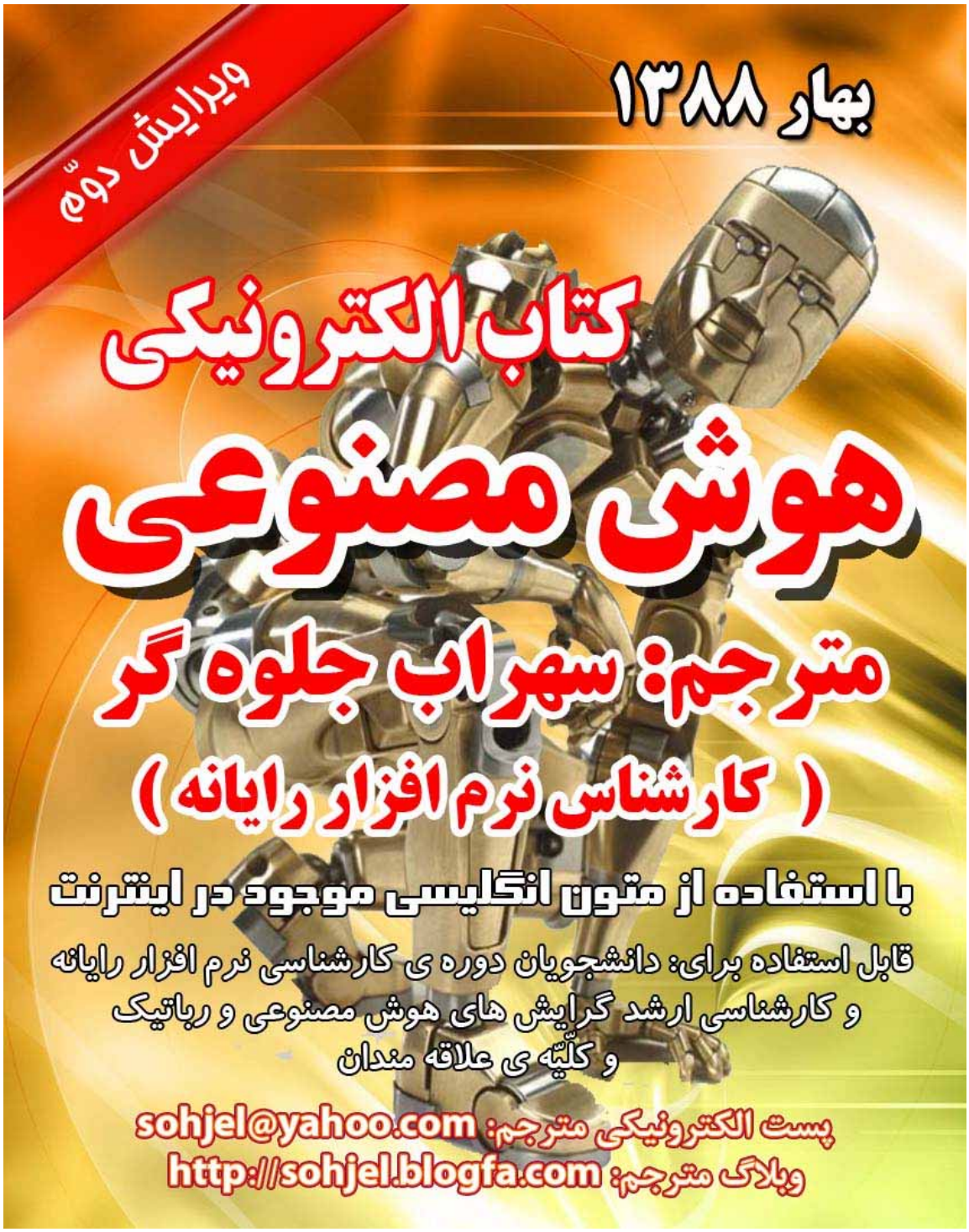
ویرایش بهار ۱۳۸۸
کتاب الکترونیکی
هوش مصنوعی

ترجمه شده توسّط : سهراب جلوه گر

به نام

خداوند

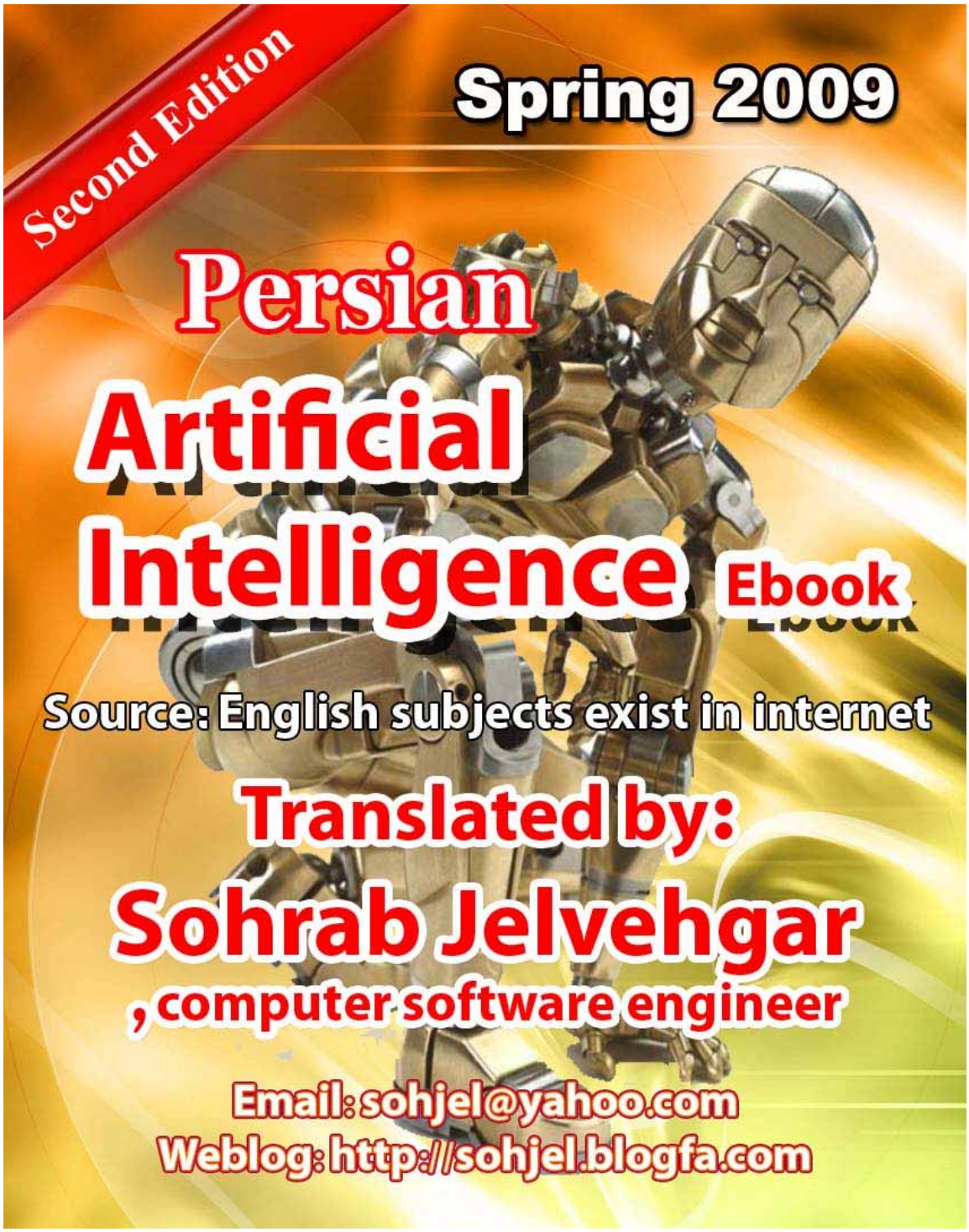
بخشنده ی مهربان


بهار ۱۳۸۸
ویرایش دوّم
کتاب الکترونیکی
هوش مصنوعی
مترجم: سهراب جلوه گر
(کارشناس نرم افزار رایانه)
با استفاده از متون انگلیسی موجود در اینترنت
قابل استفاده برای: دانشجویان دوره ی کارشناسی نرم افزار رایانه
و کارشناسی ارشد گرایش های هوش مصنوعی و رباتیک
و کلّیه ی علاقه مندان
پست الکترونیکی مترجم: sohjel@yahoo.com
وبلاگ مترجم: http://sohjel.blogfa.com

هوش مصنوعی

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸



برای آگاهی یافتن از انتشار ویرایش های جدیدتر کتاب هوش مصنوعی اینجانب به وبلاگ های زیر مراجعه نمایید :

http://sohrabejelvehgar.blogspot.com

http://sohrab-e-jelvehgar.blogspot.com

http://sohjel.blogfa.com

جویای کار هستم

در انتظار سرمایه گذار یا ناشر سرمایه گذار برای چاپ ویرایش های بعدی این کتاب هستم .

