؟

- برای انتشار (تکثیر) خبره ی نادر و پرهزینه
- برای فرمول بندی دانش خبره
- برای جمع آوری منابع دانش متمایز

دلایل ممکن دیگری هم وجود دارند . برخی از افراد دلیل می آورند که سیستم های خبره در مقایسه با انسان ها کم تر خطا می کنند و مناسب تر هستند و رک تر هستند ، یا در مقایسه با خبره های بشری بی طرف تر می باشند . البته ، سیستم های خبره معایب زیادی هم دارند ، که باید استفاده از آن ها را به دامنه های مشخص ، محدود نماییم . سیستم های خبره دارای بینش ، دلسوزی ، فهم انگیزه ی بشری ، توانایی حدس ، توانایی یادگیری (معمولا) ، دانش قضاوت صحیح (عقل سلیم) کمی می باشند . در این مورد کاربر می تواند دلسوزی و قضاوت صحیح را اضافه نماید .

خصوصیات خوب یک سیستم خبره

- در صورتی که سیستم محاوره ای باشد ، یک رابط کاربری خوب ، بسیار ضروری می باشد . توجه کنید که مکالمه ای که سیستم خبره با کاربر انجام می دهد باید به صورت "طبیعی" توسط کاربر مطرح شده باشد . که شامل مواردی نظیر موارد زیر می باشد :
  - شیوه ی سوالاتی که سیستم می پرسد باید طبیعی باشد .
  - سوالات احمقانه نباید وجود داشته باشند (کسانی که به سیستم جواب می دهند باید با دلیل ، تدبیر کرده باشند) .
  - سیستم باید قادر باشد که توضیح دهد که چرا یک سوال را می پرسد و هر نتیجه ای که به آن می رسد را توجیه نماید . باید به کاربر در مورد سیستم ، اطمینان دهد .
- در استدلال ، باید یک سیستم خبره ، قادر به انجام موارد زیر باشد :
  - باید استنتاج هایش باورکردنی و نه الزاما صحیح باشند . به عنوان مثال ، یک سیستم خبره که یک وضعیت بهداشتی را از یک مجموعه از علایم ، استنتاج می کند ؛ بعید است که وضعیت ، رشته ای منطقی از علایم (نشانه ها) باشد .
  - باید قادر به کار با دامنه ی دانش ناقص و مواردی که اطلاعات ناقص هستند ، باشد . اغلب ، دانش و / یا مورد اطلاعات ، ناکامل می باشند و دانش و / یا داده ها ممکن است قابل اعتماد نباشند (مثلا ممکن است دارای خطا باشند) و شاید دانش و / یا داده ها به صورت نادرست بیان شده باشند (به عنوان مثال اگر دارای دندان بلندی باشد ، آن گاه این خطرناک می باشد) .
  - همچنین باید سیستم ، رقابت همزمان مفروضات را در نظر داشته باشد .
- سیستم های خبره باید با نگهداشت پذیری (ویژگی مربوط به جداسازی و تعمیر یک خرابی) [2] طراحی شده باشند . باید بتوانند به سادگی اطلاعات جدید را یکی نمایند (یا از طریق مهندسی دانش بیش تر یا با استفاده از آموزش ماشینی) .

سیستم های خبره ی قانون گرا [3]

---

[1] ماده ای که مولکول های آن شامل یک یا بیش تر اتم های کربن است ، به استثنای کربنات ها ، سیانیدها ، کربیدها ، و تعدادی موارد دیگر (دایره المعارف بریتانیکا) ، در ضمن در لغت نامه ی مهندسی محیط هم ، چنین آمده است : گروه بزرگی از ترکیبات شیمیایی که معمولا شامل کربن ، هیدروژن ، نیتروژن و اکسیژن می باشند . تمام موجودات زنده از این ترکیبات آلی تشکیل شده اند .

[2] maintainability

[3] rule-based

روش های زیادی می توانند برای ساخت سیستم های خبره استفاده شوند . اما ، اکثر سیستم های خبره که تا کنون ساخته شده اند سیستم های قانون گرا می باشند : پایگاه دانش شامل واقعیت ها و قانون ها می باشد .

پایگاه دانش

پایگاه دانش شامل واقعیت ها و قوانین می باشد . تصور کنید که ما در حال ساخت یک سیستم خبره ی قانون گرا برای تشخیص پزشکی بودیم . قانون ها ارتباطات میان علایم و عبارت های پزشکی را کد خواهند کرد . واقعیت ها دانش در مورد بیماری جاری را کد خواهند کرد . در سیستم های قانون گرا ، اغلب به صورت اگر ... آن گاه ... نوشته می شوند ، اما به صورت برابر می توانند به یکی از شکل هایی که برای موارد زیر استفاده می شوند نوشته شوند :

$if \quad p_1 \quad AND...AND \quad p_n \quad then \quad q$

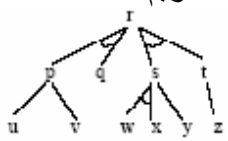
$(p_1 \wedge ... \wedge p_n) \Rightarrow q$

$q \vee \neg p_1 \vee ... \vee \neg p_n$

برای سیستم های قانون گرا استفاده از منطق گزاره ای در جایی که گزاره ها دارای هیچ آرگومانی نمی باشند ، کاملا معمولی است . بنابراین متغیرها و سمبل های تابعی در واقعیت ها و قانون ها وجود ندارند . ما در مثال های زیر خودمان را با این وضعیت ، محدود نموده ایم . نتیجه ی یک قانون شاید یک قلم تجزیه ناپذیر باشد که در قانون قبلی آمده است . ما می توانیم این زنجیره را به شکل گرافیکی به صورت یک گراف AND-OR نمایش دهیم . برای مثال ، برای قانون های زیر :

if p AND q **then** r
if s AND t **then** r
if u **then** p
if v **then** p
if w AND x **then** s
if y **then** s
if z **then** t

که گراف زیر را در مورد قانون های بالا داریم :



در زیر مثالی از پایگاه دانشی برای شناسایی حیوانات را می بینید :

if *animalGivesMilk* **then** *animalIsMammal*
if *animalHasHair* **then** *animalIsMammal*
if *animalIsMammal* AND *animalChewsCud* **then** *animalIsUngulate*
if *animalIsUngulate* AND *animalHasLongNeck* **then** *animalIsGiraffe*
if *animalIsUngulate* AND *animalIsStriped* **then** *animalIsZebra*

ما از مثال زیر استفاده خواهیم کرد .

موتور استنتاج

موتور استنتاج ، استنتاج ها را با استفاده از دانش موجود در پایگاه دانش ، استخراج می کند و با استفاده از استنتاج قانون گرا [1] انجام می دهد . از یک نقطه نظر منطقی ، اساسا از تحلیل استفاده می کند .

یک چشم انداز (دورنمای سه بعدی) استنتاج (استدلال) قانون گرا این است که در حال انجام یک جستجوی گرافی AND-OR می باشد . به طور موثر ، ما به دنبال یک مسیر که ریشه و برگ ها را به هم متصل می کند و موارد زیر را اجرا می کند می گردیم :

- در صورتی که گره ای ، یک گره ی OR می باشد ، برای نمایش این که یک مسیر برای فقط یکی از نسل ها وجود دارد ، مناسب می باشد ؛
- اگر گره ای ، یک گره ی AND باشد ، برای نمایش این که برای هر یک از نسل ها مسیری وجود دارد ، لازم می باشد ؛
- در صورتی که یک گره ، یک برگ باشد ، گره یک واقعیت را که باید درست باشد را نمایش می دهد .

در سیستم های خبره ی محاوره ای ، ما فرض نمی کنیم که همه ی واقعیات شناخته شده اند و قبلا در پایگاه دانش وجود داشته اند . بنابراین ، در زمانی که ما در تلاش برای این هستیم که ببینیم که آیا یک گره ی برگ (واقعیت) درست است یا نه ، ما پایگاه دانش را بررسی خواهیم نمود ، اما اگر در پایگاه دانش نباشد ، در این صورت ما از کاربر پرسش می کنیم که آیا واقعیت درست است یا نه . جواب کاربر می تواند به پایگاه دانش اضافه شود . (این محاوره ، چیزی است که استدلال قانون گرا را از تحلیل SLD متمایز می کند .)

زنجیره ی معکوس (زنجیره ای کردن به صورت معکوس یا زنجیره ی پس رو)

بیش تر استدلال قانون گرا به روشی انجام می شود که شبیه تحلیل SLD می باشد (همان طوری که در پاراگراف قبل گفته شد ، دلایلی وجود دارد که به ندرت با تحلیل SLD برابر می باشند) . اما ، عبارت های زنجیره ی معکوس ، استدلال مشتق شده از هدف می باشند و استدلال مشتق شده از فرض به صورت عمومی تری در جامعه ی سیستم های خبره مورد استفاده قرار می گیرند . در عبارات گراف AND-OR ، این نوع از استدلال در ریشه ی گراف شروع می شود و برای پیداکردن مسیری از ریشه به برگ ها تلاش می نماید .

زنجیره ی مستقیم

این روش ، استدلال مشتق شده از داده ها هم نامیده می شود . در عبارات گراف AND-OR ، این نوع از استدلال از برگ ها شروع می کند و تلاش می کند که مسیری را از برگ ها به طرف ریشه پیدا نماید .

زنجیره ی معکوس و مستقیم برگ برگ شده [2]

برخی از سیستم ها منحصرا از یکی از این روش ها یا هر دوی این روش ها استفاده می کنند . اما تعداد زیادی از سیستم های برگ برگی از هر دو روش استفاده می کنند که این کار خیلی طبیعی می تواند باشد . به یک مشاوره با یک دکتر بشری توجه نمایید . بیمار برخی از علایم را تشریح می کند . نتایج با استفاده از زنجیره ی مستقیم به دست می آید (شاید فقط به طور آزمایشی) . دکتر یک فرض را انتخاب می کند و با استفاده از زنجیره ی معکوس به پرسشی می رسد که برای بیمار ارایه شده است . بیمار ، دوباره صحبت می کند ، و شاید به پرسشی دیگر جواب دهد و یا از او اطلاعات دیگری خواسته شود . و این کار (پروسه) باز هم تکرار شود .

توضیح های استدلال

---

[1] Rule-Based Reasoning (RBR)
[2] Interleaved Backwards- and Forwards-Chaining

ما قبلا اهمیت داشتن توضیح روان را در یک سیستم خبره بیان کردیم . سیستم های قانون گرا معمولا دو امکان را برای توضیحات ارایه می کنند . ممکن است یا بپرسند چرا و یا بپرسند چگونه . سیستم ، سوالات را با نمایش برخی از قوانین مربوط ، جواب می دهد . برای مثال ، تصور نمایید که سیستم خبره ی شناسایی حیوانات از کاربر این پرسش را می پرسد که آیا حیوان ، نشخوار می کند . در این مورد ، کاربر می تواند به جای جواب دادن به این سوال بپرسد که : چرا شما این سوال را می پرسید ؟ برای این کار ، سیستم خبره با نمایش یک گراف AND-OR پاسخ می دهد . برای مثال ممکن است سیستم پاسخ دهد : *من از شما پرسیدم که آیا حیوان نشخوار می کند ، زیرا این در تشخیص این که حیوان جانور سم دار است کمک می کند . قبلا تشخیص داده شده که جانور ، پستاندار است . بنابراین اگر حیوان نشخوار می کند آن گاه حیوان ، سم دار می باشد . این در تشخیص این که آیا حیوان زرافه است کمک می کند . در صورتی که حیوان سم دار باشد و دارای گردن بلند باشد ، زرافه می باشد* . به بیان دیگر ، تصور نمایید که یک سیستم خبره تشخیص داده که برخی از گره ها درست هستند . در گفتن نتیجه به کاربر ، کاربر می تواند چگونگی توضیح را بپرسد : چگونه به این نتیجه رسیدی ؟ برای این کار ، سیستم خبره با نمایش قسمت های موفق گراف AND-OR ، جواب می دهد . ایده این است که برای تصدیق (توجیه کردن) یک استنتاج ، سیستم باید نشان دهد که کدام قانون ها در رسیدن به آن نتیجه اجرا می شوند . برای مثال ، تصور کنید در موردی که سیستم تشخیص می دهد که حیوان یک جانور سم دار می باشد ، کاربر ممکن است درخواست کند که چگونه به این نتیجه رسیدی و این کار ممکن است به صورت زیر باشد : این قانون برای تشخیص این که حیوان ، سم دار است استفاده شده : اگر حیوان ، پستاندار است و نشخوار می کند ، سم دار می باشد . این قانون برای تشخیص این که آیا جانور پستاندار است استفاده شده است : حیوان دارای مو می باشد ، پس پستاندار می باشد . شما به من گفتید که حیوان دارای مو می باشد . شما به من گفتید که حیوان نشخوار می کند .

بیش تر تحقیقات انقلابی در سیستم های قانون گرا در مورد سیستمی به نام MYCIN انجام شده است . ایده های درون MYCIN دارای برتری بر همه ی سیستم های خبره ی قانون گرا هستند . MYCIN برای استفاده ی یک پزشک برای تشخیص عفونت های باکتریایی خون طراحی شد . در حدود ۴۵۰ قانون دارد و تقریبا به طور کامل با استفاده از زنجیره ی معکوس کار می کند . بازده MYCIN و طبیعی بودن مکالمه ی آن با مشارکت قطعه های خیلی کوچک دانش که شاید در مدت زنجیره ی معکوس به کار گرفته شوند ، بهبود یافت . یکی از پرمعنی ترین این موارد ، استفاده از یک مجموعه از فرا قانون ها می باشد که برای کمک به تصمیم در مورد این که کدام قانون های نرمال به صورت بعدی باید استفاده شوند ، طراحی شدند . یکی دیگر از سیستم های قانون گرای خبره ی مشهور ، PROSPECTOR (برای ارزیابی مکان های معدنی دارای استعداد برای استخراج) و R1 (a.k.a. XCON) برای بررسی ، کامل کردن ، و پیکربندی تقاضاهای تجهیزات کامپیوتری مشتری ، می باشند . بعد از برخی از آزمایش ها با ساخت چند سیستم خبره ی قانون گرا ، محققان هوش مصنوعی دریافتند که فقط پایگاه دانش ، وابسته به دامنه می باشد . موتور استنتاج و رابط کاربر (نسبتا) مستقل از دامنه بودند . به طور کلی ، برای ساخت یک سیستم خبره ی جدید برای یک دامنه ی مختلف به یک پروسه ی مهندسی دانش برای دریافت کردن قانون ها نیاز داریم .

Jess

- Jess یک نوع لایه ی معروف سیستم خبره بنام CLIPS می باشد و از سی به جاوا برده شده است و دارای زبان ارایه ی دانش خودش می باشد . همچنین می تواند با کتابخانه های جاوا ارتباط برقرار نماید و یک سیستم زنجیره ی مستقیم است

- شما واقعیات و قوانین را می نویسید و Jess استنتاج های آن ها را به دست می آورد .

منابع :

http://www.cs.ucc.ie/~dgb/courses/ai/notes/notes36.pdf
http://www.cs.usfca.edu

فصل بیست و دوم

پردازش های تصمیم گیری مارکوف

پردازش های تصمیم گیری مارکوف [1]
فصل بیست و دوم

پردازش های تصمیم گیری مارکوفی – یک مدل گسسته برای تصمیم گیری تحت عدم قطعیت می باشد . چهار جزء این مدل عبارتند از : وضعیت ها ، که جهان به وضعیت هایی تقسیم شده است . عملکردها ، هر وضعیت دارای یک تعداد محدود عملکردها می باشد . تابع انتقال ، ارتباطی احتمالی میان وضعیت ها و عملکردهای قابل دسترسی برای هر وضعیت . تابع پاداش ، پاداش مورد انتظار انجام یک کار ، تحت وضعیت s . [2]

به وجود آوردن تصمیم گیری های ترتیبی - قبلا ما در مورد موارد زیر صحبت کردیم :

- به وجود آوردن تصمیم گیری های تک ضربه در یک محیط قطعی
- به وجود آوردن تصمیم گیری های ترتیبی در یک محیط قطعی
  - جستجو – استنتاج - برنامه ریزی
- به وجود آوردن تصمیم گیری های تک ضربه در یک محیط اتفاقی
  - شبکه های باور و احتمال - پیش بینی سودمندی

تصمیم گیری های ترتیبی در یک محیط اتفاقی چگونه می باشد ؟

سودمندی مورد انتظار - به یاد بیاورید که سودمندی مورد انتظار یک عملکرد ، سودمندی هر نتیجه ی ممکن است و توسط احتمالی که نتیجه اتفاق بیفتد وزن می شود . به طور صریح تر ، از وضعیت s ، یک عامل ممکن است عملکردهای $a_1, a_2, ..., a_n$ را بگیرد . هر عملکرد a_i می تواند منجر به وضعیت های $s_{i1}, s_{i2}, ..., s_{im}$ با احتمال $p_{i1}, p_{i2}, ..., p_{im}$ شود : $EU(a_i) = \sum p_{ij} s_{ij}$. ما مجموعه ای از احتمال ها و وضعیت های وابسته را مدل انتقال [3] وضعیت می نامیم . عامل باید وضعیت a' ای که EU را بیشینه می کند را انتخاب نماید .

محیط های مارکوفی - ما می توانیم این ایده را برای محیط های ترتیبی بسط دهیم . مساله : چگونه احتمال وضعیت ها را تشخیص دهیم ؟ احتمال رسیدن به وضعیت s ارایه شده توسط عملکرد a ممکن است وابسته به عملکردهای قبلی ای که انجام شده اند باشد . فرض مارکوف می گوید که احتمال انتقال وضعیت ها فقط وابسته به یک تعداد محدود از والدها می باشد . ساده ترین ، پردازه ی مرتبه ی اول مارکوف می باشد . احتمال انتقال وضعیت ها فقط وابسته به وضعیت قبلی می باشد . ما نیز روی پردازش مرتبه ی اول مارکوف تمرکز می نماییم .

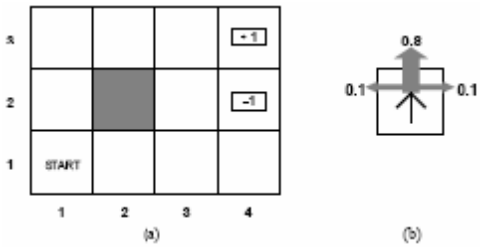
---

[1] Markov Decision Processes(MDP)
[2] http://www.sci.brooklyn.cuny.edu/~parsons/courses/790-spring-2004/notes/incremental.ppt
[3] state transition model

توزیعات ثابت - ما فرض می کنیم که توزیعمان ثابت باشد . این می گوید که احتمال رسیدن به یک وضعیت s' ارائه شده توسط عملکرد a از وضعیت s با تاریخچه ی H تغییر نمی یابد . تاریخچه های مختلف ممکن است احتمال های مختلف را تولید نمایند . ارایه ی تاریخچه های یکسان ، انتقال وضعیت های یکسان را سبب خواهد شد . ما همچنین فرض خواهیم کرد که سودمندی وضعیت تغییری را در زمینه ی مساله به وجود نمی آورد .

حل مسایل ترتیبی - سودمندی وابسته به یک رشته از وضعیت های $s_1, s_2, ..., s_n$ می باشد . بیایید هر وضعیت را به یک پاداش $R(s_i)$ انتساب دهیم . عامل می خواهد مجموع پاداش ها را بیشینه نماید . ما این فرمول بندی را یک پردازش تصمیم گیری مارکوفی می نامیم . به طور صریح ، این موارد را در MDP داریم : یک وضعیت اولیه ی s_0 ، یک مجموعه ی وضعیت ها و عملکردهای گسسته ، یک مدل انتقال : $T(s,a,s')$ که احتمال رسیدن وضعیت s' از s در زمانی که عمل a انجام می شود را نشان می دهد . یک تابع پاداش : R(s) .

مثال مساله ی توری



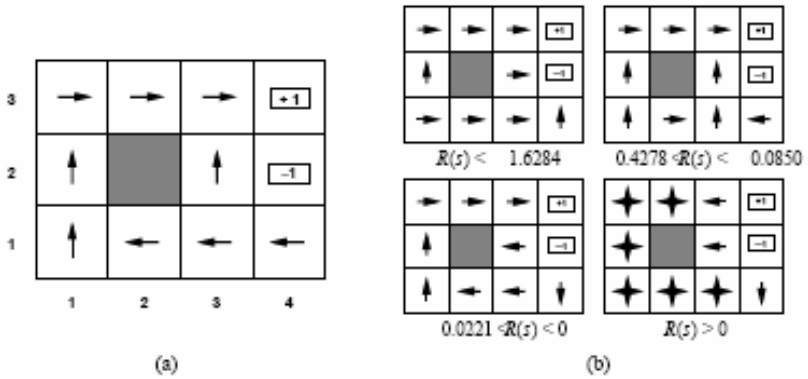
- عامل در جهت خواسته شده با احتمال ۰,۸ حرکت می کند و در یک جهت راست با احتمال ۰,۲ حرکت می کند . در هر وضعیت یک عامل باید چه کاری انجام دهد تا پاداش را بیشینه نماید ؟

راه حل های MDP - از آنجایی که محیط ، اتفاقی می باشد ، یک راه حل به صورت رشته ای از عملکرد نمی باشد . در عوض ، ما باید معین نماییم که یک عامل باید چه کاری در هر وضعیت ممکن انجام دهد . ما این خصوصیت را یک *روش* [1] می نامیم : " اگر شما زیر هدف می باشید بالا بیایید . " و اگر شما در ستون سمت چپ می باشید ، به راست حرکت نمایید . " ما یک روش را با π نشان می دهیم و $\pi(s)$ روش را برای وضعیت s مشخص می نماید .

مقایسه ی روش ها - ما می توانیم روش ها را با توجه به سودمندی مورد انتظار تاریخچه هایی که آن ها تولید می کنند مقایسه نماییم . روشی که دارای بالاترین سودمندی مورد انتظار می باشد *روش بهینه* [2] نام دارد . اولین باری که یک روش پیدا می شود ، عامل فقط می تواند بهترین عملکرد را برای هر وضعیت پیدا نماید .

مثال مساله ی توری

[1] policy
[2] optimal policy



- از آنجایی که هزینه ی بودن در یک وضعیت غیرپایانی تغییر می یابد ، بنابراین ، روش بهینه را انجام می دهد .
- هزینه ی خیلی بالا : عامل تلاش می کند که بلافاصله خارج شود ، حتی با خروج بد .
- زمینه ی میانی : عامل تلاش می نماید که از خروج بد جلوگیری نماید .

مطالب بیش تری در مورد توابع پاداش

- در حل یک MDP،یک عامل باید به مقدار عملکردهای آینده توجه نماید.انواع مختلفی از مسایل که باید به آن ها توجه نمایید ، وجود دارد :
- وسعت – آیا جهان برای همیشه ادامه می یابد ؟
  - وسعت محدود : بعد از N عملیات ، جهان متوقف می شود و هیچ پاداشی دریافت نمی شود .
  - وسعت نامحدود : جهان به صورت نامحدود ادامه می یابد یا این که ما نمی دانیم چه هنگامی متوقف می شود .
    - در وسعت نامحدود روش ها در طول زمان تغییر نمی کنند .
- ما همچنین نیاز داریم که در مورد چگونگی مقدار پاداش در آینده فکر کنیم ، صد دلار در یک روز دارای ارزش بیش تری نسبت به صد دلار در یک سال است . ما با *تخفیف* [1] در پاداش های آینده این مورد را مدل سازی می نماییم .
- $U(s_0, s_1, s_2, s_3, ...) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + ..., \gamma \in [0,1]$
- در صورتی که γ بزرگ باشد ، ما وضعیت های آینده را مقدار دهی می نماییم . در صورتی که γ کوچک باشد ما روی پاداش نزدیک تمرکز می نماییم . در موارد پولی ، یک فاکتور تخفیف γ برابر است با یک بهره با نرخ $(1/\gamma) - 1$. تخفیف به ما اجازه می دهد که مقدار محسوسی ، مسایل با وسعت نامحدود را داشته باشیم . در غیر این صورت ، همه ی EU ها باید نزدیک به مقدار نامحدود باشند .
- سودمندی مورد انتظار در صورتی که پاداش ها محدود و در محدوده باشند و $\gamma<1$ باشد ، محدود خواهند شد . ما حالا می توانیم روش بهینه ی π^* را به این صورت بیان نماییم :

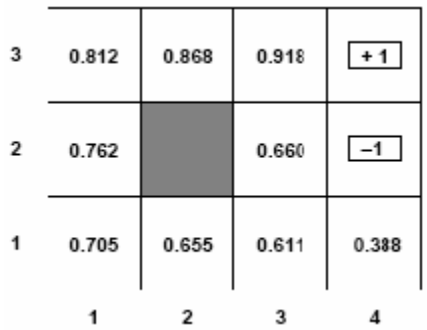
$$\pi^* = \arg\max_\pi EU(\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi)$$

تکرار مقدار - چگونه یک روش بهینه را پیدا نماییم ؟ ما با محاسبه ی سودمندی مورد انتظار هر وضعیت و سپس انتخاب عملکردهایی که سودمندی مورد انتظار را بیشینه می کنند شروع می نماییم . در یک مساله ی ترتیبی ، سودمندی یک وضعیت سودمندی مورد انتظار همه ی رشته های

[1] discounting

وضعیتی که از آن دنبال می شوند ، می باشد . این به روشی که π اجرا می شود وابسته می باشد . در اصل ، U(s) سودمندی مورد انتظار یک روش بهینه از وضعیت s می باشد .

سودمندی وضعیت ها



- توجه کنید که سودمندی ها دارای بیش ترین مقدار برای وضعیت های نزدیک به خروجی ۱+ هستند . سودمندی یک وضعیت ، پاداش فوری برای آن وضعیت به اضافه ی سود تخفیف داده شده ی مورد انتظار وضعیت بعدی می باشد و فرض بر این است که عامل ، عملکرد بهینه را انتخاب می نماید . معادله ی بلمن :

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

- مثال :

$$U(1,1)=-0.04+\gamma\max(0.8U(1,2)+0.1U(2,1)+0.1U(1,1),$$
$$0.9U(1,1)+0.1U(1,2),$$
$$0.9U(1,1)+0.1U(2,1),$$
$$0.8U(2,1)+0.1U(1,2)+0.1U(1,1))$$

برنامه نویسی پویا - معادله ی بلمن ، براساس برنامه نویسی پویا می باشد . در یک گراف انتقال بدون حلقه ، شما می توانید این ها را به صورت بازگشتی توسط معکوس کارکردن از وضعیت نهایی به وضعیت های اولیه حل نمایید . شما این کار را برای گراف های انتقال با حلقه ، به صورت مستقیم نمی توانید انجام دهید .

تکرار مقدار

- از آنجایی که سودمندی وضعیت ها در سودمندی سایر وضعیت ها تعریف می شود ، چگونه یک راه حل نزدیک را پیدا نماییم ؟ ما می توانیم از یک نگرش تکراری به صورت زیر استفاده نماییم :
  - هر وضعیت را به صورت تصادفی مقداردهی اولیه نمایید . طرف چپ جدید را برای یک وضعیت براساس مقایر همسایگانش محاسبه نمایید . برای به روز رسانی طرف راست برای سایر وضعیت ها این مورد را انتشار دهید و قانون به روز رسانی :

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U_i(s')$$

- این برای همگرایی با راه حل های معادلات بلمن ضمانت شده است .

الگوریتم تکرار مقدار

انجام دهید

برای s در وضعیت ها

$$U(s)=R(s)+\max T(s,a,s')\ U(s')$$

تا هنگامی که
تمام سودمندی ها با مقدار کم تر از دلتا تغییر نمایند

- اگر $\delta = error * (1-\gamma)/\gamma$ باشد .

تشریح مطلب

- نقاط قوت تکرار مقدار : برای نزدیک شدن به راه حل صحیح ، ضمانت شده است و الگوریتم تکرار ساده می باشد
- ضعف : نزدیک شدن می تواند به صورت کند باشد ، ما در واقع به تمام این اطلاعات نیاز نداریم و فقط به این نیاز داریم که بدانیم در هر وضعیت چه کاری را انجام دهیم .

تکرار روش

- تکرار روش به آدرس دهی این ضعف ها کمک می کند . به طور مستقیم برای روش های بهینه و ترجیحا سودمندی وضعیت ها جستجو می نماید . ایده ی شبیه : روش های به صورت تکراری برای هر وضعیت به روز رسانی را انجام می دهند .
- دو مرحله:برای یک روش ، سودمندی ها را برای هر وضعیت محاسبه نمایید و یک روش جدید را براساس این سودمندی ها محاسبه نمایید .

الگوریتم تکرار روش

Pi = بردار روش تصادفی شاخص گذاری شده توسط وضعیت
انجام دهید
U = ارزیابی سودمندی هر وضعیت برای Pi
برای s در وضعیت ها
a = عملکردی را که سودمندی مورد انتظار را برای آن وضعیت بیشینه می نماید
پیدا نمایید
Pi(s)=a
مادامی که برخی از عملکردها تغییر داده شوند

تشریح مطلب - روش و تکرار مقدار ، یک محیط کاملا قابل مشاهده را به وجود می آورند . MDP هایی که به صورت جزیی قابل مشاهده باشند محیط های قابل مشاهده به صورت جزیی را به کار می گیرند . همچنین دارای یک ساختار هدف/پاداش ثابت می باشند . برای پیدا کردن رشته های عملکرد بهینه موثر می باشد ، فرض کنید که مدل انتقال T(s,a,s') شناخته شده می باشد .

منابع :

http://www.cs.usfca.edu

فصل بیست و سوم

سیستم های طبقه بندی کننده

سیستم های طبقه بندی کننده [1]
فصل بیست و سوم

سیستم های طبقه بندی کننده - الگوریتم های ژنتیکی برای مسایل بهینه سازی خیلی خوب کار می کنند : $f(x_1, x_2, x_3, ..., x_n) = R$. در مورد مسایلی که کم تر به صورت خوب تعریف شده اند چطور ؟ آیا ما می توانیم از این ایده ها برای ساختن یک عامل جاروبرقی استفاده نماییم ؟ بله ، ولی ما نیاز داریم که کمی بادقت تر باشیم . *طبقه بندی* ، پردازه ی انتساب یک ورودی به یکی از کلاس های چندگانه می باشد .

سیستم های تولید - ماشین های تولید یک روش مشترک هوش مصنوعی هستند . اگر – آن گاه یا قانون های شرط – عملکرد . در صورتی که [۱و۱] جارو نشده باشد و همسایه باشد ، آن گاه به [۱،۱] برو . یک سیستم تولید از یک بدنه ی بزرگ از قانون ها تشکیل شده است . برخی از قوانین ممکن است باعث عمل کردن قانون های دیگر شوند . در سیستم های تجاری ، مساله این است که وقتی بیش از یک قانون مطابقت داده می شود چه کار کنیم ، ما بعدا به این مورد خواهیم پرداخت .

طبقه بندی کننده ها - الگوریتم های ژنتیکی به صورت بالقوه به ما یک روش برای استنتاج و انتخاب قانون ها را ارایه می دهند ، قانونی را که دارای بیش ترین شایستگی می باشد را انتخاب نمایید ، توسط ۰،۱ و * قانون ها را به صورت یک رشته ی بیتی کد نمایید . بیت های وضعیت : بیت های عملکرد . مطالب : چگونه شایستگی را به یک قانون انتساب دهیم ؟ و چگونه عملکرد برخط بالا را نگهداری نماییم ؟

اجزای یک سیستم طبقه بندی - یک سیستم طبقه بندی دارای سه جزء می باشد : یک قانون و سیستم پیام ، یک سیستم انتساب اعتبار و یک الگوریتم ژنتیکی برای تولید قانون های جدید .

قانون و سیستم پیام - حسگرهای عامل ، اطلاعات را که به صورت یک رشته ی بیتی کد شده اند را دریافت می نمایند ، این اطلاعات پیامی از محیط می باشد . ین پیام طبقه بندی کننده ها (قانون ها) را با تطبیق وضعیت ها فعال می نمایند . طبقه بندی کننده ها پیام هایشان را به لیست پیام ارسال می نمایند ، این پیام ها ممکن است دیگر طبقه بندی کننده ها یا عمل کننده های عامل را فعال نمایند .

مثال - فرض کنید سیستم ما دارای طبقه بندی کننده های زیر می باشد :

1. 01## : 0000
2. 00#0 : 1100

---

[1] Classifier Systems

3. 11## : 1000
4. ##00 : 0001

- پیام ۰۱۱۱ از محیط دریافت می شود .
  - o قانون یک اجرا می شود ، ۰۰۰۰ در لیست پیام قرار می گیرد .
  - o قانون دو و چهار اجرا می شود ، ۱۱۰۰ و ۰۰۰۱ در لیست پیام قرار می گیرند .
  - o قانون سه اجرا می شود ، ۱۰۰۰ در لیست پیام قرار می گیرد .
  - o قانون چهار و پیامی که در لیست پیام می باشد منطبق می شود .
- حالا چند پیام در لیست پیام قرار دارد – کدام به عمل کننده ها فرستاده می شود ؟

دسته باکت [1] - الگوریتم دسته باکت به قانون ها برای پیشنهاد اجرا براساس عملکرد قبلی اجازه می دهد . وقتی که یک قانون منطبق می شود ، در یک " مزایده [2] " شرکت می نماید . هر قانون براساس عملکرد قبلی ، دارای یک قوت می باشد ؛ یک قانون یک نسبت از قدرت را داراست . بالاترین قانون های شرکت کننده برنده می شوند و این پیشنهاد به طبقه بندی کننده (ها) ای که آن را فعال می نماید فرستاده می شود .

مثال

- به صورت اولیه (t=0) : M = ۰۱۱۱

1) 01## : 0000 S=200 B=20
2) 00#0 : 1100 S=200
3) 11## : 1000 S=200
4) ##00 : 0001 S=200

محیط S=0

- t=1

1. 01## : 0000 S=180 0000
2. 00#0 : 1100 S=200 B=20
3. 11## : 1000 S=200
4. ##00 : 0001 S=200 B=20

محیط S=20

- t=2

1. 01## : 0000 S=220 1100
2. 00#0 : 1100 S=180 0001
3. 11## : 1000 S=200 B=20
4. ##00 : 0001 S=180 B=18

محیط S=20

- t=3

1. 01## : 0000 S=220 1000
2. 00#0 : 1100 S=218 0001

[1] bucket brigade
[2] auction

3. 11## : 1000 S=180
4. ##00 : 0001 S=162 B=16

محیط S=20

- t=4

5. 01## : 0000 S=220 0001
6. 00#0 : 1100 S=208
7. 11## : 1000 S=196
8. ##00 : 0001 S=156

محیط S=20

- t=5 – درخواست به سیستم می آید و به آخرین طبقه بندی کننده ی فعال منتسب می شود .
  - 01## : 0000 S=220
  - 00#0 : 1100 S=208
  - 11## : 1000 S=196
  - ##00 : 0001 S=206

محیط S=20

تولید قوانین جدید - دسته باکت یک روش انتخاب قوانین و انتساب اعتبار را ارایه می نماید . چگونه قوانین جدید را به دست بیاوریم ؟ الگوریتم ژنتیکی پایه ی ما از یک مدل جمعیت که با هم اشتراک ندارند استفاده می نماید؛ همه ی جمعیت در زمان t ، در زمان t+1 جایگزین می شود . به خوبی برای بهینه سازی عمل می کند ، ولی برای آموزش ، زیاد مناسب نمی باشد . در عوض ما از نخبه سالاری برای حفظ برخی از قانون ها استفاده می نماییم ، از انتخاب رولت برای حفظ قانون ها استفاده نمایید که کندتر استنتاج می کند .

برنامه نویسی ژنتیکی – فراتر از قوانین - روش های تکاملی برای تولید برنامه ها به کار گرفته شده اند . ایده : یک برنامه می تواند به صورت یک S- عبارتی بیان شود : (+ 3 (* 4 x)) . ما می توانیم این را به صورت یک درخت بکشیم . ما سپس تکامل را روی یک جمعیت از برنامه ها انجام می دهیم . شایستگی ، کارایی یک برنامه برای یک عملکرد ارایه شده است . انقطاع ، زیر درخت ها را با هم عوض می نماید . جهش ، عملگرها را عوض می نماید . برای تکامل استفاده می شوند ، مثل : برنامه های فوتبال روبوت ، مدارهای کنترل آنالوگ ، فیلترها و مدارهای پیچیده . چالش ها : فضای پهناور (نامحدود) برنامه ها و کمبود ساختارهای سطح بالاتر برنامه .

منابع :

http://www.cs.usfca.edu

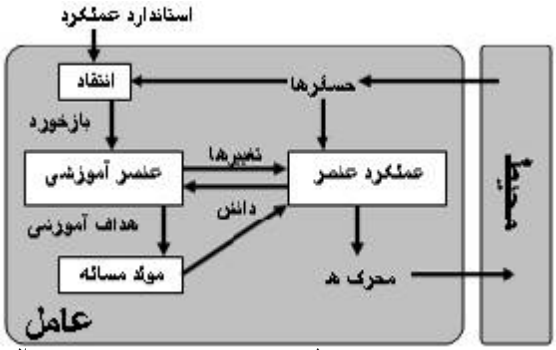
فصل بیست و چهارم

درخت های آموزشی – تصمیم گیری

درخت های آموزشی – تصمیم گیری
فصل بیست و چهارم

آموزش - برای یک عامل ، آموزش به چه معنی می باشد ؟ عامل دانش جدید را دریافت می نماید ، عامل ، رفتارش را تغییر می دهد و عامل در یک کار ارایه شده معیار کارایی خود را بهبود می بخشد .

عامل های آموزشی - به یاد بیاورید که ما قبلا در مورد عامل های آموزشی صحبت کردیم :



یک عامل آموزشی دارای یک *عنصر کارایی* [1] و یک *عنصر آموزشی* [2] می باشد . عنصر کارایی ، چیزی است که یک عامل برای تصمیم گیری در مورد این که چه کاری انجام دهد از آن استفاده می کند ، این چیزی است که ما تا کنون مطالعه کرده ایم . عنصر آموزشی ، چیزی است که به عامل برای بازنگری عنصر کارایی اجازه می دهد ، این ممکن است به معنی اضافه نمودن یا تغییر قانون ها یا واقعیات ، بازنگری یک مکاشفه ، تغییر یک تابع جانشین باشد . برای بازنگری کردن رفتارش ، یک عامل به اطلاعاتی که به عامل بگوید چگونه به خوبی کارش را انجام می دهد نیازمند می باشد ، این اطلاعات ، بازخورد [3] نام دارد .

انواع بازخورد - در اصل سه نوع عملکرد آموزشی وجود دارد که هر کدام بازخورد متفاوتی را دارد : آموزش نظارت شده ، در این مورد ، یک منبع خارجی (که اغلب آموزش دهنده نام دارد) عامل را با *نمونه های برچسب زده شده* [4] ارایه می نماید ، عامل ، موارد / عملکردهای معینی را

---

[1] performance element
[2] learning element
[3] feedback
[4] labled examples

در طول طبقه بندی اشان می بیند . آموزش بدون نظارت [1] ، در این مورد ، آموزش دهنده ای برای ارائه ی نمونه ها وجود ندارد . عامل ، معمولا برای پیداکردن الگوهایی در داده ها تلاش می نماید . آموزش تقویتی [2] ، این ، یک نوع مخصوص از آموزش است که در آن عامل فقط یک پاداش را برای انجام یک عمل دریافت می نماید . ممکن است نداند یک پاداش بهینه چگونه می باشد . بهترین عملکرد را برای انجام نمی داند .

آموزش نظارت شده - آموزش نظارت شده ، یکی از عمومی ترین شکل های آموزش می باشد . عامل با مجموعه ای از داده های برچسب زده شده ارایه می شود که باید از این داده ها برای تشخیص قانون های کلی تر استفاده نماید . نمونه ها : لیست بیماران و ویژگی ها : چه عامل هایی مرتبط با سرطان می باشند ؟ چه عواملی ، فردی را دارای خطر می داند ؟ بهترین سوالات برای طبقه بندی حیوانات چیست ؟ صورت چه کسی در این تصویر می باشد ؟ این پردازش آموزش قانون های کلی از واقعیات مشخص ، *استنتاج* [3] نام دارد .

تعریف یک مساله ی آموزشی - ما می توانیم مساله ی آموزشی را با تخمین ، به صورت یک تابع f که به ما می گوید چگونه یک مجموعه از ورودی ها را طبقه بندی نماییم تفسیر کنیم . یک مثال در این مورد یک مجموعه از ورودی های x و f(x) متناظر می باشد : <<Mammal,Eats-Meat,Black-Stripes,Tawny>,Tiger> . ما می توانیم یک عملکرد آموزشی را به این صورت ، تعریف نماییم : یک مجموعه ی ارایه شده از نمونه های f ، یک تابع H که f را برای مثال ما تخمین می زند پیدا نمایید ، H ، *فرض* [4] نام دارد .

استنتاج - ما ترجیح می دهیم H را تعمیم بدهیم ، این به این معنی است که H به درستی نمونه های دیده نشده را طبقه بندی می کند . در صورتی که فرض به درستی بتواند همه ی نمونه های آموزشی را طبقه بندی نماید ، ما آن را *فرض سازگار* [5] می نامیم . هدف : پیداکردن یک فرض سازگار که در نمونه های دیده نشده هم به خوبی کار کند . ما می توانیم آموزش را به صورت جستجو در میان یک فضا از فرض ها تصور نماییم .

بایاس استنتاجی - توجه نمایید که استنتاج ، بی عیب نمی باشد . در انتخاب یک فرض ، ما یک گمان آموزش داده شده را به وجود می آوریم . روشی که ما با استفاده از آن این گمان را به وجود می آوریم بایاس نام دارد . نمونه ها : اصل اکام [6] ، معین ترین فرض ، کلی ترین فرض ، تابع خطی

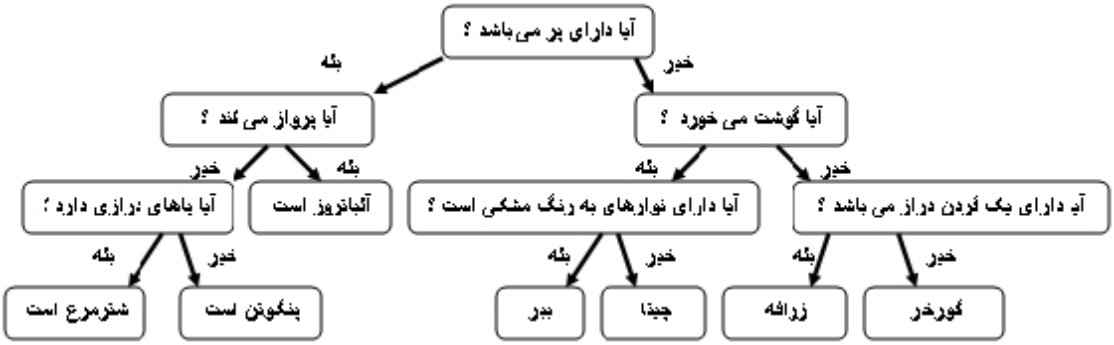
مشاهده ی داده ها - عامل ها ، ممکن است دارای ابزار مختلف مشاهده کننده ی نمونه های یک فرض باشند . یک الگوریتم آموزشی *دسته ای* [7] ، یک مجموعه ی بزرگ داده ها که همه با هم ارایه می شوند و یک فرضیه ی منفرد را انتخاب می کنند ، می باشد . یک الگوریتم آموزشی *افزایشی* ، نمونه های همه در یک زمان را دریافت می کند و دایما فرض آن را بازنگری می نماید ؛ دسته معمولا با دقت تر می باشد ، اما افزایشی ممکن است با محیط عامل ، مناسب تر باشد . یک عامل آموزشی فعال می تواند نمونه ها را انتخاب نماید . یک عامل آموزشی *غیرفعال* [8] ، دارای

---

[1] unsupervised learning
[2] reinforcement learning
[3] induction
[4] hypothesis
[5] consistent hypothesis
[6] Occam's razor ، برای دو توصیف ارایه شده ، آسان ترین آن ها را انتخاب نمایید – توضیحی که به مفروضات کم تری نیاز دارد (لغت نامه ی وب برنامه ی babylon)
[7] batch
[8] passive

نمونه های ارایه شده با آن به وسیله ی یک منبع خارجی هستند ؛ آموزش فعال توانمندتر می باشد ، اما ممکن است با محدودیت های دامنه ، مناسب نباشد .

درخت های تصمیم گیری آموزشی - درخت های تصمیم گیری ، ساختمان داده هایی هستند که یک عامل را با ابزارهای طبقه بندی کننده ی نمونه ها ارایه می نمایند . در هر گره در درخت ، یک ویژگی ، تست می شود .