صاحبان محترم مشاغل ، علاقه مندان به سرمایه گذاری و ناشرین گرامی ، می توانند پیشنهادهای خود را از طریق پست الکترونیکی sohrabejelvehgar@yahoo.com **برای اینجانب ارسال نمایند .**

نظرات ، پیشنهادات و انتقادات

لطفاً جهت بهتر شدن این کتاب ؛ نظرات ، پیشنهادات و انتقادات سازنده ی خود را از طریق ایمیل های زیر برای اینجانب ارسال نمایید .

sohrabejelvehgar@hotmail.com

sohrabejelvehgar@yahoo.com

sohjel@yahoo.com

با احترام ، سهراب جلوه گر ، کارشناس نرم افزار رایانه

شیراز – ایران – بهار ۱۳۸۸

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فهرست

فصل اوّل – هوش مصنوعی

فصل دوّم – عامل های هوشمند

فصل سوّم – حل مسأله و جستجو

فصل چهارم - جستجوی آگاهانه (مکاشفه ای)

فصل پنجم - الگوریتم های جستجوی محلّی

فصل ششم - مسایل ارضای محدودیّت

فصل هفتم - مسایل ارضای محدودیّت و برنامه نویسی ارضای محدودیّت

فصل هشتم - تئوری بازی ها

فصل نهم - عامل های منطقی

فصل دهم - منطق مرتبه ی اوّل به بیان ساده

فصل یازدهم - منطق مرتبه ی اوّل

فصل دوازدهم – استنتاج در منطق مرتبه ی اوّل

فصل سیزدهم – نامعلومی (عدم قطعیّت)

فصل چهاردهم - شبکه های بیزی

فصل پانزدهم – استنتاج در شبکه های بیزی

فصل شانزدهم – شناخت سخن یا سخن شناسی (به طور خلاصه)

فصل هفدهم - تصمیم گیری های عاقلانه (تیوری تصمیم گیری)

فصل هیجدهم - شبکه های عصبی

فصل نوزدهم - الگوریتم های ژنتیکی

فصل بیستم - سیستم های خبره

فصل بیست و یکم - پردازش های تصمیم گیری مارکوف

فصل بیست و دوّم - سیستم های طبقه بندی کننده

فصل بیست و سوّم - درخت های تصمیم گیری

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

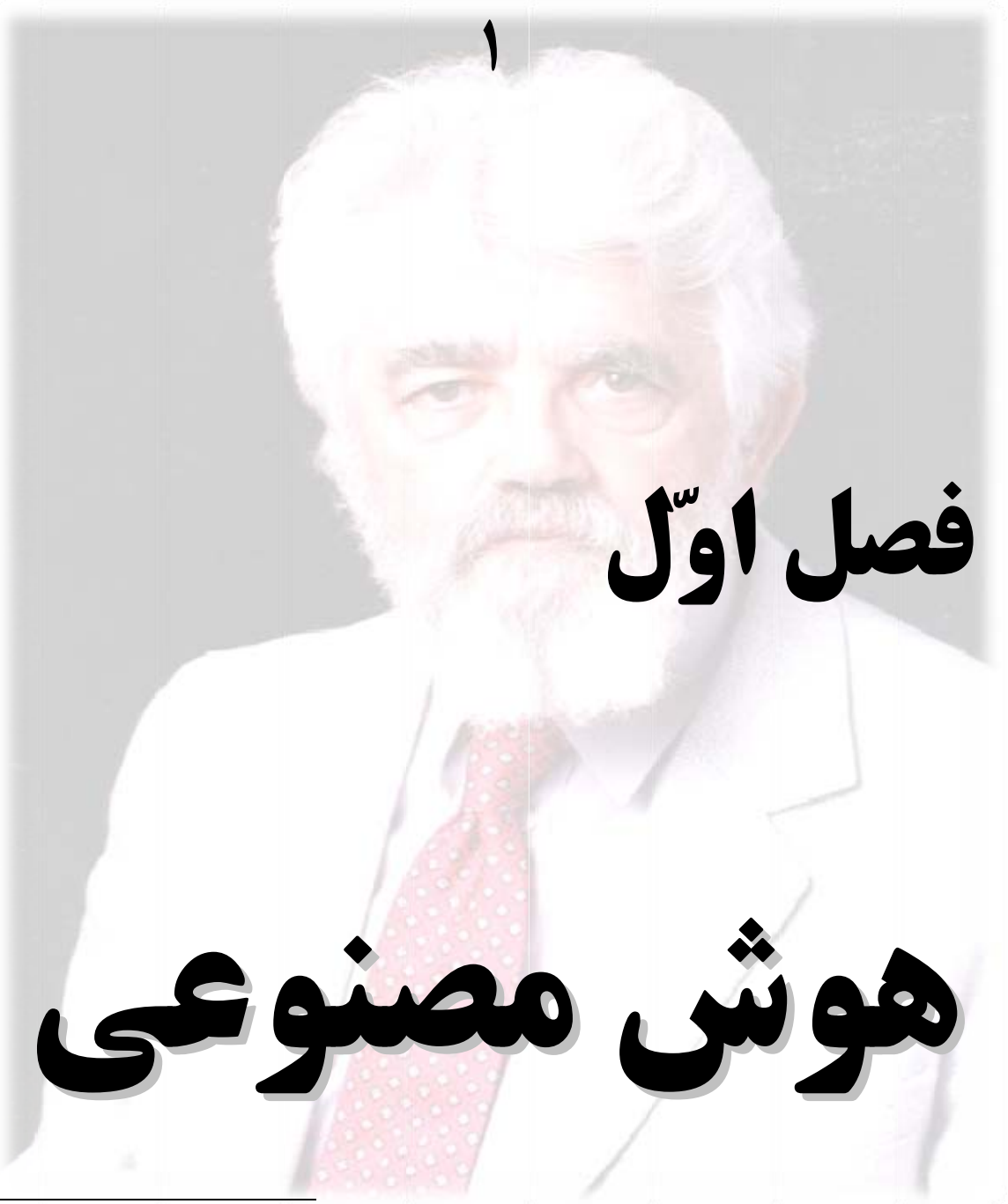
فصل بیست و چهارم - یادگیری Q ای

فصل بیست و پنجم - برنامه ریزی

فهرست برخی از منابع و مآخذ

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸



۱

فصل اوّل

هوش مصنوعی

[۱]- تصویر ، متعلّق به جان مکارتی ، ملقّب به پدر هوش مصنوعی است .

مقدّمه

هوش مصنوعی [۱] : ساخت ماشین هایی که کارهایی را انجام می دهند که آن کارها معمولاً با استفاده از هوش انسان انجام می شوند ؛ مثل ، ترجمه ی یک زبان به زبان دیگر . [۲] به عنوان تعریفی دیگر ، دانش و مهندسی ساخت ماشین های هوشمند و مخصوصاً برنامه های کامپیوتری هوشمند می باشد . هوش مصنوعی ، وابسته به کامپیوترهای مورد استفاده برای فهم هوش انسانی می باشد ، ولی لازم نیست که خودش را به روش هایی که به صورت زیستی [۳] قابل مشاهده اند محدود نماید .

---

[۱] Artificial Intelligence (AI)

[۲] Babylon / Concise Oxford English Dictionary

[۳] biologically

هوش مصنوعی

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

هوش ، توانایی به دست آوردن و به کار گرفتن دانش و مهارت ها می باشد . [۱] انواع و درجه های هوش در افراد و حیوانات و برخی از ماشین ها متنوّع است . *هنوز یک تعریف جامع از هوش که به ارتباط آن با هوش بشری وابسته نباشد وجود ندارد* ؛ مسأله این است که ما هنوز نمی دانیم که در حالت کلّی 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 . کسی نمی تواند بپرسد که "آیا این ماشین ، هوشمند است یا نه ؟" زیرا 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 .

بعضی اوقات ، امّا نه همیشه هوش مصنوعی در مورد شبیه سازی هوش بشری می باشد . 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 . محقّقان 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 .

بهره ی هوشی [۲] ، براساس میزان یا نرخی است که هوش در بچّه ها توسعه پیدا می نماید . برای بزرگسالان از روش مناسب دیگری استفاده می نماییم . بهره ی هوشی به خوبی با اندازه های متفاوت موفّقیّت یا عدم موفّقیّت در زندگی ارتباط برقرار می نماید ، امّا به طور کمّی ، ساختن کامپیوترهایی که بتوانند امتیاز بالا را در تست های بهره ی هوشی انجام دهند به مفید بودن آن ها بستگی دارد .