تولید یک درخت تصمیم گیری - ما می توانیم یک درخت تصمیم گیری را به صورت بازگشتی تولید نماییم :

۱. اگر همه ی نمونه ها مثبت یا منفی هستند ، کار تمام است .
۲. اگر هیچ نمونه ای باقی نمانده است ، ما نمونه ای از این طبقه بندی را مشاهده نکرده ایم ، بنابراین ما از طبقه بندی اکثریت والد استفاده می کنیم .
۳. در صورتی که ویژگی ای برای تست باقی نمانده باشد ، در این صورت ما نمونه هایی با توصیف یکسان ، اما طبقه بندی های مختلف را داریم .
۴. در غیر این صورت ، بهترین ویژگی را برای دو نیم کردن و تولید به صورت بازگشتی زیر درخت ها انتخاب نمایید .

انتخاب یک ویژگی - پایه ی تولید یک درخت تصمیم گیری فشرده و مناسب ، انتخاب ویژگی های موثر برای تست می باشد . بینش : ما می خواهیم تست هایی که مجموعه ی داده های ما را به نمونه های مثبت و منفی تقسیم نمایند ، انتخاب نماییم . ما می خواهیم اندازه ی اطلاعات ارایه شده توسط تست را اندازه گیری نماییم . این روشی ریاضی است که تعداد بیت های لازم برای پاسخ به یک سوال یا ارایه ی یک واقعیت را مشخص می کند . مثال : پیش بینی شیر یا خط بودن یک سکه را در یک پرتاب در نظر بگیریم ؛ قبل از پرتاب ، یک بیت (شیرها / خط ها) برای پاسخ به سوال کافی می باشد و بعد از دیدن سکه ، بیت های اضافی دارای هیچ اطلاعاتی نمی باشند ، چون که آن ها یک دانش جدید را ارایه نمی نمایند .

تئوری اطلاعات - به صورت رسمی تر ، بیایید بگوییم که n پاسخ ممکن $v_1, v_2, ..., v_n$ برای یک سوال وجود دارد و هر جواب دارای احتمال رویداد $P(v_n)$ باشد . *محتوای اطلاعات*[1] جواب برای سوال به این صورت می باشد : $I = \sum_{i=1}^{n} - P(v_i) \log_2 P(v_i)$ ، برای یک سکه ، این برابر است با : $-\frac{1}{2}\log_2 \frac{1}{2} + -\frac{1}{2}\log_2 \frac{1}{2} = 1$. سوالاتی با یک احتمال زیاد جواب دارای محتوای اطلاعاتی کمی خواهند بود . (در صورتی که سکه در یک بار ۹۹/۱۰۰ شیر بیاید ، I=0.08) ، محتوای اطلاعاتی

[1] information content

، برخی اوقات آنتروپی [1] نامیده می شود؛ اغلب در فشرده سازی و الگوریتم های انتقال داده ها استفاده می شود .

استفاده از تئوری اطلاعات - برای درخت های تصمیم گیری ، ما می خواهیم بدانیم ارزشمندی هر تست ممکن چگونه می باشد ، یا چه میزان از اطلاعات را نتیجه می دهد . ما می خواهیم احتمال جواب های ممکن از یک مجموعه ی آموزشی را تخمین بزنیم . معمولا ، یک تست منفرد برای به طور کامل مجزا نمودن نمونه های مثبت و منفی کافی نمی باشد . در عوض ، ما نیاز داریم در مورد این که چه مقدار بعد از پرسیدن یک سوال ، مفید خواهد بود فکر کنیم . این ، *سود اطلاعات* [2] نام دارد .

سود اطلاعات - در صورتی که یک مجموعه ی آموزشی دارای p نمونه ی مثبت و n نمونه ی منفی باشد ، اطلاعات در یک جواب صحیح به صورت زیر خواهد بود :

$$I(\frac{p}{p+n},\frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

- ما می خواهیم خصوصیتی که برای مجزا کردن نمونه های مثبت ومنفی نزدیک ترین می باشد را پیدا نماییم . ما با محاسبه ی باقی مانده شروع می کنیم – این اطلاعاتی است که هنوز در داده ها بعد از این که ما ویژگی A را تست نمودیم وجود دارد . می گوییم ویژگی A می تواند مقادیر ممکن v را به کار بگیرد . این تست ، v زیر مجموعه ی جدید از داده ها را به نام $E_1, E_2, ..., E_v$ خواهد ساخت . باقی مانده ، مجموع اطلاعات موجود در هر کدام از این زیرمجموعه ها می باشد .
- $$\text{Re}\,mainder(A) = \sum_{i=1}^{v}\frac{p_i+n_i}{p+n} * I(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i})$$
- سود اطلاعات سپس می تواند به صورت تفاوت میان اطلاعات اصلی (قبل از تست) و اطلاعات جدید (بعد از تست) بیان شود .
- $$Gain(A) = I(\frac{p}{p+n},\frac{n}{p+n}) - \text{Re}\,mainder(A)$$
- مکاشفه : معمولا ویژگی با بیش ترین سود اطلاعات را انتخاب می نماید . سوال : این چه نوع از جستجو می باشد ؟

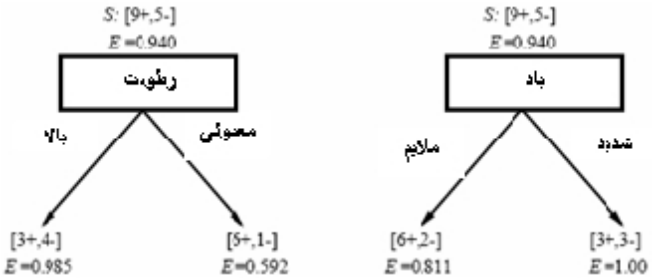
مثال

روز	پیش بینی	دما	رطوبت	باد	تنیس بازی
D1	آفتابی	گرم	بالا	ملایم	نه
D2	آفتابی	گرم	بالا	شدید	نه
D3	ابری	گرم	بالا	ملایم	بله
D4	بارانی	متوسط	بالا	ملایم	بله
D5	بارانی	خنک	معمولی	ملا	بله

[1] entropy
[2] information gain

				یم	
D6	بارانی	خنک	معمولی	شدید	نه
D7	ابری	خنک	معمولی	شدید	بله
D8	آفتابی	متوسط	بالا	ملا یم	نه
D9	آفتابی	خنک	معمولی	ملا یم	بله
D10	بارانی	متوسط	معمولی	ملا یم	بله
D11	آفتابی	متوسط	معمولی	شدید	بله
D12	ابری	متوسط	بالا	شدید	بله
D13	ابری	گرم	معمولی	ملا یم	بله
D14	بارانی	متوسط	بالا	شدید	نه

کدام ویژگی ، بهترین طبقه بندی می باشد ؟



Grain(S,رطوبت) = 0.940-(7/14).985-(7/14).592 = .151

Gain(S,باد) = .940-(8/14).811-(6/14)1.0=.048

اغتشاش - در صورتی که دو نمونه که دارای ویژگی های یکسانی باشند ولی مقدارهای مختلفی داشته باشند وجود داشته باشد ، یک درخت تصمیم گیری قادر به طبقه بندی آن ها به صورت مجزا نخواهد بود . ما می گوییم که این داده ها دارای اغتشاش می باشند . راه حل : از قانون اکثریت در گره ی والد استفاده نمایید .

خیلی مناسب کردن [1] - یک مساله ی عمومی ، رخدادن الگوریتم های آموزشی در زمانی که الگوهای تصادفی یا غیرعادی در داده ها وجود دارد می باشد . رویداد آموزشی ناگهانی در داده ها ، خیلی مناسب کردن نام دارد . در درخت های تصمیم گیری ، خیلی مناسب کردن به هرس کردن رسیدگی می کند . بار اول که درخت تولید می شود ، ما مفید بودن هر گره را ارزیابی می کنیم .

هرس کردن - تصور کنید که تست ، هیچ اطلاعاتی را ارایه نمی کند . (فرض ، تهی می باشد) . آیا داده های در فرزندان که به نظر مهم می رسند از این فرض مستثنی هستند ؟ از آزمون چی – دو [2] استفاده نمایید . وگرنه ، گره برداشته می شود و نمونه ها تاحد والد بالا می روند .

---

[1] overfitting

[2] chi-square test ، یک روش آماری ارزیابی کننده ی خوبی مناسب میان یک مجموعه از مقادیر مشاهده شده و به صورت نظری مورد انتظار

ورودی های پیوسته ی مقداردهی شده - درخت های تصمیم گیری همچنین می توانند برای کار با ورودی های صحیح و مقداردهی شده به صورت پیوسته توسعه داده شوند . گسسته کردن دامنه ، برای مثال <۷۰و> ۷۰ . چالش : چه مقداری بالاترین سود اطلاعات را نتیجه می دهد ؟ ، برای پیداکردن این مقدار از یک جستجوی تپه نوردی استفاده نمایید .

منابع :

http://www.cs.usfca.edu

فصل بیست و پنجم

آموزش Q ای

آموزش Q ای
فصل بیست و پنجم

ما به محیط های تصادفی جذب شده ایم . ما می توانیم این مساله را به صورت یک پروسه ی تصمیم گیری مارکوفی فرمول بندی نماییم . مساله دارای : یک حالت اولیه s_0 ، یک مجموعه ی گسسته از وضعیت ها و عملکردها ، یک مدل انتقالی : T(s,a,s') که احتمال رسیدن وضعیت s' از s در موقع انجام عمل a می باشد و یک تابع پاداش : R(s) می باشد .

روش ها - یک روش ، راه حلی برای یک MDP می باشد . عملکرد بهینه را برای انجام در هر وضعیت ، مشخص می نماید . عملکرد بهینه ، عملکردی است که سودمندی مورد انتظار کاهش داده شده از این وضعیت را بیشینه می نماید . بنابراین ، سودمندی یک وضعیت ، پاداشی برای آن وضعیت ، به اضافه ی پاداش کم شده ی مورد انتظار برای دنبال کردن روش بهینه از این وضعیت می باشد . $U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$ ، عبارت فوق معادله ی بلمن نام دارد .

تکرار مقدار - به صورت مستقیم نمی توان معادله ی بلمن را حل نمود . ما می توانیم از تکرار مقدار برای حل یک سیستم معادلات بلمن استفاده نماییم : سودمندی های غیر پایانه ای را برای مقدارهای تصادفی قرار دهید . برای هر وضعیت ، عملکرد بهینه ارایه کننده ی این مقادیر را محاسبه نمایید . از این برای به روز رسانی سودمندی تخمین زده شده ی وضعیت استفاده نمایید . تا زمان به هم نزدیک شدن ادامه دهید .

تکرار روش -

- مسایل با تکرار مقدار ؛ همگرایی ، کند می باشد ، حتی در زمانی که روش تغییر نمی کند و ما واقعا نمی توانیم نسبت به سودمندی یک وضعیت ، بی تفاوت باشیم .
- تکرار روش ، روش ها را به صورت مستقیم به روز رسانی می نماید .

الگوریتم تکرار روش

وضعیت های اولیه را با سودمندی های تصادفی مقداردهی اولیه نمایید
بردار روش تصادفی شاخص گذاری شده با وضعیت = Pi
انجام بده
ارزیابی سودمندی هر وضعیت برای U = Pi
برای s در وضعیت ها
عملکردی که سودمندی مورد انتظار برای آن وضعیت را بیشینه می کند را پیدا نمایید = a

Pi(s)=a

مادامی که برخی از عملکردها تغییر داده شوند

آموزش یک روش - تا حالا ، ما وجود مقدار زیادی از دانش را فرض نموده ایم . در حالت به خصوص ، ما فرض کرده ایم که مدلی از جهان شناخته شده است ؛ این ، مدل انتقال وضعیت می باشد . اگر ما یک مدل نداشته باشیم چطور ؟ همه ی چیزی که ما می دانیم این است که مجموعه ای از وضعیت ها و یک مجموعه از عملکردها وجود دارند . ما هنوز می خواهیم یک روش بهینه را یاد بگیریم

آموزش Q ای - آموزش یک روش به صورت مستقیم ، مشکل می باشد . مساله : داده های ما به یک شکل نمی باشند : < وضعیت ، عملکرد> . در عوض ، به شکل $s_1, s_2, s_3, ..., R$ می باشد . از آنجایی که ما تابع وضعیت را نمی دانیم ، این هم مشکل است که سودمندی یک وضعیت را یاد بگیریم . در عوض ، ما یک تابع Q(s,a) را یاد خواهیم گرفت . این " سودمندی " ارایه ی عملکرد a در وضعیت s را تخمین می زند . به طور صریح تر ، Q(s,a) مقدار a در وضعیت s را ارایه می کند و بعد از این به صورت بهینه عمل می کند . $Q(s,a) = R(s,a) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$. برای ارایه ی عملکرد با بالاترین مقدار Q در هر وضعیت ، روش بهینه می باشد . اگر عامل بتواند Q(s,a) را یاد بگیرد ، می تواند عملکردهای بهینه را حتی بدون دانستن تابع پاداش یا تابع انتقال داشته باشد

آموزش تابع Q - برای آموزش Q ما باید مقدار یک عملکرد را در یک وضعیت و حتی آن هایی که در میان پاداش هایمان در طول زمان پخش می شوند را تخمین بزنیم . ما می توانیم این کار را به صورت تکراری انجام دهیم . توجه کنید که $U(s) = \max_a Q(s,a)$. سپس ما می توانیم معادله را برای Q به صورت زیر بازنویسی نماییم : $Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$. بیایید تخمین Q(s,a) را به صورت $\hat{Q}(s,a)$ مشخص نماییم . ما یک جدول لیست کننده ی هر جفت وضعیت – عملکرد و مقدار Q را نگهداری می نماییم . مراقب های عامل با وضعیت s ، یک عملکرد a را انتخاب می کنند و سپس r=R(s,a) می باشد که آن را دریافت می کند و وضعیت جدید s' می باشد . سپس جدول Q را با توجه به فرمول زیر به روز می کند : $\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$ ، عامل ، تخمینی از $\hat{Q}$ را برای s' برای تخمین $\hat{Q}$ برای s ، استفاده می نماید . توجه کنید که عامل نیازی به هر دانش از R یا تابع انتقال برای اجرای این ندارد . آموزش Q ای برای نزدیک شدن به درازا دارای این موارد می باشد : پاداش ها محدود شده است ، عامل جفت وضعیت – عملکرد های این شکلی به روشی که معمولا نامحدود می باشد انتخا ب می نماید ، این به این معنی است که یک عامل باید دارای احتمالی غیر صفر از انتخاب هر a در هر s به صورت رشته ای از جفت های وضعیت – عملکرد با نگرش نامحدود باشد .

اکتشاف - بنابراین چگونه آن را به وجود بیاوریم ؟ آموزش Q ای دارای یک تفاوت مشخص از دیگر الگوریتم هایی که تا کنون دیده ایم می باشد : عامل می تواند عملکردها و دیدی که آن ها به دست می آورند را انتخاب نماید . این *آموزش پویا* [1] می باشد . مساله : عامل ، مایل به بیشینه کردن عملکرد می باشد ؛ این به این معنی است که در حال حاضر به نظر می رسد بهترین باشد انتخاب می شود ، اما اگر عامل هیچ گاه عملکردهای با " دید بد " نداشته باشد ، نمی تواند از خطاها بیرون بیاید . $\hat{Q}$ خیلی بادقت نمی باشد ، بنابراین با عملکردهای غیر بهینه را امتحان خواهیم کرد . از این پس ، در صورتی که $\hat{Q}$ بهتر شود ، ما عملکردهای بهینه را انتخاب خواهیم کرد .

[1] active learning

اکتشاف بولتزمان - یک راه برای انجام این کار استفاده از اکتشاف بولتزمان است . ما یک عمل با احتمال زیر را انجام می دهیم :

- $$P(a \mid s) = \frac{k^{\hat{Q}(s,a)}}{\sum_j k^{\hat{Q}(s,a_j)}}$$
- k پارامتر دما می باشد . این فرمولی شبیه به آن چیزی است که ما در گرم و سرد کردن شبیه سازی شده استفاده می نماییم .

آموزش مستحکم [1]

- آموزش Q ای نمونه ای از آن چه که ما آموزش مستحکم می نامیم می باشد .
- عامل ها نمونه هایی از این که چگونه عمل نمایند را نمی بینند .
- در عوض ، آن ها عملکردها را انتخاب می نمایند و پاداش ها یا مجازات ها را دریافت می نمایند .
- یک عامل می تواند یاد بگیرد ، حتی در زمانی که عمل غیر بهینه ای را انجام می دهد .
- تعمیم ها به پاداش تاخیر داده شده و MDP های غیرقطعی رسیدگی می کنند .

خلاصه

- آموزش Q ای برخی اوقات به صورت آموزش مستقل از مدل [2] شناخته می شود .
- عامل فقط نیاز به انتخاب عملکردها و دریافت پاداش ها دارد .
- مسایل :
  - چگونه تعمیم بدهد ؟
  - مقیاس پذیری [3]
  - سرعت نزدیک شدن
- آموزش Q ای برای یک الگوریتم مفید تجربی به سطح مطلوبی رسیده است .

منابع :

http://www.cs.usfca.edu

[1] reinforcement learning
[2] model-free learning
[3] scalability

فصل بیست و ششم

برنامه ریزی

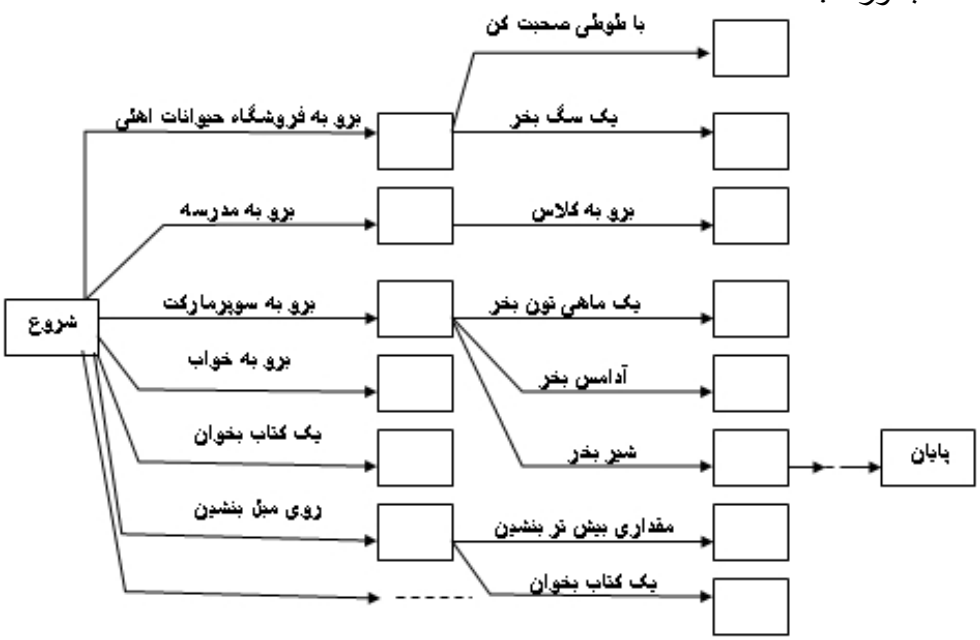
برنامه ریزی
فصل بیست و ششم

رئوس مطالب

- جستجو و برنامه ریزی
- عملگرهای STRIPS
- برنامه ریزی با مرتبه ی جزیی [1]
- برنامه ریزی با ارضا

برنامه ریزی - برنامه ریزی ، تعبیه کردن یک رشته از عملکردها برای دسترسی به یک هدف می باشد . برنامه ریزی کلاسیک : کاملا قابل مشاهده می باشد ، قطعی است ، محدود می باشد ، ثابت [2] است و گسسته است . زبان مشخصی برای ارایه ی مسایل برنامه ریزی وجود دارد

جستجو و برنامه ریزی - الگوریتم های جستجوی استاندارد به نظر می رسد که به سختی شکست بخورند :



[1] partial-order planning
[2] static

- مساله ای که با روش جستجو وجود دارد : تعداد زیادی عملکرد نامربوط وجود دارد ، پیدا کردن مکاشفه های خوب مشکل می باشد و از تجزیه ی مساله نمی توان سودی برد . به صورت خودکار مکاشفه ها از ارایه مشتق می شوند ؛ مانند مساله ی ارضای محدودیت

طراحی با منطق - سیستم های برنامه ریزی کارهای زیر را انجام می دهند :

۱. عملکرد و ارایه ی هدف را برای مجاز کردن انتخاب ، باز می کنند
۲. توسط زیر هدف سازی مشتق می کنند و غلبه می نمایند

	جستجو	**برنامه ریزی**
وضعیت ها	ساختمان داده های جاوا	عبارت های منطقی
عملکردها	کد جاوا	پیش شرط ها [1] / نتیجه ها
هدف	کد جاوا	عبارت منطقی (اجتماع)
برنامه	رشته ای از S_0	محدودیت های روی عملکردها

زبان مسایل برنامه ریزی - حل کننده ی مساله ی موسسه ی تحقیق استنفورد [2] : جهان توسط شرط های منطقی توصیف می شود ، **وضعیت به صورت اجتماع لفظ های مثبت می باشد**

- گزاره ای ؛ به عنوان مثال ، $Happy \wedge Hungry$ برای نمایش وضعیت عامل
- **زمینه ی** مرتبه ی اول و عبارت های **مستقل از تابع** ؛ به عنوان مثال ،

$$At(Plane_1, Verona) \wedge At(Plane_2, Malpensa)$$

- **فرض جهان بسته** ؛ هر شرطی که نام برده نشده غلط می باشد
- **هدف** ، وضعیتی است که به صورت جزیی مشخص شده است
  - اگر محدودیت ها همه لفظ هایی از هدف باشند یک وضعیت یک هدف را ارضا می نماید
  - به عنوان مثال ، $At(Plane_1, Verona) \wedge At(Plane_2, Malpensa)$ هدف $At(Plane_2, Malpensa)$ را ارضا می نماید

عملکردهای استریپس - عملکردها توسط موارد زیر مشخص می شوند :

- **پیش شرط ها :** زمانی که عملکرد می تواند به کار گرفته شود . **اثرات :** وضعیت توسط عملکردها تغییر می یابد
  - لیست اضافه : گزاره هایی که صحیح از آب در می آیند . لیست حذف : گزاره هایی که غلط از آب در می آیند
- عملکردها شامل متغیرها می باشند ، یک طرح عملکرد منفرد عملکردهای مختلف را نمایش می دهد (نمونه ای از متغیرها)

توصیف عملکردها به صورت منظم می باشد و زبان محدود شده است

طرح عملکرد :

---

[1] preconditions
[2] Stanford Research Institute Problem Solver (STRIPS)

Buy(x)

پیش شرط : $At(p) \wedge Sells(p,x)$. اثر : Have(x)

(نکته : هیچ اطلاعاتی در مورد چگونگی اجرای عملکرد وجود ندارد !)

- چون که زبان محدود شده است پس الگوریتم کارا می باشد
  - عملکرد : نام و لیست پارامتر را مشخص می نماید . پیش شرط : اجتماع لفظ های مثبت . اثر : اجتماع لفظ ها (مثبت یا منفی)
- یک مجموعه ی کامل از عملگرهای استریپس می تواند به یک مجموعه از اصول جانشین وضعیت ترجمه شود
  - **معانی** - با یک وضعیت ارایه شده (اجتماع لفظ ها) : اگر انتساب یک متغیر ، در ارتباط با لفظ های شامل شده ی وضعیت باشد ، پیش شرط ارضا می شود ؛ به عنوان مثال ، وضعیت $At(HW) \wedge Sell(HW, Drill)$ ، پیش شرط $At(p) \wedge Sell(p,x)$ را با انتساب p/HW و x/Drill راضی می کند .
- عملکردهای با پیش شرط های ارضا شده می توانند در این موارد به کار گرفته شوند : حذف عناصر ار لیست حذف ، اضافه نمودن عناصر به لیست اضافه ، وضعیت جدید : $At(HW) \wedge Sell(HW, Drill) \wedge Have(Drill)$

مثال : خرید

- عملکردها
  - Buy(x)

    پیش شرط : At(store) و Sells(store,x) . اثر : Have(x)
  - Go(x,y)
  - پیش شرط : At(x) . اثر : At(y) و $\neg At(x)$
- شروع
  - $At(Home) \wedge Sells(SM, Milk) \wedge Sells(SM, Banana) \wedge Sells(HWS, Drill)$
- هدف
  - $Have(Milk) \wedge Have(Banana) \wedge Have(Drill)$

مثال : انتقال محموله ی هوایی

- عملکردها
  - Load(c,p,a)

    پیش شرط : $At(c,a) \wedge At(p,a) \wedge Carg o(c) \wedge Plane(p) \wedge Airport(a)$. اثر : $\neg At(c,a) \wedge In(c,p)$
  - Unload(c,p,a)

    پیش شرط : $In(c,p) \wedge At(p,a) \wedge Carg o(c) \wedge Plane(p) \wedge Airport(a)$. اثر : $At(c,a) \wedge \neg In(c,p)$
  - Fly(p,from,to)

    پیش شرط : $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

    اثر : $\neg At(p, from) \wedge At(p, to)$

- عملکردها : Load ، Unload و نتیجه ی Fly . گزاره ها : In(0,0) و At(0,0) . انواع گزاره ها : Cargo(0) ، Plane(0) و Airport(0) . شروع :

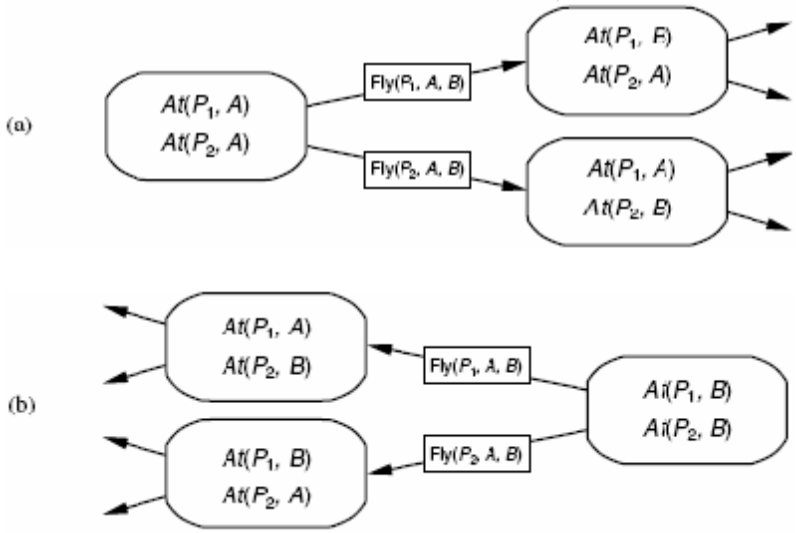
$$At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$$
$$\wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) Plane(P_2)$$
$$\wedge Airport(JFK) \wedge Airport(SFO)$$

- هدف : $At(C_1, JFK) \wedge At(C_2, SFO)$. یک راه حل به صورت زیر می باشد

$$Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$$
$$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)$$

برنامه ریزی استریپس - مساله ی برنامه ریزی استریپس : یک رشته از عملکردهایی که به هدف منجر می شوند را پیدا نمایید . وضعیت ها و هدف ها توسط یک اجتماع از لفظ ها تعریف می شوند . فضای حالت جستجو : جستجوی مستقیم از وضعیت اولیه تلاش می کند که به هدف برسد . جستجوی معکوس از هدف تلاش و طرح ریزی می کند که به وضعیت ابتدایی برسد . فضای برنامه ی جستجو : برنامه ریزی با مرتبه ی جزیی [1] ، فضای جزیی ساخته شده توسط طرح ها را جستجو می کند

فضای حالت جستجو - وضعیت اولیه ، وضعیت شروع می باشد ، آزمون هدف بررسی می کند که آیا هدف ارضا می شود یا نه ، عملکردها ، عملگرها را تعریف می کنند ، هزینه مرحله ، معمولا ۱ می باشد . در زیر ، شکل a ، مستقیم و شکل b ، معکوس می باشد :



جستجوی مستقیم - حلقه ی جستجوی اصلی : یک عملکرد را انتخاب نمایید و پیش شرط را با وضعیت یکی نمایید . در صورتی که پیش شرط ارضا می شود ، عملکرد تولید کننده ی یک وضعیت جدید را به کار ببرید . بررسی کنید که آیا وضعیت جدید ، هدف را راضی می نماید یا نه . در صورتی که فضای حالت ، محدود باشد ، برای الگوریتم های جستجوی کامل ، کامل می باشد . به دلیل عملکردهای نامربوط ناکارآمد می باشد ، به مکاشفه های خوب ، نیازمند می باشد .

جستجوی معکوس - ایده : عملکردهای مربوط را فقط با شروع از هدف انتخاب نمایید . عملکرد : در زمانی که به یکی از متحدانش دست می یابد (لیست اضافه) مربوط می باشد . در زمانی که هر لفظ مربوط را بی اثر [2] نمی کند (لیست حذف) ، سازگار می باشد . یک عملکرد مربوط و سازگار

[1] Partial-Order Planning(POP)
[2] undo

را انتخاب نمایید و وضعیت جدید را تولید نمایید : لیست اضافه را پاک نمایید . پیش شرط ها را اضافه نمایید . با یک وضعیت ارضا شده توسط وضعیت ابتدایی خاتمه می یابد . روی وضعیت های مربوط و سازگار منشعب می شود .

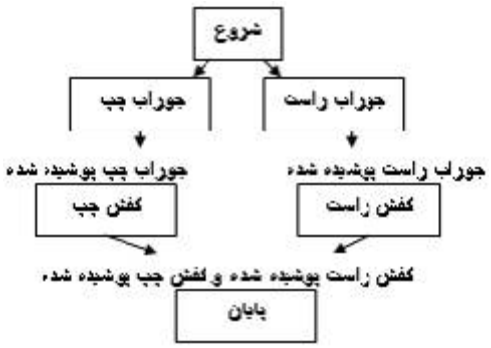
فضای حالت و فضای برنامه

- جستجوی مستقیم و معکوس رشته های خطی عملکردها را اکتشاف می نماید
- این لازم نمی باشد
  - به راه حل انتقال هوایی محموله توجه نمایید

$$Load(C_1,P_1,SFO), Fly(P_1,SFO,JFK), Unload(C_1,P_1,JFK),$$
$$Load(C_2,P_2,JFK), Fly(P_2,JFK,SFO), Unload(C_2,P_2,SFO)$$

  - تنها ترتیب لازم میان Load ، Fly و Unload می باشد
  - توسط اثرات **سببی** به هم متصل شده اند ؛ به عنوان مثال ، $Load(C_1,P_1,SFO)$ به $In(C_1,P_1)$ استفاده شده توسط پیش شرط Unload دست می یابد
  - نیازی نیست که $Load(C_2,P_2,JFK)$ را بعد از $Unload(C_1,P_1,JFK)$ قرار دهیم
- **طرح با مرتبه ی جزیی** [1] : گراف عملکردها شامل شروع و پایان می باشد (گراف ≡ مرتبه ی جزیی ϕ)

مثال : طرح با مرتبه ی جزیی ،



طرح های با مرتبه ی کلی :

طرح های منظم شده به صورت جزیی - مجموعه ای از مراحل با **مرحله ی شروع** [1] دارای توصیف وضعیت اولیه به صورت اثرش می باشد ، **مرحله ی پایانی** [2] دارای توصیف هدف به صورت پیش شرطش می باشد ، **اتصالات سببی** [3] از نتیجه های یک مرحله به پیش شرط دیگری ،

[1] partial-order plan

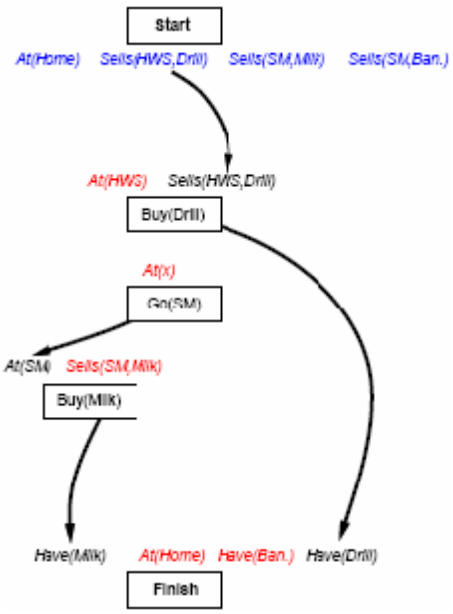
- **مرتب سازی زمانی** [4] میان جفت های مراحل

- **شرط باز** [5] = پیش شرط یک گام که هنوز به صورت سببی متصل نشده است ، **طرح** در صورتی کامل است که هر پیش شرط در دسترس باشد ، **پیش شرط** در صورتی قابل دسترس می باشد که اثر یک مرحله قبل تر باشد و هیچ مرحله ی میانی ، آن را بی اثر ننماید (تناقض) ، طرح ها نمی توانند دارای حلقه باشند

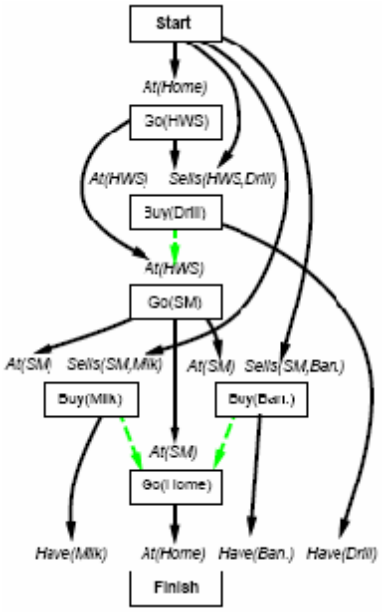
مثال

به تعریف برو

شکل ها را از چپ به راست دنبال نمایید :

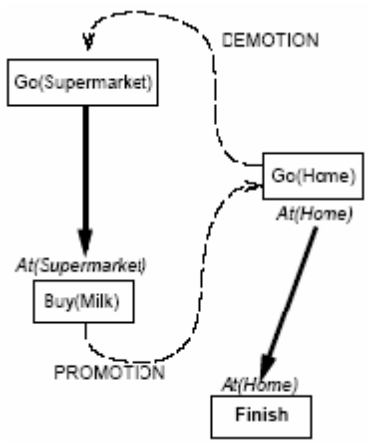


برو به تعریف



برو به تعریف

ناسازگاری ها و پیشرفت [6] **/ تنزل** [7] - یک عملکرد ناسازگار یک مرحله ی میانی بالقوه است که شرط قابل دسترس توسط اتصال سببی را خراب می نماید . به عنوان مثال ، Go(Home) ، At(Supermarket) را حذف می نماید :



تنزل : قبل از Go(Supermarket) قرار دهید
پیشرفت : بعد از Buy(Milk) قرار دهید

start step [1]
finish step [2]
causal links [3]
temporal ordering [4]
open condition [5]
promotion [6]
demotion [7]

پردازه ی برنامه ریزی - عملگرهای طرح های جزیی : یک اتصال را از یک عملکرد موجود به یک شرط باز اضافه نمایید . یک مرحله را برای انجام یک شرط باز اضافه نمایید . یک مرحله را با توجه به دیگری برای حذف تناقض های ممکن ، منظم نمایید . به تدریج از برنامه های ناقص / مبهم [1] به برنامه های کامل و صحیح حرکت نمایید . در صورتی که یک شرط باز قابل دسترس نمی باشد یا اگر یک تناقض قابل رزرو کردن نباشد به عقب برگشت نمایید [2] . برنامه های سازگار دارای هیچ حلقه و هیچ تناقضی نمی باشند . هر طرح یا برنامه ی سازگار با هیچ شرط باز ، یک راه حل می باشد

طرح تقریبی الگوریتم POP

- طرح اولیه شامل شروع و پایان می باشد
  - ترتیب به صورت $Start \pi Finish$ می باشد
  - اتصال های سببی وجود ندارند
  - همه ی پیش شرط های درون پایان ، باز می باشند
- پیش شرط باز p در عملکرد B را انتخاب نمایید و برنامه ی جانشین را برای هر عملکرد A که دسترسی کننده به p می باشد ، انتخاب نمایید

۱. اتصال سببی $A \rightarrow^{p} B$ و منظم کننده های $A \pi B$ ، $Start \pi A$ و $A \pi Finish$ را اضافه نمایید

۲. تناقض های درون اتصال سببی جدید را رفع نمایید ، این کار می تواند منظم کننده های جدید را اضافه نماید

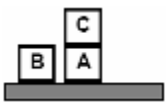
نکته : نقطه ی برگشت به عقب در عملکرد A و تحلیل ناسازگاری در مورد عدم موفقیت ها می باشد

- بررسی برای راه حل : هیچ پیش شرط بازی وجود نداشته باشد (فقط طرح های سازگار را تولید نمایید)

خصوصیات POP - POP ، صحیح ، کامل و سیستماتیک یا منظم (بدون تکرار) می باشد . هزینه ی مسیر : مراحل با پالایش های طرح ، ارتباط دارند ، اما هزینه ی اجرا ، هزینه ی واقعی می باشد ، برای مراحل پالایش ، هزینه ی صفر را انتساب دهید !

- آیا می تواند با مکاشفه های خوب مشتق شده از توضیح مساله ، موثر باشد . مخصوصا برای مسایل با تعدادی زیر هدف وابسته ی به صورت ضعیف ، خوب می باشد . برای اشتراک ، عمومی ها ، نقیض و شرطی ها توسعه می یابد .

مثال : دنیای بلوک ها - مساله ی " ناهنجاری سوسمن [3] "



وضعیت شروع
Clear(x) On(x,z)
Clear(y)

PutOn(x,y)

وضعیت پایان
Clear(x) On(x,z)

PutOnTable(x,y)

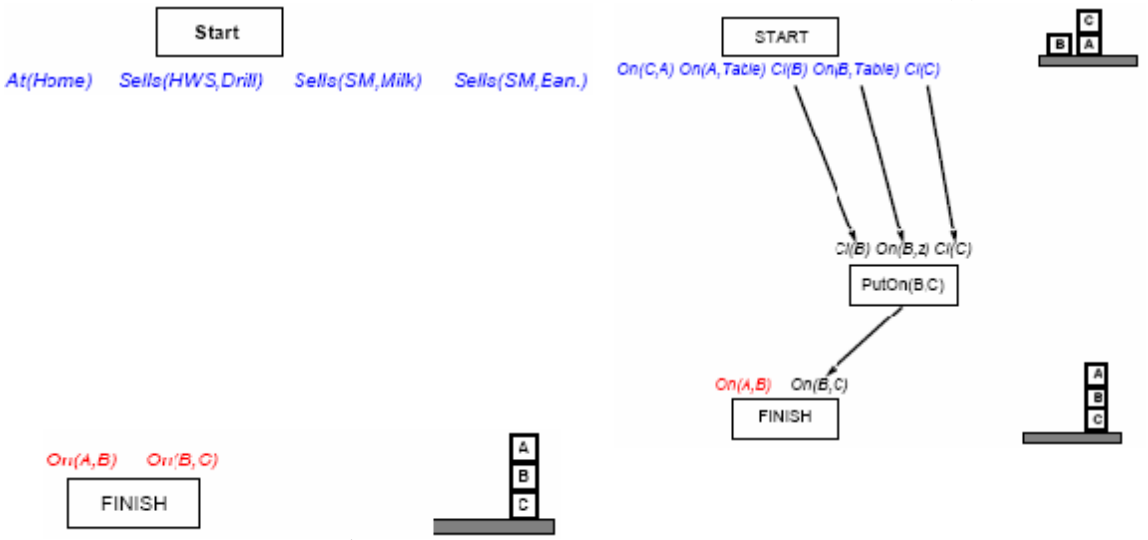
~On(x,z) Clear(z)

---

[1] vague
[2] backtrack
[3] Sussman anomaly

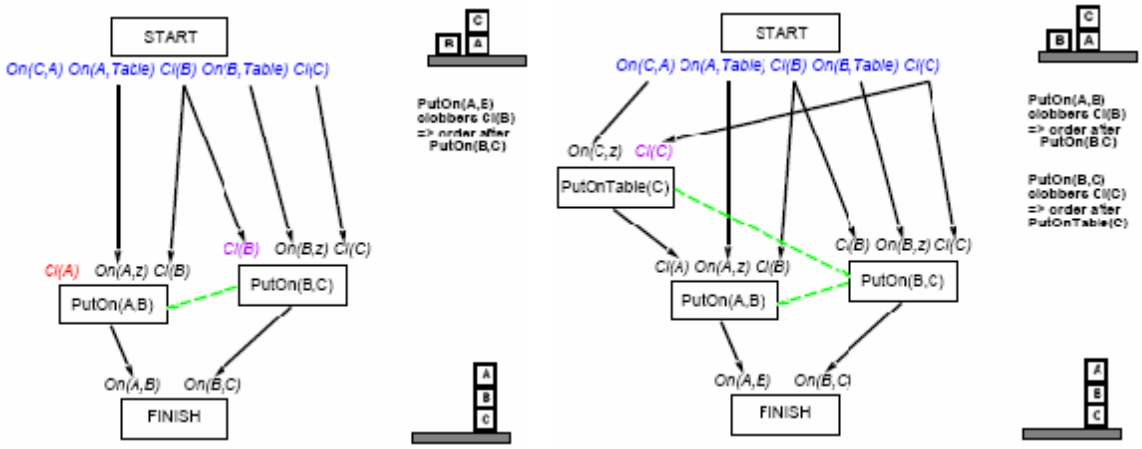
~On(z) ~Clear(y) On(x,Table)
Clear(z) On(x,y)

+ چند محدودیت عدم تساوی
شکل ها را از چپ به راست دنبال نمایید :



گام بعدی

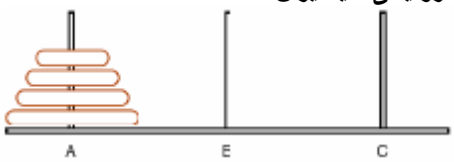
گام بعدی (شکل ها را از چپ به راست دنبال نمایید)



گام بعدی

مثال برنامه ریزی - برج هانوی [1] **:** سه میله [2] A,B,C و یک برج از دیسک ها که در اولی (A) دیسک کوچک تر بالا و کوچک تر در زیر قرار گرفته است وجود دارد . هدف این است که همه ی برج را از اولین میله (A) به سومین میله (C) منتقل نماییم ، با حرکت دادن فقط یک دیسک در هر مرتبه و در نظر گرفتن این که در هیچ مرحله ای نباید یک دیسک بزرگ تر روی یک دیسک کوچک تر قرار گیرد .

[1] the Tower of Hanoi
[2] peg



مشخص کردن دامنه ی PDDL

```
(define (domain hanoi)
        (:requirements :strips)
        (:predicates (clear ?x) (on ?x ?y) (smaller ?x ?y))
        (:action move
        :parameters (?disc ?from ?to)
        :precondition (and (smaller ?to ?disc)
                (on ?disc ?from)
                (clear ?disc) (clear ?to))
        :effect (and (clear ?from) (on ?disc ?to)
                (not (on ?disc ?from))
                (not (clear ?to))))
)
(define (problem hanoi4)
(:domain hanoi)
(:objects peg1 peg2 peg3 d1 d2 d3 d4)
(:init
        (smaller peg1 d1) (smaller peg1 d2) (smaller peg1 d3) (smaller peg1
d4)
        (smaller peg2 d1) (smaller peg2 d2) (smaller peg2 d3) (smaller peg2
d4)
        (smaller peg3 d1) (smaller peg3 d2) (smaller peg3 d3) (smaller peg3
d4)
        (smaller d2 d1) (smaller d3 d1) (smaller d3 d2) (smaller d4 d1)
        (smaller d4 d2) (smaller d4 d3)
        (clear peg2) (clear peg3) (clear d1)
        (on d4 peg1) (on d3 d4) (on d2 d3) (on d1 d2))
(:goal (and (on d4 peg3) (on d3 d4) (on d2 d3) (on d1 d2)))
)
```

برنامه ریزی با ارضا

برنامه ریزی از طریق ارضا - یک مساله ی برنامه ریزی را به صورت یک فرمول گزاره ای کد نمایید . روال تصمیم گیری ارضا را برای بررسی این که آیا فرمول قابل ارضا است یا نه استفاده نمایید . طرح از انتساب های تشخیص داده شده توسط روال تصمیم گیری ارضا توسعه داده می شود .

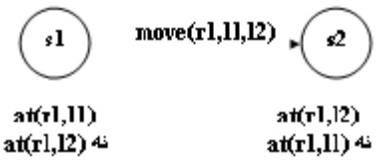
مساله ی ارضای گزاره ای

- مجموعه ای از متغیرهای گزاره ای
- رابط های منطقی : and , or , not
- با یک فرمول ارایه شده بررسی کنید که آیا یک انتساب متغیر که فرمول را صحیح نماید وجود دارد
  - *مدلی* را برای فرمول پیدا نمایید
- اتصال با مساله ی ارضای محدودیت را به یاد بیاورید

مساله ی قاب - هنگامی که کسی یک آجر [1] را بر می دارد چه اتفاقی می افتد ؟ هر بچه ای می داند که الان آجر در هوا قرار دارد ، و شیء کم تری روی زمین وجود دارد و **همه ی مطلب همین است** [2] .

وضعیت ها و فرمول گزاره ای

- مثال



- کد کننده ی بد (وضعیت گم کننده) : $(at(r1,l1) \wedge \neg at(r1,l2) \vee (\neg at(r1,l1) \wedge at(r1,l2)$
- وضعیت را به حساب ببرید : $at(r1,l1,s1) \wedge \neg at(r1,l2,s1) \wedge \neg at(r1,l1,s2) \wedge at(r1,l2,s2)$

برنامه ریزی با فرمول گزاره ای

- مساله ی برنامه ریزی را به مساله ی پیداکردن یک طرح با طول ثابت شناخته شده ، محدود نمایید
  - مساله ی برنامه ریزی محدود شده
- مساله ی برنامه ریزی محدود شده را به یک مساله ی ارضا تبدیل نمایید
  - از آنجایی که طول طرح ، ناشناخته می باشد ، شامل عملکردهای اضافی می باشد
- یک آرگومان را برای هر گزاره و سمبل عملکرد اضافه نمایید
  - آرگومان آخر مرحله ی طرح می باشد
- مثال
  - at(r1,l1) را به at(r1,l1,i) ترجمه می نماید ، اگر $0 \le i \le n$
  - Move(r1,l1,l2) را به move(r1,l1,l2,i) ترجمه می نماید $0 \le i \le n$
- مواد بیش تری مورد نیاز می باشد
  - وضعیت های آغاز و پایان
  - عملکردها : پیش شرط ها و اثرات میان مراحل

یادداشت

- روان : فرمول اتمیک ، توصیف کننده ی وضعیت ها در یک مرحله ی ارایه شده می باشد
  - به عنوان مثال ، at(r1,l1,5)
- یادداشت کوتاه :

---

[1] brick
[2] P.J. Hayes,K.M. Ford(1992)

- در صورتی که f ، at(r1,l1) باشد ، f_i در مرحله ی I ، روان می باشد : at(r1,l1,i)
- برای عملکردها هم همین طور است
  - a ، move(r1,l1,l2) است ، a_i ، move(r1,l1,l2,i) می باشد

برنامه ریزی با فرمول گزاره ای

۱. وضعیت اولیه : $\wedge_{f \in s_0} f_0 \wedge \wedge_{f \notin s_0} \neg f_0$
۲. وضعیت های هدف : $\wedge_{f \in s_n} f_n$
۳. عملکردها و اثرها : برای همه ی عملکردها و مراحل ۰ ... n-1 :

$$a_i \rightarrow \left(\wedge_{p \in precond(a)} p_i \wedge \wedge_{e \in effects(a)} e_{i+1}\right)$$

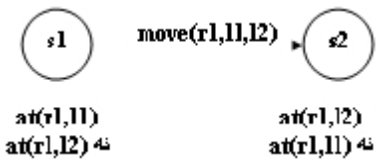
۴. در هر لحظه فقط یک عملکرد : برای هر عملکرد متمایز a,b و مراحل ۰ ... n-1

$$\neg a_i \vee \neg b_i$$

۱. اصول قاب
   a. برای هر روان (سلیس)
   b. و مرحله ی ۰ ... n

$$\neg f_i \wedge f_{i+1} \rightarrow \left(\vee_{\{a \in A \,.\, f \in effects^{+}(a)\}} a_i\right) \wedge$$
$$f_i \wedge \neg f_{i+1} \rightarrow \left(\vee_{\{a \in A \,.\, f \in effects^{-}(a)\}} a_i\right)$$

مثال



- عملگر منفرد
  : move(r,l,l')
  P:at(r,l)
  E:at(r,l'), $\neg$ at(r,l)
- طرح با طول ۱
- ضعیت اولیه : $at(r1,l1,0) \wedge \neg at(r1,l2,0)$
- هدف : $at(r1,l2,1) \wedge \neg at(r1,l1,1)$
- عملکردها :

$$move(r1,l1,l2,0) \rightarrow at(r1,l1,0) \wedge at(r1,l2,1) \wedge \neg at(r1,l1,1$$
$$move(r1,l2,0) \wedge at(r1,l1,1) \wedge \neg at(r1,l2,1)$$

- استثنا : $\neg move(r1,l1,l2,0) \vee \neg move(r1,l2,l1,0)$
- اصول قاب :

$$\neg at(r1,l1,0) \wedge at(r1,l1,1) \rightarrow move(r1,l2,l1,0)$$
$$\neg at(r1,l2,0) \wedge at(r1,l2,1) \rightarrow move(r1,l1,l2,0)$$

توضیحات

- کدکردن به فضای نمایی نیاز دارد . مساله ی SAT ، NP – کامل می باشد . کد کننده های متفاوتی وجود دارد ، کد کننده بر تعداد متغیرها و شرط ها تاثیر می گذارد

منابع :

www.inf.unibz.it

فصل بیست و هفتم

برنامه نویسی پرولوگ

فصل بیست و هفتم
برنامه نویسی پرولوگ [1]

پرولوگ = " برنامه نویسی در منطق " [2]
برتری های مهم - راحتی ارایه ی دانش ، پشتیبانی طبیعی غیر جبری [3] ، پشتیبانی طبیعی از منطبق کننده ی الگو [4] ، پشتیبانی طبیعی از فرا برنامه نویسی [5]
برتری های دیگر - هدف برنامه ها مستقل از چگونگی اجرا شدن آن هاست ، ارتباط ساده میان برنامه ها و جواب های محاسبه شده و خصوصیات و نیازی به تشخیص برنامه ها از پایگاه های داده ای نمی باشد .

موضوعات بحث شده

۱. مفاهیم اولیه
۲. واژگان تخصصی - ارزیابی های قطعی و ورودی – خروجی غیرجبری
۳. ارزیابی غیرقطعی - برتری کارایی ، یکسان سازی
۴. پردازش لیست - بررسی نوع ، مقایسه ی واژه ها و محاسبه
۵. جدایی – نقیض ، تولید و تست و تراکم
۶. کنترل جستجو
۷. فرابرنامه نویسی

مفهوم اول – تعریف روال ها : برنامه ها ، از تعریف روال ها تشکیل شده اند . یک روال منبعی برای ارزیابی چیزی می باشد . **مثال :** a:-b,c.
به صورت روالی به صورت یک روال برای ارزیابی a توسط ارزیابی b,c است . در این جا " ارزیابی " چیزی به معنی تشخیص این است که آیا با توجه به برنامه ، در کل درست است یا نه .
روال a:-b,c در منطق می تواند به صورت $a \leftarrow b \wedge c$ نوشته شود . و سپس به صورت اعلانی خوانده می شود ، a درست است در صورتی که b درست باشد و c هم درست باشد .
مفهوم ۲ – فراخوانی روال ها : اجرا ، شامل ارزیابی فراخوان ها می باشد و با یک پرس و جوی *اولیه* شروع می شود
مثال ها :

---

[1] prolog programming
[2] PROLOG = "PROgramming in LOGic "
[3] non-determinism
[4] pattern-matching
[5] meta-programming

?-a,d,e.
?-likes(chris,X).
?-flight(gatwick,Z),in_poland(Z),flight(Z,beijing).

پرس و جوها در حال پرسیدن این هستند که آیا فراخوان های درون آن ها با توجه به روال های ارایه شده در برنامه درستند یا نه . پرولوگ فراخوان های در یک پرس و جو را به صورت ترتیبی و با همان ترتیبی که از چپ به راست نوشته شده انذ ، ارزیابی می کند :

?-a,d,e

اول a ارزیابی می شود ، بعد d و سپس e ارزیابی می شود . به صورت قراردادی ، واژگان شروع شده با یک حرف بزرگ یا خط زیرین (_) ، به صورت متغیرها رفتار می نمایند :

?-likes(chris,X)

در این جا ، X یک متغیر می باشد . پرس و جوها و روال ها هر دو به کلاس عبارت های منطقی نامیده شده به صورت *شرط ها* تعلق دارند .

مفهوم ۳ – محاسبات : یک *محاسبه* ، *زنجیره ای* از *پرس و جوهای مشتق شده* می باشد . پرولوگ ، اولین فراخوان در پرس و جوی جاری را انتخاب می نماید و یک عبارت برنامه ای که ابتدای آن با فراخوان مطابقت می نماید را جستجو می کند . در صورتی که چنین عبارتی وجود داشت ، فراخوان با بدنه ی عبارت ، که ارایه کننده ی پرس و جوی مشتق شده ی بعدی می باشد جایگزین می شود . این فقط در حال به کار گرفتن روش استاندارد فراخوانی کننده در هر صورتی است .

مثال :

?-a,d,e پرس و جوی اولیه
a:-b,c شرط برنامه با ابتدای a و بدنه ی b,c

با شروع از پرس و جوی اول ، اولین فراخوان در آن با ابتدای شرط نشان داده شده منطبق می شود و بنابراین پرس و جوی مشتق شده به صورت ?-b,c,d,e. می باشد . سپس ، اجرا در مورد پرس و جوی مشتق شده به روشی مشابه عمل می کند

مفهوم ۴- محاسبات موفق - یک محاسبه در صورتی موفق است که پرس و جوی تهی را نتیجه دهد

مثال :

?-likes(bob,prolog). پرس وجو
likes(bob,prolog). عبارت درون برنامه

فراخوان با بدنه منطبق می شود و با بدنه ی (خالی) شرط منطبق می شود و بنابراین پرس و جوی مشتق شده خالی می باشد . بنابراین پرس و جو موفق شده است

مفهوم ۵ – عدم موفقیت محدود - یک محاسبه در صورتی به صورت محدود ناموفق می شود که فراخوان انتخاب شده از پرس و جو با هیچ ابتدایی از شرط ها مطابقت نکند .

مثال :

?-likes(bob,haskell). پرس و جو

این به صورت محدود ناموفق می شود اگر هیچ شروط برنامه ای که ابتدای آن با likes(bob,haskell). منطبق شود وجود نداشته باشد

مثال دیگر :

?-likes(chris,haskell). پرس و جو
likes(chris,haskell):-nice(haskell).

در صورتی که هیچ شرطی که با ابتدای nice(haskell) مطابقت نماید وجود نداشته باشد ، محاسبه بعد از اولین مرحله با عدم موفقیت مواجه می شود
مفهوم ۶ – عدم موفقیت نامحدود - یک محاسبه به صورت نامحدود دارای عدم موفقیت است در صورتی که هر پرس و جو در آن با یک پرس و جوی غیر تهی دنبال شود :
مثال :

?-a. پرس وجو
a:-a,b. شرط

این یک محاسبه ی نامحدود را ارایه می کند

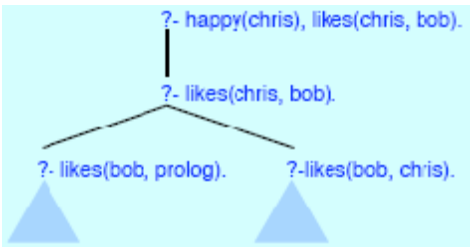
?-a.
?-a,b.
?-a,b,b.
.

این ممکن است برای به دست آوردن برخی پردازه های همیشگی ، مفید باشد
مفهوم ۷ – جواب های چندگانه : یک پرس وجو ممکن است چند محاسبه را به وجود بیاورد
مثال :

?-happy(chris).

happy(chris).
likes(chris,bob):-likes(bob,prolog).
likes(chris,bob):-likes(bob,chris).

و ...
در این صورت ما دارای یک درخت جستجو که در آن هر شاخه یک محاسبه ی مجزا می باشد هستیم



این زیردرخت ها بسته به این که "در غیر این صورت در برنامه باشد" ممکن است موفق یا ناموفق باشند .
مفهوم ۸ – جواب ها و نتیجه ها : یک محاسبه ی موفق تایید می کند که "و" ی درون پرس و جوی اولیه ، یک نتیجه ی منطقی برنامه می باشد .
مثال :

?-a,d,e.

در صورتی که این از یک برنامه ی P به دست می آید ، جواب محاسبه شده $a \wedge d \wedge e$ می باشد و ما داریم $P \models a \wedge d \wedge e$
به طور معکوس : در صورتی که برنامه ی P هیچ محاسبه ی موفقی را از پرس و جو به دست نیاورد ، در این صورت عطف پرس وجو یک نتیجه از P نمی باشد .
مفهوم ۹ – آرگومان های متغیر : متغیر هایی که درون پرس و جوها هستند به صورت سور وجودی رفتار می نمایند

مثال :

?-likes(X,prolog).

می گوید که " آیا $(\exists X) : likes(X, prolog)$ درست باشد ؟ " یا " X ای را پیدا نمایید که به ازای آن likes(X,prolog) درست باشد " . متغیرهای درون شرط های برنامه به صورت سورهای عمومی عمل می کنند .

مثال :

likes(chris,X):-likes(X,prolog).

این عبارت را بیان می نماید : $(\forall X)(likes(chris, X) \leftarrow likes(X, prolog))$

مفهوم ۱۰ – تطبیق تعمیم یافته : تطبیق یک فراخوان با ابتدای یک شرط لازم دارد که آن ها یا قبلا کاملا برابر باشند یا بتوانند برابری را به وجود آورند ، اگر لازم باشد با معرفی اتصال های متغیرهای آن ها .

مثال :

?-likes(U,chris).
likes(bob,Y):-understands(bob,Y).

در این جا ، likes(U,chris) و likes(bob,Y) با قرار دادن U/bob و Y/chris می توانند کاملا برابر باشند . این پروسه ، *یکسان سازی* نام دارد . پرس و جوی مشتق شده ?-understands(bob,chris). می باشد .

واژه های پرولوگ : واژه ها ، عناصری هستند که می توانند به صورت آرگومان های گزاره ها ظاهر شوند . آن ها می توانند به صورت داده های اساسی به کار گرفته شده در مدت اجرا دیده شوند . آن ها ممکن است به صورت ثابت در کد داده شده ی برنامه و پرس و جوی اولیه موجود باشند یا آن ها ممکن است به صورت پویا ، توسط پردازه ی یکسان سازی بیایند . واژه هایی که شامل هیچ متغیری نمی باشند *زمینه*[1] نام دارند . یک ویژگی خاص پرولوگ این است که می تواند هم داده های زمینه و هم داده های غیر زمینه را پردازش نماید . یک برنامه ی پرولوگ می تواند کارهای مفید را با یک ساختمان داده ، حتی در زمانی که ساختمان تا حدودی ناشناخته می باشد انجام دهد .

واژه های ساده

اعداد : ۳ ۵,۶ -۱۰ -۶,۳۱

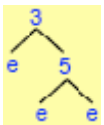
اتم ها : [ ] 'Hello there' x2 tom apple

متغیرها : _ _35 Person Left_Subtree Chris Y31 X

واژه های مرکب

واژه های پیشوندی : mother(chris)

← tree(e,3,tree(e,5,e))

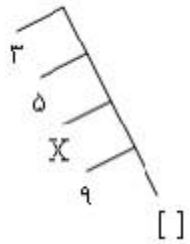


← tree(T,N,tree(e,5,e)) یک درخت دودویی که ریشه و زیردرخت سمت چپ ناشناخته می باشند

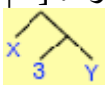
واژه های لیستی : [ ] [3,5] [3,5,X,9]

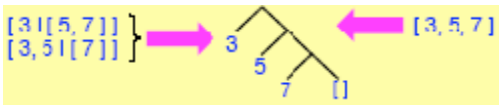
[1] ground

شکل لیست ها به صورت زیر کلاسی از درخت های دودویی می باشد ←

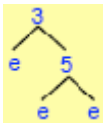


یک لوله (|) می تواند به صورت یک مجزا کننده ، برای نمایش یک لیست به شکل [لیست باقی مانده | عناصر لیست جزء به جزء] استفاده شود : [X,3|Y]





واژه های چندتایی [1] **:** ((U,V),(X,Y)) (1,2,3) (bob,chris)
← (e,3,(e,5,e))



این ها در زمان کار با ساختمان داده های با طول ثابت ، دارای برتری هستند

واژه های محاسباتی : sin(X+Y)/(cos(X)+cos(Y)) 3*X+5

گرچه این ها دارای یک شکل ریاضیاتی هستند ، آن ها به صورت ریاضی فقط با یک مجموعه از فراخوان های معین که پس از این ارایه شده اند تفسیر شده اند .

ارزیابی های قطعی - پرولوگ در حالت کلی به صورت غیرقطعی می باشد چون که ارزیابی یک پرس و جو ممکن است چند محاسبه را تولید نماید . اگر فقط **یک** محاسبه تولید می شود (چه موفقیت آمیز باشد و چه نباشد) ، ارزیابی قطعی نامیده می شود . در این صورت درخت جستجو از یک شاخه ی تک تشکیل شده است .

مثال :

```
all_bs([ ]).
all_bs([b|T]):-all_bs(T).
```

این برنامه لیستی را تعریف می نماید که در آن هر عضو ، b است . حالا به پرس و جوی ?- all_bs([b,b,b]). توجه نمایید . این یک ارزیابی قطعی را تولید خواهد کرد :

```
all_bs([ ]).
all_bs([ b | T ]) :- all_bs(T).

?- all_bs([ b, b, b ]).
?- all_bs([ b, b ]).
?- all_bs([ b ]).
?- all_bs([ ]).
?- .
```

بنابراین در این جا درخت جستجو شامل یک شاخه (محاسبه) می باشد

[1] tuple

مثال دیگر : پرولوگ الحاق لیستی ابتدایی را ارایه می نماید :

append(X,Y,Z)

در صورتی که بخواهیم می توانیم این عمل را خودمان به صورت زیر تعریف نماییم :

```
app([ ],Z,Z).
app([U|X],Y,[U|Z]):-app(X,Y,Z).
```

حال به پرس و جوی ?-app([a,b],[c,d],L). توجه نمایید :

```
app([ ],Z,Z).
app([U|X],Y,[U|Z]):-app(X,Y,Z).

?- app([a,b],[c,d],L).
```

فراخوانی ، سر شرط برنامه ی دوم را با ساخت اتصالات زیر منطبق می نماید :

L/[a|Z] Y/[c,d] X/[b] U/a

بنابراین ، ما فراخوانی را با جایگزین نمودن بدنه ی شرط انجام می دهیم ، سپس متصل کننده هایی که سبب تولید پرس و جوی مشتق شده می شوند را به کار می بریم :

```
?-app([b],[c,d],Z).
app([ ],Z,Z).
app([U|X] , Y,[U|Z]) :-app(X,Y,Z).
?-app([b],[c,d],Z).
```

مرحله ی شبیه دیگر ، Z/[b|Z2] را متصل می نماید و پرس و جوی مشتق شده ی بعدی را می سازد

```
?-app([ ],[c,d],Z2).
```

این با منطبق کردن شرط اول برنامه و چسباندن Z2/[c,d] موفق می شود . بنابراین جواب L/[a,b,c,d] می باشد . در هر مرحله ، فراخوانی ، بیش تر از یک شرط سر برنامه را منطبق نمود ، و بنابراین ارزیابی دوباره قطعی می باشد . توجه نمایید که در کل ، هر مرحله در یک محاسبه ، اتصالاتی را که یا منتشرشده برای متغیرهای پرس و جو هستند یا نگهداری شده در یک طرف ، در مورد آن هایی که در جواب نهایی شرکت می کنند می باشند را تولید می نماید . در مثال ، متصل کننده ی خروجی نهایی L/[a,b,c,d] می باشد . اتصالات یک طرف که *محیط متصل کننده*[1] ی محاسبه نامیده می شود را نگهداری می کنند . *روش*[2] پرس و جو در مثال قبلی ?-app(input,input,output). ، به این صورت می باشد : دو آرگومان اول ، دارای ورودی کاملا شناخته شده بودند ، در حالی که آرگومان سوم دارای خروجی کاملا ناشناخته می باشد . به هرحال ، ما می توانیم پرس و جوهای دارای هر ترکیب ار آرگومان ها را به هر طریقی که می خواهیم ، استفاده نماییم . بنابراین روش دومی وجود دارد که در آن پرولوگ غیرقطعی می باشد : ***برنامه روش قرار دادن پرس و جوها را تشخیص نمی دهد .***

مثال دیگر : با استفاده از همان برنامه می توانیم یک پرس و جوی دارای روش ?-app(input,input,input) را قرار دهیم . نظیر :

```
?-app([a,b],[c,d],[a,b,c,d]).
```

این ، فقط دارای یک محاسبه می باشد ، که موفق می شود ولی هیچ متصل کننده ی خروجی را برنمی گرداند .

---

[1] binding environment
[2] mode

مثـال دیگـر : بـه پـرس و جـوی .(?-app[X,[b|L],[a,E,c,d]. توجـه نماییـد . بـا داشـتن روش -? app(output,mixed,mixed) . بـــه صـــورت قطعـــی بـــا ارایـــه ی اتـــصالات خروجـــی X/[a],L/[c,d],E/b موفق می شود . این دومین نوع غیرقطعی ، *ورودی – خروجی غیر قطعی* [1] نـام دارد و پرولـوگ را از دیگـر زبـان هـای رسـمی ، متمـایز مـی نمایـد . بـا یـک برنامـه ی منفـرد پرولـوگ ، ممکن است یـک طیـف نامحـدود از پـرس و جوهـا را مطـرح نمـاییم ، امـا بـا زبـان هـای رسمی دیگر ، ما باید برنامه را در هر زمانی که می خواهیم یک نوع جدید از مساله را حل نمـاییم ، تغییر دهیم . این به این معنی نمی باشد که یک برنامه ی منفرد پرولوگ بـه همـه ی پرس و جوهـا بـا بـازدهی برابـر رسیدگی مـی نمایـد . اغلـب در توجـه کامـل بـه بـازده ، مـا ممکن است بـه خـوبی یـک برنامه ی پرولوگ را برای رسیدگی به یک نوع های جدید از پرس و جوها تغییر دهیم .

خلاصه ـ با ارایه ی یک مساله ، ما می توانیم هر پرس و جویی را کـه مـی خـواهیم و بـا هـر روشـی مطـرح نمـاییم . برخـی از پـرس و جوهـا فقـط یـک محاسـبه را بـه وجـود مـی آورنـد ، در حـالی کـه آن های دیگر تعدادی محاسبه را به وجود می آورند . محاسبات موفق چندگانـه ممکن است جـواب هـای متمایزی را نتیجه بدهند یا جواب های متمایزی را نتیجه ندهند . هر جواب ، یـک نتیجـه ی منطقـی از برنامه می باشد

ارزیابی های غیرقطعی

در زمـانی کـه فراخـوانی بـا چنـد سـر شـرط یکـی مـی شـود ارزیـابی پرولـوگ غیرقطعـی (دارای چنـد محاسبه) می باشد . در این زمان ، درخت جستجو هم دارای چند شاخه می باشد .

مثال :

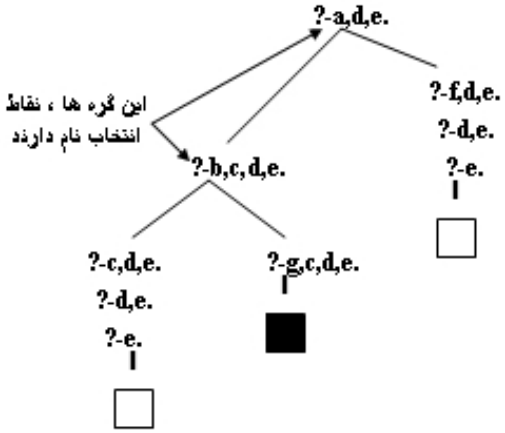
a:-b,c. دو سر شرط
a:-f. با a یکی می شود

b. دو سر شرط
b:-g با b یکی می شود

c.
d.
e.
f.

یک پرس و جو از فراخوانی با a یا b انتخاب می شود ، بنابراین دارای چند محاسبه می باشد :

[1] input-output non-determinism



در صورتی که نمایش داده شده با چند محاسبه باشد ، پرولوگ در هر لحظه فقط یکی را تولید می نماید . هر کدام از محاسباتی که در حال حاضر در حال ایجاد می باشند ، پرولوگ به آن ها می پردازد تا این که یا موفق شوند و یا به صورت محدود دارای عدم موفقیت شوند . این روش ، *جستجوی اول عمق* نام دارد . این یک روش *ناعادلانه* [1] می باشد ، که در آن ضمانتی برای تولید همه ی محاسبه ها وجود ندارد ، مگر این که همه محدود باشند . در زمانی که یک محاسبه خاتمه می یابد ، پرولوگ به شاخه هایی که زودتر ارایه شده اند و تا کنون به آن ها نرفته است ، برگشت به عقب می نماید . ارزیابی به صورت یک کل ، فقط در زمانی که هیچ نقطه ی انتخاب این جوری باقی نمانده باشد خاتمه می یابد . ترتیبی که در آن شاخه ها امتحان می شوند وابسته به مرتبه ی متن شرط های انتساب داده شده در برنامه می باشد . این ، *قانون جستجوی* [2] پرولوگ نام دارد ، که شاخه ها را در درخت جستجو ، اولویت بندی می نماید .

بازده - بازدهی که در آن پرولوگ ، یک مساله را حل می نماید وابسته به این موارد می باشد : روشی که دانش در برنامه ارایه می شود و ترتیب فراخوانی ها

مثال : پرس و جوی زودتر و برنامه را تغییر دهید

?-d,e,a. مرتبه ی فراخوانی متفاوت

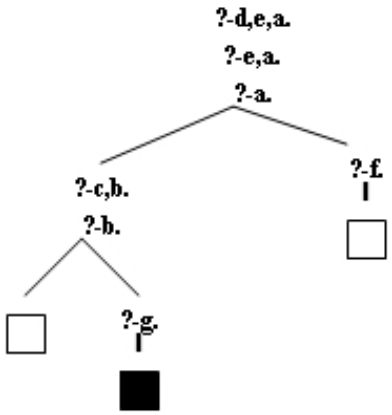
a:-c,b. مرتبه ی فراخوانی متفاوت
a:-f.

b.
b:-g.

c.
d.
e.
f.

این ارزیابی ، فقط دارای ۸ مرحله می باشد ، در حالی که ارزیابی قبلی دارای ۱۰ مرحله بود

[1] unfair
[2] search rule



روشی برای انتخاب فراخوانی بعدی ، برای پردازش *قانون محاسبه* [1] نام دارد و دارای یک برتری اساسی در بازده می باشد . بنابراین ، به یاد بیاورید ...

یک قانون محاسبه برای انتخاب پرس و جوی بعدی ، تصمیم گیری می کند . یک قانون جستجو ، تصمیم می گیرد که کدام شرط برنامه برای فراخوانی انتخاب شده ، به کار گرفته شود . در پرولوگ ، دو قانون به ترتیب ، اجرا می شوند : " **اولین** انتخاب درون پرس و جوی فعلی را انتخاب نمایید " و " **اولین** شرط برنامه ی امتحان نشده در قبل که اجراشدنی است را انتخاب نمایید " .

یکسان سازی - این پردازشی است که در آن پرولوگ تصمیم می گیرد که یک فراخوانی می تواند از یک شرط برنامه استفاده نماید یا نه . فراخوانی باید با سر ، یکی شود . دو گزاره قابل یکی شدن هستند اگر و فقط اگر آن ها دارای یک *نمونه ی مشترک* [2] باشند . **مثال :**

```
?-likes(Y,chris).
likes(bob,X):-likes(X,logic).
```

تصور نمایید θ یک مجموعه ی متصل کننده ی {Y/bob,X/chris} باشد . اگر E هر فرمول منطقی ای باشد Eθ نتیجه ی به کار گیری θ با E را مشخص می نماید و بنابراین به دست آورنده ی یک نمونه از E می باشد .

likes(Y,chris)θ = likes(bob,chris)
likes(bob,X)θ = likes(bob,chris)

از آنجایی که این دو مثال همانند هستند ، می گوییم که θ یک یکسان کننده برای گزاره های اصلی می باشد .

مرحله ی محاسبه ی کلی

پرس و جوی فعلی ?-P(args1),others.
شرط برنامه P(args2):-body.

در صورتی که θ موجود باشد ، P(args1) θ =P(args2)θ ، این شرط می تواند با این فراخوانی برای تولید ، استفاده شود . پرس و جوی مشتق شده :

?-bodyθ ,othersθ

در غیر این صورت ، این شرط نمی تواند با این فراخوانی مورد استفاده قرار گیرد .

مثال :

```
?-app(X,X,[a,b,a,b]).
```

---

[1] computation rule
[2] common instance

با محاسبه ی موفق ، داریم :

O1={X/[a|X1]}
O2={X1/[b|X2]}
O3={X2/[ ]}

{X/[a,b],X1/[b],X2/[ ]} ترکیب بندی این ها ، اتصالات خروجی در یکسان کننده ها هستند . می باشد . جواب جانشین ، {X/[a,b]} می باشد با به کاربستن آن در پرس و جوی اولیه جواب ، app([a,b],[a,b],[a,b,a,b]) خواهد بود .

پردازش لیست

لیست ها ، عادی ترین ساختارهای استفاده شده در پرولوگ می باشند و ارتباط های درون آن ها معمولا نیازمند برنامه های بازگشتی می باشد .

مثال : [ ] – یک لیست تهی می باشد ، [۳ ، ۲ ، ۱] - یک لیست دارای سه عدد می باشد ، [bob ، ted ، alice] لیستی دارای سه عبارت (ثابت) می باشد و لیست [a ، [۳ ، ۲ ، ۱] ، [ ]] – لیستی از لیست ها (و یک عبارت) می باشد .

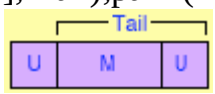
برای بررسی یک زیر قسمت یک لیست : [H | T] ، لیست به عنصر اول H و انتهای T تجزیه می شود .

مثال :

?– [H | T] = [[ ] , [1 , 2 , 3] , a] .
H = [ ]
T = [[1 , 2 , 3] , a]
y e s

مثال : برای تعریف یک متقارن [1] (عبارتی که از هر دو طرف به یک صورت خوانده می شود ، مثل کلمه ی level) داریم :

```
palin([ ]).
palin([U | Tail]) :-append(M, [U], Tail),palin(M).
```



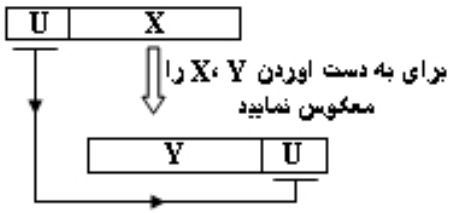
به صورت چکیده تر :

```
palin([ ]).
palin(L) :- first(L, U), last(L, U),middle(L, M), palin(M).
first([U | _], U).

last([U], U).
last([_ | Tail], U) :- last(Tail, U).

middle([ ], [ ]).
middle([_], [ ]).
middle([_ | Tail], M) :- append(M, [_], Tail).
```

[1] palindrome

مثال - برای معکوس کردن یک لیست :

```
reverse([ ], [ ]).
reverse([U | X], R) :- reverse(X, Y),append(Y, [U], R).
```



توجه کنید که برنامه به صورت بازگشتی از انتها نمی باشد . اگر ما تلاش کنیم که این کار را با مرتب سازی دوباره انجام دهیم :

```
reverse([U | X], R) :- append(Y, [U], R),
reverse(X, Y).
```

آن گاه ، ارزیابی برای برخی از موارد ، نامحدود خواهد بود . در زیر برنامه ی بازگشتی از انتها آورده شده است :

```
reverse(L, R) :- rev(L, [ ], R).
rev([ ], R, R).
rev([U | Tail], A, R) :- rev(Tail, [U | A], R).
```

با این برنامه ، زمانی برای معکوس کردن یک لیست ارایه شده وابسته به طول آن لیست ، برده می شود و محیط زمان اجرا یک پشته با فراخوان های معلق را به وجود نمی آورد .

توابع موجود برای لیست ها - پرولوگ دارای کتابخانه ای از برنامه های لیست می باشد ، نظیر :

append(X,Y,Z)	Y را به X اضافه می کند و Z را ارایه می کند
reverse(X,Y)	معکوس X ، Y می باشد
length(X,N)	طول X ، N می باشد
member(U,X)	U در X می باشد
non_member(U,X)	U در X نمی باشد
sort(X,Y)	X را مرتب می نماید و Y را ارایه می کند

برای دسترسی به این توابع در Sicstus ، در فایل خود ، عبارت زیر را اضافه نمایید :

```
?- use_module(library(lists)).
```

بررسی نوع - برای بررسی انواع آرگومان ها می توانید از موارد زیر استفاده نمایید :

atom(X) X یک اتم است ، number(X) X یک عدد است ، integer(X) X یک integer است ، var(X) X یک متغیر (بدون محدوده) است ، nonvar(X) X یک متغیر نمی باشد ، compound(X) X یک عبارت مرکب می باشد . با این توابع اولیه شما می توانید روال های بررسی نوع بیش تری را تعریف نمایید .

مثال ها : برای بررسی این که آیا یک عبارت ، لیست می باشد :

```
is_list(X) :- atom(X), X=[ ].
is_list(X) :- compound(X), X=[_ | _].
```

برای بررسی این که آیا یک عبارت ، یک درخت دودویی است :

```
bintree(X) :- atom(X), X=e.
bintree(X) :- compound(X), X=t(L, _, R),bintree(L), bintree(R).
```

مقایسه ی عبارات - پرولوگ دارای تعدادی توابع اولیه برای محاسبه ی عبارات می باشد ، شامل :

X=Y X با Y یکی می شود
به عنوان مثال ، X=a ، انتساب دهنده ی X/a می باشد
X==Y X و Y کاملا برابر هستند
به عنوان مثال ، [a,b]==[a,b] درست است ، ولی [a,b]==[a,X] نادرست می باشد
X\==Y X و Y با هم برابر نمی باشند
به عنوان مثال ، [a,b]=\=[a,X] ، بدون انتساب X ، درست است
محاسبه - عبارت های محاسباتی ، از عملگرهای استاندارد ، نظیر / * - + (در کنار دیگران) استفاده می نمایند . عملوند ها ، عبارات ساده یا عبارات محاسباتی می باشند . **مثال :**

(7 + 89 * sin(Y+1)) / (cos(X) + 2.43)

مقایسه ی عبارت های محاسباتی

E1=:=E2 بررسی می کند که آیا مقادیر E1 و E2 با هم برابرند یا نه
E1=\=E2 بررسی می کند که آیا مقادیر E1 و E2 نامساویند
E1<E2 بررسی می کند که آیا مقدار E1 کوچک تر از E2 می باشد یا نه

به علاوه ما داریم > برای بزرگ تر
>= برای بزرگ تر یا مساوی
=< برای مساوی یا کوچک تر

مثال ها :

?- X=3,(2+2)=:=(X+1). درست است
?- (2+2)=:=(X+1),X=3. خطا دارد
?- (2+2)>X. خطا دارد

مقدار یک عبارت محاسباتی E شاید محاسبه شود و به یک متغیر X توسط فراخوانی X is E نسبت داده شود . **مثال ها :**

?- X is (2+2). درست است و X/4 را انتساب می دهد
?- 4 is (2+2). خطا دارد
?- X is (Y+2). خطا دارد

توجه کنید که is را با = اشتباه نگیرید . X=Y دارای معنی " X ممکن با Y یکسان شود " می باشد و به ندرت لازم می شود .

مثال ها :

?- X=(2+2). درست است و X/(2+2) را انتساب می دهد
?- 4=(2+2). خطا نمی دهد ولی دارای عدم موفقیت می باشد
?- X=(Y+2). درست است و X/(Y+2) را انتساب می دهد

گزاره ی "is" **فقط** برای هدف خیلی معین استفاده می شود . ***متغیر ، عبارت محاسباتی ای است که باید ارزیابی شود***

مثال - جمع کردن لیستی از اعداد :

```
sumlist([ ], 0).
sumlist([ N | Ns], Total) :-sumlist(Ns, Sumtail), Total is N+Sumtail.
```

این بازگشتی از انتها نمی باشد – طول پرس و جو متناسب با طول لیست ورودی ، افزایش خواهد یافت . اجرای غیر بازگشتی از انتهای معمولی :

```
?- sumlist([ 2, 5, 8 ], T).
```

```
?- sumlist([ 5, 8 ]), T is 2+T1.
?- sumlist([ 8 ], T2), T1 is 5+T2, T is 2+T1.
?- sumlist([ ], T3), T2 is 8+T3, T1 is 5+T2, T is 2+T1.
?- T2 is 8+0, T1 is 5+T2, T is 2+T1.
?- T1 is 5+8, T is 2+T1.
?- T is 2+13.
?- .
```

با انتساب خروجی T/15 ، درست است

مثال - انجام مثال قبلی با انتهای به صورت بازگشتی :

```
sumlist(Ns, Total) :- tr_sum(Ns, 0, Total).
tr_sum([ ], Total, Total).
tr_sum([ N | Ns ], S, Total) :-Sub is N+S,tr_sum(Ns, Sub, Total).
```

در این جا ، tr_sum(Ns,S,T) دارای معنی $T = S + \sum Ns$ می باشد . اجرا به صورت بازگشتی از انتها :

```
?- sumlist([ 2, 5, 8 ], T).
?- tr_sum([ 2, 5, 8 ], 0, T).
?- Sub is 2+0, tr_sum([ 5, 8 ], Sub, T).
?- tr_sum([ 5, 8 ], 2, T).
:
?- tr_sum([ 8 ], 7, T).
:
?- tr_sum([ ], 15, T).
?- .
```

و دوباره با T/15 ، درست می باشد . در این جا ، طول پرس و جو ، هیچ گاه با دو فراخوانی درست نمی باشد و هر پرس و جوی مشتق شده می تواند روی محتوای قبلی حافظه نوشته شود .

مثال : جستجوی اول عمق – جستجوی اول عمق از یک وضعیت شروع X :

```
d f s (X) :– g o a l (X) .
d f s (X) :– s u c c e s s o r (X, S ) , d f s (S ) .
```

نیازی به داشتن حلقه برای S نداریم : successor/2 برای همه موفق می باشد .

مثال : عضویت در یک لیست – isInList(H,L) در صورتی که H عضوی از لیست L باشد ، موفق است . اگر عنصر ، ابتدا (سر) لیست می باشد ، در این صورت یک عضو می باشد :

```
isInList(H, [H|T]).
```

ولی ما در مورد انتهای لیست ، بی تفاوت می باشیم :

```
isInList(H, [H| _ ]).
```

اگر سر لیست نباشد ، باید انتهای لیست را بررسی نماییم :

```
isInList(Item, [H|T]) :- isInList(Item, T).
```

ولی ما در مورد ابتدای لیست ، بی تفاوت می باشیم :

```
isInList(Item, [ |T]) :- isInList(Item, T).
```

کاربرد طبیعی :

```
?– i s I n L i s t ( 1 , [ 2 , 1 , 4 ] ) .
```

```
y e s

?- i s I n L i s t ( 1 , [ 2 , 5 , 4 ] ) .
no
```

اما پرولوگ به دنبال یک دلیل می باشد ، بنابراین ...

```
?- i s I n L i s t (X, [ 2 , 5 , 4 ] ) .
X=2
X=5
X=4
```

مثال : متصل کردن دو لیست – $append(L_1, L_2, L_3)$ در صورتی که L_3 حاصل الحاق L_1 و L_2 باشد .

```
append ( [ ] , Y,Y) .
append ( [X| L ] ,Y, [ X| Z ] ) :- append (L ,Y, Z ) .
```

پرس و جو : append([1,2],[3],X) ؟ جواب ها : X=[1,2,3]
پرس و جو : append(A,B,[1,2]) ؟
جواب ها :

```
A=[ ] B=[1 ,2]
A=[1] B=[2]
A=[1 ,2] B=[ ]
```

مثال : شمارش عناصر یک لیست – شمارش همه ی عناصر

```
count ( [ ] , 0 ) .
count ( [ _ |T] , N) :- count (T,N1 ) , N i s N1 + 1 .
```

شمارش عناصر متمایز یک لیست

```
c o u n t _d i s t i n c t ( [ ] , 0 ) .
c o u n t _d i s t i n c t ( [H|T] , N) :- member (H,T) ,c o u n t _d i s t i n c t
(T,N) .
c o u n t _d i s t i n c t ( [H|T] , N) :- \+ member (H,T) ,c o u n t _d i s t i n c t
(T,N1 ) ,N i s N1 + 1 .
```

جدایی

جدایی میان فراخوانی ها می تواند همیشه با استفاده از روال های ارایه ی شرط های متناوب ، بیان شود . **مثال :**

```
out_of_range(X, Low, High) :- X<Low.
out_of_range(X, Low, High) :- X>High.
```

به طور هم ارز ، از کلمه ی ربط دهنده ی جدایی سمیکلن [1] " ; " پرولوگ استفاده نمایید . **مثال :**

```
out_of_range(X, Low, High) :- X<Low ; X>High.
```

برای استفاده از اجتماع و جدایی به صورت مخلوط ، از پرانتز برای جلوگیری از ابهام استفاده کنید ، **مثال :**

[1] semi-colon

a :- b, (c ; (d, e)).

نقیض - پرولوگ دارای کلمه ی ربط صریحی برای انجام نقیض به صورت کلاسیکی نمی باشد ،
مثال : innocent(X) ← ¬ guilty(X)
در منطق کلاسیک ، در عمل ما بی گناهی X [1] را با اثبات نقیض " X گناهکار [2] می باشد " انجام نمی دهیم . در عوض ، ما این کار را با *ناموفق به صورت محدود برای اثبات* [3] " X گناهکار است " انجام می دهیم . پرولوگ دارای یک کاراکتر مخصوص به صورت +\ که به صورت " ناموفق به صورت محدود برای اثبات " خوانده می شود ، می باشد . بنابراین در پرولوگ ، ما باید بنویسیم :
innocent(X) :- \+guilty(X).
معنای +\ در عمل به این صورت می باشد : P+\ درست است ، اگر P به صورت محدود ، غلط باشد و P+\ غلط می باشد ، اگر P ، درست باشد . **مثال :**

```
person(bob). likes(bob, frank).
person(chris).
person(frank).
sad(X) :-person(X),person(Y), X \== Y, \+likes(Y, X).
```

" X ، غمگین است اگر کس دیگری X را دوست نداشته باشد " . با استفاده از داده ها ، bob,chris و frank غمگین هستند ، زیرا در هر مورد کس دیگری آن ها را دوست ندارد . +\ به طورکامل نقیض کلاسیکی را شبیه سازی نمی نماید . **مثال :** $p \leftarrow \neg p$ به صورت کلاسیک بر p ، دلالت می کند ، اما ، p:-\+p. نمی تواند p-? را حل نماید (به صورت نامحدود دارای عدم موفقیت خواهد بود ، نه به صورت محدود) . بنابراین ، p نتیجه ی منطقی در مورد اول می باشد ، اما در دومی یک نتیجه ی به صورت منطقی قابل محاسبه نمی باشد . **مثال دیگر :**

```
person(bob). likes(bob, frank).
person(chris).
person(frank).
very_sad(X) :-person(X),\+ (person(Y), X \== Y, likes(Y, X)).
```

" X در صورتی که کس دیگری دوستش نداشته باشد خیلی غمگین می باشد " . در این جا ، فقط bob و chris خیلی غمگین هستند ، زیرا در هر مورد ، هیچ کس دیگری آن ها را دوست ندارد .
نکته ی گرامری - قرار دادن یک فاصله میان +\ و (ضروری می باشد . برخی ار پرولوگ ها (اما نه Sicstus) برای زمینه شدن در نمونه ای که برای ارزیابی انتخاب می شود نیازی به P+\ ندارند . می توانیم مثال قبلی را به صورت زیر ، فرمول بندی نماییم :

```
very_sad(X) :- person(X), \+liked(X).
liked(X) :- person(Y), Y \== X, likes(Y, X).
```

این انتخابی مناسب می باشد : در صورتی که پرولوگ ما فراخوان های غیر زمینه ای +\ را رد نماید ، آن گاه ممکن است به صورت مستقیم جواب هایی غلط در زمان ارزیابی آن ها بدهد . در بالا ، فراخوانی liked(X)+\ در زمانی که انتخاب می شود ، زمینه می باشد ، زیرا فراخوانی person(X) ، قبلا X را زمینه نموده است . عملگر +\ به طور جزیی جبران را برای سر یک شرط محدود شده با یک گزاره ی منفرد انجام می دهد . در صورتی که ما بخواهیم دانشی را که می

[1] innocence
[2] guilty
[3] finitely failing to prove

گوید $A \vee B \leftarrow C$ را استفاده نماییم ، ما می توانیم آن را با A:-C,\+B. یا با B:-C,\+A. یا با استفاده همزمان از هر دو ، تخمین بزنیم .