---

[۱] Babylon / Concise Oxford English Dictionary

[۲] Intelligence Quotient

هوش مصنوعی

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

آرتور آر. جِنسِن [۱] ، یک محقّق 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 کمّی و وضعیّت های فیزیولوژیکی هستند را پیشنهاد می کند " . که ما آن ها را به صورت سرعت [۲] ، حافظه ی کوتاه مدت [۳] ، توانایی به صورت صحیح و حافظه های بازیافتنی طولانی مدت [۴] تقسیم بندی می کنیم . 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 امّا توانایی عقلانی آن ها با توانایی عقلانی طرّاحان برنامه ها بستگی دارد . به احتمال قوی ، سازماندهی مکانیزم های عقلانی برای هوش مصنوعی می تواند با سازماندهی مکانیزم های عقلانی برای افراد ، 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 طرّاحان برنامه فهم مکانیزم های عقلانی لازم برای انجام کار را به طور مناسب نداشته اند .

بعد از جنگ جهانی دوّم تعدادی از افراد به صورت مستقل کار بر روی ماشین های هوشمند را شروع کردند . ریاضیدان انگلیسی ، آلن تورینگ [۵] شاید اوّلین آن ها باشد . وی یک سخنرانی را در مورد هوش مصنوعی در سال ۱۹۴۷ میلادی ارایه نمود . همچنین وی شاید اوّلین 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

---

[۱] Arthur R. Jensen

[۲] speed

[۳] short term memory

[۴] retrievable long term memory

[۵] Alan Turing

وجود داشت و بیش تر آن ها اساس کار خود را برمبنای برنامه نویسی کامپیوترها گذاشته بودند.

برخی از محقّقان گفته اند که می خواهند افکار انسان را با استفاده از هوش مصنوعی در کامپیوتر پیاده سازی نمایند ؛ فکر بشری دارای موارد زیادی می باشد و هر کسی به طور کامل نمی تواند همه ی آن ها را تقلید نماید .

اهداف هوش مصنوعی در سطح هوش انسانی است ؛ 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 . 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 [۱] واقعیّات ، با استفاده از زبان هایی که اکنون برای بیان دانش استفاده می شوند ، قابل دسترسی می باشند . به هر حال ، بیش تر محقّقان هوش مصنوعی تصوّر می کنند که ایده های بنیادین جدید مورد نیازند ، و بنابراین *قابل پیش بینی نیست که سطح هوش انسانی قابل دسترسی باشد* .

کامپیوترها ابزار مناسبی برای پیاده سازی هوش مصنوعی هستند ؛ کامپیوترها برای شبیه سازی هر نوع از ماشین ها می توانند برنامه ریزی شوند . برخی از افراد فکر می کنند که کامپیوترهای به مراتب سریع تر برای ایده های جدید لازم هستند . نظر شخصی جان مک کارتی [۲] 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 . البتّه به طور مجزا از بلند پروازی های محقّقان هوش مصنوعی ، کامپیوتر ها روند سریع تر شدن را دنبال می کنند .

---

[۱] knowledge base

[۲] یکی از استادان دانشگاه استنفورد ایالات متحده ی آمریکا

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

فرزند ماشینی [۱] می تواند با خواندن و آموزش و از راه تجربه یاد بگیرد و شروع بحث آن از سال ۱۹۴۰ میلادی بوده است . امّا 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 یاد بگیرند .

یک سیستم هوش مصنوعی می تواند خودش را به سطح های بالاتر از هوش برساند . امّا ، ما هم اکنون در مرحله ای از هوش مصنوعی نمی باشیم که این کار بتواند انجام شود .

توضیحی در مورد چند بازی

بازی شطرنج [۲] – بازی ای است دو نفره که در آن هر نفر شانزده مهره را با استفاده از قوانینی ثابت در یک صفحه ی شطرنجی [۳] حرکت می دهد و تلاش می کند که مهره ی شاه حریف را کیش و مات [۴] نماید [۵] . برنامه های بازی شطرنج در حال حاضر در سطح استادی اجرا می شوند ، امّا آن ها این کار را با مکانیزم های محدود شده نسبت به انسان و با انجام تعداد زیادی محاسبات برای فهم راه ، انجام می دهند . ما این مکانیزم ها را بهتر می فهمیم و می توانیم برنامه های شطرنج را در سطح انسانی بسازیم طوری که محاسبه های کم تری را نسبت به برنامه های فعلی انجام دهند . نمونه ای از صفحه ی این بازی را به همراه مهره های آن در شکل زیر می توانید ببینید .

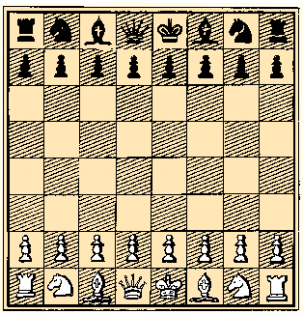
---

[۱] child machine
[۲] chess
[۳] checkerboard
[۴] checkmate
[۵] Babylon / Merriam-Webster

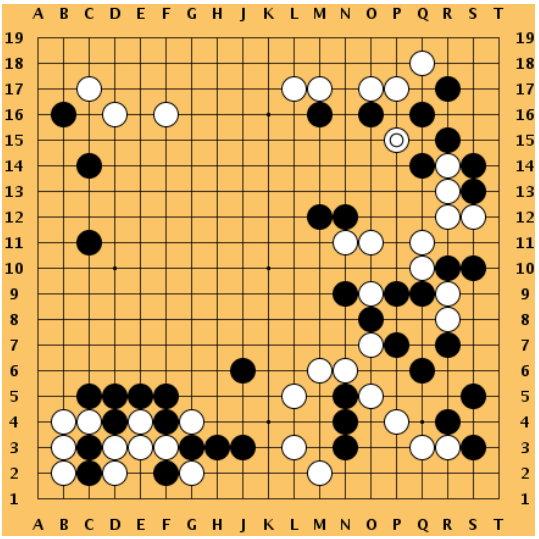
هوش مصنوعی

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸



بازی Go – بازی دو نفره ی چینی و ژاپنی Go ، بازی ای روی مقوایی دارای نوزده خط افقی و نوزده خط عمودی است که در آن معمولاً یکی از بازی کنندگان دارای مهره های سفید رنگ و یکی دیگر از بازی کنندگان دارای مهره های سیاه رنگ می باشند . در زیر تصویری از صفحه ی این بازی به همراه مهره های آن را مشاهده می نمایید .



برخی از افراد می گویند که هوش مصنوعی ایده ی بدی می باشد ؛ فیلسوفی به نام جان سِرل [۱] می گوید که ایده ی ماشینی غیر بیولوژیکی که می تواند هوشمند شود ، ایده ای نقض کننده می باشد . وی

---

[۱] John Searle (1980)

هوش مصنوعی

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

استدلال اطاق چینی [۱] را پیشنهاد می کند [۲] . فیلسوفی به نام هیوبرت دریفس [۳] می گوید که هوش مصنوعی غیرممکن می باشد . دانشمند علوم کامپیوتری به نام جزف ویزنباوم [۴] 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 مأیوس می شوند .

استدلال اطاق چینی : یک گمان آزمایشی طرّاحی شده توسط جان سرل برای کم ارزش کردن ادّعاهای مطرح شده توسط هوش مصنوعی قوی [۵] و همچنین کارکردگرایی [۶] می باشد . [۷] از این استدلال همچنین برای نشان دادن این که مغز ، یک کامپیوتر نمی باشد و آزمایش تورینگ ، ناقص است استفاده شده است . [۸]

هوش مصنوعی ضعیف [۹] : اوّلین هدف هوش مصنوعی (هوش مصنوعی ضعیف) ساختن چیزها (موجودیت ها) ی هوشمند می باشد .

هوش مصنوعی قوی : فهمیدن چیزها (موجودیّت ها) ی هوشمند و شاید حتی فهمیدن و مهندسی هوش انسان می باشد . [۱] دیدی است که می گوید مغز بشر یک ابزار محاسباتی می باشد و

---

[۱] Chinese room argument
[۲] http://www-formal.stanford.edu/jmc/chinese.html
[۳] Hubert Dreyfus
[۴] Joseph Weizenbaum
[۵] Strong AI
[۶] functionalism
[۷] en.wikipedia.org/wiki/Chinese_room_argument
[۸] Babylon / Dictionary of Philosophy of Mind
[۹] weak artificial intelligence

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

کامپیوترها به طور کلّی قادرند که فکر کنند [۲] . 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 [۳] باشد امّا ممکن است پردازش های فکری شبیه بشر را داشته باشد یا نداشته باشد . [۴]

تیوری تجزیه پذیری [۵] و پیچیدگی محاسباتی [۶] روش هایی برای هوش مصنوعی نمی باشند [توجّه کنید که افراد عامی و مبتدیان در علم کامپیوتر نمی توانند در این موارد اظهار نظر کنند ، این ها کاملاً شاخه های منطقی ریاضی و علم کامپیوتر می باشند و جواب این موارد باید تا حدودی تکنیکی باشد .] ؛ این تیوری ها مورد استفاده قرار می گیرند ولی مسایل اساسی هوش مصنوعی را جوابگو نمی باشند . در سال ۱۹۳۰ منطقدانان ریاضیّات و مخصوصاً کورت گودل [۷] 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 اوّل [۸] یک مثال می باشد و یک معادله ی چند جمله ای با چند متغیّر مثال درست 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 (معمولاً با برخی اضافات) که کامپیوترها ذاتاً قادر به انجام کارهایی که افراد می توانند انجام دهند نمی باشند

---

[۱] پروفسور گرینوالد (Professor Greenwald) ، استاد دانشگاه برون (Brown) ایالات متحده ی آمریکا

[۲] www.ucd.ie/philosop/documents/2.%20definitions%20of%20some%20key%20terms.htm

[۳] self-aware

[۴] en.wikipedia.org/wiki/Strong_AI

[۵] computability theory – بحثی در علم نظری کامپیوتر می باشد که می گوید مسایل می توانند توسط هر کامپیوتری حل شوند . Babylon / FOLDOC

[۶] computational complexity – هزینه ی حل یک مسأله در محاسبات علمی گسترده ، که با استفاده از تعداد عملیّات لازم به علاوه ی مقدار حافظه ی استفاده شده برای مساله و ترتیبی که مساله حل می شود اندازه گیری می شود . (Babylon / Britannica Concise Encyclopedia)

[۷] Kurt Godel

[۸] First Order Logoc(FOL)

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

، راجر پنروز [۱] این مطلب را ادّعا 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 مخصوصاً استیو کوک [۲] و ریچارد کارپ [۳] تیوری مسایل با دامنه ی **NP – کامل** [۴] را توسعه دادند . مسایل در این دامنه ها قابل حل می باشند ، امّا 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 ، امّا در حالت کلّی 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 ، امّا 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 . موفّقیّت در مسایل حل شده توسط بشر و توسط برنامه های هوش مصنوعی به نظر می رسد که وابسته به خصوصیات مسایل و متدهای حل مسأله ای که نه توسط محقّقان و نه توسط اجتماع هوش مصنوعی به دقت قابل تعریف باشند وابسته باشد . تیوری پیچیدگی الگوریتمی که به نحوی توسط سولومونوف [۵] ، کولموگوروف [۶] و چایتین [۷] (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

---

[۱] Roger Penrose
[۲] Steve Cook
[۳] Richard Karp
[۴] NP-complete
[۵] Solomonoff
[۶] Kolmogorov
[۷] Chaitin

است یک مسأله 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 – یک نوع از مسایل محاسباتی هستند که هیچ الگوریتم مناسبی برای حل آن ها پیدا نشده است . مسأله ی فروشنده ی دوره گرد [۱] مثالی از این نوع مسایل می باشد . [۲] در تیوری پیچیدگی ، مسایل NP- کامل ، سخت ترین مسایل در NP هستند ، در صورتی که شما بتوانید یک راه برای حل یک مسأله 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

در زیر یک لیست از شاخه های هوش مصنوعی آمده است ، امّا 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 [۳] – این که یک برنامه در حالت کلّی ، واقعیّات وضعیّت 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 .

---

[۱] traveling salesman problem – یک مسأله ی بهینه سازی در تیوری گراف ها است که در آن گره ها (شهرها) ی یک گراف به وسیله ی خط هایی مستقیم (یال) به هم وصل شده است و طول هر خط به فاصله ی دو شهر از هم بستگی دارد به عبارت دیگر هر چقدر طول خط بیش تر باشد فاصله ی دو شهر بیش تر خواهد بود . حال مسأله این است که مسیری مناسب را پیدا کنیم که از هر شهر یکبار بگذرد و در ضمن کوتاه ترین مسیر هم باشد . (Babylon / Britannica.com)

[۲] Babylon / Britannica.com

[۳] logical AI

برنامه در مورد عملی که باید انجام دهد با استنتاج عملکردهای معین که برای به دست آوردن اهداف ، مناسب هستند تصمیم گیری می نماید . اوّلین مقاله ی پیشنهاد شده در این مورد 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 [۱] – برنامه های هوش مصنوعی معمولاً تعداد زیادی از احتمالات را برّرسی می کنند ، به عنوان مثال حرکت های درون یک بازی شطرنج .

الگو شناسی یا شناخت الگو [۲] – در علم کامپیوتر ، تشخیص داده های ورودی مثل سخن ، تصویرها و رشته های متنی با شناخت و تشریح خصوصیات و تشخیص ارتباط های میان آن ها است . [۳]

نمایش [٤] – واقعیّات یک محیط باید به طریقی نمایش داده شوند . در این مورد معمولاً از زبان های منطقی ریاضی استفاده می شود .

استنتاج [٥] – از برخی از واقعیّات ، دیگر واقعیّات می تواند استنتاج شود . استنتاج منطقی ریاضی برای برخی از اهداف ، کافی می باشد ، امّا متدهای استنتاج جدید غیر یکنواخت از سال ۱۹۷۰ به منطق اضافه شده اند . ساده ترین نوع استدلال غیر یکنواخت [٦] ، استدلال پیش فرض است که در آن یک نتیجه گیری به صورت پیش فرض استنتاج می شود ، امّا 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

---

[۱] search
[۲] pattern recognition
[۳] Babylon / Britannica Concise Encyclopedia
[۴] representation
[۵] inference
[۶] non-monotonic

کند ، امّا 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 . معمولاً 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 [۱] – محیطی است که در آن هوش مصنوعی از سطح انسانی خیلی دور می باشد ، باوجود این واقعیّت 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 واقعیّات عقل سلیم می باشد .

یادگیری از تجربه [۲] – برنامه ها این کار را انجام می دهند . براساس پیوندگرایی [۳] و شبکه های عصبی [۴] 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 فراگیری ماشین یا آموزش ماشینی [۵] می باشد . برنامه ها فقط می توانند واقعیّات 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 اطّلاعات می باشند .

---

[۱] common sense knowledge and reasoning

[۲] learning from experience

[۳] connectionism – یک تیوری شناختی که به مغز به صورت یک شبکه ی بزرگ که با هم کار می کنند (تعامل دارند) با تعداد زیادی متصل کننده بین هر گره نگاه می کند . (Babylon / Concise Oxford English Dictionary)

[۴] neural nets – یک سیستم کامپیوتری از روی مغز انسان و سیستم عصبی ساخته شده است . (Babylon / Concise Oxford English Dictionary)

[۵] machine learning – توانایی یک ماشین برای بهتر کردن عملکرد خود براساس نتیجه های قبلی . شبکه های عصبی یک مثال از آن می باشند . (Babylon / FOLDOC)

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

برنامه ریزی [۱] – برنامه ریزی تلاش می کند که عملیّات را برای رسیدن به هدف ها مرتّب نماید . کاربردهای برنامه ریزی شامل تدارکات [۲] ، زمانبندی ساخت [۳] و برنامه ریزی ساخت مراحلی برای تولید محصول مطلوب می باشد . با یک برنامه ریزی بهتر می توان مقادیر زیادی در هزینه صرفه جویی نمود . برنامه ریزی برنامه ها با [شناخت] واقعیت های عمومی در مورد جهان (مخصوصاً واقعیّاتی در مورد اثرات عملکردها) شروع می شوند ، واقعیّات ، در مورد وضعیّت 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 ، استراتژی فقط یک ترتیب از عملکردها می باشد .

شناخت شناسی [۴] – یک شاخه از فلسفه که در مورد منبع [۵] ، طبیعت [۶] ، روش ها و محدودیّت های دانش بشری بحث می کند . به عبارت دیگر شناخت شناسی ، مطالعه در مورد این است که چه ما می دانیم و چگونه می دانیم . [۷] به بیان دیگر ، یک مطالعه بر روی انواع دانش که برای حل مسایل در محیط (جهان) لازم می شوند می باشد .