تولید و آزمایش - تولید و آزمایش یک ویژگی تعدادی از الگوریتم ها می باشد . می تواند به صورت زیر فرمول سلزی شود . عناصر ارضا کننده ی خصوصیت P را تولید نمایید ، آزمایش کنید که آیا آن ها خصوصیت Q را راضی می نمایند یا نه . P به صورت یک تولید کننده عمل می نماید و Q به صورت یک آزمایش کننده عمل می کند . **مثال :** " X ، خوشحال است اگر همه ی دوستان X ، منطق را دوست داشته باشند " . در منطق کلاسیک ، ما می توانیم این مطلب را به این صورت زیر بیان نماییم : $happy(X) \leftarrow (\forall Y)(friend(X,Y) \rightarrow likes(Y, \log ic))$ ، در پرولوگ ما می توانیم این مطلب را به این صورت بازنویسی نماییم : happy(X) :- forall(friend(X, Y), likes(Y, logic)). هر دوست Y ، از X را تولید خواهد کرد و که در آن forall ، آزمایش خواهد کرد که آیا Y ، منطق را دوست دارد یا نه .

مثال دیگر - نشان دهید که L لیستی از اعداد مثبت می باشد : - all_pos_nums(L) :- is_list(L),forall(member(U, L), (number(U), U>0)).

و برخی از روال های مناسب is_list . برخی از پرولوگ ها forall را پشتیبانی می کنند (اما نه Sicstus) . اگر لازم باشد ما می توانیم forall را خودمان به صورت زیر تعریف نماییم :

forall(P, Q) :- \+ (P, \+ Q).

" هیچ راه حل P برای حل Q با عدم موفقیت مواجه نمی شود " . توجه نمایید که forall ، به صورت کامل شبیه سازی $\forall$ نمی باشد

$(\forall ...)(P \rightarrow P)$ در منطق کلاسیک ، درست می باشد

اما ، forall(P,P) فقط اگر تعداد راه های حل کننده ی P ، محدود باشند موفق می شود .

واژه های فراخوانی - یک واژه ی فراخوانی ، هر چیزی است که مفسر پرولوگ می تواند برای ارزیابی منطقی ، پرسیده شود ، نظیر : یک فراخوانی اتمیک ، یک فراخوانی P+\ که P ، واژه ی فراخوانی می باشد ، یک اجتماع از واژه های فراخوانی ، یک جدایی از واژه های فراخوانی ، به علاوه ی دیگران ... . در یک فراخوانی forall(P,Q) ، آرگومان های P و Q شاید هر واژه ی فراخوانی باشند ، اما اگر آن ها اتمیک نباشند آن گاه به پرانتز بندی نیاز دارند .

تراکم - اغلب ما می خواهیم در یک لیست تک همه ی عناصر راضی کننده ی برخی از خصوصیات را جمع نماییم . پرولوگ ، برای این کار این عبارت را دارا می باشد : (لیست ، واژه ی فراخوانی ، واژه)findall

مثال :

```
likes(frank, chris).
likes(chris, logic).
likes(chris, frank).
```

برای پیداکردن همه ی کسانی که chris دوست می دارد :

```
?- findall(X, likes(chris, X), L).
```

که این عبارت ، L/[logic,frank] را برمی گرداند .

مثال دیگر : برای پیدا کردن همه ی زیر لیست های دارای طول ۲ لیست [a,b,c] :

```
?- findall([ X, Y ], sublist([ X, Y ], [ a, b, c ]), S).
```

که S/[[a,b],[b,c]] را برمی گرداند .

مثال دیگر : با داشتن لیست X ، لیست Y را که با جایگزینی هر عضو X با E به دست آمده را به دست آورید :

```
replace(X, E, Y) :- findall(E, member(_, X), Y).
?- replace([ a, b, c ], e, Y).
```

که Y/[e,e,e] را برمی گرداند

```
?- replace([ a, b, c ], [ 0 ], Y).
```

Y/[[0],[0],[0]] را برمی گرداند

مثال دیگر - یک لیست L از جفت های (X,F) که X ، فرد است و F یک لیست از همه ی دوستان X است را تولید نمایید :

```
friend_list(L) :-findall( (X, F),( person(X), findall(Y, friend(X, Y), F) ), L ).
```

بنابراین ، در این جا ما یک findall درون یک findall دیگر داریم

مثال دیگر : یک لیست L از افرادی که هر کدام هر کاری که chris انجام می دهد را انجام می دهند بسازید :

```
clones_of_chris(L) :-findall( X,( person(X),forall(does(chris, Y), does(X, Y))
), L ).
```

بنابراین ، ما در این جا یک forall قرار گرفته در findall دیگر را داریم .

مثال دیگر - با داشتن یک لیست L از کلاس ها ، بررسی نمایید که آیا در همه ی آن ها تعداد خانم ها بیش تر از آقایان است :

```
mostly_female_classes(L) :-forall( ( member(C, L),findall(F, (member(F, C),
female(F)), Fs),findall(M, (member(M, ), male(M)), Ms),length(Fs,
NF),length(Ms, NM) ), NF > NM ).
```

بنابراین ما دراین جا findall هایی قرار گرفته در forall داریم .

کنترل جستجو

اندازه ای برای این که کدام درخت جستجو تولید می شود می تواند با استفاده از "cut" که با "!" مشخص می شود ، کنترل شود . در زمان اجرا ، یک برش ، برخی تکه های درخت جستجو را هرس می کند . این با یک خواست برای متوقف کردن برخی از محاسبات ، به وجود آمده است . می تواند در هر جای یک پرس و جو یا برنامه در جایی که یکی ممکن است در جای دیگر یک فراخوانی معمولی را موجب شود قرار داده شود . از هر تعداد از برش ها می توانیم استفاده نماییم . یک شرط برنامه دارای یک برش شبیه زیر می باشد :

فراخوانی های دیگر ، ! ، فراخوانی های دیگر - : head

برش فقط در زمانی که به صورت فراخوانی بعدی برای ارزیابی انتخاب می شود ، عمل می کند ، و سپس ***همه ی راه های امتحان نشده ی ارزیابی کننده ی هر کدام از فراخوانی هایی که موجب برش شامل شرط شده اند را هرس نمایید*** و **همه ی راه های امتحان نشده ی ارزیابی کننده ی فراخوانی هایی که در این شرط ، در برش ، مقدم می باشند را هرس نمایید .**

```
s:-p,b.
s:-a.
```

اگر انتخاب شود ، برش ، هرس خواهد کرد

همه ی راه های امتحان نشده ی ارزیابی p

```
p:-c.
p:-
q,!,r.
p:-d.
```

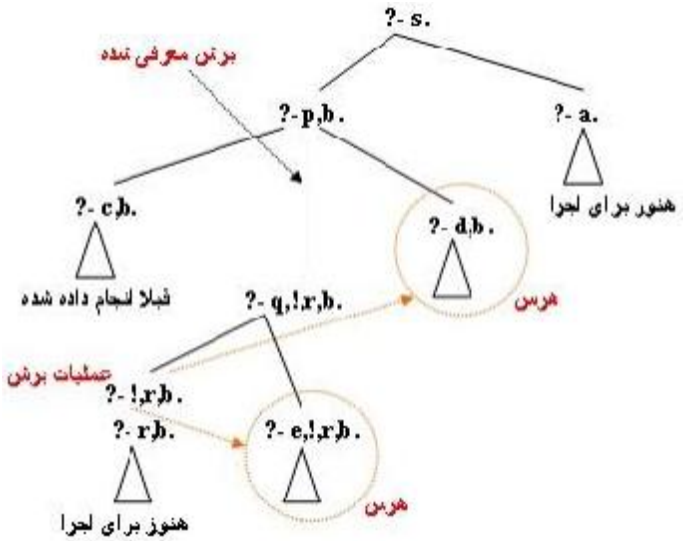
و

همه ی راه های امتحان نشده ی ارزیابی کننده ی q

q.

q:-e. همه ی شاخه های دیگر که در درخت انتظار تولید خواهند شد در این عمل برش ، باقی خواند ماند

و ...

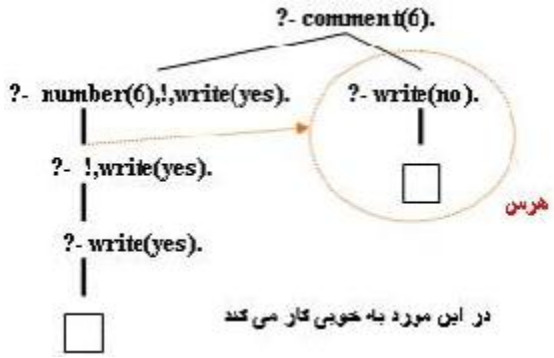


مثال :

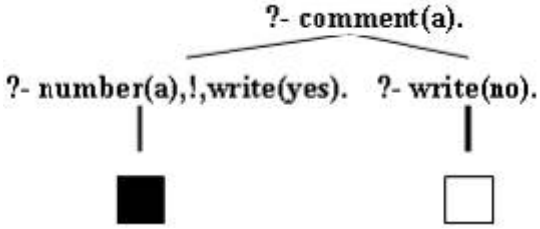
```
comment(X) :- number(X), !, write(yes).
comment(X) :- write(no).
```

این برنامه یک واژه ی X را بررسی می کند و یک توضیح را چاپ می کند . این به این معنی است که اگر X ، یک عدد باشد ، آن گاه ، توضیح yes می باشد و در غیر این صورت no خواهد بود . آیا با فرض این که X زمینه باشد هم کار خواهد کرد ؟

```
comment(X) :- number(X), !, write(yes).
comment(X) :- write(no).
```




```
comment(X) :- number(X), !, write(yes).
comment(X) :- write(no).
```



همچنین در مورد بالا هم به خوبی کار می کند . اما تصور نمایید که ما شرط ها را به صورت زیر بنویسیم :

```
comment(X) :- write(no).
comment(X) :- number(X), !, write(yes).
```

مثال دیگر : least(X,Y,L) را به این صورت تعریف نمایید که " L ، کم ترین X و Y می باشد "

```
least(X, Y, X) :- X<Y, !.
least(X, Y, Y).

?- least(1, 2, L).
```

در حال حاضر با انتساب L/1 موفق می باشد

```
?- least(2, 1, L).
```

در حال حاضر با انتساب L/1 موفق می باشد
اما ...

```
?- least(1, 2, 2).
```

اشتباها درست است

```
?- least(a, b, b).
```

اشتباها درست است
این مطلب با وجودی که شرط ها به صورت مرتب هستند درست می باشد .
نصیحت های مهم - در صورتی که به صورت موجه می توانید از برش اجتناب نمایید ، این کار را انجام دهید . در صورتی که باید از برش استفاده نمایید ، به ترتیب شرط توجه نمایید . در هر رویداد ، فقط **حقیقت** را محاسبه نمایید .
مثال :

```
comment(X) :- number(X), write(yes).
comment(X) :- \+number(X), write(no).
```

این برنامه دارای هیچ برشی نمی باشد ، به صورت بالقوه member(X) را وابسته به پرس و جو ارزیابی می کند .

فرا برنامه نویسی

مربوط به برنامه هایی می باشد که دارای دسترسی به طور متفاوت هستند و دیگر برنامه ها یا اجزای آن ها را کنترل یا تحلیل می نماید . این تخمینی قابل مقایسه برای استفاده از توابع با مرتبه ی بالاتر در یک زبان برنامه نویسی تابعی است . در پرولوگ ، بیش تر فرا برنامه نویسی از این واقعیت استفاده می کند که : ***واژه ها و گزاره ها دارای ساختار نحوی کاملا برابر می باشند .***

مثال :

```
overcome_with_joy(X) :- user_of(X, prolog).
```

در بالا ، user_of(X,prolog) یک گزاره می باشد

```
overcome_with_joy(X) :- true_that(user_of(X, prolog)).
```

در بالا ، user_of(X,prolog) یک آرگومان (واژه) می باشد .

فراگزاره های موجود - ما قبلا با برخی از آن ها آشنا شده ایم :

```
\+P
forall(P, Q)
findall(Term, Q, List)
```

در این جا ، P و Q آرگومان های سطح شی ء می باشند ، اما به صورت واژه های فراخوانی در فرا سطح ، تفسیر می شوند . استفاده در زمان اجرای آن ها می تواند از مکانیسم یکسان سازی همانندی به صورت استفاده شده برای واژه های در سطح شی ء استفاده نماید .

مثال :

```
choose(X, wants(chris, X)).
?- choose(Y, Q), forall(nice(Y), Q).
```

از این پرس و جو ما می توانیم پرس و جوی مشتق شده ی زیر را داشته باشیم

```
?- forall(nice(Y), wants(chris, Y)).
```

با انتساب X/Y,Q/wants(chris,Y)

THE=.. : این یکی دیگر از فرا گزاره های موجود می باشد . یک واژه را به یک لیست متضمن اجرا و آرگومان های اصلی واژه ی ، ارتباط می دهد .

مثال ها :

```
chris =.. L
```

L / [chris] را انتساب می دهد

```
happy(chris) =.. L
```

L / [happy, chris] را انتساب می دهد

```
likes(X, prolog) =.. L
```

L / [likes, X, prolog] را انتساب می دهد

```
T =.. [append, X, Y, Z]
```

T / append(X, Y, Z) را انتساب می دهد

```
T =.. [s, s(0)]
```

T / s(s(0)) را انتساب می دهد

یک مثال سخت تر : از هر واژه ی غیر متغیر داده شده ، یک لیست L را از همه ی اجراکننده های واژگان با آریتی آن ها توسعه دهید . برای مثال ، ما پرس و جوی زیر را داریم :

```
?- functors(p(a, f(X, g(b)), Y), L).
```

برای برگرداندن L / [(p, 3), (a, 0), (f, 2), (g, 1), (b, 0)]

نکته ی گرامری : اتم های پرولوگ فقط اجراکننده هایی هستند که آریتی آن ها برابر صفر می باشد . در این جا ، یک برنامه را می بینید :

```
functors(Term, [ ]) :- var(Term), !.
functors(Term, [(F, Arity) | Functors]) :-Term =.. [F | Args],length(Args, Arity),findall(E, ( member(Arg, Args),functors(Arg, Es),member(E, Es)), Functors).
```

مطمین شوید که آن را فهمیده اید .

فرا متغیرها - این ها متغیرهای معمولی هستند اما این انتظار از آن ها می باشد که برای واژه هایی که به خود می گیرند به صورت واژه های فراخوانی رفتار نمایند .

مثال : در زیر برنامه ای را می بینید که X+\ را شبیه سازی می نماید :

```
our_not(X) :- X, !, fail.
our_not(X).
```

نکته – "fail" همیشه به صورت محدود ناموفق می شود . در این جا X به صورت یک فرا متغیر عمل می کند :

```
our_not(X) :- X, !, fail.
our_not(X).

?- our_not(happy(chris)).
```

در شرط اول ، X/happy(chris) را انتساب می دهد ، تا این که X یک واژه ی فراخوانی در مثالی که برای ارزیابی انتخاب شده است باشد . پرس و جوی بالا دقیقا به صورت ?- happy(chris)+\ رفتار می نماید .

مثال دیگر :

```
aspect(logical).   person(chris).   logical(chris).
aspect(strict).    person(susan).   strict(susan).
aspect(fair).      person(marek).   fair(susan).
aspect(crazy).     person(mike).    fair(marek).
tell_us_about(X, Y) :-person(X), aspect(Y), Test=..[Y, X],Test.
```

?- tell_us_about(susan, Y). Y/strict یا Y/fair را برمی گرداند

?- tell_us_about(X, logical). X/chris را برمی گرداند

شرط های پویا - شرط ها می توانند ساخته شوند ، همفکری کنند و یا به صورت پویا حذف شوند . ارتباط های سر آن ها می تواند به صورت "پویا" ظاهر شود ، ولی Sicstus در این مورد سماجت نمی کند ، مگر این که ارتباطات آن ها به صورت اضافی توسط روال های صریح ، تعریف شوند ، به عنوان مثال ، :- dynamic likes/2. ، likes را وادار می کند که به صورت دینامیک بشود . عمومی ترین عمل کننده ها روی شرط های پویا به صورت زیر می باشند :

clause یک بدنه ی شرط که ارتباط سر آن داده شده را پیدا می نماید

assert یک شرط را به وجود می آورد

retract یک شرط را حذف می نماید

"CLAUSE"

یک فراخوانی برای آن دارای شکل زیر می باشد

clause(H,B) H هر گزاره ای است که در آن لااقل نام ارتباط داده شده است

اگر و فقط اگر H با θ توسط سر یک شرط پویای موجود یکی شود ، موفق می شود
Head :- Body.
که در نتیجه ی آن B به صورت Bodyθ برگردانده می شود

مثال :

likes(chris, X) :- likes(X, prolog).
likes(chris, X) :- honest(X), praises(X, chris).
likes(frank, prolog).

?- clause(likes(chris, frank), B).

دو مقدار مربوط به B را برمی گرداند

B / likes(frank, prolog)
B / (honest(frank), praises(frank, chris))

?- clause(likes(frank, X), B).

X/prolog و B/true را برمی گرداند

"ASSERT" - دارای شکل (شرط)assert می باشد .

مثال ها :

?- assert(likes(chris, prolog)).

به مبنای شرطی پویای [1] شرط اضافه می شود

likes(chris, prolog).
?- assert((likes(X, prolog) :- wise(X))).

به مبنای شرطی پویای شرط اضافه می شود

likes(X, prolog) :- wise(X).

"RETRACT" - دارای شکل (شرط) retract می باشد

مثال :

?- retract((likes(X, haskell) :- crazy(X))).

از مبنای شرطی پویای شرط ، حذف می شود

likes(X, haskell) :- crazy(X).

نکته ی اضافی - برای جمع کردن همه ی شرط های پویای جاری برای یک ارتباط P ، فراخوانی retract(P(…)) که در آن هر آرگومان P یک " _ " می باشد را اجرا نمایید ، نظیر retractall(likes(_, _))

مثال : شبیه سازی انتساب مخرب - تصور نمایید که یک آرایه ی دو- بعدی a از اعداد ، توسط یک مجموعه از ادعاها که قبلا با استفاده از assert تنظیم شده اند باشد : a(I,J,V) ، a[I,J]=V را ارایه می نماید . تصور نمایید که ما حالا می خواهیم a را برای این که هر عنصر قبلی کوچک تر از صفر به ۱۰ تغییر داده شود . ما می توانیم این کار را با ارزیابی واژه ی فراخوانی انجام دهیم :

forall((a(I, J, V), V<0),
(retract(a(I, J, V)), assert(a(I, J, 10))))

[1] dynamic-clause-base

فرا مفسرها ـ برنامه هایی که راه های اجرای پرس و جوها را با استفاده از برنامه های دیگری که به صورت داده ها رفتا ر می نمایند وجود دارند .

مثال :

```
solve([ ]).
solve(Query) :-select(Call, Query,Othercalls), clause(Call,
Body),callterm_to_list(Body, Bodycalls),combine(Bodycalls, Othercalls,
DQ),solve(DQ).
```

این عبارت ها رفتار یک مفسر ترتیبی و اول عمق خواسته شده برای ارزیابی یک لیست از فراخوانی های ارایه شده به صورت Query را بیان می نمایند . قانون محاسبه ی استفاده شده وابسته به چگونگی select و combine تعریف می شوند . برای بیان قانون محاسبه ی پرولوگ :

```
select(Call, [Call | Othercalls], Othercalls].
combine(A, B, C) :- append(A, B, C).
```

سپس ، نتیجه یک مفسر می باشد که به زبان پرولوگ نوشته شده است و رفتار پرولوگ را شبیه سازی می نماید

مثال دیگر : یک قانون محاسبه که اغلب برای پرولوگ عالی می باشد ، ***قانون تعویق*** [1] (یک مکاشفه ی استاندارد در هوش مصنوعی) می باشد : *" هر فراخوانی ای که به کم ترین تعداد شرط ها می تواند استناد کند را انتخاب نمایید "* . برای گرفتن این رفتار ، ما باید یک تعریف مناسب را برای select بنویسیم . تعریف select برای قانون تعویق :

```
select(Call, Query, Othercalls) :-findall((N, C),(member(C, Query),findall(_,
clause(C, _), Cs),length(Cs, N)), NCs),sort(NCs, [(_, Call) | _]),del(Call,
Query, Othercalls).
del(_, [ ], [ ]).
del(U, [V | X], X) :- U==V, !.
del(U, [V | X], [V | Y]) :- U\==V, del(U, X, Y).
```

و همان طوری که قبلا دیدیم combine را به صورت زیر تعریف نمایید :

```
combine(A, B, C) :- append(A, B, C).
```

منابع :

http://www.doc.ic.ac.uk/~cjh/prologslides

یا

http://wwwhomes.doc.ic.ac.uk/~cjh/prologslides
http://www.inf.unibz.it/~tessaris/teaching/AI/handouts/prolog.pdf

[1] procrastination principle

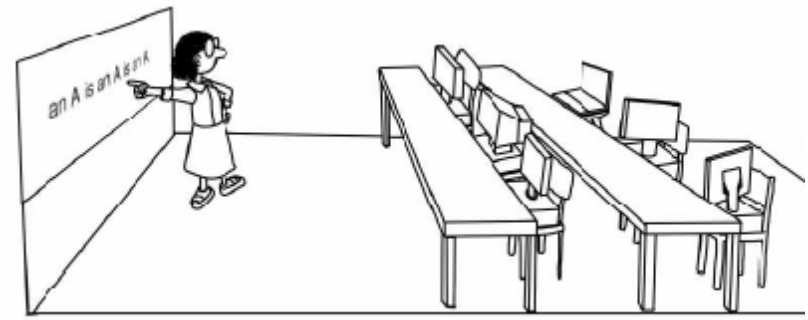
فصل بیست و هشتم

آموزش ماشینی

فصل بیست و هشتم
آموزش ماشینی (Machine Learning)

آشنایی با آموزش ماشینی
۱. آموزش ماشینی چیست ؟
۲. انواع مسائل و وضعیت ها
۳. محتوای زمینه
۴. اثبات با مدرک (مستندسازی)

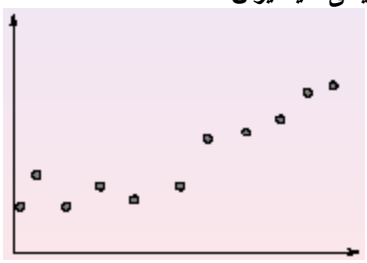
۱. **آموزش ماشینی چیست ؟** در زیر دیدی گرافیکی از آموزش ماشینی را مشاهده می کنید :



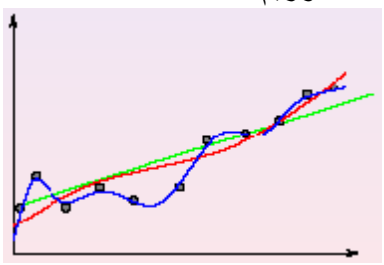
- آموزش یک خصوصیت ذاتی و حیاتی بشر می باشد
- آموزش به معنای *تغییر* دادن برای *بهتر* شدن (با توجه به یک *ضابطه یا معیار* داده شده) در موقعی که وضعیت شبیه می رسد ، می باشد
- آموزش ، آموزش توسط قلب یا عاطفه *نمی باشد*
- هر کامپیوتر ی می تواند به وسیله ی عاطفه یاد بگیرد (موارد عاطفی را یاد بگیرد) ، سختی ، در تولید یک رفتار در یک وضعیت جدید می باشد

چرا آموزش ، مشکل یا دشوار می باشد ؟

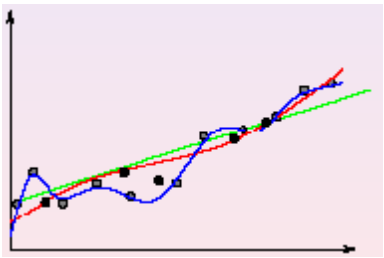
- با داشتن داده های آموزشی محدود ، شما باید یک ارتباط را برای یک دامنه ی نامحدود به دست آورید
- در واقع ، یک تعداد نامحدود از ارتباط های این چنینی وجود دارد



- چگونه باید ارتباط را به دست آوریم ؟



- کدام ارتباط ، مناسب ترین می باشد ؟



اصل اوکام ریزور [1]

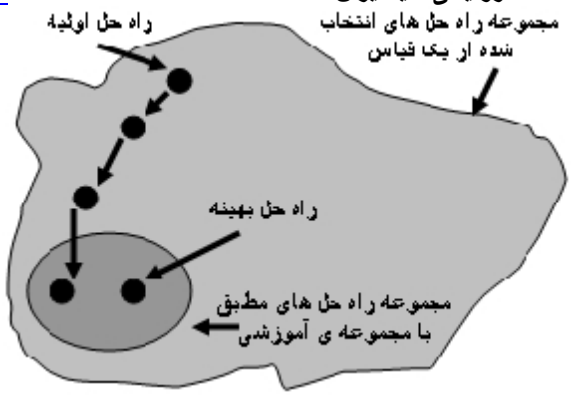
- ویلیام اوکام : راهبی بود که در قرن چهاردهم میلادی زندگی می کرد
- *اصل پارکیمونی* :

چیزی بیش تر از آنچه لازم است نباید افزایش داده شود و تعداد موجودیت ها برای توصیف هر چیز لازم هستند

- در زمانی که *تعداد زیادی* راه حل برای یک مساله ی داده شده وجود داشته باشد ، ما باید ساده ترین راه حل را انتخاب نماییم .
- اما منظور ما از ساده چیست ؟
- ما از دانش مساله برای حل کردن و تعریف کردن این که یک راه حل ساده چیست ، استفاده خواهیم کرد

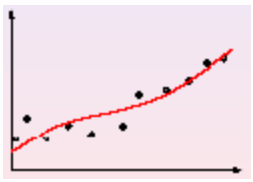
آموزش به صورت یک مساله ی جستجو

[1] Occam's Razor's Principle

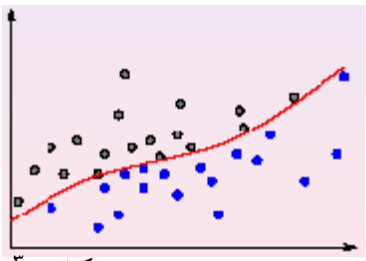


۲. انواع مسائل و راه حل ها

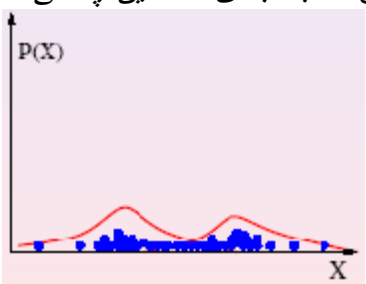
- انواع مسائل
  - سه نوع مساله وجود دارد
    - رگرسیون [1]



  - سه نوع مساله وجود دارد :
    - رگرسیون ، *طبقه بندی* [2]



    - رگرسیون ، طبقه بندی ، *تخمین چگالی* [3]



انواع آموزش

آموزش نظارت شده (تحت نظارت) :

---

[1] regression
[2] classification
[3] density estimation

- داده های آموزشی ، شامل رفتار مطلوب هستند
- (کلاس مطلوب ، نتیجه ، ...)

آموزش تقویتی

- داده های آموزشی ، شامل اهداف جزیی هستند
- (برای مثال ، به طور ساده ماشین ، چه به خوبی کار کند و چه به خوبی کار نکند)

آموزش نظارت نشده

- داده های آموزشی ، خام هستند ، کلاس یا هدف ، ارایه نشده است
- اغلب ، یک هدف پنهان در آن عمل وجود دارد (فشرده سازی ، بیشینه ی احتمال [1] و ...)

کاربرد

- پردازش منظره (تصور) [2]
  - آشکار سازی چهره [3] / بازبینی [4]
  - دست خط شناسی
- پردازش سخن
  - صوت (واج) / کلمه / جمله / شخص شناسی
- موارد دیگر
  - شاخص گذاری : گوگل ، استخراج متن [5] ، بازیابی اطلاعات
  - دارایی [6] : پیش گویی دارایی ، دارایی و مدیریت ریسک
  - مخابرات : پیش گویی ترافیک مخابراتی
  - استخراج داده ها : استفاده از مجموعه های داده ای بزرگ نگهداری شده توسط شرکت های بزرگ
  - بازی ها : تخته نرد ، go
  - کنترل : روبوت ها
- ... و تعداد زیادی از موارد دیگر !

محتوای زمینه

- نتایج نظری (تئوری)
  - *مبانی نظری* برای آموزش آماری چیستند ؟
  - چگونه ما می توانیم کارایی مورد انتظار مدل را *اندازه گیری* نماییم ؟
- نتایج مدلسازی (نمونه سازی)
  - مدل ها برای طبقه بندی ، رگرسیون ، توزیعات ، رشته ها ، شکل ها و غیره ، تخصص یافته اند
  - برای هر مدل ، ما باید یک *الگوریتم آموزشی* را طراحی نماییم
- موارد دیگر
  - نتایج مفید دیگر ، نظیر انتخاب ترکیب (چهره) ، اشتراک پارامتر و ...

---

[1] maximum likelihood : یک روش (متد) آماری برای تخمین پارامترهای جمعیتی (میانگین و واریانس) از داده های نمونه (برگرفته از لغتنامه ی Merriam-Webster نرم افزار بیبیلون)
[2] Vision Processing
[3] face detection
[4] verification
[5] text mining
[6] finance

مستندسازی

روزنامه ها و کنفرانس ها

- **روزنامه ها :**
  - روزنامه ی تحقیق آموزش ماشینی
  - محاسبه ی عصبی
  - تبادل های IEEE در شبکه های عصبی
  - تبادل های IEEE در تحلیل الگو و هوش ماشینی
- کنفرانس ها :
  - NIPS : اطلاعات عصبی سیستم های پردازشی
  - COLT : تئوری آموزش محاسبه ای
  - ICML : کنفرانس بین المللی آموزش ماشینی

منابع الکترونیکی

- موتورهای جستجو :
  - http://nips.djvuzone.org : NIPS online
  - http://citeseer.ist.psu.edu/ : Giteseer
  - پژوهشگر گوگل : http://scholar.google.com/
- کتابخانه های آموزش ماشینی :
  - *http://www.Torch.ch : Torch*
  - http://lush.sf.net : Lush

تئوری آموزش آماری

۱. داده ها ، توابع و ریسک

۲. ظرفیت

۳. متدلوژی (گفتار در روش و اسلوب)

۱. داده ها ، توابع و ریسک

داده ها

داده های آموزشی قابل دسترس

- تصور کنید $Z_1, Z_2, ..., Z_n$ یک نمونه ی تصادفی n- تایی از یک توزیع ناشناخته ی چگالی p(z) باشد .
- همه ی Z_i ها به صورت مستقل و توزیع شده به صورت یکنواخت می باشند [1]
- تصور کنید که D_n یک مثال خاص باشد = $(z_1, z_2, ..., z_n)$.

اشکال مختلف داده ها

- طبقه بندی : $Z = (X, Y) \in R^d \times \{-1,1\}$
  هدف : با داشتن یک x جدید ، P(Y|X=x) را تخمین بزنید
- رگرسیون : $Z = (X, Y) \in R^d \times R$

[1] Independently and Identically Distributed (IID)

هدف : با داشتن یک x جدید ، E[Y|X=x] را تخمین بزنید

- تخمین چگالی : $Z \in R^d$

هدف : با داشتن یک Z جدید ، p(z) را تخمین بزنید

فضای تابع

آموزش : جستجو برای یک تابع خوب در یک فضای تابع F

مثال های توابع $f(.;\theta) \in F$:

- رگرسیون : $\hat{y} = f(x;a,b) = a.x + b$
- طبقه بندی : $\hat{y} = f(x;a,b) = sign(a.x + b)$
- تخمین چگالی :

$$\hat{p}(z) = f(z;\mu,\sum) = \frac{1}{(2\pi)^{\frac{|x|}{2}}\sqrt{|\sum|}}\exp(-\frac{1}{2}(z-\mu)^T\sum^{-1}(z-\mu))$$

تابع اتلاف

آموزش : جستجو برای یک تابع خوب در یک فضای تابع F

مثال های تابع های اتلاف L:Z*F

- رگرسیون : $L(z,f) = L((x,y),f) = (f(x)-y)^2$
- طبقه بندی :
  - در صورتی که f(x)=y باشد L(z,f)=L((x,y),f) برابر صفر خواهد بود
  - و در غیر این صورت ، برابر یک خواهد بود
- تخمین چگالی : L(z,f)=-log p(z)

ریسک و ریسک تجربی

آموزش : جستجو برای یک تابع خوب در یک فضای تابع F

- ریسک مورد انتظار در F را کمینه نمایید ، که برای یک f داده شده به صورت زیر تعریف شده است :

$$R(f) = E_Z[L(z,f)] = \int_z L(z,f)p(z)dz$$

- اصل استقرا :
  - $f^* = \arg\min_{f\in F} R(f)$ را انتخاب نمایید
  - مساله ها : p(z) ناشناخته می باشد و ما به همه ی L(z,f) ها دسترسی نداریم !!!
- ریسک تجربی :

$$\hat{R}(f,D_n) = \frac{1}{n}\sum_{i=1}^{n} L(z_i,f)$$

ریسک تجربی

ریسک تجربی ، یک تخمین تحت تاثیر واقع نشده (بدون تعصب) از ریسک می باشد

$$E[\hat{R}(f,D)] = E[\frac{1}{n}\sum_{i=1}^{n} L(f,Z_i)] = \frac{1}{n}\sum_{i=1}^{n} E[L(f,Z_i)]$$

Z_i ها مستقل می باشند

$$= \frac{1}{n}\sum_{i=1}^{n} E[L(f,Z)]$$

Z_i ها به طور یکنواخت توزیع شده اند

$$= \frac{1}{n} nE[L(f,Z)] = R(f)$$

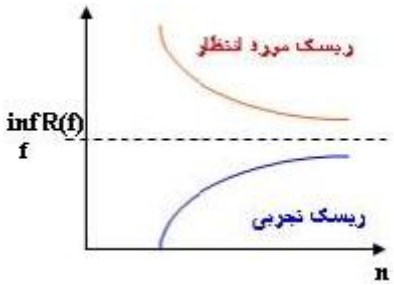
کمینه سازی ریسک تجربی [1]

اصل کمینه سازی ریسک تجربی : $f^*(D_n) = \arg\min_{f\in F} \hat{R}(f, D_n)$

سازگاری اصل ERM

- تصور کنید $f^* = \arg\min_{f\in F} R(f)$ و $f^*(D_n) = \arg\min_{f\in F} \hat{R}(f, D_n)$
- ما می گوییم که اصل ERM برای F ، سازگار است اگر

$$\hat{R}(f^*(D_n), D_n) \xrightarrow[n\to\infty]{p} R(f^*) \quad \text{و} \quad R(f^*(D_n)) \xrightarrow[n\to\infty]{p} R(f^*)$$



ریسک و خطای آموزشی

- خطای آموزشی :

$$\hat{R}(f^*(D_n), D_n) = \min_{f\in F} \hat{R}(f, D_n)$$

- *آیا خطای آموزشی یک تخمین تحت تاثیر قرار گرفته از ریسک می باشد ؟ بله .*

$$E[R(f^*(D_n)) - \hat{R}(f^*(D_n), D_n)] \geq 0$$

- راه حل $f^*(D_n)$ پیداشده با کمینه سازی خطای آموزشی ، با D_n بهتر از هر مجموعه ی دیگر D_n' به دست آمده از p(z) می باشد .

محدود کردن ریسک

آیا ما می توانیم تفاوت میان *خطای آموزشی* و *خطای تعمیم* را محدود نماییم ؟

$$| R(f^*(D_n)) - \hat{R}(f^*(D_n), D_n) | \leq ?$$

جواب : تحت شرایط معین F ، بله .
این شرایط ، وابسته به ظرفیت h ، F هستند .

ظرفیت

- ظرفیت h(F) ، سنجیدن اندازه اش و یا پیچیدگی اش می باشد .
- طبقه بندی :

[1] Empirical Risk Minimization (ERM)

ظرفیت h(F) ، بزرگ ترین n ای است که در یک مجموعه از نمونه های D_n وجود دارد و همیشه می تواند یک $f \in F$ که جواب صحیح را برای همه ی نمونه های در D_n ، برای هر برچسب ممکن را ارایه می کند ، پیدا کند .

- مثال : برای مجموعه ای از توابع خطی (y=w.x+b) با d بُعد (دیمانسیون) ، ظرفیت برابر است با d+1 .
- رگرسیون و تخمین چگالی : ظرفیت هم وجود دارد ، اما برای به دست آوردن ، به مراتب پیچیده تر می باشد (برای مثال ، ما همیشه می توانیم یک مساله ی رگرسیون را به یک مساله ی طبقه بندی کاهش دهیم) .

محدودکردن ریسک

براساس ریسک مورد انتظار ، محدود نمایید :

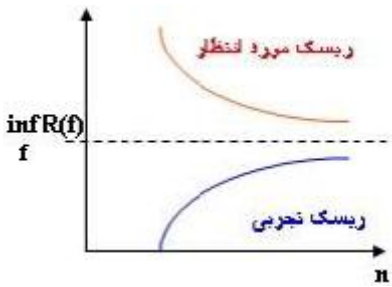
- قرار دهید : τ=sup L-inf L
- به ازای $\forall \eta$ ما داریم

$$P\left(\sup_{f \in F} | R(f) - \hat{R}(f, D_n) | \leq 2\tau \sqrt{\frac{h\left(\ln \frac{2n}{h} + 1 \right) - \ln \frac{\eta}{9}}{n}} \right) \geq 1 - \eta$$

- h ، ظرفیت F و n ، تعداد مثال های آموزشی در D_n می باشد

کمینه سازی ساختاری ریسک – با n ثابت

سازگاری – با h ثابت



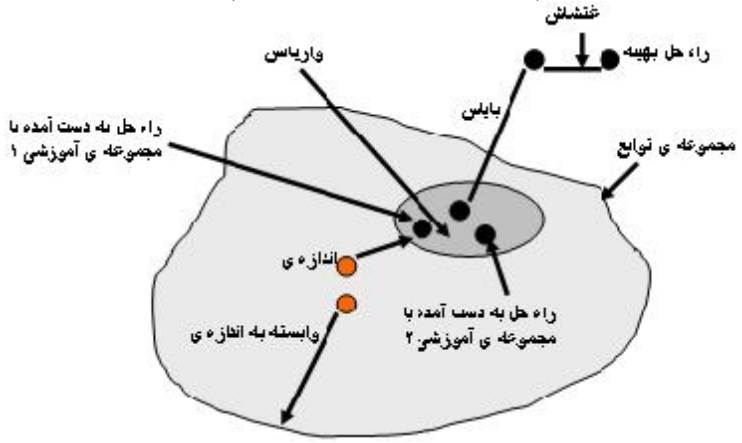
وضعیت دشوار بایاس - واریانس

خطا می تواند به سه قسمت تقسیم شود :

- بایاس : به علت این واقعیت که مجموعه ای از توابع F ، شامل راه حل بهینه نمی باشند ،

- واریانس : به علت این واقعیت که ما مجموعه ی دیگر D'_n به دست آمده از توزیع یکسان $p(Z)$ را استفاده کرده ایم و بنابراین ما باید یک راه حل متفاوتی را داشته باشیم ،
- اغتشاش (نوفه) : حتی راه حل بهینه می تواند اشتباه باشد (به عنوان مثال اگر برای یک x داده شده ، چند مقدار y وجود داشته باشد)

وضعیت دشوار بایاس – واریانس (نمایش به صورت گرافیکی)



وضعیت دشوار بایاس – واریانس (نمایش به صورت گرافیکی)

وضعیت دشوار بی واسطه : زمانی که ظرفیت h(F) بیش تر می شود ، بایاس کاهش می یابد ، اما واریانس ، افزایش می یابد !