هستی شناسی [۸] – شاخه ای از متافیزیک است که در مورد طبیعت موجودات صحبت می کند . [۹] به عبارت دیگر ، هستی شناسی ، مطالعه ی انواع چیزهایی است که موجودند . در برنامه ها و عبارات به انواع

---

[۱] planning
[۲] logistics
[۳] manufacturing scheduling
[۴] epistemology
[۵] origin
[۶] nature
[۷] Babylon / Learning , Performance and Training Definitions
[۸] ontology
[۹] Babylon / Concise Oxford English Dictionary

مختلفی از اشیا توجّه می شود و ما این انواع و خصوصیات اساسی آن ها را مطالعه می کنیم . تأکید بر هستی شناسی از سال ۱۹۹۰ شروع شد .

ابتکارها یا اکتشافات [۱] – برای افزایش احتمال حلّ برخی از مسأله ها به کار می روند . [۲] این واژه به صورت متنوّع در هوش مصنوعی به کار می رود . از توابع مکاشفه ای [۳] 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 [۴] دو گره را در یک درخت جستجو برای دیدن این که کدام بهتر از دیگری است مقایسه می کند .

برنامه نویسی ژنتیک [۵] – یک روش برنامه نویسی است که الگوریتم ژنتیکی [۶] را برای تمام برنامه های کامپیوتری توسعه می دهد . در برنامه نویسی ژنتیکی ، جمعیّت برنامه ها برای حل مسأله ها توسعه می یابد . از برنامه های ژنتیکی می توان برای حل مسایل طبقه بندی ، کنترل ، روبوتیک ، بهینه سازی ، تیوری بازی ها و الگو شناسی استفاده نمود . [۷] برنامه نویسی ژنتیک توسط گروه جان کوزا [۸] توسعه داده شد .

کاربردهای هوش مصنوعی

برخی از کاربردهای هوش مصنوعی در این جا آمده است :

---

[۱] heuristics
[۲] Babylon / WordNet 2.0
[۳] heuristic functions
[۴] heuristic predicates
[۵] genetic programming (GP)
[۶] Genetic Algorithm (GA)
[۷] Babylon / FOLDOC
[۸] John Koza

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

تیوری بازی ها [۱] – بازی ها مواردی خوب برای تحقیق می باشند زیرا بازی ها کوچک و جامع هستند . و بنابراین به آسانی برنامه ریزی می شوند . بازی ها مدل های خوبی از وضعیّت های رقابتی می توانند باشند ، بنابراین روش های طرّاحی شده برای تیوری بازی ها شاید بتوانند در مسایل عملی هم به کار گرفته شوند . 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 ، امّا 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 وضعیّت ها نگاه می کنند .

سخن شناسی [۲] – سخن شناسی ، توانایی سیستم های کامپیوتری برای پذیرش ورودی سخن و کار بر روی آن یا تبدیل آن به صورت نوشتاری است . [۳] در دهه ی ۱۹۹۰ ، سخن شناسی کامپیوتری به سطحی کاربردی برای اهدافی محدود رسید . خطوط هوایی ایالات متحده ی آمریکا صفحه کلیدی درختی را که برای اطّلاعات 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 کاملاً مناسب بود .

فهم زبان طبیعی [۴] – فقط دریافت (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 .

تصویر مجازی در کامپیوتر [۵] – موردی در روبوتیک است که در آن برنامه ها تلاش می کنند اشیایی که به صورت تصاویر دیجیتالی توسط دوربین های ویدیویی دریافت کرده اند را تشخیص دهند و در

---

[۱] game playing

[۲] speech recognition

[۳] Babylon / Britannica Concise Encyclopedia

[۴] understanding natural language – زبان طبیعی متضاد کد کامپیوتری یا زبان مصنوعی می باشد .

[۵] computer vision

نتیجه ربات ها بتوانند ببینند . جهان از اشیایی سه بعدی تشکیل شده است امّا 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 ، امّا تصویر مجازی کامپیوتری کامل ، اطّلاعات سه بعدی ناتمام که فقط مجموعه ای از دیدهای دو بعدی نیستند را لازم دارد . در حال حاضر فقط راه های محدودی برای نمایش اطّلاعات سه بعدی به صورت مستقیم وجود دارد و این راه ها از قرار معلوم به خوبی راه هایی که انسان ها استفاده می کنند نمی باشد .

سیستم های خبره [۱] – سیستم های خبره تلاش می کنند که دانش یک انسان خبره (ماهر) را بگیرند و آن را در یک سیستم کامپیوتری پیاده سازی نمایند . از سیستم های خبره انتظار می رود که بتوانند کارهایی که به یک فرد خبره نیاز دارند را انجام دهند ، مثل پزشکی ، زمین شناسی و مشورت در سرمایه گذاری . سیستم های خبره برخی از موفق ترین کاربردهای هوش مصنوعی بوده اند چون که این برنامه ها باید در دنیای واقعی کار کنند و با برخی از مشکلات مهم موجود در هوش مصنوعی مواجه شده اند ؛ مثل : کمی اطّلاعات ورودی مناسب و استدلال بر پایه ی احتمال . یکی از اوّلین سیستم های خبره در سال ۱۹۷۴ 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 تأثیر آن چه که خبرگان به آن ها می گفتند در یک چارچوب کاری معین قرار داشت . در وضعیّت فعلی هوش مصنوعی ، این مطلب باید درست باشد .

طبقه بندی اکتشافی [۲] – یکی از بیش ترین انواع عملی (امکان پذیر) سیستم ها ی خبره ی ارایه شده توسط هوش مصنوعی برای قراردادن برخی از اطّلاعات 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

---

[۱] expert systems

[۲] heuristic classification

پیشنهاد شده ، ثبت پرداخت او ، همچنین چیزی که او می خرد و ثبت مؤسّسه ای که او از آن خرید می نماید ، می باشد .

ارتباط میان هوش مصنوعی و فلسفه : هوش مصنوعی دارای ارتباطات زیادی با فلسفه می باشد ، مخصوصاً با فلسفه ی تحلیلی مدرن [۱] ارتباط زیادی دارد . هر دو فکر و عقل سلیم را مطالعه می کنند .

پیش نیازهای هوش مصنوعی : باید ریاضیّات و مخصوصاً منطق ریاضیّات [۲] 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 [۳] ، لیسپ [۴] و پرولوگ [۵] 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 [۶] بود .

برخی از سازمان ها و موسساتی که در زمینه ی هوش مصنوعی فعالیت می کنند :
انجمن آمریکایی هوش مصنوعی [۷] ، کمیته ی هماهنگ کننده ی اروپا برای هوش مصنوعی [۸] و جامعه ی

---

[۱] modern analytic philosophy

[۲] mathematical logic

[۳] C

[۴] Lisp

[۵] Prolog

[۶] Java

[۷] The American Association for Artificial Intelligence (AAAI) با آدرس اینترنتی http://www.aaai.org

[۸] European Coordinating Committee for Artificial Intelligence (ECCAI) به آدرس اینترنتی http://eccai.org

هوش مصنوعی

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

هوش مصنوعی و شبیه سازی رفتار [۱] جوامع علمی علاقه مند به هوش مصنوعی می باشند . شرکت ماشین آلات محاسبه کننده [۲] یا SIGART یک گروه مهم در زمینه ی هوش مصنوعی می باشد . کنفرانس بین المللی هوش مصنوعی [۳] . تعاملات الکترونیک با هوش مصنوعی [۴] و هوش مصنوعی [۵] و روزنامه ی تحقیقات تحقیقات هوش مصنوعی [۶] و تعاملات IEEE با تحلیل الگو و ماشین های هوشمند [۷] 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 .

---

[۱] Society for Artificial Intelligence and Simulation of Behavior (AISB) با آدرس اینترنتی http://www.cogs.susx.ac.uk/aisb

[۲] Association for Computing Machinery (ACM) به آدرس اینترنتی http://www.acm.org/sigart

[۳] The International Joint Conference on AI (IJCAI) به آدرس اینترنتی http://www.ijcai.org

[۴] *Electronic Transactions on Artificial Intelligence* با آدرس اینترنتی http://www.ida.liu.se/ext/etai

[۵] به آدرس اینترنتی http://www.elsevier.nl/locate/artint

[۶] *Journal of Artificial Intelligence Research* به آدرس اینترنتی http://www.jair.org

[۷] 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 ؛ ساخت سیستم های هوشمند ؛ انجام بعضی از کارها مثل بازی شطرنج که به نظر می رسد برای انجام آن ها می توانیم از هوش مصنوعی کمک بگیریم ؛ انجام کارهایی مثل خنثی کردن مین

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

یا تمیز کردن استخر شنا که برای انسان خطرناک یا کسل کننده هستند ، از دلیل هایی هستند که ما هوش مصنوعی را مطالعه می نماییم .

هوش مصنوعی چیست؟

در اینجا چهار تعریف زیر را ارایه می کنیم :

۱- **سیستم هایی که شبیه انسان ها عمل می کنند.**
۲- **سیستم هایی که همانند انسان ها فکر می کنند.**
۳- **سیستم هایی که معقولانه [1] فکر می کنند.**
۴- **سیستم هایی که معقولانه عمل می کنند.**

دوّمین و سوّمین مورد می توانند در طبقه بندی هوش مصنوعی قوی قرار گیرند و موارد اوّل و چهارم بیش تر در طبقه بندی هوش مصنوعی ضعیف قرار می گیرند . در زیر توضیحاتی در مورد تعریف هایی که ذکر شد آمده است :

عمل کردن مثل انسان : آزمایش تورینگ [2] – 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 کاملاً 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 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 .

---

[1] rationally
[2] Turing test

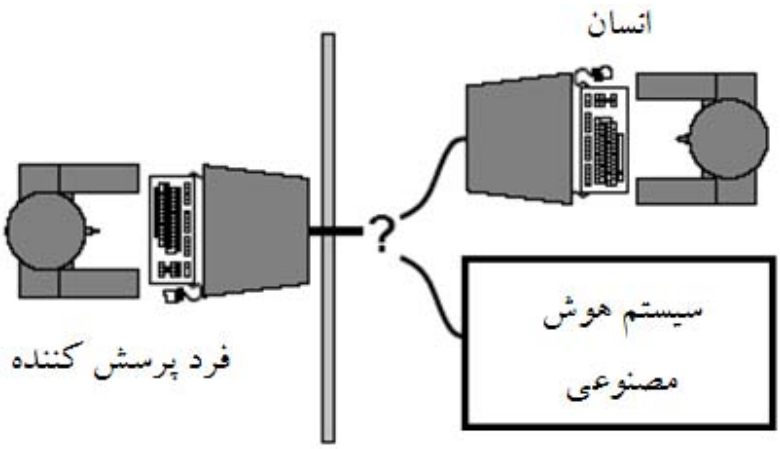
هوش مصنوعی

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

یک ماشین که آزمایش را می گذراند باید کاملاً هوشمند باشد ، ولی یک ماشین ، برای تقلید ، بدون دانستن به اندازه ی کافی در مورد افراد می تواند هوشمند باشد . کتاب مغز کودکان [۱] دانیل دنت [۲] 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 تصوّر می کنند که یک برنامه ی نسبتا گنگ هوشمند می باشد . [۳] اجزای اصلی پیشنهاد شده برای هوش مصنوعی عبارتند از : دانش ، استدلال ، زبان و آموزش .

اشکالات آزمایش تورینگ این بود که : آزمایش تورینگ تجدید پذیر [۴] ، سودمند [۵] یا جوابگو [۶] برای تحلیل ریاضی نیست.



---

[۱] Brainchildren

[۲] Daniel Dennett

[۳] http://www-formal.stanford.edu/jmc/whatisai/node1.html

[۴] reproducible

[۵] constructive

[۶] amenable

هوش مصنوعی

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

هنوز هیچ برنامه ای از آزمایش تورینگ ، سربلند بیرون نیامده است . یک برنامه در صورتی آزمایش تورینگ را با موفّقیّت می گذراند که قادر باشد زبان طبیعی را بفهمد ؛ چیزهایی که می داند (دانش) را بتواند بیان (ارایه) کند ؛ دانش داشته باشد و بتواند استدلال کند .

فکر کردن مانند انسان : علم شناخت [۱]

علم شناخت برای به وجود آوردن تیوری هایی که چگونگی عملکرد مغز انسان را توضیح می دهند تلاش می کند . در این راه از مدل های کامپیوتری هوش مصنوعی و روش های تجربی روانشناسی استفاده می نماید . لازم است توجّه کنیم که بیش تر روش های هوش مصنوعی به طور مستقیم براساس مدل های شناختی نمی باشند ؛ اغلب دشوار است که روش های هوش مصنوعی به صورت برنامه های کامپیوتری ترجمه شوند . علم شناخت اساسا از هوش مصنوعی جدا 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 مسأله ، 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 اطّلاعات فکری انسان ، تشبیه "***مغز به صورت کامپیوتر***" به صورت لفظ به لفظ کاملاً انجام می شود . [۲]

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 اوّل می باشند . هدف دیگر ، استفاده از هوش مصنوعی کامپیوتری برای فهمیدن چگونگی فکر بشر

---

[۱] cognitive science

[۲] http://www.aaai.org/AITopics/html/cogsci.html

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 . " [۱]

علم عصب شناسی [۲] : شاخه ای از علم اعصاب است که اصول زیستی اتّفاقات روانی را مطالعه می کند . [۳]

علم شناخت و علم عصب شناسی 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 [۴] بیش از تشریحی حاکم می باشند ، چند مدرسه ی یونانی شکل های مختلف منطقی را توسعه دادند : نمادها [۵] و قوانین استنتاج [۶] برای تفکرات ، برای فکر مکانیزاسیون ممکن است به کار روند یا ممکن است به کار نروند. مشکلات این روش یکی این است که همه ی رفتارهای هوش با برّرسی های منطقی انجام نمی شوند . و دیگری این که میان حل یک مسأله در کل و حل آن در عمل تحت محدودیّت منابع گوناگون نظیر زمان ، محاسبه و دقت تفاوت وجود دارد .

عمل کردن به صورت منطقی (معقولانه)

---

[۱] همان منبع قبلی
[۲] cognitive neuroscience
[۳] WordNet 2.0
[۴] normative
[۵] notation
[۶] rules of derivation

هوش مصنوعی

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

رفتار منطقی [۱] **:** کار درست را انجام می دهد . **کار درست ،** کاری است که بیش ترین دستیابی به هدف را داراست . بدون فکر هم می توانیم کار درست را انجام دهیم ؛ امّا فکر کردن باید در انجام کار معقولانه وجود داشته باشد. مزایای عمل کردن به صورت منطقی یکی این است که کلّی تر است و دیگری این است که هدفش به خوبی تعریف شده است . ارسطو [۲] می گوید : هر هنر ، هر تحقیق و غیره ، هر عمل و عکس العمل برای انجام صحیح ، احتیاج به فکر دارد .

تاریخچه ی هوش مصنوعی

در سال ۱۹۴۳ ، مک کلوچ و پیتز [۳] : 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 اوّلیه ، شامل برنامه ی چک کننده ی ساموئیل [۴] ، نظریّه ی منطقی نوئل و سیمون [۵] ، ماشین هندسی گلرنتر [۶] | سال ۱۹۵۶ نشست دارتمود [۷] : " هوش مصنوعی " پذیرفته شد ، الگوریتم کامل رابینسون [۸] برای استدلا ل منطقی | در سال های۱۹۶۶ تا ۱۹۷۴ ، پیچیدگی محاسباتی پیدا می کند و پژوهش در مورد شبکه ی عصبی [۹] تقریبا ناپدید می شود | در سال های ۱۹۶۹ تا ۱۹۷۹ توسعه ی اوّلیه ی سیستم های برمبنای دانش [۱۰] | در سال های ۱۹۸۰ – ۸۸ توسعه ی عظیم سیستم های خبره ی صنعتی | در سال های ۱۹۹۸

---

[۱] rational behavior

[۲] Aristotle

[۳] McCulloch وPitts

[۴] samuel

[۵] Newell & Simon

[۶] Gelernter

[۷] Dartmouth meeting

[۸] Robinson

[۹] neural network

[۱۰] knowledge-based systems

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

–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 عامل ها و ..... | سال ۲۰۰۳ به سطح انسانی هوش مصنوعی پرداخته می شود .

زندگی مصنوعی [۱] – 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 چیزهای ابتدایی 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 [۲] 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 مسأله استفاده می نمایند . [۳]

وضعیّت دانش

– در حال حاضر کدامیک از موارد زیر می تواند انجام شود؟

اجرای یک بازی تنیس . (می شود) | رانندگی با ایمنی در یک جاده ی مارپیچی کوهستانی . (می شود) | خرید مایحتاج هفتگی از فروشگاه های های وب (می شود) | اجرای یک بازی پل (می شود) | ترجمه

---

[۱] Al یا ALife یا Artificial Life

[۲] neomorphic

[۳] http://www.aaai.org/AITopics/html/alife.html

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) .