تنظیم [1]

- ما دیدیم که آموزش = جستجو در یک مجموعه از توابع
- این مجموعه نباید خیلی کوچک باشد [2]
- این مجموعه نباید خیلی بزرگ باشد [3]
- یک راه حل : تنظیم
- توابع f را با توجه به دانش قبلی جریمه نمایید
- برای مثال ، توابعی که دارای پارامترهای بزرگ هستند را جریمه نمایید

$$f^*(D_n) = \arg\min_{f \in F} \hat{R}(f, D_n) + H(f)$$

- تابع H(f) تابعی است که با توجه به قبلی ، جریمه می کند

regularization [1]
underfitting [2]
overfitting [3]

- برای مثال ، در برخی از مدل ها :

پارامترهای کوچک ← راه حل های ساده تر ← ظرفیت کم تر

در ابتدا متوقف کردن [1]

- روش دیگری برای تنظیم ، در ابتدا متوقف کردن می باشد .
- در زمانی که آموزش ، یک *پردازه ی تکراری* می باشد کار می کند .
- به جای انتخاب تابعی که ریسک تجربی D_n را کمینه می کند ، ما می توانیم :
  - مجموعه ی آموزشی D_n تان را به دو قسمت تقسیم کنید
    - *مجموعه ی آموزشی* $D^{tr} = \{z_1, z_2, ..., z_{tr}\}$
    - *مجموعه ی صحت (اعتبار)* [2] $D^{va} = \{z_{va+1}, z_{tr+2}, ..., z_{tr+va}\}$
    - $Tr + va = n$
  - فرض کنید $f^t(D^{tr})$ تابع به تازگی پیدا شده در تکرار t باشد
  - فرض کنید $\hat{R}(f^t(D^{tr}), D^{va}) = \frac{1}{va} \sum_{z_i \in D^{va}} L(z_i, f^t(D^{tr}))$ باشد
  - آموزش را در تکرار t^* متوقف کنید که $t^* = \arg\min_t \hat{R}(f^t(D^{tr}), D^{va})$ و تابع برگشتی $f(D_n) = f^{t^*}(D^{tr})$

متدلوژی

- ابتدا : هدف را شناسایی نمایید ! هدف می تواند موارد زیر باشد
  ۱. برای ارایه ی بهترین مدل ، شما می توانید یک مجموعه ی آموزشی ارایه شده را به دست آورید ؟
  ۲. برای ارایه ی کارایی مورد انتظار یک مدل به دست آمده توسط کمینه سازی ریسک تجربی ، یک مجموعه ی آموزشی ارایه شده است ؟
  ۳. برای ارایه ی بهترین مدل و کارایی مورد انتظار ، شما می توانید یک مجموعه ی آموزشی ارایه شده را به دست آورید ؟
- در صورتی که هدف ، (۱) می باشد : *انتخاب مدل* را انجام دهید
- در صورتی که هدف ، (۲) می باشد ، شما نیاز به تخمین ریسک دارید
- اگر هدف (۳) مورد نظر باشد ، شما به انجام هر دو مورد فوق نیاز دارید
- روش های مختلفی وجود دارد که می تواند یا برای تخمین ریسک استفاده کنید یا برای انتخاب مدل :
  - صحت (اعتباریابی) ساده
  - اعتباریابی متقاطع (k- لا [3] ، حذف کردن [4])

انتخاب – تایید اعتبار (صحت) مدل

- خانواده ای از توابع با پارامتر فوق العاده [5] θ را انتخاب نمایید
- مجموعه ی آموزشی D_n تان را به دو قسمت تقسیم نمایید

---

[1] Early Stopping
[2] Validation set
[3] k-fold
[4] Leave-one-out
[5] Hyper-terminal

- $D^{tr} = \{z_1, z_2, ..., z_{tr}\}$
- $D^{va} = \{z_{tr+1}, z_{tr+2}, ..., z_{tr+va}\}$
- tr+va=n

- برای هر مقدار θ_m پارامتر فوق العاده ی θ
  - $f^*_{\theta_m}(D^{tr}) = \arg\min_{f \in F_{\theta_m}} \hat{R}(f, D^{tr})$ را انتخاب نمایید
  - $\hat{R}(f^*_{\theta_m}, D^{va}) = \frac{1}{va}\sum_{z_i \in D^{va}} L(z_i, f^*_{\theta_m}(D^{tr}))$ را با $R(f^*_{\theta_m})$ تخمین بزنید
- $\theta^*_m = \arg\min_{\theta_m} R(f^*_{\theta_m})$ را انتخاب نمایید
- $f^*(D_n) = \arg\min_{f \in F_{\theta^* m}} \hat{R}(f, D_n)$ را برگردان

انتخاب مدل - وارسی اعتبار [1]

- خانواده ای از توابع با پارامتر فوق العاده ی θ را انتخاب نمایید
- مجموعه ی آموزشی D_n تان را به K قطعه ی مساوی و متمایز $D^1, ..., D^k, ..., D^K$ تقسیم نمایید
- برای هر مقدار θ_m پارامتر فوق العاده ی θ
  - برای هر قطعه ی D^k (و همتای $\overline{D}^k$ ی آن)
    - $f^*_{\theta_m}(\overline{D}^k) = \arg\min_{f \in F_{\theta_m}} \hat{R}(f, \overline{D}^k)$ را انتخاب نمایید
    - $R(f^*_{\theta_m}(\overline{D}^k))$ را با $\hat{R}(f^*_{\theta_m}(\overline{D}^k), D^k) = \frac{1}{|D^k|}\sum_{x_i \in D^k} L(z_i, f^*_{\theta_m}(\overline{D}^k))$ تخمین بزنید
  - $R(f^*_{\theta_m}(D_n))$ را با $\frac{1}{K}\sum_k R(f^*_{\theta_m}(D^k))$ تخمین بزنید
- $\theta^*_m = \arg\min_{\theta_m} R(f^*_{\theta_m}(D))$ انتخاب نمایید
- $f^*(D_n) = \arg\min_{f \in F_{\theta^*_m}} \hat{R}(f, D_n)$ را برگردان

تخمین ریسک تایید اعتبار

- مجموعه ی آموزشی D_n تان را به دو قسمت تقسیم نمایید

[1] cross-validation

- $D^{tr} = \{z_1, z_2, ..., z_{tr}\}$
- $D^{te} = \{z_{tr+1}, z_{tr+2}, ..., z_{tr+te}\}$
- tr + te = n
- $f^*(D^{tr}) = \arg\min_{f \in F} \hat{R}(f, D^{tr})$

(این پردازش بهینه سازی ، می تواند شامل انتخاب مدل باشد)

- $R(f^*(D^{tr}))$ را بـــا $\hat{R}(f^*(D^{tr}), D^{te}) = \frac{1}{te} \sum_{z_i \in D^{te}} L(z_i, f^*(D^{tr}))$ تخمین بزنید

تخمین ریسک - وارسی اعتبار

- مجموعـه ی آموزشـی D_n تـان را بـه K قسمت متمـایز و برابـر $D^1, ..., D^k, ..., D^K$ تقسیم نمایید
- برای هر قسمت D^k
  - تصور نمایید D^k مجموعه ای از نمونه هایی که در D_n هستند ولی در D^k نمی باشند ، باشد
  - $f^*(D^k) = \arg\min_{f \in F} \hat{R}(f, D^k)$ را انتخاب نمایید

این پروسه ممکن است شامل انتخاب مدل باشد

  - $R(f^*(\overline{D}^k))$ را بـا $\hat{R}(f^*(D^k), D^k) = \frac{1}{|D^k|} \sum_{z_i \in D^k} L(z_i, f^*(D^k))$

تخمین بزنید

- $R(f^*(D_n))$ را با $\frac{1}{K} \sum_k R(f^*(\overline{D}^k))$ تخمین بزنید
- زمانی که k=n بود : وارسی اعتبار را ترک نمایید

تخمین ریسک و انتخاب مدل

- در زمانی که شما هر دوی بهترین مدل و ریسک مورد انتظارش را می خواهید
- شما به *ادغام* روش هایی که قبلا ارایه شده اند نیاز دارید . برای مثال :
  - تست تایید اعتبار آموزشی : سه مجموعه ی داده ای مجزا ، لازم هستند
  - وارسی اعتبار + تست : وارسی اعتبار در مجموعه ی آموزشی و سپس تست در مجموعه ی مجزا
  - وارسی اعتبار مضاعف : برای هر زیر مجموعه ، به انجام یک وارسی اعتبار با K-1 زیر مجموعه ی دیگر داریم
- دیگر صورت های متدولوژیکی مهم :
  - نتایج خود را با روش های دیگر ، *مقایسه نمایید* !!!!
  - از تست های آماری برای *بازنگری اهمیت* ، استفاده نمایید
  - مدل خود را در بیش تر از یک مجموعه ی داده ای ، بازنگری نمایید

تست اعتبار آموزشی

- خانواده ای از توابع با پارامتر فوق العاده ی θ را انتخاب نمایید

- مجموعه ی آموزشی D_n تان را به سه قسمت D^{tr} ، D^{va} و D^{te} تقسیم نمایید
- برای هر مقدار θ_m پارامتر فوق العاده ی θ
  - $f^*_{\theta_m}(D^{tr}) = \arg\min_{f \in F_{\theta_m}} \hat{R}(f, D^{tr})$ را انتخاب نمایید
  - قرار دهید $\hat{R}(f^*_{\theta_m}(D^{tr}), D^{va}) = \frac{1}{va}\sum_{z_i \in D^{va}} L(z_i, f^*_{\theta_m}(D^{tr}))$
- انتخاب نمایید $\theta^*_m = \arg\min_{\theta_m} \hat{R}(f^*_{\theta_m}(D^{tr}), D^{va})$
- انتخاب نمایید $f^*(D^{tr} \cup D^{va}) = \arg\min_{f \in F_{\theta^*_m}} \hat{R}(f, D^{tr} \cup D^{va})$
- $R(f^*(D^{tr} \cup D^{va}))$ را با $\frac{1}{te}\sum_{z_i \in D^{te}} L(z_i, f^*(D^{tr} \cup D^{va}))$ تخمین بزنید

وارسی اعتبار + تست

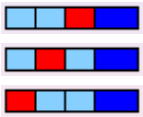
- خانواده ای از توابع با پارامتر فوق العاده ی θ را انتخاب نمایید
- مجموعه ی داده ای D_n را به دو قسمت تقسیم نمایید
  یک مجموعه ی آموزشی D^{tr} و یک مجموعه ی تستی D^{te}
- برای هر مقدار θ_m پارامتر فوق العاده ی θ
  $R(f^*_{\theta_m}(D^{tr}))$ را با D^{tr} استفاده کننده از وارسی اعتبار ، تخمین بزنید
- $\theta^*_m = \arg\min_{\theta_m} R(f^*_{\theta_m}(D^{tr}))$ انتخاب نمایید
- *آموزش دوباره* [1] $f^*(D^{tr}) = \arg\min_{f \in F_{\theta^*_m}} \hat{R}(f, D^{tr})$
- $R(f^*(D^{tr}))$ را با $\frac{1}{te}\sum_{z_i \in D^{ta}} L(z_i, f^*(D^{tr}))$ تخمین بزنید

وارسی اعتبار مضاعف

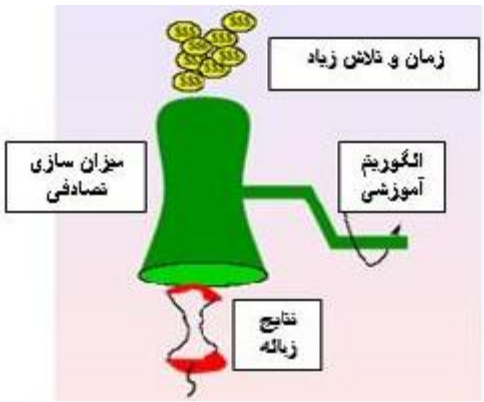
- یک خانواده از توابع با پارامتر فوق العاده ی θ را انتخاب نمایید
- مجموعه ی آموزشی D_n تان را به K قسمت برابر و متمایز $D^1,...,D^k,...,D^K$ تقسیم نمایید
- برای هر قطعه ی D^k
  - بهترین مدل $f^*(\overline{D}^k)$ را با وارسی اعتبار در $\overline{D}^k$ انتخاب نمایید
  - $R(f^*(\overline{D}^k))$ را با $\hat{R}(f^*(D^k), D^k) = \frac{1}{|D^k|}\sum_{z_i \in D^k} L(z_i, f^*(D^k))$ تخمین بزنید

[1] retrain

- $R(f^*(D))$ را با $\frac{1}{K}\sum_k R(f^*(\overline{D}^k))$
- *نکته :* این پروسه ، تنها یک تخمین از ریسک را ارایه می دهد ، اما یک مدل نمی باشد . در صورتی که شما به مدل هم نیازمند هستید ، شما باید یک پروسه ی انتخاب مدل مجزا را انجام دهید !

همه ی مجموعه ی داده ای به ۳ قسمت تقسیم می شود	
دو قطعه ی اول به ۳ قسمت تقسیم می شود ، سپس یک وارسی اعتبار ۳ – لا برای انتخاب بهترین پارامتر فوق العاده انجام می شود	
بهترین پارامتر فوق العاده برای آموزش مجدد در ۲ قطعه ی اصلی و تست بر روی قطعه ی دیگر ، استفاده می شود	
... برای هر قطعه کار های مشابه آن چه که در بالا گفته شد را برای تخمین ریسک انجام دهید	

از جادوی آموزش ماشینی برحذر باشید



مدل های کلاسیکی

۱. پیشینه نماها یا سابقه نماها یا هیستوگرام ها [1] ، K نزدیک ترین همسایه و پنجره های پارزن [2]
۲. بیشینه ی احتمال و تصمیم گیری بیزی
۳. K- میانگین
۴. رگرسیون خطی

پارامتری یا نه ؟

- اغلب مشخص شده است که فضای F ، پارامتری است یا نه .
- **پارامتری :** فضا خیلی کوچک می باشد و توسط تعداد کمی پارامتر ، مشخص شده است .
  - مثال ها : یک توزیع گاوسی یا یک تابع خطی
- **غیرپارامتری** : فضا ، نامحدود می باشد ، فقط توسط داده های آموزشی ، محدود شده است
  - **مثال ها :** K نزدیک ترین همسایه
- **نیم پارامتری :**
  - **مثال ها :** بیش تر الگوریتم های آموزش ماشینی !

۱. هیستوگرام ها ، K نزدیک ترین همسایه و پنجره های پارزن

هیستوگرام ها – مثال

- برای طبقه بندی یا رگرسیون : z = (x,y)
- فرض نمایید که x ، یک بردار k- بعدی باشد
- برای هر بعد d ، مقدارهای ممکن x_d را به m_d قسمت تقسیم نمایید
- مثال ، برای ۱۴ نمونه ی آموزشی با ورودی بعد (دیمانسیون) ۲ :

	$x_1 < 5$	$5 \le x_1 < 7$	$7 \le x_1$
$x_2 = red$	y(0)=-3 y(1)=-4 y(2)=-2.8	y(3)=2 y(4)=1	y(5)=3 y(6)=4 y(7)=2.8 y(8)=2.5
$x_2 = blue$	y(9)=-4.5 y(10)=-4	y(11)=0.1	y(12)=0.1 y(13)=0.65

هیستوگرام ها - آموزش

- **مدل :** مقدار میانگین $\hat{y}$ مطابق با هر قسمت را محاسبه نمایید (در مجموعه ی آموزشی) :

	$x_1 < 5$	$5 \le x_1 < 7$	$7 \le x_1$
$x_2 = red$	$\hat{y} = -3.3$	$\hat{y} = 1.5$	$\hat{y} = 3.1$
$x_2 = blue$	$\hat{y} = 4.25$	$\hat{y} = 0.1$	$\hat{y} = 0.38$

[1] histograms
[2] Parzen Windows

- **تست :** با داشتن یک نمونه ی جدید x ، قسمت مطابق و خروجی وابسته به $\hat{y}$ را انتخاب نمایید
- می تواند برای طبقه بندی ، توسعه داده شود .
- ظرفیت ، توسط مجموع تعداد قطعه ها ، کنترل شده است .
- مجموع تعداد قطعه ها $= \prod_{d=1}^{k} m_d$

مساله : بدی بعد (دیمانسیون)

انفجار ترکیبی

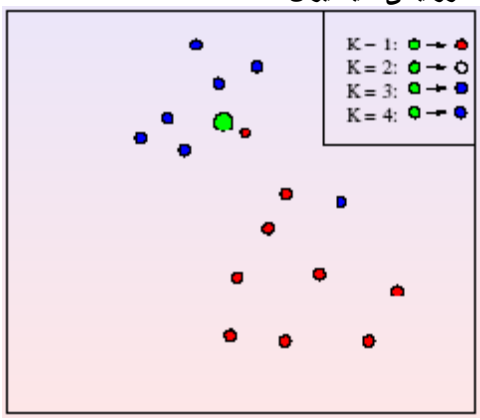
- زمانی که تعداد ابعاد ورودی ، افزایش می یابد چه اتفاقی می افتد ؟
- تعداد قسمت ها به صورت نمایی سریع تر افزایش می یابد !
- بیش تر قسمت ها هیچ نمونه ی مثال آموزشی را به دست نمی آورند
- چگونه ما می توانیم یک نمونه ی جدید را که در یکی از آن قسمت ها است تخمین بزنیم ؟؟؟؟
- در واقع ، حتی قسمت های با برخی نمونه های آموزشی هم احتمالا به درستی تخمین زده نمی شوند ...

K نزدیک ترین همسایه [1]

- یک روش خیلی ساده هیچ آموزشی نیاز نیست
- احتیاجات :
  - یک مجموعه ی آموزشی $D_n = \{z_1, z_2, ..., z_n\}$ با $z_i = (x_i, y_i)$
  - یک تابع فاصله $L(x_1, x_2)$. برای مثال ، $(x_1 - x_2)^2$
  - یک پارامتر K
- برای هر نقطه ی تستی x
  - در D_n ، K نمونه که با توجه به $L(x, x_i)$ به x ، نزدیک ترین می باشند و اندیس شان را (از D_n) در $\{s_1, ..., s_K\}$ نگهداری می کنند ، انتخاب نمایید
  - تصمیم گیری :
    - رگرسیون : $\hat{y} = \frac{1}{K} \sum_{i=1}^{K} y_{s_i}$
    - طبقه بندی : $\hat{y} = sign\left(\frac{1}{K} \sum_{i=1}^{K} y_{s_i} \right)$
- ظرفیت ، توسط K کنترل شده است

K – نزدیک ترین همسایه (دید تصویری)

[1] K-NN



برخی توضیحات

- نزدیک ترین بودن به یک نمونه به چه معنی می باشد ؟
- اغلب از سیستم متریک (متری) استفاده شده است : فاصله ی اقلیدسی [1] ، یا l^2- نورم

$$d = \sqrt{\sum_i (x_i - t_i)^2}$$

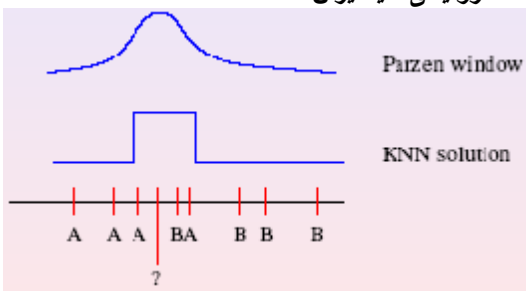
- برای K- نزدیک ترین همسایه $\sqrt{\ }$ لازم نمی باشد
- K را چگونه انتخاب نماییم ؟؟؟
- یادآوری : K ، ظرفیت را کنترل می کند ...
- بنابراین ما می توانیم از روش *انتخاب مدل* استفاده نماییم

فاصله ها و بدی دیمانسیون (بعد)

- به یک شبکه ی منظم با b قسمت در هر بعد در یک فرا مکعب [2] d – بعدی توجه نمایید .
  ما دارای b^d قسمت هستیم .
- چه تعداد از قسمت ها روی سطح فرا مکعب نمی باشند ؟
  به داده های یکسان توجه نمایید
  $\left(\frac{b-2}{b}\right)^d$ شانس بودن در مرکز وجود دارد . $\longleftarrow 0$
- *وقتی که d بزرگ باشد ، همه ی داده ها روی سطح قرار می گیرند !*
- چه تعداد از نقاط مجموعه ی آموزشی می توانند روی سطح یکسان باشند ؟ از آنجایی که d افزایش می یابد ، به طور میانگین ، کم تر از یکی
- بنابراین هر نقطه ، دورتر از نقاط دیگر می باشد
- بنابراین ، همه ی روش های براساس *فاصله ی اقلیدسی* فقط به کار بر روی ابعاد کوچک محدود شده اند

K – نزدیک ترین همسایه در مقابل پنجره های پارزن

[1] Euclidean distance
[2] hypercube = مکعبی با بیش از سه بعد



پنجره های پارزن

- روشی بسیار ساده ، به آموزش نیاز ندارد
- احتیاجات :
  - یک مجموعه ی آموزشی $D_n = \{z_1, z_2, ..., z_n\}$ با $z_i = (x_i, y_i)$
  - یک تابع هسته (شالوده)[1] $K(x_1, x_2)$. برای مثال ، $\exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$
- برای هر نقطه ی تستی x (یا z برای تخمین چگالی)
  - تصمیم گیری :
    - رگرسیون : $\hat{y}_r = \frac{\sum_{i=1}^{n} y_i K(x, x_i)}{\sum_{i=1}^{n} K(x, x_i)}$
    - طبقه بندی : $\hat{y} = sign(\hat{y}_r)$
    - تخمین چگالی : $\hat{p}(z) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} K(x, x_i)$
- ظرفیت توسط σ کنترل شده است

۲. بیشینه ی احتمال و تصمیم گیری بیزی

بیشینه ی احتمال برای تخمین چگالی

- با داشتن یک مجموعه از نمونه های $D_n = \{z_1, z_2, ..., z_n\}$
- هدف : پیداکردن یک توزیع p(z) که احتمال داده های آینده را بیشینه می کند
- یک مجموعه از توزیع های p(z|θ) را با پارامترهای θ انتخاب نمایید .
- احتمال D_n (همه ی نمونه ها iid هستند) :

$$L(D_n \mid \theta) = \prod_{i=1}^{n} p(z_i \mid \theta)$$

با این وجود ما جستجو می کنیم برای :

$$\theta^* = \arg\max_{\theta} \prod_{i=1}^{n} p(z_i \mid \theta) = \arg\max_{\theta} \sum_{i=1}^{n} \log(p(z_i \mid \theta))$$

[1] kernel function = توابعی ساده (معمولا گاوسی) که به هم اضافه شده اند و در نقاط داده ای شناخته شده قرار دارند ، برای تخمین یک توزیع نمونه (برگرفته از لغتنامه ویکیپدیا)

بیشینه ی احتمال برای گاوسی ها

- خانواده ی گاوسی های یک بعدی با $\theta = \{\mu, \sigma\}$

$$\hat{p}(z \mid \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2} \right)$$

- بنابراین ، احتمال لگاریتمی برای یک مجموعه از n داده برابر است با :

$$l = \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(z_i-\mu)^2}{2\sigma^2} \right)$$

- برای پیداکردن راه حل با بیشینه ی احتمال ، ما باید قرار دهیم

$$\frac{\partial l}{\partial \theta} = 0$$

برای پارامترهای $\theta = \{\mu, \sigma\}$

راه حل - روش با بیشینه ی احتمال

$$l = \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(z_i-\mu)^2}{2\sigma^2} \right)$$

$$= -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \sum_{i=1}^{n} \frac{(z_i-\mu)^2}{2\sigma^2}$$

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^{n} \frac{z_i - \mu}{\sigma^2} = \sum_{i=1}^{n} \frac{z_i}{\sigma^2} - \sum_{i=1}^{n} \frac{\mu}{\sigma^2}$$

$$0 = \sum_{i=1}^{n} \frac{z_i}{\sigma^2} - \frac{n\mu}{\sigma^2}$$

$$\Rightarrow \mu = \frac{1}{n} \sum_{i=1}^{n} z_i$$

راه حل با بیشینه ی احتمال – واریانس ها

$$l = \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(z_i-\mu)^2}{2\sigma^2} \right)$$

$$= -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \sum_{i=1}^{n} \frac{(z_i-\mu)^2}{2\sigma^2}$$

$$\frac{\partial l}{\partial \sigma^2} = -\frac{n}{2}\frac{1}{\sigma^2} + \sum_{i=1}^{n} \frac{(z_i-\mu)^2}{2\sigma^4} = 0$$

$$\Rightarrow \sum_{i=1}^{n} \frac{(z_i-\mu)^2}{2\sigma^4} = \frac{n}{2\sigma^2}$$

$$\frac{1}{n} \sum_{i=1}^{n} (z_i-\mu)^2 = \sigma^2$$

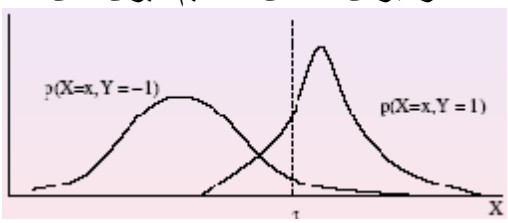
تصمیم گیری بیزی

- طبقه بندی : $z=(x,y)\in R^d \times\{-1,1\}$
- با داشتن : توزیع درست قبلی P(Y=y|X=x)
- می توان نشان داد که تصمیم گیری $\hat{y}=\arg\max_{i\in\{-1,1\}} P(Y=i\mid X=x)$ در موردی که تعداد خطاهای طبقه بندی را کمینه می کند ، بهینه می باشد .
- این تصمیم گیری برای *طبقه بندی بیشینه ی یک معیار تجربی* [1] ، مناسب می باشد

چرا طبقه بندی MAP ، خطا را کمینه می نماید ؟

$$\hat{y}=\arg\max_{i\in\{1,-1\}} P(Y=i\mid X=x)$$
$$=\arg\max_{i\in\{1,-1\}} \frac{p(X=x\mid Y=i).P(Y=i)}{p(X=x)}$$
$$=\arg\max_{i\in\{1,-1\}} p(X=x\mid Y=i).P(Y=i)$$
$$=\arg\max_{i\in\{1,-1\}} p(X=x,Y=i)$$

- اجازه دهید که یک آستانه را برای همه ی تصمیم گیری های X=τ مان ، انتخاب نماییم .



- نسبت خطاها می تواند به دو عبارت تقسیم شود :
  - در زمانی که X>τ باشد و Y=-1 :

$$p(X>\tau,Y=-1)=\int_{x>\tau} p(X=x,Y=-1)dx$$

  - در زمانی که X<τ و Y=1 باشد

$$=p(X<\tau,Y=1)=\int_{x<\tau} p(X=x,Y=1)dx$$

- کدام τ برای کمینه کردن خطا مناسب می باشد ؟

$$\tau^*=\arg\min_{\tau} p(X>\tau,Y=-1)+p(X<\tau,Y=1)$$

- که دقیقا در زمانی اتفاق می افتد که

$$p(X=\tau,Y=-1)=p(X=\tau,Y=1)$$

طبقه بندی کننده های بیزی

- هدف : براساس معیار MAP ، تصمیم گیری نمایید :

$$\hat{y}=\arg\max_{i\in\{1,-1\}} p(X=x\mid Y=i).P(Y=i)$$

- بنابراین شما به تخمین موارد زیر نیاز دارید :

[1] Maximum A Posteriori criterion (MAP)

- چگالی شرطی p(X=x|Y=i) برای هر کلاس i
- کلاس قبلی P(Y=i) برای هر کلاس i

- مزیت : هر کلاس به صورت مستقل تخمین زده می شود
- بدی : شما ارتباط های غیر لازم را یاد می گیرید
- با این وجود این روش اغلب در *پردازش سخن* استفاده می شود

طبقه بندی کننده های ساده ی بیزی

- طبقه بندی تصمیم گیری با توجه به یک طبقه بندی کننده ی بیزی :

$$\hat{y} = \arg\max_{i\in\{1,-1\}} p(X = x \mid Y = i).P(Y = i)$$

- P(Y=i) می تواند با شمارش در مجموعه ی آموزشی ، تخمین زده شود .
- ما به راهی برای ارایه ی p(X=x|Y=i) نیاز داریم .
- بیایید تصور کنیم که $X \in R^d$ و همه ی X_j ها مستقل می باشند
- بنابراین مدل *ساده ی بیزی* فرض می کند :

$$p(X = x \mid Y = i) = p(X_1 = x_1,..., X_d = x_d \mid Y = i) = \prod_{j=1}^{d} p(X_j = x_j \mid Y = i)$$

- بنابراین ، طبقه بندی کننده ی ساده ی بیزی می شود :

$$\hat{y} = \arg\max_{i\in\{1,-1\}} P(Y = i).\prod_{j=1}^{d} p(X_j = x_j \mid Y = i)$$

۳. K - میانگین

دسته بندی توسط K - میانگین

- با داشتن یک مجموعه از نمونه های $D_n = \{z_1, z_2,..., z_n\}$
- به دنبال K نمونه ی اولیه μ_k زیر مجموعه های گسسته ی S_k از D_n برای کمینه سازی بگردید

$$L = \sum_{k=1}^{K} \sum_{j\in S_k} \left\| z_j - \mu_k \right\|^2$$

که μ_k ، میانگین نمونه های درون زیر مجموعه ی S_k می باشد :

$$\mu_k = \frac{1}{|S_k|} \sum_{j\in S_k} z_j$$

- ما می توانیم از هر فاصله ای و نه فقط فاصله ی اقلیدسی ، استفاده نماییم

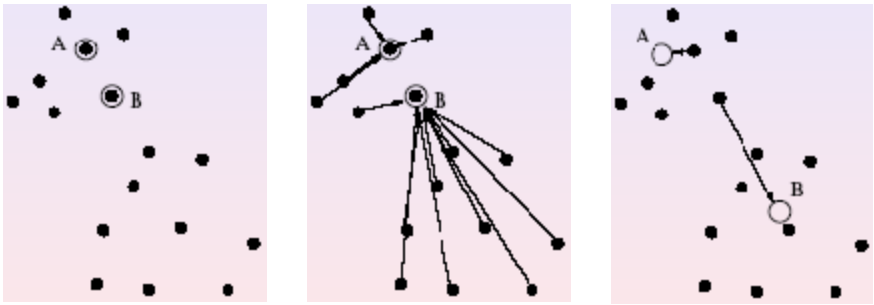
K – میانگین دسته ای و تصادفی

- مقدار دهی اولیه : به صورت تصادفی K نمونه z_j در D_n را به صورت مقدارهای اولیه ی هر μ_k انتخاب نمایید
- در هر تکرار دسته ای :

- برای هر نمونه ی اولیه μ_k ، در مجموعه ی خالی S_k ، نمونه های D_n که نزدیک تر به μ_k نسبت به سایر $\mu_k \neq k$ هستند را قرار دهید .
- مقدار هر μ_k به صورت میانگین نمونه های درون S_k را مجددا محاسبه نمایید .

- الگوریتم زمانی که دیگر هیچ نمونه ی اولیه ای تغییر نکند ، متوقف می شود .
- این می تواند نشان دهنده ی این باشد که معیار K – میانگین هرگز افزایش نخواهد یافت .
- یک نوع تصادفی K – میانگین نیز می تواند به دست آید : با داشتن یک η ی کوچک ، برای هر نمونه ، z_j به نزدیک ترین μ_k به صورت زیر تغییر می یابد :

$$\mu_k = \mu_k + \eta(z_j - \mu_k)$$



K – میانگین (دید تصویری) ۱ **K – میانگین (دید تصویری) ۲** **K – میانگین (دید تصویری) ۳**

انحراف K - میانگین

- تصور نمایید که μ^t مجموعه ای از گروه ها در زمان t باشد
- $s(z_i, \mu^t) = \arg\min_k \left\| z_i - \mu_k^t \right\|^2$ را بهترین گروه در μ^t برای z_i قرار دهید .
- بیایید $L(\mu^t) = \sum_{i=1}^{n} \left\| z_i - \mu^t_{s(z_i,\mu^t)} \right\|$ را بازنویسی نماییم
- ما می خواهیم نشان دهیم که

$$L(\mu^{t+1}) - L(\mu^t) \leq 0$$

- قرار دهید $\mu_k^{t+1} = \frac{1}{|S_k|} \sum_{i \in S_k} z_i$ ، که S_k مجموعه ی z_i انتساب داده شده به μ_k می باشد
- قرار دهید $Q(\mu^{t+1}, \mu^t) = \sum_{i=1}^{n} \left\| z_i - \mu^{t+1}_{s(z_i,\mu^t)} \right\|$

$$L(\mu^{t+1}) - L(\mu^t) = L(\mu^{t+1}) - Q(\mu^{t+1}, \mu^t) + Q(\mu^{t+1}, \mu^t) - L(\mu^t)$$

$$L(\mu^{t+1}) - Q(\mu^{t+1}, \mu^t) = \sum_{i=1}^{n} \left(\left\| z_i - \mu^{t+1}_{s(z_i,\mu^{t+1})} \right\|^2 - \left\| z_i - \mu^{t+1}_{s(z_i,\mu^t)} \right\|^2 \right) \leq 0$$

$$Q(\mu^{t+1}, \mu^t) - L(\mu^t) = \sum_{i=1}^{n} \left(\left\| z_i - \mu^{t+1}_{s(z_i,\mu^t)} \right\|^2 - \left\| z_i - \mu^{t}_{s(z_i,\mu^t)} \right\|^2 \right) \leq 0$$

$$\Rightarrow L(\mu^{t+1}) - L(\mu^t) \leq 0$$

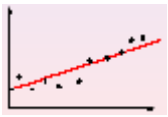
K – میانگین – برخی توضیحات

- با توجه به KNN ، ما می توانیم متریک را تغییر دهیم
- برای مثال ، ما می توانیم داده ها را نرمال سازی نماییم
- چگونه K را انتخاب نماییم ؟؟؟
- یادآوری : با توجه به KNN ، K ظرفیت ... را کنترل می کند
- بنابراین ، ما می توانیم از یک روش انتخاب مدل ، استفاده نماییم
- نکته : K – میانگین برای مقدار دهی اولیه حساس است . دیگر مکاشفه ها وجود دارد یا شما می توانید چند بار آموزش مجدد بدهید
- کاربرد : *استخراج خصیصه* [1]

هر نمونه ی z را توسط اندیس نزدیک ترین نمونه ی اولیه ، ارایه می نماید

۴. رگرسیون خطی

- ما یک مجموعه از نمونه های آموزشی $D_n = \{z_1, z_2, ..., z_n\}$ را داریم
- با $z_i = (x_i, y_i) \in R^d \times R$
- فضای تابع خطی : $\hat{y} = w.x + b$ با پارامترهای (w,b)
- تابع اتلاف : $L(y, \hat{y}) = (y - \hat{y})^2$



حل رگرسیون خطی

- مجموع خطا به صورت زیر می باشد :

$$C = \sum_i L(y, \hat{y}) = \sum_i (y_i - \hat{y})^2 = \sum_i (y - w.x_i - b)^2$$

- ما نیاز داریم که همزمان $\frac{\partial C}{\partial w}$ و $\frac{\partial C}{\partial b}$ را برابر صفر قرار دهیم
- برای اقتباس ریاضی ساده تر از نمایش ماتریسی استفاده می کنیم

حل رگرسیون خطی با ماتریس وارون

- $r_i = [x_i \; 1]$ را بردار ورودی نمونه ی i تقویت شده توسط مقدار ۱ باشد .
- فرض کنید R ، (n*(d+1)) ماتریس بردارهای r_i باشد .
- تصور کنید که Y ؛ ماتریس مقصد (n*1) باشد .
- v=[w b] ، (d+1) – بعدی ، بردار متصل کننده ی w و b باشد .
- مجموع هزینه برابر است با :

$$C = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y - (w.x_i + b.1))^2 = (Y - Rv)'(Y - Rv)$$

راه حل مساله ی رگرسیون خطی

- هزینه :

$$C=(Y-Rv)'(Y-Rv)$$

[1] feature extraction

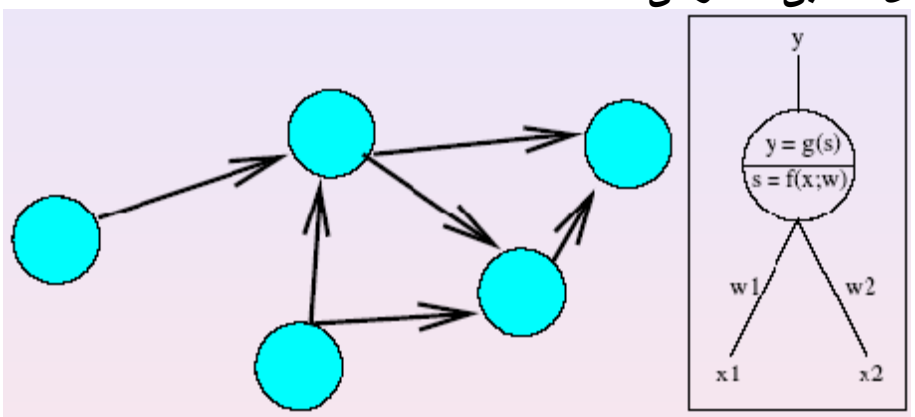
- کمینه ی آن باید $\frac{\partial C}{\partial v} = 0$ را برآورده کند
- اجازه بدهید که $\frac{\partial C}{\partial v} = -2R'(Y - Rv) = 0$ را حل نماییم

بنابراین : $\hat{v} = (R'R)^{-1}R'Y$

پرسپترون های چندلایه ای

۱. کلیات

شبکه های عصبی مصنوعی [1]



- یک ANN ، یک مجموعه از قسمت ها (نورون ها) ی متصل شده به هم می باشد
- هر قسمت ممکن است دارای چند ورودی باشد اما دارای یک خروجی باشد
- هر قسمت دارای دو تابع می باشد :
  - مجتمع سازی (انتگرال گیری) : $s=f(x;\theta)$
  - انتقال : $y=g(s)$

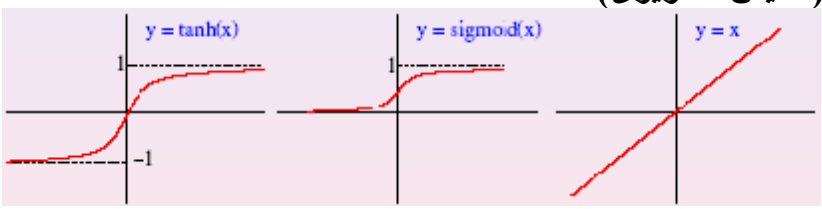
شبکه های عصبی مصنوعی : توابع

- مثال تابع مجتمع سازی : $s = \theta_0 + \sum_i x_i.\theta_i$
- مثال های تابع های انتقال :
  - tanh : $y=\tanh(s)$
  - sigmoid : $y=\frac{1}{1+\exp(-s)}$

[1] Artificial Neural Network (ANN)

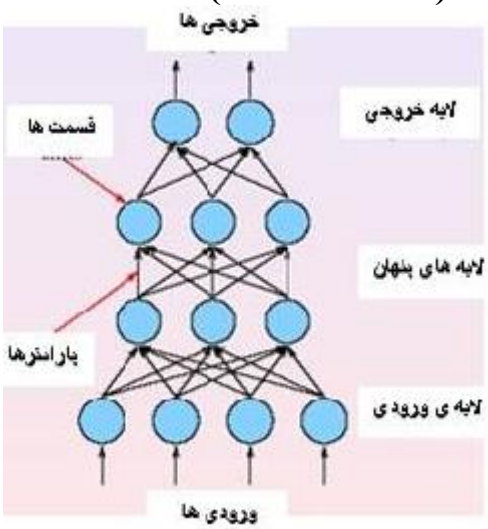
- برخی از قسمت ها ورودی ها را از دنیای خارج دریافت می کنند .
- برخی از قسمت ها خروجی ها را برای دنیای خارج به وجود می آورند
- سایر قسمت ها ، اغلب *پنهان* نامیده می شوند .
- بنابراین ، از خارج ، یک شبکه ی عصبی مصنوعی می تواند به صورت یک *تابع* دیده شود .
- انواع مختلفی از شبکه های عصبی مصنوعی وجود دارند . رایج ترین آن ها پرسپترون چند لایه ای می باشد [1] .

توابع انتقال (نمایش تصویری)



۲. پرسپترون های چندلایه ای

پرسپترون های چندلایه ای (نمایش تصویری)



پرسپترون های چندلایه ای

- یک MLP ، یک تابع می باشد : $\hat{y} = MLP(x;\theta)$
- پارامترها : $\theta = \{w'_{i,j}, b'_i \quad : \quad \forall i, j, l\}$
- از حالا تصور کنید که $x_i(p)$ مقدار i^{th} در نمونه ی p^{th} ارایه شده توسط بردار x(p) باشد (و در موقع ممکن ، اجازه دهید که p را رها نماییم) .
- هر لایه ی l ($1 \le l \le M$) کاملا به لایه ی قبلی متصل شده است
- مجتمع سازی : $s'_i = b'_i + \sum_j y'^{-1}_j . w'_{i,j}$
- انتقال : $y'_i = \tanh(s'_i)$ یا $y'_i = sigmoid(s'_i)$ یا $y'_i = s'_i$
- خروجی لایه ی صفر (اول) شامل ورودی های $y^0_i = x_i$

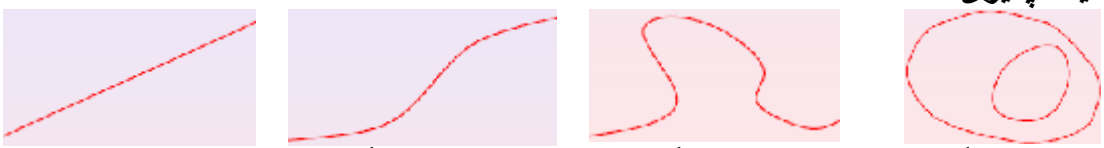
[1] Multi Layer Perceptron (MLP)

- خروجی لایه ی آخر M شامل خروجی های $\hat{y}_i = y_i^M$

خصوصیات MLP ها

- یک MLP می تواند هر تابع پیوسته ای را تخمین بزند
- اما ، لااقل به یک لایه ی مخفی و قسمت های کافی در هر لایه نیازمند می باشد (برخی از اوقات با دو لایه ی مخفی کار ساده تر است)
- به علاوه ، ما باید مقدار صحیح پارامترهای θ را پیدا نماییم
- این یک مساله ی NP- کامل می باشد !!!!
- چگونه می توانیم این پارامترها را پیدا نماییم ؟
- جواب : یک معیار داده شده را با استفاده از یک روش گرادیان ، بهینه نمایید .
- نکته : ظرفیت توسط تعداد پارامترها کنترل شده است

تفکیک پذیری



خطی | خطی + سیگما (سیگموید) | خطی + سیگما + خطی | خطی + سیگما + خطی + سیگما + خطی

۳. گرادیان توارث

- هدف : یک ملاک C را در یک مجموعه از داده های D_n ، کمینه نمایید :

$$C(D_n,\theta) = \sum_{p=1}^{n} L(y(p),\hat{y}(p))$$

که $\hat{y}(p) = MLP(x(p);\theta)$

- ما به دنبال بهترین پارامترهای θ هستیم :

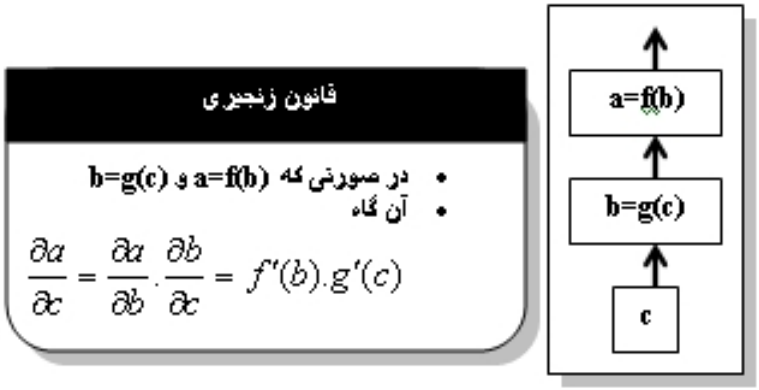
$$\theta^* = \arg\min_{\theta} C(D_n,\theta)$$

- *توارث گرادیان :* یک روال تکراری ، در هر تکرار s ما پارامترهای θ را ویرایش می کنیم :

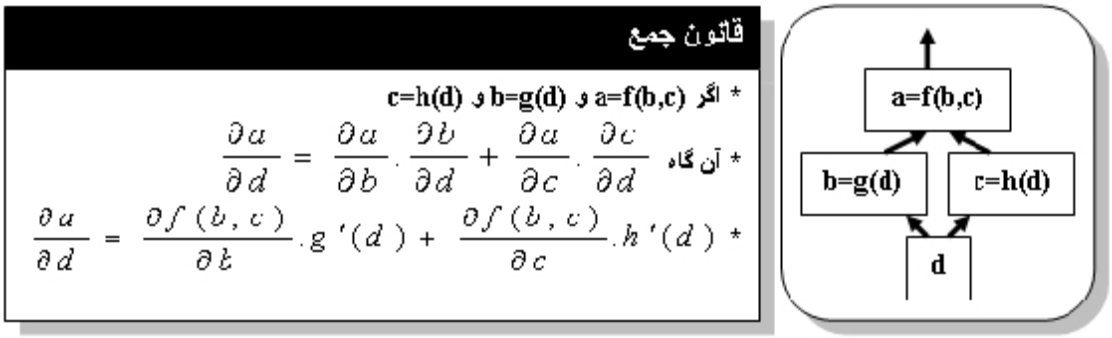
$$\theta^{s+1} = \theta^s - \eta \frac{\partial C(D_n,\theta^s)}{\partial \theta^s}$$

که η میزان آموزش می باشد . هشدار : بهینه های محلی

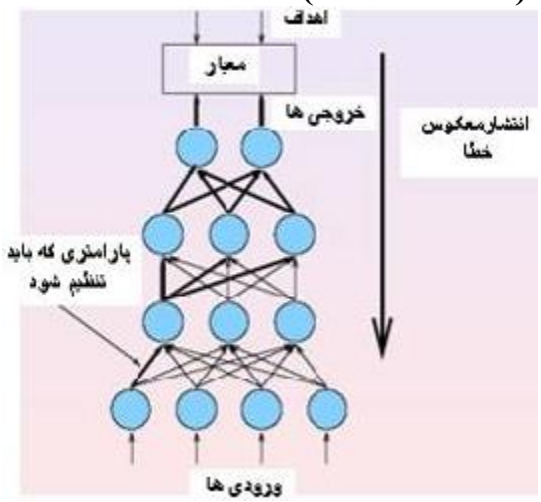
توارث گرادیان : پایه ها



توارث گرادیان : پایه ها (۲)



پایه های توارث گرادیان (نمایش تصویری)



توارث گرادیان : معیار

- اول : ما نیاز داریم که گرادیان را با معیار ارسال نماییم
- معیار عمومی C :

$$C(D_n,\theta) = \sum_{p=1}^{n} L(y(p),\hat{y}(p))$$

- مثال : ملاک خطای مربع میانگین [1] :

[1] Mean Squared Error (MSE)

$$L(y,\hat{y}) = \sum_{i=1}^{d} \frac{1}{2}(y_i - \hat{y}_i)^2$$

- و مشتق با رعایت خروجی $\hat{y}_i$:

$$\frac{\partial L(y,\hat{y})}{\partial \hat{y}_i} = \hat{y}_i - y_i$$

توارث گرادیان : لایه ی آخر

- دوم : مشتق با رعایت پارامترهای لایه ی آخر M

$$\hat{y}_i = y_i^M = \tanh(s_i^M)$$
$$s_i^M = b_i^M + \sum_j y_j^{M-1}.w_{i,j}^M$$

- بنابراین مشتق با رعایت $w_{i,j}^M$ برابر است با :

$$\frac{\partial \hat{y}_i}{\partial w_{i,j}^M} = \frac{\partial \hat{y}_i}{\partial s_i^M}.\frac{\partial s_i^M}{\partial w_{i,j}^M} = (1-(y_i^M)^2).y_j^{M-1}$$

- و مشتق با توجه به b_i^M برابر است با :

$$\frac{\partial \hat{y}_i}{\partial b_i^M} = \frac{\partial \hat{y}_i}{\partial s_i^M}.\frac{\partial s_i^M}{\partial b_i^M} = (1-(y_i^M)^2).1$$

توارث گرادیان : سایر لایه ها

- سوم : مشتق با توجه به خروجی لایه ی پنهان y_j^l

$$\frac{\partial \hat{y}_i}{\partial y_j^l} = \sum_k \frac{\partial \hat{y}_i}{\partial y_k^{l+1}}.\frac{\partial y_k^{l+1}}{\partial y_j^l}$$

در جایی که

$$\frac{\partial y_k^{l+1}}{\partial y_j^l} = \frac{\partial y_k^{l+1}}{\partial s_k^{l+1}}.\frac{\partial s_k^{l+1}}{\partial y_j^l} = (1-(y_k^{l+1})^2).w_{k,j}^{l+1}$$

و

$$\frac{\partial \hat{y}_i}{\partial y_{k\neq i}^M} = 0 \quad \text{و} \quad \frac{\partial \hat{y}_i}{\partial y_i^M} = 1$$

توارث گرادیان : سایر پارامترها

- چهارم : مشتق با توجه به پارامترهای لایه ی مخفی y_j^l

$$\frac{\partial \hat{y}_i}{\partial w_{j,k}^l} = \frac{\partial \hat{y}_i}{\partial y_j^l}.\frac{\partial y_j^l}{\partial w_{j,k}^l} = \frac{\partial \hat{y}_i}{\partial y_j^l}.y_k^{l-1}$$

و

$$\frac{\partial \hat{y}_i}{\partial b_j^l} = \frac{\partial \hat{y}_i}{\partial y_j^l}.\frac{\partial y_j^l}{\partial b_j^l} = \frac{\partial \hat{y}_i}{\partial y_j^l}.1$$

توارث گرادیان : الگوریتم عمومی

برای هر تکرار

۱. مقداردهی اولیه ی گرادیان ها $\frac{\partial C}{\partial \theta_i} = 0$ برای هر θ_i