توضیحی در مورد بازی پل ، بازی ای کارتی (با ورق) و چهار نفره می باشد . مانند سایر بازی های کارتی ، یکی از خصوصیات اوّلیه ی این بازی این است که بازی ای با اطّلاعات ناقص می باشد که در آن بازی کننده های مختلف دارای اطّلاعات مختلفی در مورد وضعیّت واقعی بازی می باشند . این یکی از خصوصیات این بازی می باشد که باعث می شود ماشین ها بازی کننده های شایسته ای برای بازی پل نباشند .[۱] در زیر تصویری از کارت های این بازی را مشاهده می کنید :

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

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل دوّم

عامل های هوشمند [۱]

[۱] Intelligent agents

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

ریوس مطالب

- عامل ها و محیط ها
- عقلانیّت [۱]
- PEAS (ارزیابی عملکرد [۲] ، محیط [۳] ، عمل کننده ها [۴] و حسگرها [۵])
- انواع محیط
- انواع عامل ها

---

[۱] rationality

[۲] Performance measure یا معیار کارایی

[۳] Environment

[۴] Actuators

[۵] Sensors

عامل ها

* یک عامل می تواند هر چیزی باشد که در یک **محیط** عمل می کند ؛ [وضعیّت] محیط خود را با استفاده از **حسگرها** دریافت می کند ؛ روی محیط خود با استفاده از **عمل کننده ها** عمل می نماید و طوری عمل می کند که بیش ترین نزدیکی را به اهدافش داشته باشد .

تعریف های دیگری در مورد عامل :

- **تعریف راسل و نورویگ (R & N) :** هر چیزی که می تواند وضعیّت محیط خود را با استفاده از حسگرها دریافت کند و توسط محرّک ها بر روی محیط عمل کند یک عامل است .

- **وولریج [1] :** یک عامل ، یک سیستم کامپیوتری است که در یک محیط قرار داده می شود و قادر است به صورت خودمختار عمل نماید .

* **ابزار ادراکی برای تحلیل سیستم ها :** روبوت ها [2] ، سافتبوت ها [3] ، چراغ های ترافیک ، ترموستات ها [4] و غیره

* عامل های هوشمند از این جهت برای ما اهمّیّت دارند که به هوش مصنوعی کمک می کنند .

مثال هایی از عامل ها

---

[1] Woolridge
[2] robots
[3] softbots
[4] thermostats

عامل انسانی مثل : چشم ها ، گوش ها ، پوست ، حس چشایی [۱] و ... برای حسگرها و دست ها ، انگشتان ، پاها ، دهان [۲] و ... برای عمل کننده ها

برای روبوت : دوربین ، دوربین مادون قرمز [۳] ، ضربه گیر یا سپر [۴] و ... برای حسگرها و گیرنده ها (ابزارهایی که چیزی را می گیرند) [۵] ، تایرها ، نورها ، بلند گوها و ... برای عمل کننده ها

برای عامل نرم افزاری یا سافتبوت ، حسگرها به صورت توابع هستند و ورودی توابع ، اطّلاعات آماده شده به صورت رشته های بیتی یا نشان ها (سمبل ها) ی کد شده هستند . و توابع ، که عمل کننده ها نیز هستند خروجی ها را اجرا می کنند . پس توابع هم به صورت حسگر هستند و اطّلاعات را به صورت رشته های بیتی یا نشان ها (سمبل ها) ی کد شده می گیرند و هم به صورت عمل کننده هستند که بر روی محیط ، عمل می کنند .

خودمختاری [۶] در عامل ها

عامل ها وظایف خود را تا حد زیادی به صورت مستقل انجام می دهند و به وسیله ی کاربران یا برنامه های دیگر برنامه ریزی می شوند و این برنامه ها عامل را برای انجام کارش راهنمایی می کند . مبنای سیستم های خودمختار ، تجربه و دانش آن ها می باشد . عامل ها به دانش اوّلیه نیاز دارند و باید بتوانند یاد بگیرند و برای کارهای به مراتب پیچیده تر دارای انعطاف باشند .

مطلب دیگری در مورد خودمختاری

---

[۱] taste buds
[۲] mouth
[۳] infrared
[۴] bumper
[۵] grippers
[۶] autonomy

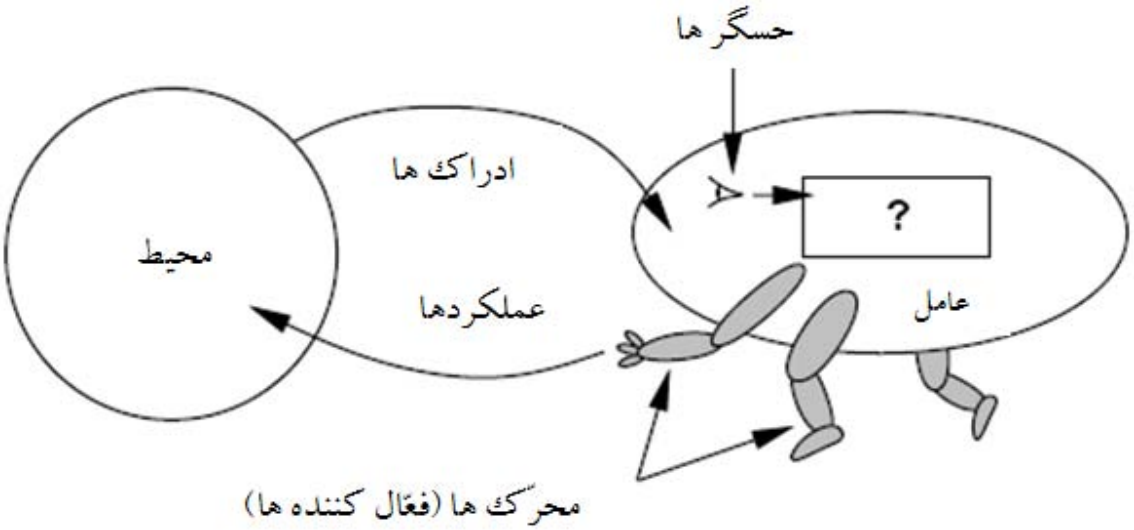
عامل ها اغلب دارای خود مختاری هستند ولی این خودمختاری ، اغلب به صورت کامل نمی باشد . حال مشکلی در این مورد وجود دارد و آن این است که عامل خود مختار چه هنگام باید درخواست کمک کند ؟ . یک عامل خودمختار قادر است با توجّه به دریافت هایش تصمیم بگیرد .

چگونه با این دردسرها رفتار کنیم ؟ : عامل ها دوست دارند که کاری که انجام می دهند دارای پیش نیاز نباشد . مشخص کردن این که در هر موردی باید چه کاری انجام دهند مشکل است . در مورد عامل ها بهتر است فرض را بر این بگذاریم که آن ها هوشمند هستند و سپس در مورد آن ها صحبت کنیم .

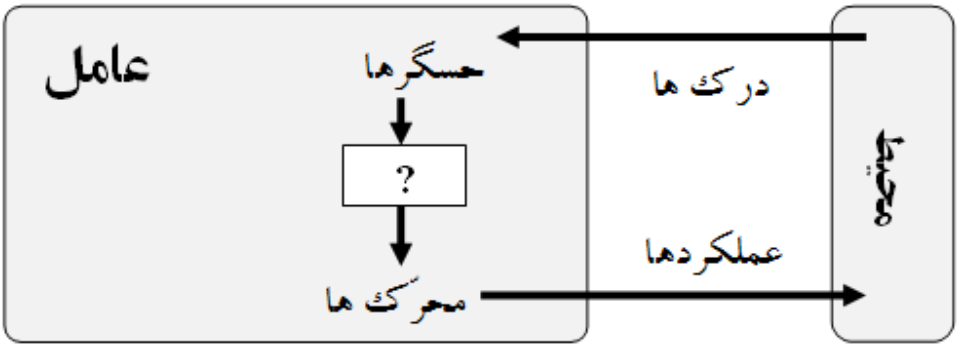
عامل ها و محیط ها

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 " عامل ها " می باشند . تحقیقات دانشمندان هوش مصنوعی در حال افزایش توانایی ، استقلال و ... عامل ها می باشد .[۱] تابع عامل از تاریخچه های ادراکی ، به عملیّات نگاشت می کند : $f : p^* \rightarrow A$. برنامه ی عامل ، در معماری فیزیکی برای تولید f اجرا می شود.

[۱] http://www.aaai.org/AITopics/html/agents.html



یا



ادراک ها (دریافت ها) : اطّلاعات فرستاده شده به حسگرهای یک عامل می باشند . مثل ، نور ، صدا ، امواج الکترو مغناطیسی و علایم (سیگنال ها) .