۲. برای هر نمونه ی z(p)=(x(p),y(p))

a. فاز مستقیم : $\hat{y}(p) = MLP(x(p), \theta)$ را محاسبه کنید

b. $\frac{\partial L(y(p), \hat{y}(p))}{\partial \hat{y}(p)}$ را محاسبه نمایید

c. برای هر لایه ی l از M تا ۱ :

i. $\frac{\partial \hat{y}(p)}{\partial y_j^l}$ را محاسبه نمایید

ii. $\frac{\partial y_j^l}{\partial b_j^l}$ و $\frac{\partial y_j^l}{\partial w_{j,k}^l}$ را محاسبه کنید

iii. گرادیان ها را جمع کنید :

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial b_j^l} + \frac{\partial C}{\partial L} \cdot \frac{\partial L}{\partial \hat{y}(p)} \cdot \frac{\partial \hat{y}(p)}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial b_j^l}$$

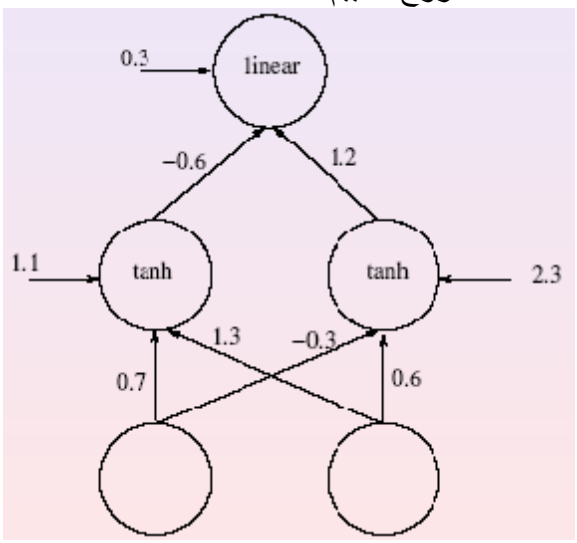
$$\frac{\partial C}{\partial w_{j,k}^l} = \frac{\partial C}{\partial w_{j,k}^l} + \frac{\partial C}{\partial L} \cdot \frac{\partial L}{\partial \hat{y}(p)} \cdot \frac{\partial \hat{y}(p)}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{j,k}^l}$$

۳. پارامترها را به روز نمایید :

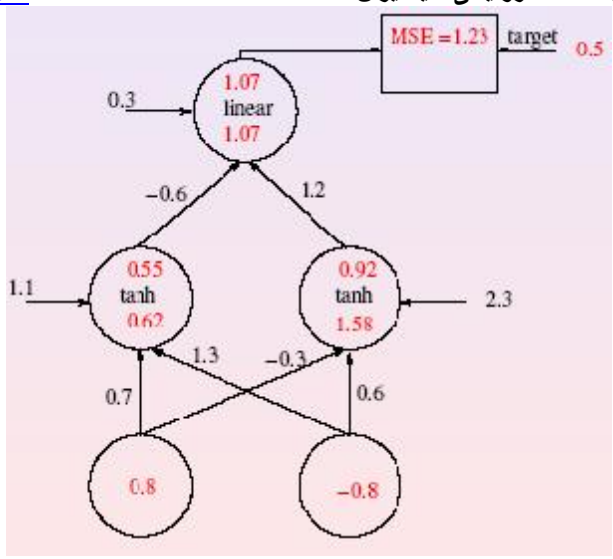
$$\theta_i^{s+1} = \theta_i^s - \eta . \frac{\partial C}{\partial \theta_i^s}$$

توارث گرادیان : یک مثال

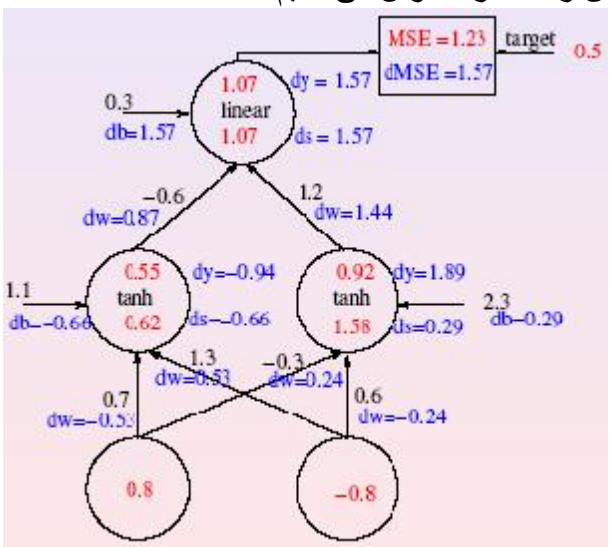
اجازه دهید که با یک MLP ساده شروع نماییم :



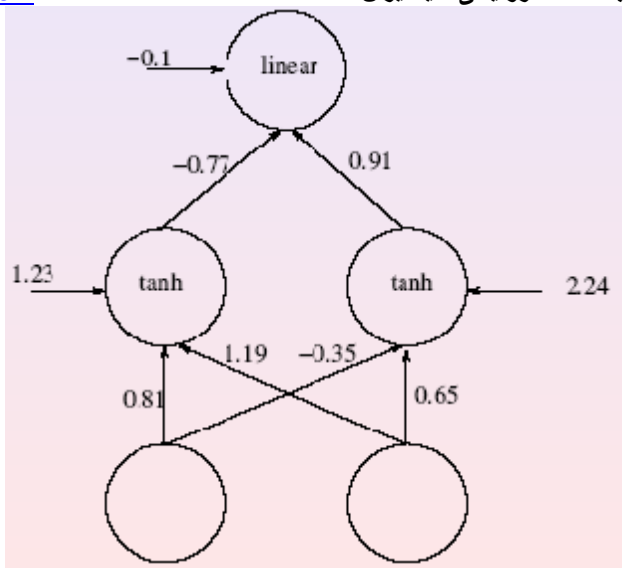
ما MSE یک نمونه را محاسبه می نماییم :



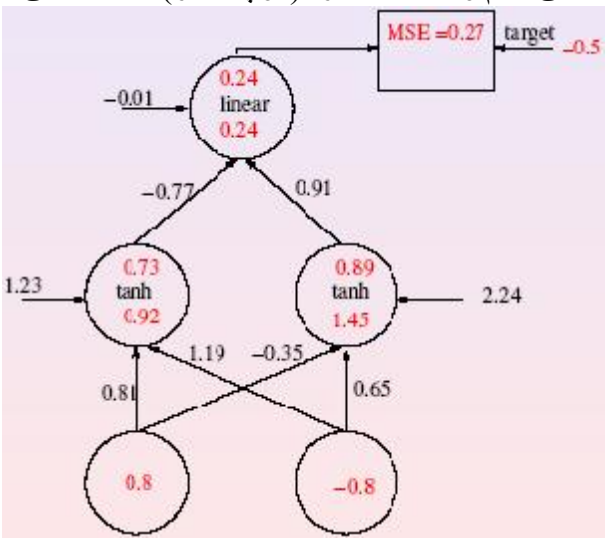
ما در هر مکان ، گرادیان را انتشارمعکوس می دهیم :



ما هر پارامتر را با میزان آموزشی ۰,۱ بازنگری می نماییم :



ما نمونه ای شبیه را ارایه می کنیم و MSE آن را (کوچک تر) محاسبه می کنیم :



MLP ، تخمین زننده های عمومی است

- می تواند نشان دهد که ، تحت مفروضات معقولانه ، می تواند هر تابع سلیس با یک MLP و با یک لایه از قسمت های مخفی را تخمین بزند .
- درک مستقیم اول :
  - اجازه بدهید که یک عملیات طبقه بندی را در نظر بگیریم
  - و توابع انتقال سخت را برای قسمت های مخفی داشته باشیم :

y=step(s) در صورتی که s>0 باشد برابر ۱ خواهد بود و در غیر این صورت برابر با صفر خواهد بود .

  - اجازه دهید که برای قسمت های خروجی توابع انتقال خطی داشته باشیم : y=s
  - تلاش اول :

$$\hat{y} = c + \sum_{i=1}^{N} v_i . sign\left(\sum_{j=1}^{M} x_j . w_{i,j} + b_i \right)$$

تصویر : تخمین زننده های عمومی

(۱) (۲) (۳) (۴) (۵)

تخمین عمومی با کسینوس ها

- بیایید توابعی ساده با دو متغیر $y(x_1, x_2)$ را در نظر بگیریم
- تجزیه ی فوریه

$$y(x_1, x_2) \cong \sum_s A_s(x_1)\cos(sx_2)$$

که ضرایب A_s ، توابعی از x_1 هستند .

- تجزیه ی فوریه ی بیش تر :

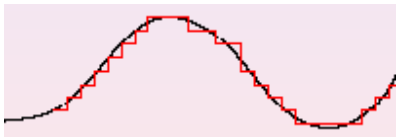
$$y(x_1, x_2) \cong \sum_s \sum_l A_{s,l}\cos(lx_1)\cos(sx_2)$$

- ما می دانیم که $\cos(\alpha)\cos(\beta) = \frac{1}{2}\cos(\alpha+\beta) + \frac{1}{2}\cos(\alpha-\beta)$:

$$y(x_1, x_2) \cong \sum_s \sum_l A_{s,l}\left[\frac{1}{2}\cos(lx_1 + sx_2) + \frac{1}{2}\cos(lx_1 - sx_2)\right]$$

- تابع کسینوس می تواند با ترکیبات خطی توابع مرحله ای تخمین زده شود :

$$f(z) = f_0 + \sum_i (f_{i+1} - f_i)step(z - z_i)$$



- بنابراین $y(x_1, x_2)$ می تواند با یک ترکیب خطی از توابع مرحله ای که آرگومان های آن ترکیبات خطی x_1 و x_2 هستند ، تخمین زده شوند و می توانند توسط توابع tanh هم تخمین زده شوند .

۴. طبقه بندی

ANN برای طبقه بندی دودویی

- یک خروجی با مقصد به صورت {-1,1} یا {0,1} وابسته به تابع خروجی لایه ی آخر (خطی ، سیگما ، tanh و ...) کد می شود .
- برای یک خروجی داده شده ، کلاس وابسته برای نزدیک ترین مقصد ، مناسب می باشد .
- چگونه می توانیم کلاس احتمال های قبلی را دریافت نماییم :
  - از یک سیگما با اهداف {0,1} استفاده نمایید
  - در صورتی که مدل به درستی آموزش داده شده (برای مثال با معیار MSE)
    $\Rightarrow \hat{y}(x) = E[Y \mid X = x] = 1.P(Y = 1 \mid X = x) + 0.P(Y = 0 \mid X = x)$
  - بنابراین ، خروجی P(Y=1|X=x) را کد خواهد کرد
- نکته : ما به طور مستقیم طبقه بندی خطا را بهینه نمی کنیم

ANN برای طبقه بندی با چندکلاس

- ساده ترین راه حل : کد گذاری یک - گرم

- یک خروجی در هر کلاس ، برای مثال به صورت (۰،...،۱، ... ،۰) رمزگذاری (کد) می شود
- برای یک خروجی داده شده ، کلاس مرتبط شده برای اندیس مقدار بیشینه در بردار خروجی ، مناسب می باشد
- چگونه احتمال های کلاس قبلی را دریافت نماییم :
  - از یک سافتمکس [1] استفاده نمایید : $\hat{y}_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$
  - هر خروجی i ، P(Y=i|X=x) را کد خواهد نمود

- در غیر این صورت : هر کلاس برای یک کد دودویی ، مناسب می باشد
  - برای مثال برای یک مساله ی ۴ – کلاس ، ما می توانیم برای هر کلاس ، یک کد ۸ – بعدی داشته باشیم
  - برای یک خروجی داده شده ، کلاس مرتبط ، با نزدیک ترین کد ، مناسب می باشد (با توجه به یک فاصله ی داده شده)
  - مثال : کدهای خروجی صحیح کننده ی خطا [2]

کدهای خروجی صحیح کننده ی خطا

- بیایید یک مساله ی ۴- کلاسی با ۶ قسمت را ارایه کنیم :

کلاس ۱ : ۱۱۰۰۰۱

کلاس ۲ : ۱۰۰۰۱۰

کلاس ۳ : ۰۱۰۱۰۰

کلاس ۴ : ۰۰۱۰۰۰

- سپس ما ۶ طبقه بندی کننده را به وجود می آوریم (یا ۱ طبقه بندی کننده با ۶ خروجی)
- برای مثال : اولین طبقه بندی کننده تلاش می کند که کلاس های ۱ و ۲ را از کلاس های ۳ و ۴ ، مجزا نماید
- در موقعی که یک نمونه ی جدید می آید ، ما فاصله ی میان کد دریافت شده توسط ۶ طبقه بندی کننده و ۴ کلاس را محاسبه می کنیم :

دریافت شده : ۰۱۱۱۱۰

فاصله ها : (بیایید فاصله ی مانهاتان را در نظر بگیریم)

برای کلاس ۱ : ۵ ، برای کلاس ۲: ۴ ، برای کلاس ۳ : ۲ ، برای کلاس ۴ : ۳

یک کد خروجی صحیح کننده ی خطا چیست

- چگونه یک کد خروجی صحیح کننده ی خطای خوب را طراحی نماییم ؟
- بهینه سازی دستی [3] کمینه ی میان هر جفت از کلمات کلیدی را بیشینه نمایید .
- یک ECOC خوب باید دو خصوصیت زیر را داشته باشد :

---

[1] softmax
[2] Error Correcting Output Codes (ECOC)
[3] Hamming distance

- جداسازی ردیفی . (بهینه سازی دستی)
- جداسازی ستونی . توابع ستونی باید تا حد ممکن با سایرین ، ناهمبسته باشند .

۵. رمزهای تجارت

توارث گرادیان تصادفی

- روش توارث گرادیان به صورت *دسته ای* می باشد :
  - اول گرادیان همه ی نمونه ها را جمع کنید ، سپس پارامترها را تنظیم نمایید
  - در صورتی که مجموعه ی داده ای خیلی بزرگ باشد و شامل افزونگی [1] هایی باشد چه اتفاقی می افتد ؟
- راه حل دیگر : توارث گرادیان تصادفی
  - در عوض (به جای آن) ، پارامترها را بعد از هر نمونه ، مرتب نمایید
  - تصادف : ما گرادیان کامل را با تخمین آن در هر نمونه ، تقریب می زنیم
  - با این وجود ، انحراف باز هم برای این روش وجود دارد
  - به علاوه : برای مجموعه های داده ای بزرگ ، به مراتب سریع تر می باشد !!!
- سایر روش های گرادیان : روش های مرتبه ی دوم ، نظیر گرادیان در هم آمیخته (مرکب) [2] برای مجموعه های داده ای کوچک ، مناسب می باشند .

مقداردهی اولیه

- ما چگونه باید پارامترهای یک ANN را مقداردهی اولیه نماییم ؟
- یک مساله ی عمومی : *اشباع (سیری)* [3]

در زمانی که مجموع توزین شده ، بزرگ می باشد ، خروجی tanh (یا سیگما) اشباع می شود و گرادیان به سمت صفر میل می نماید .

بنابراین ، ما باید پارامترها را مقداردهی اولیه کنیم ، میانگین مجموع وزن در قسمت خطی تابع انتقال می باشد :

- برای جزئیات به کار تحقیقی لئون بوتو نگاه کنید
- داده های ورودی : نرمال سازی شده میانگین صفر و واحد واریانس ،
- اهداف :
  - رگرسیون : نرمال سازی شده با میانگین صفر و واحد واریانس
  - طبقه بندی :
    - تابع انتقال خروجی tanh می باشد : ۰٫۶ و ۰٫۶-
    - تابع انتقال خروجی ، سیگما می باشد : ۰٫۸ و ۰٫۲
    - تابع انتقال خروجی ، خطی می باشد : ۰٫۶ و ۰٫۶-

---

[1] redundency
[2] conjugate gradient
[3] saturation

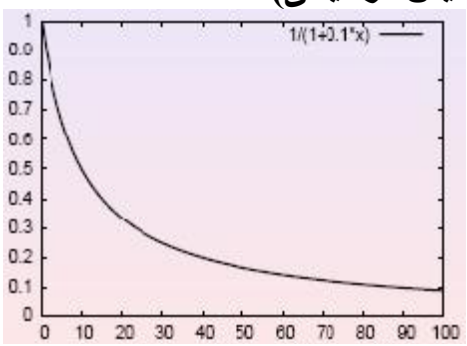
- پارامترها : به صورت یکنواخت در $\left[\frac{-1}{\sqrt{fan\,in}}, \frac{1}{\sqrt{fan\,out}}\right]$ توزیع شده اند

میزان یادگیری و کم شدن (تنزل) میزان یادگیری

- چگونه میزان یادگیری η را انتخاب کنیم ؟
- در صورتی که η خیلی بزرگ باشد : بهینه سازی کم می شود
- در صورتی که η خیلی کوچک باشد : بهینه سازی خیلی کند خواهد بود و در کمینه های محلی گیر بیفتد
- یک راه حل : کاهش پیشرفت [1]
  - میزان آموزش اولیه : $\eta=\eta_0$
  - کاهش میزان آموزش η_d
  - در هر تکرار s :

$$\eta(s) = \frac{\eta_0}{(1+s.\eta_d)}$$

کاهش میزان یادگیری (نمایش گرافیکی)



کاهش وزن

- یک راه برای کنترل ظرفیت : *تنظیم* [2]
- برای MLP ها ، زمانی که وزن ها به سمت صفر میل می کند ، توابع سیگما یا tanh *تقریبا خطی* می باشند ، بنابراین دارای ظرفیت کمی می باشند
- *کاهش وزن* : راه حل های با تاوان (جریمه) با وزن های بالا و بایاس (در دامنه)

$$C(D_n,\theta) = \sum_{p=1}^{n} L(y(p),\hat{y}(p)) + \frac{\beta}{2}\sum_{j=1}^{|\theta|}\theta_j^2$$

که β کاهش وزن را کنترل می کند .

- به کار گیری آسان می باشد :

$$\theta_j^{s+1} = \theta_j^s - \sum_{p=1}^{n}\eta\frac{\partial L(y(p),\hat{y}(p))}{\partial\theta_j^s} - \eta.\beta.\theta_j^s$$

مثال های ملاک های آموزشی

- خطای میانگین مربع ، برای رگرسیون :

---

[1] progressive decay
[2] regularization

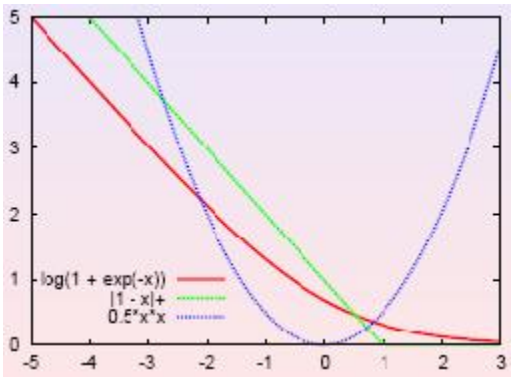
$$L(y,\hat{y}) = \sum_{i=1}^{d} \frac{1}{2}(y_i - \hat{y}_i)^2$$

- افت (آنتروپی) متقاطع ، برای طبقه بندی (اهداف $\in \{-1,1\}$) :

$$L(y,\hat{y}) = \log(1+\exp(-y_i \hat{y}_i))$$

- نوع سخت :

$$L(y,\hat{y}) = \|1 - y_i \hat{y}_i\| +$$

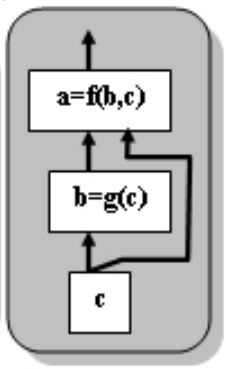


سایر شبکه های عصبی مصنوعی

۱. توابع با پایه های انشعابی [1]
۲. شبکه های عصبی بازگشت کننده
۳. شبکه های خود شرکت پذیر
۴. مخلوط خبره ها
۵. TDNN ها و LeNet

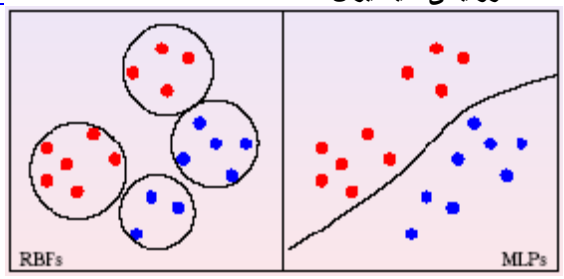
مکانیزم کاهش گرادیان نوعی

- در صورتی که $a=f(b,c;\theta_f)$ تشخیص پذیر می باشد و $b=g(c;\theta_g)$ تشخیص پذیر است
- آن گاه شما باید قادر باشید که گرادیان را با رعایت a,b,c,θ_f,θ_g و ... محاسبه نمایید
- بنابراین فقط تصور شما از به وجود آوردن سایر شبکه های عصبی ، جلوگیری می کند !



۱. توابع با پایه ی انشعابی
تفاوت میان RBF ها و MLP ها

[1] Radial Basis Function (RBF)



مدل های RBF

- MLP معمولی اما با لایه ی پنهان l به صورت زیر ، کد می شود:

$$s_i^l = -\frac{1}{2}\sum_j (\gamma_{i,j}^l)^2 . (y_j^{l-1} - \mu_{i,j}^l)^2$$

$$y_i^l = \exp(s_i^l)$$

- پارامترهای چنین لایه ی l ای $\theta_l = \{\gamma_{i,j}^l, \mu_{i,j}^l : \forall i, j\}$ می باشند
- این لایه ها برای توسعه ی خصوصیات محلی (از آنجایی که لایه های tanh خصوصیات سراسری را توسعه می دهند) مفیدند

کاهش گرادیان برای RBF ها

- $s_i^l = -\frac{1}{2}\sum_j (\gamma_{i,j}^l)^2 . (y_j^{l-1} - \mu_{i,j}^l)^2$
- $\leftarrow \quad y_i^l = \exp(s_i^l)$
- $\frac{\partial y_i^l}{\partial s_i^l} = \exp(s_i^l) = y_i^l$
- $\frac{\partial s_i^l}{\partial y_j^{l-1}} = -(\gamma_{i,j}^l)^2 . (y_j^{l-1} - \mu_{i,j}^l)$
- $\frac{\partial s_i^l}{\partial \mu_{i,j}^l} = (\gamma_{i,j}^l)^2 . (y_j^{l-1} - \mu_{i,j}^l)$
- $\frac{\partial s_i^l}{\partial \gamma_{i,j}^l} = -\gamma_{i,j}^l . (y_j^{l-1} - \mu_{i,j}^l)^2$

هشدار در مورد واریانس ها

- مقداردهی اولیه : برای نمونه از K- میانگین استفاده نمایید
- در مورد آموزش γ با کاهش گرادیان باید خیلی دقت کنیم
- یادآوری :

$$y_i^l = \exp\left(-\frac{1}{2}\sum_j (\gamma_{i,j}^l)^2 . (y_j^{l-1} - \mu_{i,j}^l)^2\right)$$

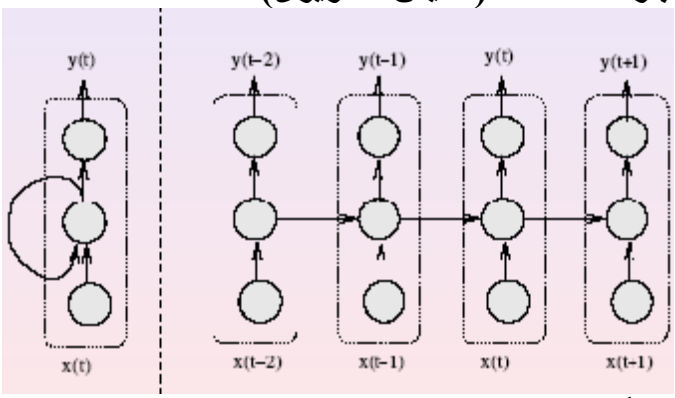
- در صورتی که γ خیلی بزرگ شود ، خروجی RBF ممکن است به صورت انفجاری باشد !
- یک راه حل : آن ها را در یک دامنه ی معقولانه ، محدود نمایید
- در غیر این صورت ، آن ها را آموزش ندهید (تخمین K- میانگین را برای γ نگهداری نمایید)

۲. شبکه های عصبی بازگشت کننده

- این جور مدل ها لایه های l با توابع مجتمع $s_i^l = f\left(y_j^{l+k}\right)$ که $k \geq 0$ و بنابراین حلقه ها یا بازگشت ها را قبول می کنند
- این قبیل لایه های l ، مفهوم یک وضعیت زمانی را کد می کنند
- برای جستجو برای ارتباط های درون داده های زمانی مفید می باشند
- احتیاج به معین کردن تاخیر دقیق در ارتباط نداریم
- برای محاسبه ی گرادیان ، در زمانی که همه ی ارتباط های میان داده ها به صورت زیر است ، یکی باید مورد توجه قرار گیرد :

  $s_i^l(t) = f\left(y_j^{l+k}(t-1)\right)$ که $k \geq 0$ می باشد
- بنابراین ، نیاز به نمایش کل گراف وابسته به زمان میان رشته ی ورودی و رشته ی خروجی داریم
- اخطار : می تواند نشان دهد که گرادیان به صورت نمایی با زمان از بین می رود

شبکه های عصبی بازگشت کننده (نمایش تصویری)



شبکه های عصبی بازگشت کننده – مثال اشتقاق

- به شبکه ی عصبی بازگشت کننده ی ساده ی زیر توجه نمایید :

$$h_t = \tanh(d.h_{t-1} + w.x_t + b)$$

$$\hat{y}_t = v.h_t + c$$

که $\{d,w,b,v,c\}$ مجموعه ای از پارامترها می باشند

- هزینه لازم برای کمینه کردن یک رشته :

$$C = \sum_{t=1}^{T} C_t = \sum_{t=1}^{T} \frac{1}{2}(y_t - \hat{y}_t)^2$$

اشتقاق گرادیان

- ما باید حاصل عبارت زیر را به دست بیاوریم :

$$\frac{\partial C}{\partial d}, \frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}, \frac{\partial C}{\partial v}, \frac{\partial C}{\partial c}$$

- بیایید این کار را برای $\frac{\partial C}{\partial w}$ انجام دهیم .

$$\frac{\partial C}{\partial w}=\sum_{t=1}^{T}\frac{\partial C_t}{\partial w}=\sum_{t=1}^{T}\frac{\partial C_t}{\partial \hat{y}_t}.\frac{\partial \hat{y}_t}{\partial w}=\sum_{t=1}^{T}\frac{\partial C_t}{\partial \hat{y}_t}.\frac{\partial \hat{y}_t}{\partial h_t}.\frac{\partial h_t}{\partial w}$$

$$\frac{\partial C}{\partial w}=\sum_{t=1}^{T}\frac{\partial C_t}{\partial \hat{y}_t}.\frac{\partial \hat{y}_t}{\partial h_t}.\frac{\partial h_t}{\partial w}=\sum_{t=1}^{T}\frac{\partial C_t}{\partial \hat{y}_t}.\frac{\partial \hat{y}_t}{\partial h_t}.\sum_{s=1}^{t}\frac{\partial h_t}{\partial h_s}.\frac{\partial h_s}{\partial w}$$

$$\frac{\partial C_t}{\partial \hat{y}_t}=\hat{y}_t-y_t$$

$$\frac{\partial \hat{y}_t}{\partial h_t}=v$$

$$\frac{\partial h_t}{\partial h_s}=\prod_{i=s+1}^{t}\frac{\partial h_i}{\partial h_{i-1}}=\prod_{i=s+1}^{t}\left(1-h_i^2\right).d$$

$$\frac{\partial h_s}{\partial w}=\left(1-h_s^2\right)x_s$$

وابستگی های بلند مدت

- تصور کنید که ما می خواهیم یک رشته را با توجه به قاب اولش طبقه بندی نماییم اما مقصد فقط در انتها شناخته شده می باشد

- متاسفانه گرادیان از بین می رود :

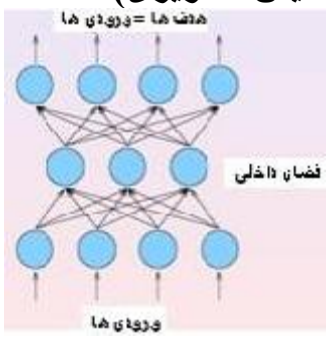
$$\frac{\partial C_T}{\partial w}=\sum_{t=1}^{T}\frac{\partial C_T}{\partial h_t}\frac{\partial h_t}{\partial w}\to 0$$

- این به این دلیل است که برای t<<T

$$\left|\frac{\partial h_\tau}{\partial h_{\tau-1}}\right|<1 \quad \text{و} \quad \frac{\partial C_T}{\partial h_t}=\frac{\partial C_T}{\partial h_T}\prod_{\tau=t+1}^{T}\frac{\partial h_\tau}{\partial h_{\tau-1}}$$

۳. شبکه های خود شرکت پذیر

شبکه های خود شرکت پذیر (نمایش تصویری)



شبکه های خود برگشت پذیر

- هدف آشکار : یاد بگیرید که ورودی را نوسازی نمایید
- در مدل های این چنینی ، بردار مقصد ، شبیه بردار ورودی می باشد !

- هدف واقعی : یک ارایه ی داخلی داده ها را یاد بگیرید
- در صورتی که یک لایه ی پنهان از قسمت های خطی وجود دارد ، سپس بعد از آموزش ، مدل یک *تحلیل جزئی اساسی* را با اولین N جزء اساسی پیاده سازی می کند (N تعداد قسمت های مخفی می باشد) .
- در صورتی که هیچ حالت خطی و لایه های مخفی بیش تر وجود نداشته باشد ، آن گاه سیستم یک نوع غیرخطی تحلیل جزئی اساسی را پیاده سازی می کند .

مخلوط خبره ها

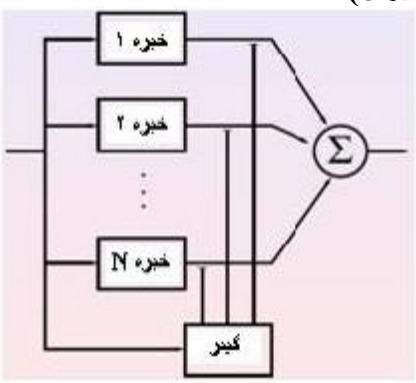
- تصور نمایید $f_i(x;\theta_{f_i})$ یک تابع پارامتری قابل تشخیص باشد
- تصور نمایید N تابع این چنینی f_i وجود داشته باشد .
- تصور کنید که $g(x;\theta_g)$ یک گیتر [1] باشد : یک تابع قابل تشخیص با N خروجی مثبت

$$\sum_{i=1}^{N} g(x;\theta_g)[i] = 1$$

- بنابراین یک مخلوط خبره ها یک تابع $h(x;\theta)$ می باشد :

$$h(x;\theta) = \sum_{i=1}^{N} g(x;\theta_g)[i].\, f_i(x;\theta_{f_i})$$

مخلوط خبره ها (نمایش تصویری)



مخلوط خبره ها ـ آموزش

- ما می توانیم گرادیان را با توجه به هر پارامتر ، محاسبه نماییم :
  - پارامترهای در خبره ی f_i :

$$\frac{\partial h(x;\theta)}{\partial \theta_{f_i}} = \frac{\partial h(x;\theta)}{\partial f_i(x;\theta_{f_i})}.\frac{\partial f_i(x;\theta_{f_i})}{\partial \theta_{f_i}} = g(x;\theta_g)[i].\frac{\partial f_i(x;\theta_{f_i})}{\partial \theta_{f_i}}$$

  - پارامترهای درون گیتر g :

$$\frac{\partial h(x;\theta)}{\partial \theta_g} = \sum_{i=1}^{N} \frac{\partial h(x;\theta)}{\partial g(x;\theta_g)[i]}.\frac{\partial g(x;\theta_g)[i]}{\partial \theta_g} = \sum_{i=1}^{N} f_i(x;\theta_{f_i}).\frac{\partial g(x;\theta_g)[i]}{\partial \theta_g}$$

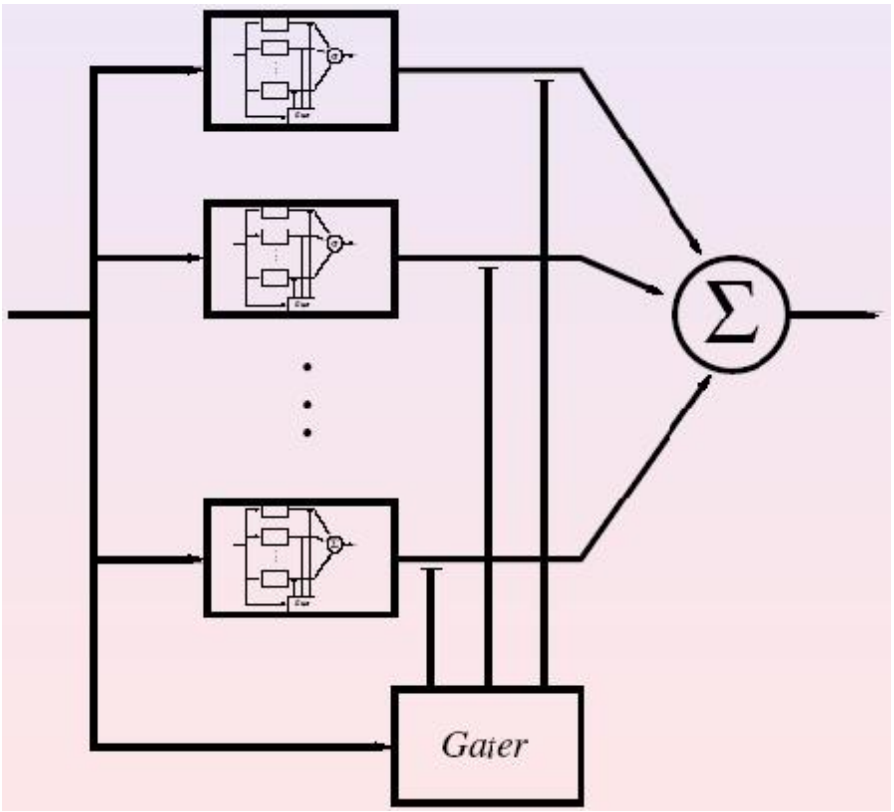
مخلوط خبره ها ـ بحث

[1] gater

- گیتر ، یک قسمت نرم فضای ورودی را پیاده سازی می کند (در مقایسه با K- میانگین باید بگوییم که K- میانگین ، قطعه ی سخت می باشد)
- در زمانی مناسب می باشد که رژیم هایی در داده ها وجود داشته باشد
- مورد خاص : در زمانی که خبره ها بتوانند توسط EM آموزش داده شوند ، مخلوط هم می تواند توسط EM آموزش داده شود .

مخلوط سلسله مراتبی خبره ها

در زمانی که خبره ها خودشان به صورت مخلوط هایی از خبره ها ارایه شده اند :



۵. TDNN ها و LeNet

زمان تاخیر شبکه های عصبی [1]

- TDNN ها مدل هایی برای تحلیل رشته ها یا سری های زمانی هستند .
- فرض ها : برخی از قواعد در طول زمان وجود دارند .
- الگوهای شبیه ، برخی از مواقع در مدت سری های زمانی یکسان (یا حتی در چند سری زمانی) می توانند دیده شوند .
- اولین ایده : یک قسمت پنهان برای مدل هر الگو را مشخص نمایید
  - این قسمت های پنهان باید دارای پارامترهای مرتبط که در طول زمان یکسان هستند ، باشند

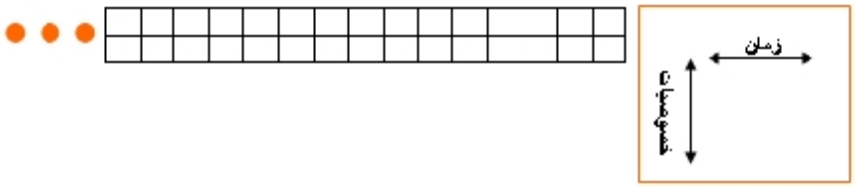
[1] Time Delay Neural Networks (TDNNs)

- بنابراین ، قسمت پنهان ربط داده شده به یک الگوی داده شده ی p_i در زمان t ، پارامترهایی به صورت قسمت پنهان ربط داده شده برای پارامتر یکسان p_i در زمان t+k را تقسیم می کند .

- توجه کنید که ما همچنین قصد داریم که یاد بگیریم که پارامترها چیستند !

TDNN ها : پیچیدگی ها

- چگونه اولین ایده را رسمی کنیم ؟ با استفاده از یک عملگر **پیچیدگی** .
- این عملگر می تواند برای نه فقط میان ورودی و اولین لایه ی پنهان استفاده شود ، بلکه در میان هر لایه ی پنهان می تواند استفاده شود .



پیچیدگی ها : معادله ها

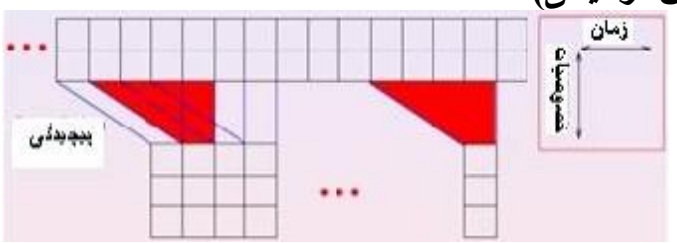
- تصور کنید $s^l_{t,i}$ مقدار ورودی در زمان t قسمت i از لایه ی l باشد .
تصور نمایید $y^l_{t,i}$ مقدار ورودی در زمان t قسمت i از لایه ی l باشد .
(مقدار ورودی ها : $y^0_{t,i}=x_{t,i}$)
فرض کنید $w^l_{i,j,k}$ وزن میان قسمت i لایه ی l در هر زمان t و قسمت j لایه ی l در زمان t-k باشد .
فرض کنید b^l_i بایاس قسمت i در لایه ی l باشد .
- عملگر پیچیدگی برای پنجره های با اندازه ی K :

$$s^l_{t,i} = \sum_{k=0}^{K-1} \sum_j w^l_{i,j,k} . y^{l-1}_{t-k,j} + b^l_i$$

- انتقال :

$$y^l_{t,i}=\tanh(s^l_{t,i})$$

پیچیدگی ها (نمایش گرافیکی)



- نکته : وزن های $w^l_{i,j,k}$ و بایاس های b^l_i وابسته به زمان نمی باشند .
- بنابراین تعداد پارامترهای این چنین مدلی مستقل از طول سری های زمانی می باشد .
- هر قسمت $s^l_{t,i}$ مقدار تابع یکسان در هر مرحله ی زمانی را ارایه می نماید .

TDNN ها : زیرنمونه

- توابع پیچیدگی ، معمولا با یک پنجره با اندازه ی ثابت کار می کنند (K در این مورد ، می تواند برای هر لایه / قسمت متفاوت باشد) .
- برخی از قواعد ممکن است در دانه های مختلف ، موجود باشند .

- بنابراین ، ایده ی دوم : زیر نمونه (در واقع بیش از یک نوع از عملگر هموار کننده می باشد) .
  - در میان هر لایه ی پیچیدگی ، اجازه بدهید که یک لایه ی زیر نمونه را اضافه نماییم .
  - این لایه ی زیر نمونه یک روش را برای تحلیل سری های زمانی در یک سطح خشن ، مهیا می کند .

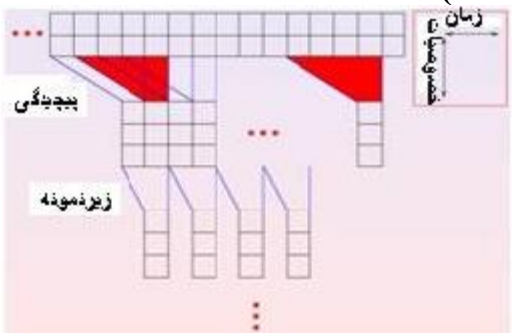
زیر نمونه : معادلات

- چگونه این ایده ی دوم را رسمی نماییم ؟
- تصور کنید $y^l_{t,i}$ مقدار خروجی در زمان t از قسمت i از لایه ی l باشد . (مقادیر ورودی : $(y^0_{t,i}=x_{t,i})$.
- فرض کنید r نسبت زیر نمونه باشد . این اغلب مجموعه ای برای مقدارهای نظیر ۲ تا ۴ می باشد .
- عملگر زیر نمونه :

$$y^{l}_{t,i} = \frac{1}{r}\sum_{k=0}^{r-1} y_{rt-k,i}\; i-1$$

- فقط مقدارهای $y^l_{t,i}$ $(t \bmod r) = ۰$ را محاسبه می کند .
- نکته : هیچ پارامتری در لایه ی زیر نمونه وجود ندارد .

TDNN ها (نمایش تصویری)



آموزش در TDNN ها

- TDNN ها می توانند توسط روش کاهش گرادیان نرمال ، آموزش داده شوند .
- توجه کنید که ، همان طوری که برای MLP ها داشتیم ، هر لایه یک تابع قابل تشخیص می باشد .
- ما فقط نیاز به محاسبه ی گرادیان محلی داریم :
- لایه های پیچیدگی :

$$\frac{\partial C}{\partial w^{l}_{i,j,k}} = \sum_t \frac{\partial C}{\partial s^{l}_{t,i}} \cdot \frac{\partial s^{l}_{t,i}}{\partial w^{l}_{i,j,k}} = \sum_t \frac{\partial C}{\partial s^{l}_{t,i}} \cdot y^{l-1}_{t-k,j}$$

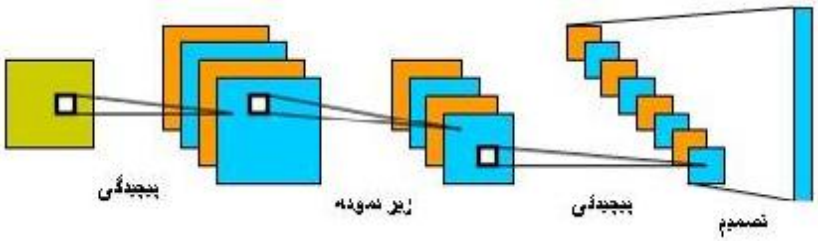
- لایه های زیر نمونه :

$$\frac{\partial C}{\partial y^{l-1}_{t-k,i}} = \frac{\partial C}{\partial y^{l}_{t,i}} \cdot \frac{\partial y^{l}_{t,i}}{\partial y^{l-1}_{rt-k,i}} = \frac{\partial C}{\partial y^{l}_{t,i}} \cdot \frac{1}{r}$$

کاربرد LeNet در تصویرها

- TDNN ها برای به کارگیری قواعد سری های زمانی مفید هستند (داده های یک بعدی)
- می توانیم از روشی مشابه برای تصویرها هم استفاده نماییم (داده های ۲ بعدی) ؟
- به هر حال ، قواعد ، اغلب در تصویرها واضح می باشند .
- این مورد با نام LeNet می باشد .

LeNet (نمایش تصویری)



مدل های مخلوط گاوسی [1]

۱. مقدمه
۲. الگوریتم EM
۳. کاربردهای EM در GMM ها
۴. نتایج عملی

۱. مقدمه

یادآوری : پایه های احتمال

تعداد کمی از معادلات که اغلب استفاده می شوند :

۱. (احتمال های شرطی

$$P(A,B)=P(A|B).P(B)$$

۲. (قانون بیز)

$$P(A \mid B) = \frac{P(B \mid A).P(A)}{P(B)}$$

۳. اگر $(\forall B_i = \Omega)$ و $(\forall i, j \neq i \;\; (B_i \cap B_j = \phi)$ آن گاه

$$P(A) = \sum_i P(A, B_i)$$

مدل مخلوط گاوسی چیست ؟

- یک مدل مخلوط گاوسی ، یک توزیع می باشد
- احتمال داده شده برای یک توزیع گاوسی به صورت زیر می باشد :

$$\aleph(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

[1] Gaussian Mixture Models (GMMs)

که d ، بعد (دیمانسیون) x می باشد و μ میانگین و Σ ماتریس کوواریانس گاوس می باشد . Σ اغلب به صورت قطری می باشد .

- احتمال داده شده برای یک GMM به صورت زیر می باشد :

$$p(x) = \sum_{i=1}^{N} w_i . \aleph(x \mid \mu_i, \Sigma_i)$$

که N تعداد گاوسی ها و w_i وزن گاوسی i می باشد ، با

$$\forall i : w_i \geq 0 \quad \text{و} \quad \sum_i w_i = 1$$

ویژگی های یک GMM

- از آنجایی که ANN ها تخمین زننده های عمومی توابع می باشند ،
- GMM ها تخمین زننده های عمومی چگالی می باشند . (البته به شرطی که به اندازه ی کافی گاوسی وجود داشته باشد)
- حتی GMM های قطری هم تخمین زننده های عمومی می باشند .
- GMM های سلسله مراتبی قطری ، برای کاربرد ، ساده نمی باشند : تعداد پارامترها ، مربع تعداد ابعاد می باشد .
- GMM ها می توانند توسط بیشینه ی احتمال با استفاده از یک الگوریتم مناسب آموزش داده شوند : *بیشینه سازی امید* [1]

کاربردهای عملی با استفاده از GMM ها

- تایید زیست سنجی فرد (با استفاده از صدا ، صورت ، دست خط و ...) :
  - یک GMM برای مشتری [2]
  - یک GMM برای سایرین
  - اگر تصمیم بیزی باشد ، در نتیجه نسبت احتمالی می باشد
- هر عملیات طبقه بندی با عدم تعادل بالا
  - یک GMM در هر کلاس ، با بیشینه ی احتمال ، تنظیم می شود
  - اگر تصمیم بیزی باشد ، در نتیجه نسبت احتمالی می باشد
- کاهش اندازه
- تدریج

۲. الگوریتم EM

پایه های بیشینه سازی امید

- هدف : بیشینه کردن احتمال $p(X|\theta)$ داده های X به دست آمده از یک توزیع ناشناخته ، با داشتن مدل پارامتربندی شده توسط θ :

$$\theta^* = \arg \max_{\theta} p(X \mid \theta) = \arg \max_{\theta} \prod_{p=1}^{n} p(x_p \mid \theta)$$

- ایده های اساسی EM :
  - یک متغیر پنهان که دانش آن بیشنه سازی $p(X| \theta)$ را ساده می کند ، معرفی نمایید

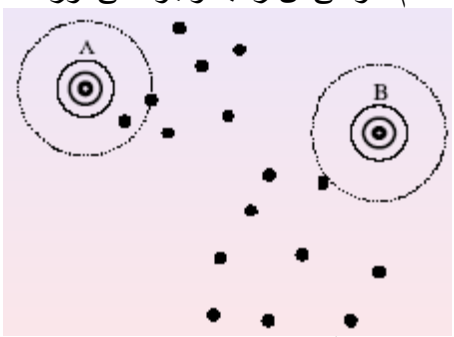
---

[1] Expectation - Maximization
[2] client

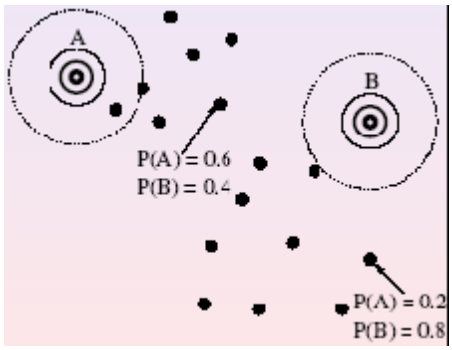
- در هر تکرار الگوریتم :
  - گام E : توزیع متغیر پنهان ارایه کننده ی داده ها و مقدار جاری پارامترها را تخمین بزنید
  - گام M : پارامترها را برای بیشینه کردن توزیع پیوسته ی داده ها و متغیر پنهان ، تغییر دهید

کاربردهای EM برای GMM (دید تصویری)

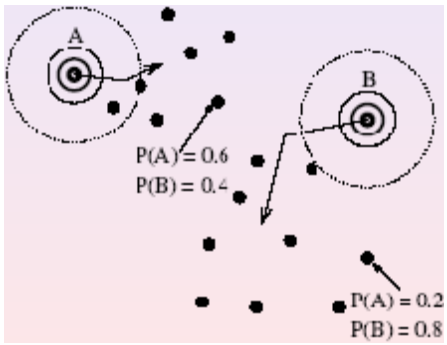
متغیر پنهان : برای هر نقطه ، کدام گاوسی آن را به وجود می آورد ؟



گام E : برای هر نقطه ، احتمالی که هر گاوسی آن را تولید کرده را تخمین بزنید



گام M : پارامترها را با توجه به متغیر پنهان برای بیشینه کردن احتمال داده ها (و متغیر پنهان) ، تغییر دهید



EM : رسمی تر

- اجازه دهید که متغیرپنهان را Q بنامیم .
- به تابع کمکی زیر توجه نمایید :

$$A(\theta,\theta^s) = E_Q[\log \quad p(X,Q \mid \theta) \mid X,\theta^s]$$

- این می تواند نشان دهد که بیشینه کردن A

$$\theta^{s+1} = \arg \max_{\theta} A(\theta, \theta^s)$$

همیشه احتمال داده های $p(X|\theta^S)$ را افزایش می دهد و یک بیشینه ی A ، برای یک بیشینه ی احتمال ، مناسب می باشد .

EM : اثبات پیچیدگی

اول بیایید تابع احتمال را توسعه بدهیم :

$$A(\theta, \theta^s) = E_Q[\log \quad p(X, Q \mid \theta) \mid X, \theta^s]$$

$$= \sum_{q \in Q} P(q \mid X, \theta^s) \log \quad p(X, q \mid \theta)$$

$$= \sum_{q \in Q} P(q \mid X, \theta^s) \log(P(q \mid X, \theta). p(X \mid \theta))$$

$$= \left[\sum_{q \in Q} P(q \mid X, \theta^s) \log \quad P(q \mid X, \theta) \right] + \log \quad p(X \mid \theta)$$

سپس اگر ما آن را براساس θ^S ارزیابی می کنیم

$$A(\theta^s, \theta^s) = \left[\sum_{q \in Q} P(q \mid X, \theta^s) \log \quad P(q \mid X, \theta^s) \right] + \log \quad p(X \mid \theta^s)$$

- تفاوت میان دو احتمال log پی در پی داده ها می تواند به صورت زیر نوشته شود :

$$\log \quad p(X \mid \theta) - \log \quad p(X \mid \theta^s) = A(\theta, \theta^s) + \sum_{q \in Q} P(q \mid X, \theta^s) \log \frac{P(q \mid X, \theta^s)}{P(q \mid X, \theta)}$$

- بنابراین ،
  - از آنجایی که آخرین قسمت معادله یک دیورژنس کولبک – لیبلر می باشد همیشه مثبت یا تهی است و
  - در صورتی که A افزایش یابد ، لگاریتم احتمال داده ها هم افزایش می یابد
  - علاوه بر این ، می تواند نشان دهد که در زمانی که A بیشینه می باشد ، احتمال داده ها هم در یک سطح ماکزیمم قرار دارد .

۳. کاربرد EM برای GMM ها

کاربرد EM برای GMM : متغیر پنهان

- برای GMM ، متغیر پنهان Q توضیح خواهد داد که کدام گاوسی هر نمونه را به وجود آورده است .
- در صورتی که Q دیده شد ، آن گاه بیشینه کردن احتمال داده ها ، ساده خواهد شد .
- به علاوه ، ما خواهیم دید که به سادگی می توانیم Q را تخمین بزنیم
- اجازه دهید که اول مخلوط مدل گاوسی را برای یک x_i بنویسیم :

$$p(x_i \mid \theta) = \sum_{j=1}^{N} P(j \mid \theta) p(x_i \mid j, \theta)$$

- اجازه بدهید که حالا متغیر نمایشگر [1] زیر را معرفی نماییم :

[1] indicator variable

$q_{i,j}$ در صورتی برابر یک می باشد که j ، x_i را منتشر نماید و در غیر این صورت برابر صفر می باشد

کاربرد EM برای GMM : تابع کمکی

- ما حالا می توانیم احتمال پیوسته ی هر X و q را بنویسیم :

$$p(X,Q\mid\theta)=\prod_{i=1}^{n}\prod_{j=1}^{N}P(j\mid\theta)^{q_{i,j}}\;p(x_i\mid j,\theta)^{q_{i,j}}$$

- که با استفاده از لگاریتم داریم :

$$\log\;\;p(X,Q\mid\theta)=\sum_{i=1}^{n}\sum_{j=1}^{N}q_{i,j}\log\;\;P(j\mid\theta)+q_{i,j}\log\;\;p(x_i\mid j,\theta)$$

- بگذارید حالا تابع کمکی مناسب را بنویسیم :

$$A(\theta,\theta^s)=E_Q\left[\log p(X,Q\mid\theta)\mid X,\theta^s\right]$$

$$=E_Q\left[\sum_{i=1}^{n}\sum_{j=1}^{N}q_{i,j}\log P(j\mid\theta)+q_{i,j}\log p(x_i\mid j,\theta)\mid X,\theta^s\right]$$

$$=\sum_{i=1}^{n}\sum_{j=1}^{N}E_Q\left[q_{i,j}\mid X,\theta^s\right]\log P(j\mid\theta)+E_Q\left[q_{i,j}\mid X,\theta^s\right]\log p(x_i\mid j,\theta)$$

کاربرد EM برای GMM : گام E و گام M

$$A(\theta,\theta^s)=\sum_{i=1}^{n}\sum_{j=1}^{N}E_Q\left[q_{i,j}\mid X,\theta^s\right]\log P(j\mid\theta)+E_Q\left[q_{i,j}\mid X,\theta^s\right]\log p(x_i\mid j,\theta)$$

- بنابراین ، گام E ، قبلی را تخمین می زند :

$$E_Q\left[q_{i,j}\mid X,\theta^s\right]=1.P(q_{i,j}=1\mid X,\theta^s)+0.P(q_{i,j}=0\mid X,\theta^s)$$

$$=P(j\mid x_i,\theta^s)=\frac{p(x_i\mid j,\theta^s)P(j\mid\theta^s)}{p(x_i\mid\theta^s)}$$

- و گام M ، پارامترهای θ ای که A را بیشینه می کند را پیدا می کند ، بنابراین جستجو می کند برای

$$\frac{\partial A}{\partial\theta}=0$$

برای هر پارامتر (μ_jو واریانس های σ_j^2 و وزن های w_j).

- توجه نمایید که w_j باید به یک برسد .

کاربرد EM برای GMM : کاربرد گام M برای میانگین

$$A(\theta,\theta^s)=\sum_{i=1}^{n}\sum_{j=1}^{N}E_Q\left[q_{i,j}\mid X,\theta^s\right]\log P(j\mid\theta)+E_Q\left[q_{i,j}\mid X,\theta^s\right]\log p(x_i\mid j,\theta)$$

$$\frac{\partial A}{\partial\mu_j}=\sum_{i=1}^{n}\frac{\partial A}{\partial\log p(x_i\mid j,\theta)}\frac{\partial\log p(x_i\mid j,\theta)}{\partial\mu_j}$$

$$= \sum_{i=1}^{n} P(j \mid x_i, \theta^s) \frac{\partial \log p(x_i \mid j, \theta)}{\partial \mu_j}$$

$$= \sum_{i=1}^{n} P(j \mid x_i, \theta^s) . \frac{(x_i - \mu_j)}{\sigma_j^2} = 0$$

$$\sum_{i=1}^{n} P(j \mid x_i, \theta^s) . \frac{(x_i - \mu_j)}{\sigma_j^2} = 0$$

← (برداشتن عبارت های ثابت در جمع)

$$\sum_{i=1}^{n} P(j \mid x_i, \theta^s) . x_i - \sum_{i=1}^{n} P(j \mid x_i, \theta^s) . \mu_j = 0$$

$$\frac{\sum_{i=1}^{n} P(j \mid x_i, \theta^s) . x_i}{\sum_{i=1}^{n} P(j \mid x_i, \theta^s)} = \hat{\mu}_j$$

کاربرد EM برای GMM : کاربرد گام M برای واریانس ها

$$A(\theta, \theta^s) = \sum_{i=1}^{n} \sum_{j=1}^{N} E_Q\left[q_{i,j} \mid X, \theta^s\right] \log P(j \mid \theta) + E_Q\left[q_{i,j} \mid X, \theta^s\right] \log p(x_i \mid j, \theta)$$

$$\frac{\partial A}{\partial \sigma_j^2} = \sum_{i=1}^{n} \frac{\partial A}{\partial \log p(x_i \mid j, \theta)} \frac{\partial \log p(x_i \mid j, \theta)}{\partial \sigma_j^2}$$

$$= \sum_{i=1}^{n} P(j \mid x_i, \theta^s) \frac{\partial \log p(x_i \mid j, \theta)}{\partial \sigma_j^2}$$

$$\sum_{i=1}^{n} P(j \mid x_i, \theta^s) . \left(\frac{(x_i - \mu_j)^2}{2\sigma_j^4} - \frac{1}{2\sigma_j^2} \right) = 0$$

$$\sum_{i=1}^{n} P(j \mid x_i, \theta^s) . \left(\frac{(x_i - \mu_j)^2}{2\sigma_j^4} - \frac{1}{2\sigma_j^2} \right) = 0$$

$$\sum_{i=1}^{n} \frac{P(j \mid x_i, \theta^s)(x_i - \mu_j)^2}{2\sigma_j^4} - \sum_{i=1}^{n} \frac{P(j \mid x_i, \theta^s)}{2\sigma_j^2} = 0$$

$$\sum_{i=1}^{n} \frac{P(j \mid x_i, \theta^s)(x_i - \mu_i)^2}{\sigma_j^2} - \sum_{i=1}^{n} P(j \mid x_i, \theta^s) = 0$$

$$\frac{\sum_{i=1}^{n} P(j \mid x_i, \theta^s)(x_i - \hat{\mu}_j)^2}{\sum_{i=1}^{n} P(j \mid x_i, \theta^s)} = \hat{\sigma}_j^2$$

کاربرد EM برای GMM : کاربرد گام M برای وزن ها

ما دارای این محدودیت هستیم که همه ی وزن های w_j باید مثبت و بالغ بر ۱ باشند

$$\sum_{j=1}^{N} w_j = 1$$

ترکیب کردن آن در سیستم :

$$J(\theta, \theta^s) = A(\theta, \theta^s) + \left(1 - \sum_{j=1}^{N} w_j\right).\lambda_j$$

که λj ها تکثیر کننده های زبان می باشند .

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial A(\theta, \theta^s)} \frac{\partial A(\theta, \theta^s)}{\partial w_j} - \lambda_j$$

$$= 1.\left(\sum_{i=1}^{n} P(j \mid x_i, \theta^s).\frac{1}{w_j}\right) - \lambda_j = 0$$

$$\hat{w}_j = \frac{\sum_{i=1}^{n} P(j \mid x_i, \theta^s)}{\lambda_j}$$

با یکی کردن محدویت احتمال ، داریم

$$\hat{w}_j = \frac{\sum_{i=1}^{n} P(j \mid x_i, \theta^s)}{\sum_{k=1}^{N}\sum_{i=1}^{n} P(k \mid x_i, \theta^s)} = \frac{1}{n}\sum_{i=1}^{n} P(j \mid x_i, \theta^s)$$

کاربرد EM برای GMM : به روز رسانی قانون ها

میانگین ها :

$$\hat{\mu}_j = \frac{\sum_{i=1}^{n} x_i.P(j \mid x_i, \theta^s)}{\sum_{i=1}^{n} P(j \mid x_i, \theta^s)}$$

واریانس ها :

$$\sigma_j^2 = \frac{\sum_{i=1}^{n} \left(x_i - \hat{\mu}_j\right)^2.P(j \mid x_i, \theta^s)}{\sum_{i=1}^{n} P(j \mid x_i, \theta^s)}$$

وزن ها : $\hat{w}_j = \frac{1}{n}\sum_{i=1}^{n} P(j \mid x_i, \theta^s)$

۴. نتیجه های عملی

مقداردهی اولیه

- EM یک روال تکراری می باشد که نسبت به شرایط اولیه ، خیلی حساس می باشد !
- در صورتی که شروع با زوائد باشد ، پایان نیز با زوائد خواهد بود .
- بنابراین ما به یک روال خوب و سریع مقدار دهی اولیه نیازمندیم .
- از K- میانگین ، اغلب استفاده می شود .
- سایر انتخاب ها : K- میانگین سلسله مراتبی و چند دسته ای شدن (شکافتگی) گاوسی .

کنترل ظرفیت

- چگونه ظرفیت را با GMM ها کنترل نماییم ؟
  - انتخاب تعدادی از گاوسی ها
  - محدود کردن مقدار واریانس ها برای این که از صفر دور شوند (اگر واریانس ها کوچک باشند ، در نتیجه ، ظرفیت ، افزایش می یابد)
- از وارسی اعتبار در معیار مطلوب (بیشینه ی احتمال ، طبقه بندی و ...) استفاده نمایید

روش های انطباق

- در برخی از موارد ، شما فقط به تعداد کمی از نمونه ها که از توزیع هدف می آیند دسترسی دارید
- ... اما تعداد زیادی از یک توزیع همسایه می آیند !
- چگونه ما می توانیم از مجموعه داده ای بزرگ سود ببریم ؟؟؟
- راه حل : از روش های تطبیق استفاده نمایید .
- معروف ترین و پراستفاده ترین روش برای GMM ها : تطبیق بیشینه ی یک قبلی [1]

تطبیق طرح

- آموزش بیشینه ی احتمال نرمال برای یک مجموعه ی داده ای X :

$$\theta^* = \arg\max_{\theta} p(X \mid \theta)$$

- آموزش بیشینه ی یک قبلی :

$$\theta^* = \arg\max_{\theta} p(\theta \mid X)$$

$$= \arg\max_{\theta} \frac{p(X \mid \theta)P(\theta)}{p(X)}$$

$$= \arg\max_{\theta} p(X \mid \theta)p(\theta)$$

که $p(\theta)$ ، تصور قبلی شما را از توزیع پارامترهای θ ارایه می کند .

پیاده سازی

- کدام نوع از توزیع قبلی برای $p(\theta)$ استفاده می شود ؟
- دو هدف :
  - محدود کردن θ با مقدارهای معقولانه

[1] Maximum A Posteriori (MAP)

- نگهداری الگوریتم EM قابل کنترل
- از در هم آمیختن قبلی ها [1] استفاده نمایید :
  - توزیع دیریچلت [2] برای وزن ها
  - چگالی گاوسی برای میانگین ها و واریانس ها

در هم آمیختن قبلی چیست ؟

- یک در هم آمیختن قبلی با توجه به وابسته های قبلی برای خانواده ی تابعی به صورت قبلی ، انتخاب می شود .
- بنابراین ما ترجیح می دهیم که $p(X|\theta)p(\theta)$ با توجه به خانواده ی یکسان به صورت $p(\theta)$ و قابل کنترل ، توزیع شود .
- مثال :
  - احتمال ، گاوسی می باشد : $p(X \mid \theta) = K_1 \exp\left(-\frac{(x_1-\mu_1)^2}{2\sigma_1^2}\right)$
  - قبلی ، گاوسی می باشد : $p(\theta) = K_2 \exp\left(-\frac{(x_2-\mu_2)^2}{2\sigma_2^2}\right)$
  - قبل تری ، گاوسی می باشد :

$$p(X \mid \theta)p(\theta) = K_1K_2 \exp\left(-\frac{(x_1-\mu_1)^2}{2\sigma_1^2} - \frac{(x_2-\mu_2)^2}{2\sigma_2^2}\right) = K_3 \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

در هم آمیختن چندجمله ای [3]

- توزیع چندجمله ای :

$$P(X_1 = x_1,..., X_n = x_n \mid \theta) = \frac{N!}{\prod_{i=1}^n x_i!}\prod_{i=1}^n \theta_i^{x_i}$$

که x_i ها اعداد صحیح نامنفی هستند و $\sum_{i=1}^n x_i = N$

- توزیع دیریچلت با پارامتر u :

$$P(\theta \mid u) = \frac{1}{Z(u)}\prod_{i=1}^n \theta_i^{u_i-1}$$

که $\theta_1,...,\theta_n \geq 0$ و $\sum_{i=1}^n \theta_i = 1$ و $u_1,...,u_n \geq 0$.

- در هم آمیختن قبلی = دیریچلت با پارامتر x+u :

$$P(X,\theta \mid u) = \frac{1}{Z}\prod_{i=1}^n \theta_i^{x_i+u_i-1}$$

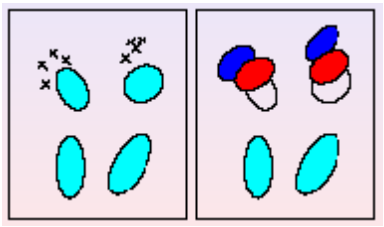
مثال های در هم آمیختن قبلی ها

قبلی p(θ|x)	در هم آمیختن قبلی	احتمال p(x| θ)

[1] conjugate priors
[2] Dirichlet
[3] multinomial

	p(θ)	
Gaussian(θ,σ)	Gaussian(μ_0,σ_0)	Gaussian(μ_1,σ_1)
Binomial[1](N,θ)	Beta(r,s)	Beta(r+n,s+N-n)
Position(θ)	Gamma(r,s)	Gamma(r+n,s+1)
Multinomial(θ_1,…, θ_k)	Dirichlet(α_1,…,α_k)	Dirichlet(α_1+n_1,…, α_k+n_k)

پیاده سازی ساده برای MAP-GMM ها



پیاده سازی ساده

- یک مدل قبلی کلی p با مقدار زیادی از داده های قابل دسترس را آموزش دهید

$$\Rightarrow \{w_j^p, \mu_j^p, \sigma_j^p\}$$

- یک فرا پارامتر : $\alpha \in [0,1]$: اعتقاد در مدل قبلی
- وزن ها :

$$\hat{w}_j = \left[\alpha w_j^p + (1-\alpha)\sum_{i=1}^{n} P(j \mid x_i)\right]\gamma$$

که γ فاکتور نرمال سازی می باشد (بنابراین $\sum_i w_j = 1$)

- میانگین ها :

$$\hat{\mu}_j = \alpha\mu_j^p + (1-\alpha)\frac{\sum_{i=1}^{n} P(j \mid x_i)x_i}{\sum_{i=1}^{n} P(j \mid x_i)}$$

- واریانس ها :

$$\hat{\sigma}_j = \alpha\left(\sigma_j^p + \mu_j^p \mu_j^{p'}\right) + (1-\alpha)\frac{\sum_{i=1}^{n} P(j \mid x_i)x_i x_i'}{\sum_{i=1}^{n} P(j \mid x_i)} - \hat{\mu}_j \hat{\mu}_j'$$

GMM های تطبیق داده شده برای شناسایی فرد

- عملیات شناسایی فرد :

دسترسی را قبول کن اگر $P(S_i \mid X) > P(\overline{S}_i \mid X)$

که S_i یک مشتری ، $\overline{S}_i$ همه ی افراد دیگر و X یک دسترسی نسبت داده شده به S_i .

[1] دو جمله ای

- با استفاده از قضیه ی بیز داریم :

$$\frac{p(X \mid S_i)}{p(X \mid \overline{S}_i)} > \frac{P(\overline{S}_i)}{P(S_i)} = \Delta_{S_i} \approx \Delta$$

- $p(X \mid \overline{S}_i)$ در یک مجموعه ی داده ای بزرگ ، آموزش داده می شود
- $p(X \mid S_i)$ ، MAP تطبیق داده شده از $p(X \mid \overline{S}_i)$ می باشد .
- Δ در یک مجموعه ی معتبرسازی مجزا برای بهینه سازی یک معیار داده شده ، پیدا می شود .

مدل های پنهان (مخفی) مارکوف

۱. مدل های مارکوفی
۲. مدل های پنهان مارکوف
۳. کاربرد HMM ها برای سخن شناسی
۴. صورت های عملی

۱. مدل های مارکوفی

- پردازش اتفاقی یک رشته ی زمانی : توزیع احتمال متغیر q در زمان t وابسته به متغیر q در زمان های t-1 تا ۱ می باشد .

$$P(q_1, q_2, ..., q_T) = P(q_1^T) = P(q_1^T) = P(q_1)\prod_{t=2}^{T} P(q_t \mid q_1^{t-1})$$

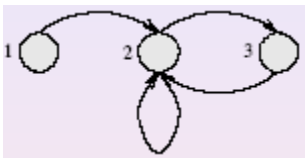
- پردازه ی مرتبه ی اول مارکوف :

$$P(q_t \mid q_1^{t-1}) = P(q_t \mid q_{t-1})$$

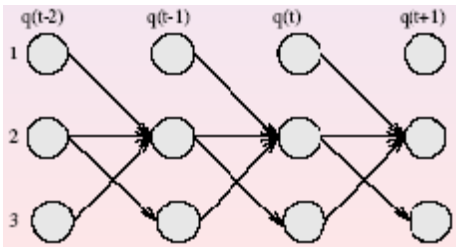
- مدل مارکوف : مدل یک پردازش مارکوفی با وضعیت های گسسته .

مدل های مارکوف (نمایش تصویری) :

- یک مدل مارکوف :



- یک مدل مارکوف نمایش داده شده در زمان :



آموزش مدل های مارکوف

- یک مدل مارکوف توسط همه ی احتمال های انتقال ، نمایش داده می شود :

$$P(q_t = i \mid q_{t-1} = j) \quad \forall i, j$$

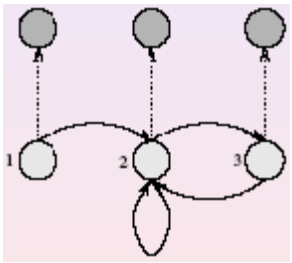
- با داشتن یک مجموعه ی آموزشی رشته های X ، میانگین های آموزشی ، این احتمال ها را مجددا تخمین می زنند .
- به سادگی ، آن ها را برای به دست آوردن راه حل با بیشینه ی احتمال ، شمارش نمایید :

$$P(q_t = i \mid q_{t-1} = j) = \frac{\#(q_t = i \quad and \quad q_{t-1} = j \mid X)}{\#(q_{t-1} = j \mid X)}$$

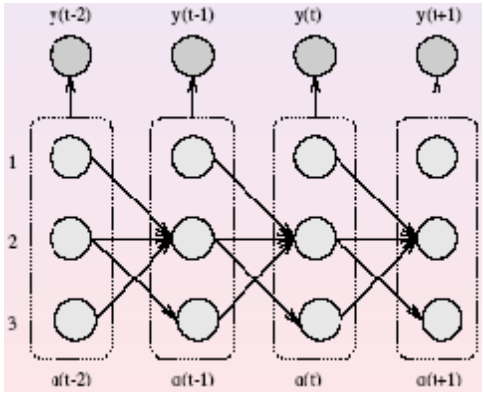
- مثال : هوای امروز را با فرض این که وابسته به هوای روز گذشته می باشد ، در نظر بگیرید .

۲. مدل های مخفی مارکوف

- یک مدل مخفی مارکوف :



- یک مدل مخفی مارکوف نمایش داده شده در زمان :



اجزای یک HMM

- مدل مخفی مارکوف : مدل مارکوف وضعیت آن مشاهده نمی شود ، اما از هر وضعیت می تواند جلوه ای را مشاهده نمود (یک متغیر x_t فقط به q_t وابسته می باشد) .
- یک تعداد محدود از وضعیت های N .
- احتمال های انتقال میان وضعیت ها ، که فقط به وضعیت قبلی وابسته می باشد : $P(q_t=i|q_{t-1}=j,\theta)$.
- انتشار احتمال ها ، که فقط وابسته به وضعیت جاری می باشد : $p(x_t|q_t=i, \theta)$ (در زمانی که x_t مشاهده می شود) .
- احتمال های وضعیت اولیه : $P(q_0=i|\theta)$.
- هر کدام از این سه مجموعه از احتمال ها دارای پارامترهای θ برای تخمین زدن می باشند .

سه اشکال HMM ها

- مدل HMM دارای سه مساله ی ریشه ای متفاوت می باشد :
- دارای یک HMM پارامتربندی شده توسط θ می باشد ، آیا می توانیم احتمال یک رشته ی $X = x_1^T = \{x_1, x_2, ..., x_T\}$ را محاسبه نماییم : $p(x_1^T \mid \theta)$
- با داشتن یک HMM پارامتربندی شده توسط θ و یک مجموعه از رشته های D_n ، آیا می توانیم پارامترهای θ^* ای که دارای شرط زیر باشند را انتخاب نماییم :

$$\theta^* = \arg\max_{\theta} \prod_{p=1}^{n} p(X(p) \mid \theta)$$

- با داشتن یک HMM پارامتربندی شده توسط θ ، آیا ما می توانیم مسیر بهینه ی Q را در میان فضای حالت داده شده توسط یک رشته ی X ، محاسبه نماییم :

$$Q^* = \arg\max_{Q} p(X, Q \mid \theta)$$

HMM ها به صورت پردازش های تولیدی

HMM ها می توانند برای تولید رشته ها ، استفاده شوند :

- اجازه دهید که یک مجموعه از وضعیت های شروع با احتمال های اولیه ی $P(q_0=i)$ را تعریف نماییم .
- همچنین اجازه دهید که یک مجموعه از وضعیت های پایانی را تعریف کنیم .
- سپس برای تولید هر رشته :
  ۱. یک وضعیت اولیه ی j را با توجه به $P(q_0)$ انتخاب نمایید .
  ۲. وضعیت بعدی i را با توجه به $P(q_t=i|q_{t-1}=j)$ انتخاب نمایید .
  ۳. یک خروجی را با توجه به توزیع انتشار $P(x_t|q_t=i)$ به وجود آورید .
  ۴. در صورتی که i یک وضعیت پایانی می باشد ، در این صورت متوقف شوید ، در غیر این صورت حلقه ای را با رفتن به مرحله ی ۲ تشکیل دهید .

فرض های مارکوفی

- خروج ها : احتمال خروج x_t در زمان t در وضعیت $q_t=i$ وابسته به هیچ چیز دیگری نمی باشد :

$$p(x_t \mid q_t = i, q_1^{t-1}, x_1^{t-1}) = p(x_t \mid q_t = i)$$

- انتقال ها : احتمال رفتن از وضعیت j به وضعیت i در زمان t وابسته به هیچ چیز دیگری نمی باشد :

$$P(q_t = i \mid q_{t-1} = j, q_1^{t-2}, x_1^{t-1}) = P(q_t = i \mid q_{t-1} = j)$$

- به علاوه ، این احتمال ، وابسته به زمان t نمی باشد :

$P(q_t = i \mid q_{t-1} = j)$ برای همه ی t ها یکسان می باشد

- ما می گوییم که این جور مدل ها ، مشابه هستند .

اشتقاق متغیر مستقیم α

احتمال داشتن رشته ی x_1^t و بودن در وضعیت i در زمان t :

$$\alpha(i,t) \overset{def}{=} p(x_1^t, q_t = i)$$
$$= p(x_t \mid x_1^{t-1}, q_t = i)\, p(x_1^{t-1}, q_t = i)$$

$$= p(x_t \mid q_t = i)\sum_j p(x_1^{t-1}, q_t = i, q_{t-1} = j)$$

$$= p(x_t \mid q_t = i)\sum_j P(q_t = i \mid x_1^{t-1}, q_{t-1} = j)\, p(x_1^{t-1}, q_{t-1} = j)$$

$$= p(x_t \mid q_t = i)\sum_j P(q_t = i \mid q_{t-1} = j)\, p(x_1^{t-1}, q_{t-1} = j)$$

$$= p(x_t \mid q_t = i)\sum_j P(q_t = i \mid q_{t-1} = j)\alpha(j, t-1)$$

از α تا احتمال

- یادآوری : $\alpha(i,t) \overset{def}{=} p(x_1^t, q_t = i)$
- وضعیت اولیه :

$\alpha(i,0) = P(q_0 = i)$ ← احتمال های قبلی هر وضعیت i

- سپس ، اجازه دهید که $\alpha(i,t)$ را برای هر وضعیت i و هر زمان t یک رشته ی داده شده ی x_1^T محاسبه نماییم
- پس از آن ، ما می توانیم احتمال را به صورت زیر محاسبه نماییم :

$$p(x_1^T) = \sum_i p(x_1^T, q_T = i) = \sum_i \alpha(i,T)$$

- بنابراین ، برای محاسبه ی $p(x_1^T)$ ، ما به $o(N^2.T)$ عملیات نیاز داریم ، که N تعداد وضعیت ها می باشد

آموزش EM برای HMM

- برای HMM ، متغیر پنهان Q توصیف خواهد کرد که در کدام وضعیت HMM برای هر مشاهده ی x_t از یک رشته ی X می باشد .
- بنابراین ، احتمال پیوسته ی همه ی رشته های $X(l)$ و متغیر پنهان Q برابر است با :

$$p(X, Q \mid \theta) = \prod_{l=1}^{n} p(X(l), Q \mid \theta)$$

- بیایید متغیر نمایش دهنده ی زیر را معرفی نماییم :

$q_{i,t}$ در صورتی که q_t برابر i باشد ، برابر ۱ خواهد بود و در غیر این صورت برابر با صفر خواهد بود .

احتمال پیوسته

حالا بیایید از متغیرهای نمایش دهنده ی q برای نمونه سازی Q استفاده نماییم :

$$p(X, Q \mid \theta) = \prod_{i=1}^{n} p(X(l), Q \mid \theta) = \sum_{l=1}^{n} \left(\prod_{i=1}^{N} P(q_0 = i)^{q_{i,0}} \right).$$

$$\prod_{t=1}^{T_l} \prod_{i=1}^{N} p(x_t(l) \mid q_t = i)^{q_{i,t}} \prod_{j=1}^{N} P(q_t = i \mid q_{t-1} = j)^{q_{i,t} \cdot q_{j,t-1}}$$

احتمال پیوسته ی لگاریتم

$$\log p(X, Q \mid \theta) = \sum_{l=1}^{n} \sum_{i=1}^{N} q_{i,0} \log P(q_0 = i) +$$

$$\sum_{l=1}^{n}\sum_{t=1}^{T_l}\sum_{i=1}^{N} q_{i,t}\log p(x_t(l)\mid q_t=i)+$$

$$\sum_{l=1}^{n}\sum_{t=1}^{T_l}\sum_{i=1}^{N}\sum_{j=1}^{N} q_{i,t}.q_{j,t-1}\log P(q_t=i\mid q_{t-1}=j)$$

تابع کمکی

بیایید حالا تابع کمکی مناسب را بنویسیم :

$$A(\theta,\theta^s)=E_Q\left[\log p(X,Q\mid\theta)\mid X,\theta^s\right]$$

$$=\sum_{l=1}^{n}\sum_{i=1}^{N} E_Q[q_{i,0}\mid X,\theta^s]\log)P(q_0=i)+$$

$$\sum_{l=1}^{n}\sum_{t=1}^{T_1}\sum_{i=1}^{N} E_Q[q_{i,t}\mid X,\theta^s]\log p(x_t(l)\mid q_t=i)+$$

$$\sum_{l=1}^{n}\sum_{t=1}^{T_1}\sum_{i=1}^{N}\sum_{j=1}^{N} E_Q[q_{i,t}.q_{j,t-1}\mid X,\theta^s]\log P(q_t=i\mid q_{t-1}=j)$$

از حالا بیایید برای سادگی از اندیس l صرف نظر نماییم .

اشتقاق معکوس متغیر β

احتمال به وجود آمدن باقی مانده ی رشته ی x_{t+1}^T با دانستن این که ما در وضعیت i و در زمان t هستیم :

$$\beta(i,t)\overset{def}{=}p(x_{t+1}^T\mid q_t=i)$$

$$=\sum_j p(x_{t+1}^T,q_{t+1}=j\mid q_t=i)$$

$$=\sum_j p(x_{t+1}\mid x_{t+2}^T,q_{t+1}=j,q_t=i)\,p(x_{t+2}^T,q_{t+1}=j\mid q_t=i)$$

$$=\sum_j p(x_{t+1}\mid q_{t+1}=j)\,p(x_{t+2}^T\mid q_{t+1}=j,q_t=i)P(q_{t+1}=j\mid q_t=i)$$

$$=\sum_j p(x_{t+1}\mid q_{t+1}=j)\,p(x_{t+2}^T\mid q_{t+1}=j)P(q_{t+1}=j\mid q_t=i)$$

$$=\sum_j p(x_{t+1}\mid q_{t+1}=j)\beta(j,t+1)P(q_{t+1}=j\mid q_t=i)$$

جزئیات نهایی در مورد β

- باقی مانده : $\beta(i,t)=p(x_{t+1}^T\mid q_t=i)$
- وضعیت نهایی : β(i,t) در صورتی که i یک وضعیت پایانی باشد برابر با صفر می باشد و در غیر این صورت برابر با صفر می باشد
- بنابراین ، برای محاسبه ی همه ی متغیرهای β ، ما به $o(N^2.T)$ عمل نیاز داریم ، که N تعداد وضعیت ها می باشد

گام E برای HMM ها

- قبلی در توزیعات خروجی :

$$E_Q[q_{i,t} \mid X, \theta^s] = P(q_t = i \mid x_1^T, \theta^s) = P(q_t = i \mid x_1^T)$$
$$= \frac{p(x_1^T, q_t = i)}{p(x_1^T)}$$
$$= \frac{p(x_{t+1}^T \mid q_t = i, x_1^t)\, p(x_1^t, q_t = i)}{p(x_1^T)}$$
$$= \frac{p(x_{t+1}^T \mid q_t = i)\, p(x_1^t, q_t = i)}{p(x_1^T)}$$
$$= \frac{\beta(i,t).\alpha(i,t)}{\sum_j \alpha(j,T)}$$

- قبلی در توزیع های انتقال :

$$E_Q[q_{i,t}.q_{j,t-1} \mid X, \theta^s] = P(q_t = i, q_{t-1} = j \mid x_1^T, \theta^s)$$
$$= \frac{p(x_1^T, q_t = i, q_{t-1} = j)}{p(x_1^T)}$$
$$= \frac{p(x_{t+1}^T \mid q_t = i) P(q_t = i \mid q_{t-1} = j)\, p(x_t \mid q_t = i)\, p(x_1^{t-1}, q_{t-1} = j)}{p(x_1^T)}$$
$$= \frac{\beta(i,t) P(q_t = i \mid q_{t-1} = j)\, p(x_t \mid q_t = i) \alpha(j, t-1)}{\sum_j \alpha(j,T)}$$

- قبلی در توزیع وضعیت اولیه :

$$E_Q[q_{i,0} \mid X, \theta^P) = P(q_0 = i \mid x_1^T, \theta^s) = P(q_0 = i \mid x_1^T)$$
$$= \frac{p(x_1^T, q_0 = i)}{p(x_1^T)}$$
$$= \frac{p(x_1^T \mid q_0 = i) P(q_0 = i)}{p(x_1^T)}$$
$$= \frac{\beta(i,0).P(q_0 = i)}{\sum_j \alpha(j,T)}$$

گام M برای HMM ها

- پارامترهای θ را که A را بیشینه می کند پیدا نمایید ، بنابراین جستجو می کنیم برای :

$$\frac{\partial A}{\partial \theta} = 0$$

- در زمانی که توزیع های انتقال به صورت جدول هایی ارایه شده اند ، با استفاده از یک تکثیر کننده ی لاگرانژ داریم :

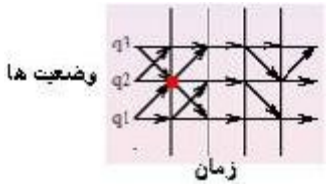
$$P(q_t = i \mid q_{t-1} = j) = \frac{\sum_{t=1}^{T} P(q_t = i, q_{t-1} = j \mid x_1^T, \theta^s)}{\sum_{t=1}^{T} P(q_{t-1} = j \mid x_1^T, \theta^s)}$$

- در زمان انتشار توزیع ها به صورت GMM ها پیاده سازی می شوند ، از معادلاتی که قبلا داده شده که توسط انتشار های قبلی $P(q_t=i|x_1^T,\theta^s)$ وزن شده اند ، استفاده نمایید .

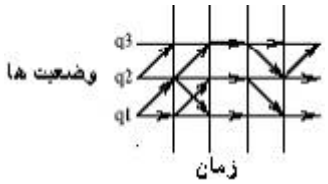
محتمل ترین مسیر (نمایش تصویری)

- الگوریتم ویتربی ، بهترین رشته ی وضعیت را پیدا می نماید .

مسیر های جزئی را محاسبه نمایید



به موقع ، به عقب برگشت نمایید



الگوریتم ویتربی برای HMM ها

الگوریتم ویتربی ، بهترین رشته ی وضعیت را پیدا می کند .

$$V(i,t) \overset{def}{=} \max_{q_1^{t-1}} p(x_1^t, q_1^{t-1}, q_t = i)$$

$$= \max_{q_1^{t-1}} p(x_t \mid x_1^{t-1}, q_1^{t-1}, q_t = i)\, p(x_1^{t-1}, q_1^{t-1}, q_t = i)$$

$$= p(x_t \mid q_t = i) \max_{q_1^{t-2}} \max_{j} p(x_1^{t-1}, q_1^{t-2}, q_t = i, q_{t-1} = j)$$

$$= p(x_t \mid q_t = i) \max_{q_1^{t-2}} \max_{j} p(q_t = i \mid q_{t-1} = j)\, p(x_1^{t-1}, q_1^{t-2}, q_t = i, q_{t-1} = j)$$

$$= p(x_t \mid q_t = i) \max_{j} p(q_t = i \mid q_{t-1} = j) \max_{q_1^{t-2}} p(x_1^{t-1}, q_1^{t-2}, q_{t-1} = j)$$

$$= p(x_t \mid q_t = i) \max_{j} p(q_t = i \mid q_{t-1} = j) \vee (j, t-1)$$

از ویتربی تا رشته ی وضعیت

- یادآوری : $V(i,t) = \max_{q_1^{t-1}} p(x_1^t, q_1^{t-1}, q_t = i)$
- بیایید V(i,t) را برای هر وضعیت i و هر زمان t یک رشته ی داده شده ی x_1^T محاسبه نماییم
- به علاوه ، بیایید برای هر V(i,t) ، argmax وابسته ی وضعیت قبلی j را نگهداری نماییم
- سپس از وضعیت $i = \arg\max_j V(j,T)$ برای کد کردن محتمل ترین رشته ی وضعیت به عقب برمی گردیم .
- بنابراین ، برای محاسبه ی همه متغیرهای V(i,t) ، ما به $o(N^2.T)$ عمل نیازداریم ، که N تعداد وضعیت ها می باشد

کاربردهای HMM ها

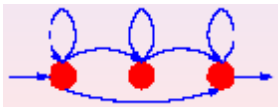
- طبقه بندی رشته هایی نظیر

- رشته های DNA (کدام خانواده)
- رشته های حرکت
- رشته های تصویری
- رشته های صوتی
- و غیره

- رمزگشایی کردن رشته هایی نظیر
  - سخن شناسی پیوسته
  - دست خط شناسی
  - رشته ی رویدادها (ملاقات (جلسه) ، نظارت ، بازی ها و ...)

۳. کاربرد HMM ها برای سخن شناسی

- کاربرد : سخن شناسی پیوسته :
  یک رشته از صوت ها (کلمه ها) درون یک رشته ی صوتی را پیدا نمایید
- ایده : از مدل صوتی استفاده نمایید

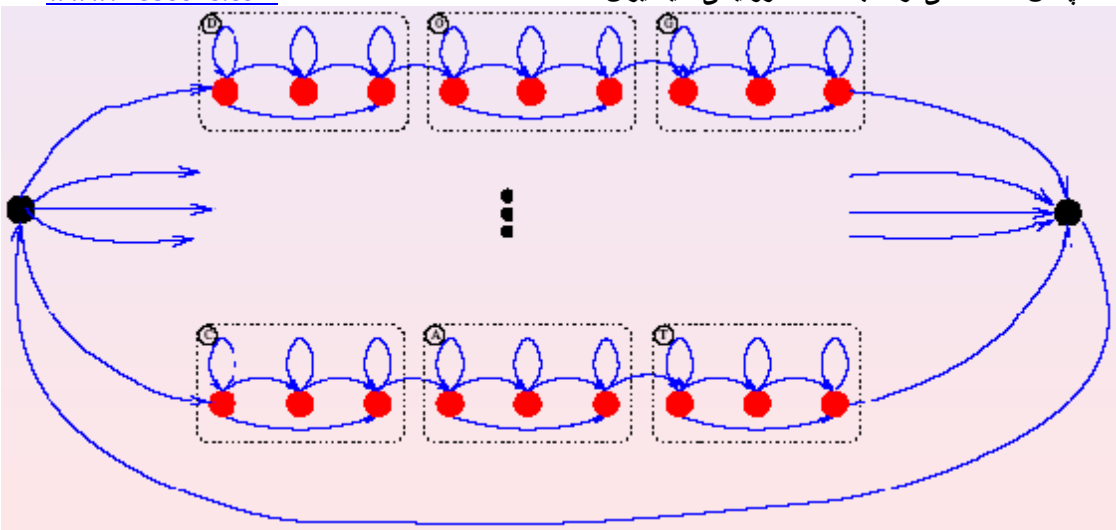


آموزش به کار رفته در HMM ها

- برای هر رشته ی صوتی در مجموعه ی آموزشی ، یک HMM جدید به صورت الحاق HMM های ارایه کننده ی رشته ی در زیر قرار گرفته از صوت ها بسازید .
- احتمال رشته های آموزشی را بیشینه نمایید .

HMM ها : رمزگشایی یک عبارت

- تصمیم بگیرید که لغت پذیرفته شده چیست .
- به صورت اختیاری یک مدل زبان را اضافه نمایید : (رشته ی کلمه) P
- الگوریتم مناسب برای پیدا کردن مسیر بهینه در رمزگشایی HMM :



اندازه گیری خطا

- چگونه ما کیفیت یک شناسایی کننده ی سخن را اندازه گیری نماییم ؟
- مساله : راه حل مقصد ، یک عبارت است ، راه حل داده شده هم یک عبارت می باشد ، اما ممکن است دارای اندازه ی متفاوت باشند !
- راه حل پیشنهاد شده : ویرایش فاصله :
  - فرض کنید که شما به عملگرهای insert ، delete و substitute دسترسی داشته باشید ،
  - کوچک ترین تعداد عملگرهای این چنینی که ما برای رفتن از عبارت داده شده به عبارت مقصد نیاز داریم چیست ؟
  - یک الگوریتم مناسب برای محاسبه ی این وجود دارد .
- در نهایت ما خطا را به صورت زیر اندازه گیری می کنیم :

$$WER = \frac{\#ins+\#del+\#subst}{\#words}$$

- توجه نمایید که نرخ خطای کلمه [1] می تواند بزرگ تر از یک باشد

بیشینه ی اطلاعات دوطرفه

- استفاده از بیشینه ی معیار احتمال برای یک کار طبقه بندی ممکن است برخی اوقات بدتر از استفاده از یک روش دارای تبعیض باشد
- در مورد تغییر معیار برای تبعیض آمیز تر بودن چطور ؟
- بیشینه ی اطلاعات دوطرفه [2] میان کلمه (W) و رشته های صوتی :

$$I(A,W) = \log \frac{P(A,W)}{P(A)P(W)}$$

$$=\log P(A|W)P(W)-\log P(A)-\log P(W)$$

$$=\log P(A|W)-\log P(A)$$

$$= \log P(A \mid W) - \sum_{w} \log P(A \mid w)P(w)$$

[1] Word Error Rate (WER)
[2] Maximum Mutual Information (MMI)

- گرادیان صعودی را به کار ببرید : $\frac{\partial I(A,W)}{\partial \theta}$

۴. صورت های عملی

- ظرفیت توسط فرا پارامترهای زیر تنظیم می شود :
  - تعداد وضعیت ها (یا مقدارهای متغیر پنهان که می تواند بگیرد)
  - انتقال های غیر صفر (اتصال کامل ، چپ به راست و ...)
  - ظرفیت مدل های انتشاری که در زیر آمده
  - تعداد تکرارهای آموزشی
- مقداردهی اولیه :
  - در صورتی که مجموعه ی آموزشی تنظیم شده است ، این اطلاعات را استفاده نمایید
  - در غیر این صورت ، برای انتقال ها و K میانگین ها برای انتشارهای GMM گرا ، یکسان سازی نمایید
- کنترینت محاسباتی [1] :
  - در دامنه ی لگاریتمی کار کنید !

عدم تعادل میان وضعیت ها و انتشارها

- یک مساله اغلب در سخن شناسی دیده شده است
- رمز گشایی با ویتربی :

$$V(i,t) = p(x_t \mid q_t = i) \max_j P(q_t = i \mid q_{t-1} = j) \vee (j, t-1)$$

- خروج های ارایه شده توسط GMM ها : چگالی ها وابسته به تعداد ابعاد x_t می باشد .
- تخمین های عملی در پایگاه داده ی نامبرز ۹۵ (۳۹ بعد) :

	واریانس
Log $P(q_t \mid q_{t-1})$	۹,۸
Log $p(x_t \mid q_t)$	۱۱۴۸۶,۰

مقایسه ی واریانس های توزیعات log در مدت رمزگشایی

پشتیبانی از ماشین های برداری [2]

۱. پشتیبانی از ماشین های برداری خطی
۲. هسته های پشتیبانی از ماشین های برداری غیرخطی
۳. آموزش پشتیبانی از ماشین های برداری
۴. سایر روش های هسته

---

[1] computational contraint
[2] Support Vector Machines (SVM)

۱. پشتیبانی از ماشین های برداری خطی

برپاکردن

- مجموعه ی آموزشی :

$$(x_i, y_i)_{i=1...n} \in R^d \times \{-1,1\}$$

- ما مایلیم که یک فراطرح [1] را پیدا نماییم

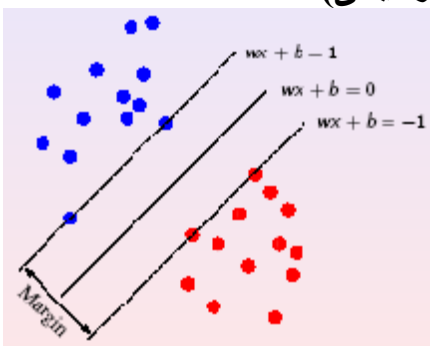
$$wx + b = 0 \, (w \in R^d, b \in R)$$

که دو کلاس را متمایز می نماید

حاشیه [2]

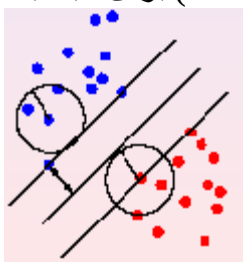
- تصور نمایید d_+ کوتاه ترین فاصله از فراطرح به نزدیک نرین نمونه ی مثبت باشد .
- تصور نمایید d_- کوتاه ترین فاصله از فراطرح به نزدیک نرین نمونه ی منفی باشد .
- حاشیه ی فراطرح را $d_+ + d_-$ تعریف نمایید
- ساده ترین SVM به دنبال مجزاکننده ی فراطرح با بیش ترین حاشیه می باشد .

SVM و حاشیه (نمایش گرافیکی)



چرا بیشینه نمودن حاشیه مفید می باشد ؟

- چند توجیه برای طرفداری کردن از حاشیه های بزرگ وجود دارد ؛ برای مثال :
- در صورتی که آموزش و تست داده ها از توزیع یکسان باشند و همه ی داده های تستی در برخی فاصله ی Δ از نقطه های آموزشی باشند
- در این صورت یک حاشیه ی $(2.\Delta)$ برای طبقه بندی همه ی نقطه ها مناسب می باشد :



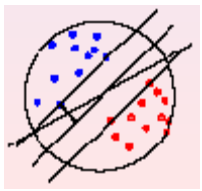
- در صورتی که همه ی نقاط در یک فاصله ی لااقل Δ از مجزاکننده قرار گیرند و همه ی نقطه ها در یک کره ی محدود شده باشند ، آن گاه یک اختلال کوچک تعریف مجزاکننده زیانی را وارد نخواهد کرد .

---

[1] hyperplane
[2] margin

- بنابراین می تواند از قسمت های کم تری برای رمزگشایی مجزاکننده ی فراطرح استفاده نماید .
- این وابسته به اصل کمینه ی توصیف طول [1] می باشد :

بهترین توصیف داده ها در موارد تعمیم خطا ، باید به کم ترین قسمت ها برای نگهداری نیاز داشته باشد .



فرمول بندی مساله ی SVM

- ما می توانیم محدودیت های زیر را تعریف نماییم :

$y_i=+1$ برای $wx_i + b \geq +1$

$y_i=-1$ برای $wx_i + b \leq -1$

- این ها می توانند به صورت زیر ترکیب شوند :

$$y_i(wx_i + b) - 1 \geq 0 \quad \forall i$$

- می توان نشان داد که $d_+ = d_- = \frac{1}{\|w\|}$ که $\|w\|$ نرم اقلیدسی w می باشد . بنابراین ، حاشیه برابر است با : $\frac{2}{\|w\|}$
- بنابراین ما تمایل داریم که $\frac{\|w\|^2}{2}$ کمینه نماییم .

با محدودیت های $y_i(wx_i + b) - 1 \geq 0 \quad \forall i$

یک مساله ی بهینه سازی محدود شده

- روشی نرمال برای حل یک مساله ی بهینه سازی با هزینه ی C(w) و پارامتر w : $\frac{\partial C}{\partial w} = 0$. مثال : $C(w) = \frac{w^2}{2} - 3w$ را کمینه نمایید .

بنابراین $\frac{\partial C}{\partial w} = w - 3 = 0 \Rightarrow w = 3$

- در موقعی که محدودیت های $c_i \geq 0$ وجود دارد ، تکثیرکننده های زبان استفاده نمایید و راه حل را با شرط های Karush-Kuhn-Tucker (KKT) بازنگری نمایید .
- شکل لاگرانژ با تفریق یک عبارت برای هر محدودیت $c_i \geq 0$ ، وزن شده با تکثیرکننده ی مثبت زبان :

$$L(w, \alpha) = C(w) - \sum_i \alpha_i c_i$$

- ما حالا باید L را با در نظرگرفتن w در ارتباط با موارد زیر ، کمینه نماییم

[1] Minimum Description Length principle (MDL)

- $\frac{\partial L}{\partial \alpha_i} = 0$
- $\forall i \quad \alpha_i \geq 0$

- ما می توانیم به طور معادل ، مساله ی دوگانه را حل نماییم : L را با در نظر گرفتن α در روابط زیر بیشینه نمایید
  - $\frac{\partial L}{\partial w} = 0$
  - $\forall i \quad \alpha_i \geq 0$
- مساله ی کلی برای پیدا کردن یک نقطه ی زینی :

$$\max_{\alpha} \min_{w} L(w,\alpha)$$

فرمول بندی لاگرانژی برای SVM ها

- ما یک تکثیرکننده ی لاگرانژی α_i که i=1,…,n می باشد را معرفی می کنیم ، برای هر نامساوی محدود :

$$L(w,b,\alpha) = \frac{\|w\|^2}{2} - \sum_{i=1}^{n} \alpha_i (y_i (wx_i + b) - 1)$$

- L باید با توجه به متغیرهای اصلی w و b بیشینه شده با توجه به متغیرهای دوگانی α_i کمینه شود .
- در اکسترمم (حداکثر یا حداقل تابع ریاضی) [1] ، ما داریم :

$$\frac{\partial L}{\partial b} = 0 \quad \text{و} \quad \frac{\partial L}{\partial w} = 0$$

حل لاگرانژ

- ما داریم :

$$L = \frac{\|w\|^2}{2} - \sum_{i=1}^{n} \alpha_i (y_i (wx_i + b) - 1)$$

- ما می خواهیم $\frac{\partial L}{\partial w} = 0$:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0$$

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

- ما می خواهیم $\frac{\partial L}{\partial b} = 0$:

$$\frac{\partial L}{b} = \sum_{i=1}^{n} \alpha_i y_i = 0$$

[1] extremum

جانشین کردن برای به دست آوردن دوگانی

$$L = \frac{\|w\|^2}{2} - \sum_{i=1}^{n} \alpha_i \left(y_i (wx_i + b) - 1 \right)$$

$$= \frac{\sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j}{2} - \sum_{i=1}^{n} \alpha_i \left(y_i \left(\sum_{j=1}^{n} \alpha_j y_j x_j x_i + b \right) - 1 \right)$$

$$= \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j - \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_j x_i - b \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i$$

$$= -\frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j + \sum_{i=1}^{n} \alpha_i$$

$$L = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j$$

فرمول بندی دوگانی

- ما باید عبارت زیر را بیشینه نماییم :

$$L = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j$$

- در ارتباط با

$$\alpha_i \geq 0, \forall i \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

- این می تواند با استفاده از بسته های بهینه سازی برنامه ریزی غیرخطی کلاسیکی ، برمبنای نمونه ی محدودشده ی گرادیان کاهشی حل شود .

عبارت های KKT

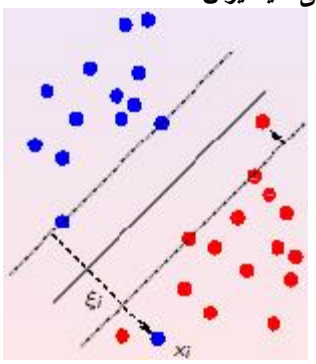
عبارت های KKT زیر در راه حل راضی شده اند .

- $\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0$
- $\frac{\partial L}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0$
- $y_i (wx_i + b) - 1 \geq 0, \forall i$
- $\alpha_i \geq 0, \forall i$
- $\alpha_i (y_i (wx_i + b) - 1) = 0, \forall i$

این می تواند برای تخمین زدن b ای که بعد از w در مدت آموزش پیدا شده است استفاده شود .

یک اشکال

کمینه سازی مساله در صورتی که دو کلاس به صورت مجزا نباشند دارای هیچ راه حلی نمی باشد .



برطرف کردن اشکال : حاشیه ی "نرم"

- محدودیت ها را آرام نمایید : از یک حاشیه ی نرم به جای یک حاشیه ی سخت استفاده نمایید .
- ما می خواهیم $\frac{\|w\|^2}{2} + C\sum_{i=1}^{n}\xi_i$ را تحت محدودیت های $y_i(wx_i + b) \geq 1 - \xi_i \quad \forall i, \xi_i \geq 0 \quad \forall i$ کمینه نماییم

فرمول بندی دوگانی غیرقابل تشخیص

- ما باید عبارت زیر را بیشینه نماییم :

$$L = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j x_i x_j$$

- در رابطه با

$$0 \leq \alpha_i \leq C, \quad \forall i,...,n$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

- سپس ما w و b را به صورت زیر به دست می آوریم :

$$w = \sum_i \alpha_i y_i x_i$$

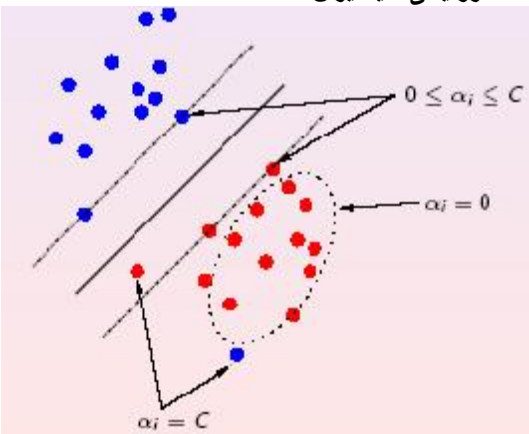
$$\alpha_i[1 - \xi_i - y_i(wx_i + b)] = 0$$

لغات بردار پشتیبانی

- توجه کنید که تابع تصمیم گیری می تواند به این صورت بازنویسی شود :

$$\hat{y} = sign\left(\sum_i \alpha_i y_i x_i x + b\right)$$

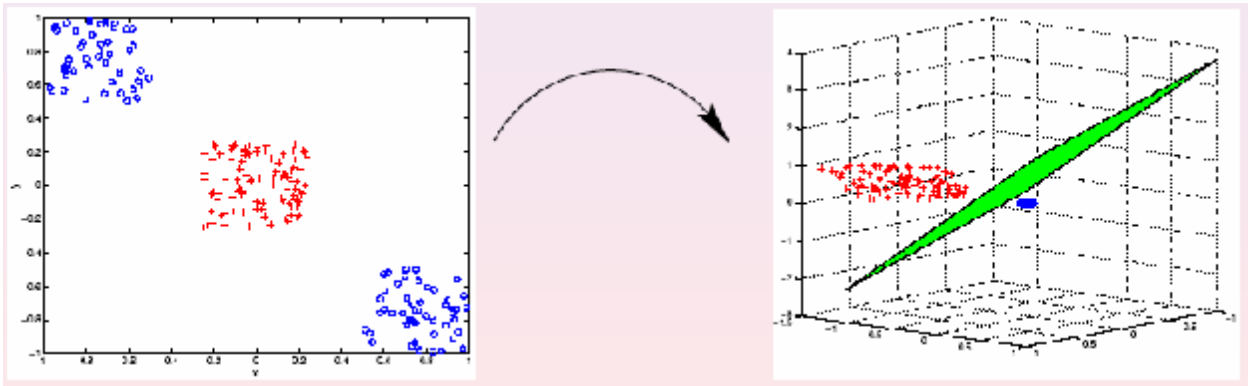
- نمونه های آموزشی x_i با $\alpha_i \neq 0$ بردارهای پشتیبانی هستند .



۲. هسته های پشتیبانی از ماشین های برداری غیرخطی

SVM های غیرخطی

- داده ها را در یک فضای ابعادی بالاتر طرح نمایید : مجزا کردن دو کلاس باید آسان تر باشد .
- با داشتن یک تابع $\phi : R^d \to F$ با $\phi(x_i)$ به جای کارکردن با x_i کار کنید .



فوت و فن هسته

- توجه کنید که ما فقط ضرب های نقطه ای $\phi(x_i)\phi(x_j)$ را برای محاسبه داریم .
- متاسفانه ، این می تواند در یک فضای ابعادی بالا خیلی گران باشد .
- به جای هسته استفاده نمایید : یک تابع $k(x,z)$ که یک ضرب نقطه ای در محیط با خصوصیت پنهان را ارایه می نماید .

$$k(x,z) = \phi(x)\phi(z)$$

- مثال : به جای

$$\phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

از $k(x,z)=(xz)^2$ استفاده نمایید

هسته های عمومی

- چندجمله ای :

$$k(x,z) = (uxz)^p \quad (u \in R, v \in R, p \in N_+^*)$$

- گاوسی :

$$k(x,z)=\exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)\quad(\sigma\in R_+^*)$$

- *تابع $k(x,z)=\tanh(uxz+v)$ یک هسته نمی باشد !*

عبارت مرکز

- کدام توابع هسته اند ؟؟؟
- با یک نگاشت ϕ و یک گسترش

$$k(x,z)=\sum_i \phi(x)_i\phi(x)_i$$

- اگر و فقط اگر برای هر g(x) که $\int g(x)^2dx$ محدود باشد آن گاه $\int k(x,z)g(x)g(z)dxdz\geq 0$
- در عمل ، یک هسته افزایشی را به یک ماتریس نیم نامحدود ارایه می نماید (مثال یک ماتریس شباهت متقارن)

راه حل نهایی

- $\sum_{i=1}^n\alpha_i-\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n\alpha_i\alpha_j y_i y_j k(x_i,x_j)$ را تحت محدودیت های $0\leq\alpha_i\leq C$ و $\sum_i\alpha_i y_i=0$ بیشینه نمایید .
- برای $0<\alpha_i<C$ ، b را با استفاده از $1-y_i\left[\sum_j\alpha_j y_j k(x_j,x_i)+b\right]=0$ محاسبه نمایید
- تابع تصمیم گیری : $\hat{y}=sign\left(\sum_i\alpha_i y_i k(x_i,x)+b\right)$

عبارت های KKT

عبارت های KKT زیر در راه حل ، ارضا می شوند .

- $y_i(wx_i+b)-1\geq 0,\quad\forall i$
- $y_i(wx_i+b)=1$ در رابطه با $\forall i$ ، $0<\alpha_i<C$
- $y_i(wx_i+b)\leq 1$ در رابطه با $\forall i$ ، $\alpha_i=C$
- و توجه کنید که $\alpha_i=0$ برای همه ی بردارهای غیر پشتیبان

واقعیت هایی برای به خاطر سپردن

- SVM ها حاشیه را در فضای ویژگی ، بیشینه می نمایند
- از روش حاشیه ی نرم استفاده نمایید
- داده ها را در یک فضای ابعادی بالاتر برای ارتباط های غیرخطی ، طرح نمایید
- هسته ها ، محاسبه را ساده می کنند
- یک روش لاگرانژی به یک مساله ی بهینه سازی درجه دوم تحت محدودیت ها منجر می شود .

SVM ها در عمل

- برای تنظیم ظرفیت ، هسته ، مهم ترین پارامتر برای انتخاب می باشد

- هسته ی چندجمله ای : افزایش درجه ، باعث افزایش ظرفیت خواهد شد .
- هسته ی گاوسی : افزایش σ ، ظرفیت را کاهش خواهد داد .

- C را تنظیم نمایید و میان حاشیه و خطاها سبک سنگین نمایید .
  - برای مجموعه های داده ای غیر پراغتشاش ، C معمولا دارای اثر زیادی نمی باشد .
  - C را بادقت برای مجموعه های داده ای پراغتشاش انتخاب نمایید : مقدار های کوچک ، معمولا نتایج بهتری را ارایه می نمایند .

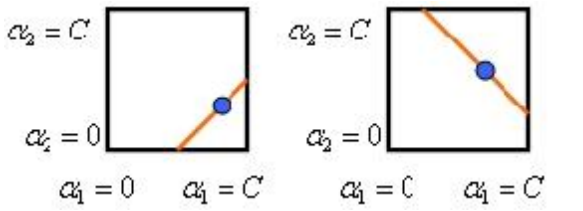
۳. آموزش پشتیبانی از ماشین های برداری

پیچیدگی مساله ی QP

- شما به n^2 فضا از حافظه برای نگهداری ماتریس هسته نیاز دارید !
- روش بهینه سازی ساده لااقل n^3 تا n^4 خواهد بود .
- در مورد مجموعه های داده ای با ۱۰۰۰۰۰ نمونه یا بیش تر چطور ؟؟؟
- روش های مختلفی پیشنهاد شده است :
  - چانک کردن [1] : در هر مرحله ، مساله ی QP را با همه ی α_i های غیر صفر ، از مرحله ی قبل و M نمونه ی بد مختلف عبارت های KKT حل نمایید .
  - تجزیه : یک سری از مسایل QP کوچک تر را حل نمایید ، که هر یک یک نمونه که عبارت های KKT را متمایز می کند را اضافه می نمایند .
  - بهینه سازی کمینه ی ترتیبی [2] : کوچک ترین مساله ی بهینه سازی را در هر تکرار حل نمایید .

چهارچوب SMO

- در هر مرحله ، دو تکثیرکننده ی زبان α_i را برای بهینه سازی با هم انتخاب نمایید . با لااقل یک عبارت KKT مختلف .
  چند فوت و فن برای انتخاب مختلف ترین ... وجود دارد
- مقدار بهینه را برای این دو α_i پیدا نمایید و مدل SVM را به روز نمایید .



$y_1 \neq y_2 \rightarrow \alpha_1 - \alpha_2 = k$ $\quad y_1 = y_2 \rightarrow \alpha_1 + \alpha_2 = k$

این روال به بهینه ها ، همگرایی پیدا می کند .

۴. سایر روش های هسته ای

یک باغ وحش از روش های هسته ای در ادبیات

- برای مسایل رگرسیون : از رگرسیون برداری پشتیبانی نمایید
- برای تخمین چگالی یا ارایه : PCA هسته ای
- برای مدل های تولیدی : هسته ی فیشر

---

[1] Chunking
[2] Sequential Minimal Optimization (SMO)

- برای رشته های گسسته : هسته ی رشته ای
- ...

چگونه یک هسته طراحی نماییم ؟ دانش قبلی !!!

- با انتخاب یک اندازه ی تشابه میان ۲ نمونه در داده ها
- با انتخاب یک ارایه ی خطی از داده ها
- با انتخاب یک فضای ویژگی برای یادگیری

درخت های تصمیم گیری

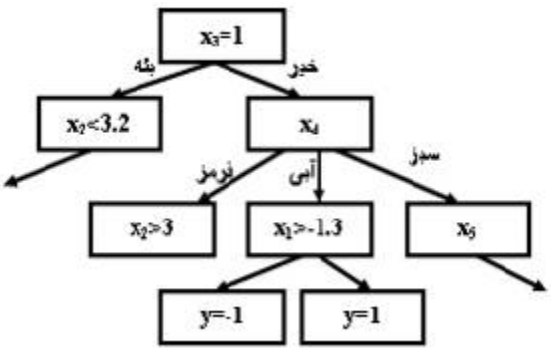
۱. مقدمه
۲. آموزش یک درخت تصمیم گیری
۳. کنترل ظرفیت یک درخت تصمیم گیری

۱. مقدمه

تصمیم گیری های قانون گرا

- تصور نمایید که D_n یک مجموعه ی آموزشی از n جفت (x^i,y^i) باشد
- تصور کنید که y یک کلاس (۱- یا ۱) باشد
- و x یک بردار مشخص کننده ی خصوصیات d باشد $\{x_j \mid 1 \le j \le d\}$.
- یک قانون تصمیم گیری می تواند به صورت
  اگر x_3 درست باشد و $x_5 \ge ۳,۲$ باشد آن گاه y=1
- چگونه یک تصمیم گیری این چنینی را به وجود بیاوریم ؟
- این خانواده ای از درخت های تصمیم گیری می باشد .

یک درخت تصمیم گیری



۲. آموزش یک درخت تصمیم گیری

آموزنده ترین ویژگی

- در هر گره داده شده ی i درخت ، ما دارای یک زیرمجموعه ی D_i از نمونه های آموزشی می باشیم .
- ما مایلیم که یک ویژگی j که ما D_i را به صورت عاقلانه با توجه به عملیات تقسیم می کنیم را انتخاب نماییم .

- هدف : بعد از قطعه بندی D_i ، همه ی نمونه های در یک گره باید به صورت ایده ال از کلاس یکسان باشند : این مطلب ، خلوص [1] نام دارد .
- چند مکاشفه برای حرکت به سمت خلوص وجود دارد .
- بیایید اول نگاهی به موردی برای ویژگی های گسسته داشته باشیم .
- انتخاب معمولا توسط بیشینه کردن سود اطلاعات انجام می شود .

سود اطلاعات

بیشینه کردن سود اطلاعات

$$IG_j=H(Y)-H(Y|X_j)$$

که $H(Y)$ آنتروپی (افت) متغیر تصادفی Y (میانگین تعداد قسمت های لازم برای انتقال Y) می باشد

$$H(Y) = -\sum_k P(Y=k)\log P(Y=k)$$

و $H(Y|X)$ افت شرطی می باشد

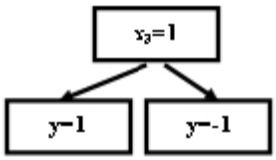
$$H(Y \mid X_j) = -\sum_m P(X_j=m)\sum_k P(Y=k \mid X_j=m)\log P(Y=k \mid X_j=m)$$

بنابراین ما انتخاب می کنیم

$$j^* = \arg\max_j \; IG_j$$

کنده های [2] تصمیم گیری

- یک کنده به طور ساده یک درخت تصمیم گیری یک سطحی می باشد
- بنابراین ، شما خصوصیت (مشخصی) که سود اطلاعات را در کل مجموعه ی داده ای تان را بیشینه می کند را انتخاب نمایید



مرحله ی بازگشت

- مجموعه ی داده ای اصلی را در نظر بگیرید
- با توجه به مقدار خصوصیتی که با توجه به آن تقسیم را انجام داده ایم ، آن را قطعه بندی نمایید
- برای هر قطعه ، یک کنده ی تصمیم گیری به وجود آورید
- و این کار را به صورت بازگشتی انجام دهید تا نمونه ی بیش تری برای تقسیم کردن وجود نداشته باشد
- شما یک درخت تصمیم گیری را دارید !

متغیرهای ورودی پیوسته

- برای متغیرهای ورودی غیر گسسته چه کار کنیم ؟
- به جای قطعه بندی با توجه به $X_j=k$

---

[1] purity
[2] Stumps

- مقدار $X_j=t$ را پیدا نمایید که

$$H(Y \mid X_{j,t}) = \max_t H(Y \mid X_j < t)P(X_j < t) + H(Y \mid X_j \geq t)P(X_j \geq t)$$

- سود اطلاعات را به طور معمول ، محاسبه نمایید :

$$IG_j = H(Y) - H(Y \mid X_{j,t})$$

- در صورتی که IG_j در میان ویژگی ها بیشینه می باشد ، X_j را با توجه به این که $X_j<t$ است یا نه قطعه بندی نمایید .

۳. کنترل ظرفیت یک درخت تصمیم گیری

- در صورتی که شما یک درخت کامل را به وجود آوردید ، توسط همه ی مجموعه ی آموزشی شما آموخته خواهد شد
- بنابراین ، ظرفیت به صورت نامحدود مجازی می باشد
- ما به ساخت درخت های با ظرفیت کنترل شده نیازمندیم
- برای این کار راه های مختلفی وجود دارد :
  - درخت کامل را به وجود نیاورید مگر این که قبلا متوقف شده باشید
  - درخت کامل را بسازید و سپس آن را هرس نمایید

انتخاب ویژگی

۱. انگیزه (محرک) [1]
۲. فیلترها
۳. بسته بندی کننده ها [2]
۴. وزن کردن ویژگی

۱. انگیزه

چرا ما ویژگی ها را انتخاب می کنیم ؟

- برخی از مسایل توسط ۱۰۰ یا حتی ۱۰۰۰ ویژگی ورودی تعریف می شوند
- بیش تر روش های آموزش ماشینی باید پارامترهایی را برای به کارگیری این خصوصیت ها مشخص نمایند (اغلب لااقل به همان اندازه و به صورت خطی)
- بنابراین ، ظرفیت توسط تعداد ویژگی ها تشخیص داده می شود
- در صورتی که بیش تر ویژگی ها دارای اغتشاش می باشند ، بنابراین بیش تر پارامتر ها بی استفاده خواهند بود ، در نتیجه ظرفیت به صورت بیهوده تلف می شود
- بدتر ، این که الگوریتم ممکن است در ویژگی های داده های آموزشی دارای قاعده ی غلطی بشود و از ظرفیتی دارای اتلاف شده برای نمایش آن ها استفاده نماید !
- مساله ی دیگر : بدی اندازه
- در نهایت : برای قابلیت تفسیر و کارایی بیش تر

کلاس های روش های انتخاب ویژگی

---

[1] motivation
[2] wrappers

روش هایی گسترده وجود دارد :

- روش های فیلتر :
  - بهترین ویژگی ها را با توجه به یک ملاک معقولانه انتخاب نمایید
  - ملاک ، مستقل از مساله ی واقعی می باشد
- روش های بسته بندی :
  - بهترین ویژگی ها را با توجه به معیار نهایی ، انتخاب نمایید
  - برای هر زیر مجموعه از ویژگی ها ، تلاش نمایید که مساله را حل نمایید
- در هر مورد ، $\sum_{p=1}^{n} C_n^p = \sum_{p=1}^{n} C_n^p = \sum_{p=1}^{n} \frac{n!}{p!(1-p)!}$ ترکیب وجود دارد
- چاره : روش های وزن کننده

۲. فیلترها

روش های فیلتر

- ایده ی اساسی : بهترین ویژگی ها را با توجه به برخی دانش های گذشته انتخاب نمایید
- نمونه های دانش قبلی :
  - در صورتی که ما انتقال ویژگی ها را پذیرفتیم
    - ویژگی ها باید غیر وابسته باشند ، در نتیجه یک PCA را انجام دهید و فقط بردارهای ایجن [1] مطابق با x% واریانس را نگهداری نمایید .
    - ایده های مشابه : تحلیل مشخص کننده ی خطی [2] و تحلیل اجزای مستقل [3]
  - ویژگی ها باید دارای ارتباط قوی با هدف باشند ، در نتیجه k ویژگی را که بیش ترین وابستگی خطی وابسته به هدف را دارند انتخاب نمایید
  - ویژگی ها باید دارای ارتباط قوی با هدف باشند ، در نتیجه k ویژگی را که دارای بالاترین اطلاعات دوطرفه با مقصد هستند را انتخاب نمایید :

$$I(x,y) = \sum_i \sum_j p(x=i, y=j) \log \left[\frac{p(x=i, y=j)}{p(x=i)\, p(y=j)} \right]$$

۳. بسته بندی کننده ها

روش های بسته بندی

- الگوریتم پایه (و ساده) :
  - برای هر زیر مجموعه از ویژگی ها ، مساله را حل نمایید .
  - بهترین زیرمجموعه را انتخاب نمایید .
- غیرممکن است زیرا مساله به صورت نمایی بزرگ می باشد !
- چاره : مکاشفه های حریصانه نظیر انتخاب مستقیم یا حذف معکوس

انتخاب مستقیم

۱. تصور نمایید که $P = \phi$ مجموعه ی جاری ویژگی های انتخاب شده باشد
۲. و Q مجموعه ی کامل ویژگی ها باشد
۳. مادامی که اندازه ی P کوچک تر از ثابت داده شده می باشد

---

[1] eigenvectors
[2] Linear Discriminant Analysis (LDA)
[3] Independent Component Analysis (ICA)

a. برای هر $v \in Q$
   i. $P' \leftarrow \{v\} \cup P$
   ii. مدل را با P' آموزش دهید و اعتبار کارایی را نگهداری نمایید
b. قرار دهید $P \leftarrow \{v^*\} \cup P$ که v^* با بهترین اعتبار کارایی به دست آمده در مرحله ی ۳,۱ مطابقت می کند
c. قرار دهید $Q \leftarrow Q \setminus \{v^*\}$
d. اعتبار کارایی به دست آمده با P جاری را نگهداری نمایید

۴. بهترین مجموعه ی P را برگردان

حذف معکوس

۱. تصور نمایید که P مجموعه ی کاملی از ویژگی ها باشد
۲. مادامی که اندازه ی P بزرگ تر از یک ثابت داده شده می باشد
   a. برای هر $v \in P$
      i. قرار دهید $P' \leftarrow P \setminus \{v\}$
      ii. مدل را با P' آموزش دهید و اعتبار کارایی را نگهداری نمایید
   b. قرار دهید $P \leftarrow P \setminus \{v^*\}$ در جایی که v^* با بدترین اعتبار کارایی به دست آمده از مرحله ی ۲,۱ مطابقت می نماید
   c. اعتبار کارایی به دست آمده با P جاری را نگهداری نمایید
۳. بهترین مجموعه ی P را به دست آورید

مقایسه : بسته بندی کننده ها و فیلترها

- هر دو مدل سرانجام مکاشفه ای هستند ، به دلیل مانع ترکیبی .
- بسته بندی کننده ها برای حل مساله ی واقعی تلاش می کنند ، بنابراین شما واقعا ملاک تان را بهینه می نمایید .
- فیلترها یک مساله ی متفاوت را حل می کنند . ممکن است مناسب نباشند
- بسته بندی کننده ها به صورت بالقوه خیلی زمان گیر می باشند : شما باید مساله را در زمان زیادی حل نمایید .
- فیلترها به مراتب سریع تر می باشند ، زیرا مساله ای که آن ها حل می کنند در کل ساده تر می باشد .

۴. وزن کردن ویژگی

روش های وزن کردن ویژگی

- به جای انتخاب یک زیر مجموعه از ویژگی ها ، کدام یک مساله ی ترکیبی می باشد ، چرا به سادگی آن ها را وزن نمی کنید ؟
- بیش تر روش های وزن کننده ی ویژگی براساس روش بسته بندی هستند
- مکاشفه ها برای وزن کردن ویژگی :
  - کاهش گرادیان در فضای ورودی ، در نتیجه با همه ی ویژگی ها آموزش دهید ، سپس پارامترها را ثابت نمایید و اهمیت هر ورودی را تخمین بزنید و تکرار کنید
  - در زمانی که هر مدل ، فقط در یک ویژگی آموزش داده شده است از ایدابوست [1] کنید (در نتیجه ، راه حل نهایی ، یک ترکیب خطی می باشد)

[1] AdaBoost

منابع :

http://www.idiap.ch/~bengio/lectures/intro.pdf
http://www.idiap.ch/~bengio/lectures/theory.pdf
http://www.idiap.ch/~bengio/lectures/classical.pdf
http://www.idiap.ch/~bengio/lectures/mlp.pdf
http://www.idiap.ch/~bengio/lectures/ann.pdf
http://www.idiap.ch/~bengio/lectures/gmm.pdf
http://www.idiap.ch/~bengio/lectures/hmm.pdf
http://www.idiap.ch/~bengio/lectures/svm.pdf
http://www.idiap.ch/~bengio/lectures/dtree.pdf

http://www.idiap.ch/~bengio/lectures/feat_sel.pdf

فصل بیست و نهم

آشنایی با روبوتیک

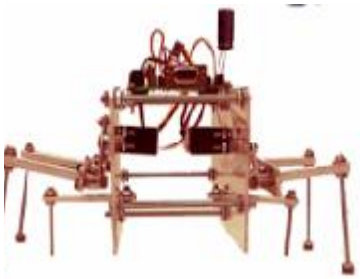
فصل بیست و نـهم
آشنـایی با روبوتیـک




روبوت A255

کاواساکی



رئوس مطالب

- تعریف
- انواع
- کاربردها
- تاریخچه
- اجزای کلیدی
- کاربردها
- آینده

* تعریف

- کلمه ی روبوت به وسیله ی یک رمان نویس کشور چک [1] به نام کارل کاپک [2] در سال ۱۹۲۰ در یک بازی با عنوان Rassum's Universal Robots (RUR) به کار گرفته شد .
- روبوت در زبان چکی ، واژه ای برای کارگر یا پیشخدمت می باشد
- **تعریف روبوت**

- هر ماشین ساخته شده توسط یکی از اعضای ما : موسسه ی روبوت آمریکا [3] ☺
- یک روبوت یک دست ماشینی قابل برنامه ریزی مجدد و چندکاربردی طراحی شده برای حرکت دادن مواد ، قطعه ها ، ابزار یا وسایل مخصوص از میان حرکت های برنامه ریزی شده ی متغیر برای کاربردهای مختلف می باشد : موسسه ی روبوت آمریکا ، ۱۹۷۹

* انواع روبوت

دست ماشینی

دست ماشینی

دست ماشینی

روبوت پادار

روبوت چرخ دار

وسیله ی زیر آبی خودمختار

وسیله ی بی سرنشین هوایی

* کاربردهای روبوت

کارهایی که برای بشر خطرناک می باشند

روبوت ضدعفونی کننده پمپ اصلی سیار در کارگاه انرژی هسته ای را تمیز می کند

[1] Czech
[2] Karel Capek
[3] Robot Institute of America

کارهای تکراری که برای بشر ، کسل کننده ، پردغدغه یا کار بر می باشند

روبوت جوشکاری کننده

کارهای پست که بشر دوست ندارد آن ها را انجام دهد

روبوت اسکرابمیت [1]

تاریخچه ی روبوتیک

* اولین روبوت صنعتی : UNIMATE

* ۱۹۵۴ : اولین روبوت قابل برنامه ریزی توسط جورج دول [2] طراحی شد ، بعدا به Unimation که نام اولین شرکت روبوت شد ، تغییر نام پیدا نمود . (۱۹۶۲) Unimate در ابتدا ساخت لامپ تصویر تلویزیون را خودکار نمود

۱۹۷۸ : پوما [3] (ماشین کلی قابل برنامه ریزی برای مونتاژ یا اسمبلی) که توسط Unimation با پشتیبانی از طرح General Motors ، توسعه داده شد .

---

[1] SCRUBMATE Robot
[2] George Devol
[3] Puma (Programmable Universal Machine for Assembly)

دست ماشینی پوما ۵۶۰

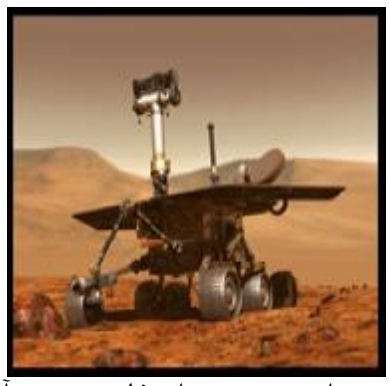
دهه ی ۱۹۸۰ : صنعت روبوت به یک مرحله ی رشد سریع رسید . تعداد زیادی موسسه ، برنامه ها و زمینه هایی را در روبوتیک معرفی نمودند . زمینه های روبوتیک در مهندسی مکانیک ، مهندسی برق و دانشکده های علوم کامپیوتر منتشر شدند .

دست ماشینی فناوری بارت

روبوت نزدیک Cognex

روبوت ماهر SCARA

۱۹۹۵ تا حالا : تحولاتی در روبوتیک کوچک و گرداننده ی روبوت های متحرک ، یک دور دوم رشد کمپانی ها و تحقیق



۲۰۰۳ : کاوشگر مریخ ، جستجویی را در مریخ برای تاریخچه ی آب در آن سیاره اجرا نمود

پایگاه دانش برای روبوتیک

- پایگاه دانش معمولی برای طراحی و کار سیستم های روبوتیک
- سیستم مدلسازی و تحلیل پویا
- کنترل بازخورد
- حسگرها و شایسته سازی سیگنال

- محرک ها (ماهیچه ها) و نیروهای الکترونیکی
- سخت افزار / واسط کامپیوتر
- برنامه نویسی کامپیوتر

ترتیب ها : ریاضیات ، فیزیک ، زیست ، مهندسی مکانیک ، مهندسی برق ، مهندسی کامپیوتر و علم کامپیوتر

اجزای کلیدی

پایه ی روبوت : ثابت و متحرک

پایه های متحرک ، معمولا سکوهایی با چرخ ها یا شیارهای متصل شده هستند . به جای چرخ ها یا شیارها ، برخی از روبوت ها از پایه هایی برای حرکت استفاده می کنند .

بازوهای ماشینی روبوتیک در کارخانه ها ، نمونه هایی از روبوت های ثابت می باشند . آن ها نمی توانند پایه ی خود را حرکت دهند .

مکانیزم روبوت

اجزای مکانیکی

- حس های بشری : جا ، صدا ، لمس ، مزه و بو اطلاعات حیاتی را برای کار و بقای ما مهیا می کنند
- حسگرهای روبوت : پیکربندی / وضعیت و محیطش را ارزیابی می کنند و اطلاعات این چنینی را به کنترلر روبوت به صورت علامت های الکترونیکی می فرستند (به عنوان مثال ، وضعیت بازو و حس گاز مسموم)

- روبوت ها اغلب به اطلاعاتی که ماورای ۵ حس بشر هستند نیاز دارند (به عنوان مثال ، دیدن در تاریکی ، تشخیص مقدارهای کم پرتوهای نامرئی ، اندازه گیری حرکت هایی که آن قدر سریع یا کوچک اند که برای انسان ها قابل دیدن نمی باشند)

شتاب سنج با استفاده از اثر فیزوالکتریک [1]

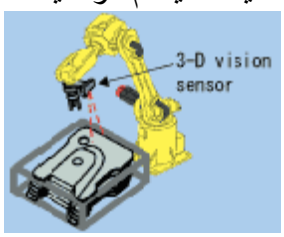
حسگر فلکسیفورس [2]

حسگرهای بینایی

به عنوان مثال ، انتخاب مواد اولیه ، انجام بازرسی و ...

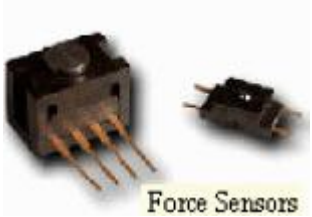
حسگرهای با دید نزدیک

انتخاب قسمت : روبوت می تواند با استفاده از حسگر دید سه بعدی ، قطعه هایی را که به صورت تصادفی به شکل توده هستند را انتخاب نمایند . از آنجایی که عملیات تطبیق ، یک خط تغذیه ی مخصوص و یک پالت لازم نمی باشند ، یک سیستم اتوماتیک می تواند به بهایی ارزان تولید شود .



حسگرهای نیرو

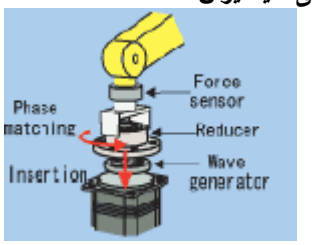
به عنوان مثال ، جاسازی قطعات و جاسازی و بازخورد نیرو در جراحی



جاسازی قطعات و جاسازی : روبوت ها می توانند جاسازی دقیق و جاسازی قطعات ماشین را با استفاده از حسگر نیرو انجام دهند . یک روبوت می تواند قطعات را به صورت بهتر ، برای ساده کردن جاسازی آن ها قرار دهد . می تواند کارهای دارای مهارت زیاد را انجام دهد .

---

[1] piezoelectric
[2] flexiforce



حسگرهای نزدیکی

روبوت KOALA

مثال ←

حسگر مسافت یابی مادون قرمز

- ۶ مبدل صوت یاب فراصوت برای اکتشاف ناحیه های پهن و باز
- مانع یاب برای یک دامنه ی پهن ۱۵ سانتیمتر تا ۳ متر
- ۱۶ حسگر نزدیکی مادون قرمز نزدیکی (دامنه ی ۵ الی ۲۰ سانتیمتر)
- حسگرهای مادون قرمز به صورت یک "ضربه گیر مجازی" عمل می کنند و برای طی کردن فضاهای سفت اجازه می دهند

Devantech SRF04

مسافت یاب ماورا صوت

حسگرهای شیب (کجی)

حسگرهای شیب : به عنوان مثال برای متعادل کردن (بالانس کردن) یک روبوت

روبوت دارای دو پای سطحی [1]

مثال ←

حسگر شیب

[1] Planar Bipedal Robot

محرک ها / ماهیچه ها

- محرک های روبوتیک رایج از ترکیب های ابزارهای الکترومکانیکی مختلف استفاده می کنند
  - موتور همزمان
  - موتور پله ساز
  - موتور خود تنظیم AC
  - موتور خود تنظیم بی زغال DC
  - موتور خود تنظیم با زغال DC

موتور DC

موتور پله ساز

سیلندر گازی

موتور روغنی

موتور خود متحرک

وایر ماهیچه ای

موتور گازی

فهرست منابع

http://www.nsbe.org