حسگرها : روش های های یک عامل برای جمع آوری اطّلاعات در مورد محیطش می باشند . مثل : چشم ها ، گوش ها ، سلول های فتوالکتریک و ...

محرّک ها : عامل ، به وسیله ی محرّک ها بر روی محیط ، عمل می نماید . تایرها ، بازوها ، رادیوها ، نورها و ... مثال هایی از محرّک ها هستند .

عملکرد : کار یا عملی است که بر روی محیط انجام می شود ؛ مثل : حرکت و غلطیدن .

ساختار عامل های هوشمند

عامل ، از معماری و برنامه تشکیل شده است . *معماری ، ابزاری است که به وسیله ی آن می توان برنامه ی عامل را اجرا کرد .* مثل : کامپیوتر همه منظوره ، ابزار تخصصی ، روبوت و ...

برنامه ی عامل ها

ما می توانیم رفتار عامل مان را با یک تابع F توصیف نماییم . داریم :

عملکرد = (تاریخچه ی ادراکی ، ادراک فعلی) F

به کار گیری این تابع ، کار یک برنامه ی عامل می باشد .

مثال : دنیای جاروبرقی

بیایید با یک مثال خیلی ساده شروع کنیم ؛ فرض کنید که دو فضای A و B وجود دارند که هر فضا یا می تواند تمیز یا کثیف باشد و این ، محیط عامل است . حسگر ها عبارتند از : حسگر کثیفی و تعیین محل . محرّک ها عبارتند از : جارو برقی و چرخ ها . ادراک ها هم تمیز و پاک بودن محیط می باشد . عملکردها ؛ حرک به چپ ، راست ، مکش و هیچ کار می باشند . در این مثال ساده ، ما می توانیم همه ی رشته های ادراکی ممکن و عملکردهای وابسته را لیست نماییم و این ، یک عامل برمبنای جدول [1] نام دارد .

---

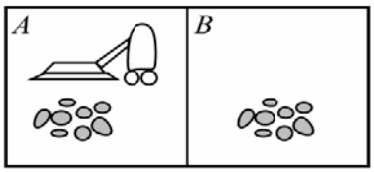
[1] table-based

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

عقلانیّت [۱]

به طورکلّی ، عقلانیّت به معنی " انجام چیز درست می باشد " در این مورد دقّت بیش تری لازم می باشد ؛ "چیز درست " ، چیست ؟ ما به یک تعریف از موفّقیّت هم نیازمندیم . یک عملکرد عقلانی (عاقلانه یا از روی عقل) ، عملکردی است که معیار کارایی (ارزیابی عملکرد) یک عامل ، ادراک ها و عملکردهای آن را بیشینه می نماید . راسل و نورویگ در این باره می گویند : 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 اطّلاعات هم ممکن است یک عملکرد عقلانی باشد ؛ به عنوان مثال ، عبور بدون توجّه از خیابان ، نامعقولانه می باشد . عامل های عقلانی باید دارای توانایی یادگیری (آموزش) [۲] 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 . آموزش ، ممکن است به معنی کاهش نامعلومی (احتمال) یا ارایه ی ملاحظاتی به حساب یا گزارش باشد .



عامل های یک جارو برقی

---

[۱] rationality

[۲] learning

تـابع Reflex-Vacuum-Agent([location,status]) یـک عملکـرد را برمـی گردانـد . در صورتی که وضعیّت [1] برابر با کثیف [2] باشد مکش را برمی گرداند و در غیر این صورت ، در صورتی که محل [3] برابر با A باشد ، راست را برمی گرداند و در صورتی که محل برابر با B بود ، چپ را بر می گرداند .

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]
⋮	⋮

عامل عقلانی [1] - یک عامل هوشمند هر کدام از اعمالی که مقدار معیارکارایی ارایـه شـده توسـط رشته ی ادراکی مورد نظر را به بیش ترین مقدار افزایش می دهد انتخاب می نماید . عامـل هوشـمند ، بـه همـه

---

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

چیز آگاهی [۲] ندارد . دریافت ادراکی ممکن است همه ی اطّلاعات را به وجود نیاورد. عامل هوشمند ، نهان بین [۳] هم نمی باشد . عملکرد به وجود آمده ممکن است به صورتی که انتظار داشتیم ، نباشد . در این صورت ، عامل هوشمند ، موفّقیّت آمیز نمی باشد . عامل هوشمند دارای ابتکار [۴] ، یادگیری [۵] و خودمختاری می باشد .

PEAS - برای طرّاحی یک عامل هوشمند ، ما باید محیط عملیّاتی [۶] را مشخص کنیم . مثلا درطرّاحی یک تاکسی خودکار ، موارد زیر را باید مشخص نماییم : معیارکارایی ؟؟ | محیط ؟؟ | عمل کننده ها (محرّک ها) ؟؟ | حسگرها ؟؟

مثال ، طرّاحی یک تاکسی خودکار ، معیارکارایی ؟؟ ایمنی [۷] ، مقصد [۸] ، مزایا [۹] ، رعایت قانون [۱۰] ، آسایش و راحتی [۱۱] و ..... | محیط ؟؟ خیابان های ایالات متحده ی آمریکا / آزاد راه های ایالات متحده ی آمریکا ، ترافیک [۱۲] ، عابران پیاده [۱۳] ، هوا [۱] و ..... | عمل کننده ها ؟؟ فرمان [۲] ، گاز ، ترمز [۳] ، بوق

---

[۱] rational agent
[۲] omniscient
[۳] clairvoyant
[۴] exploration
[۵] learning
[۶] task environment
[۷] safety
[۸] destination
[۹] profit
[۱۰] legality
[۱۱] comfort
[۱۲] traffic
[۱۳] pedestrains

بوق [۴] ، بلندگو یا نمایش دهنده [۵] و ..... | حسگرها ؟؟ ویدئو [۶] ، شتاب سنج [۷] ، درجه ها [۸] ، حس گرهای موتور [۹] ، صفحه کلید [۱۰] ، GPS [۱۱] و .....

مثال : عامل های خرید [۱۲] اینترنتی ، معیارکارایی ؟؟ | محیط ؟؟ | عمل کننده ها ؟؟ | حسگرها ؟؟ را باید مشخص نماییم . معیارکارایی ؟؟ قیمت [۱۳] ، کیفیت [۱۴] ، تناسب [۱۵] ، کارایی [۱۶] | محیط ؟؟ سایت های فعلی و آینده ی www ، فروشنده ها [۱۷] ، حمل کننده ها [۱] | عمل کننده ها ؟؟ نمایش دادن به کاربر [۲] ، دنبال کردن URL ، پرکردن فرم | حسگرها ؟؟ صفحات HTML (متنی ، گرافیکی ، اسکریپتی)

---

[۱] weather
[۲] steering
[۳] brake
[۴] horn
[۵] speaker/display
[۶] video
[۷] accelerometer
[۸] gauges
[۹] engine sensors
[۱۰] keyboard
[۱۱] مخفف Global Positioninig System است و یک سیستم مکان یابی ماهواره ای می باشد . (Babylon / Concise Oxford English Dictionary)
[۱۲] shopping
[۱۳] price
[۱۴] quality
[۱۵] appropiateness
[۱۶] efficiency
[۱۷] vendors

محیط ها

یک معیار برای وجود یک عامل ، وجود آن در یک محیط می باشد . لازم نیست که یک عامل به صورت فیزیکی در محیط باشد ؛ ممکن است محیط ، به صورت یک محیط نرم افزاری باشد . به نظر راسل و نورویگ ، محیط عملیّاتی تشکیل شده از : معیار کارایی ، محیط و محرّک هایی که برای عامل قابل دسترسی می باشند .

ویژگی محیط ها

عبارتند از : قابل مشاهده بودن [۳] ؛ قطعی / اتّفاقی بودن ؛ دوره ای (اپیزودیک) و ترتیبی بودن ؛ ثابت (استاتیک) و پویا (دینامیک) بودن ؛ گسسته و پیوسته بودن ؛ تک عاملی و چند عاملی بودن .

قابل مشاهده بودن : در صورتی که حسگرهای عامل همیشه اطّلاعات کاملی در مورد اجزای مربوط محیط بدهند ، محیط ، کاملاً قابل مشاهده است . به عنوان مثال ، محیط بازی شطرنج ، کاملاً قابل مشاهده می باشد . محیط 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 / اتّفاقی بودن : ما می توانیم فکر کنیم که جهان دارای انتقال میان وضعیّت ها می باشد . در این مورد داریم ، وضعیّت جاری * عملکردهای عامل ← (نتیجه می دهد) وضعیّت جدید . در صورتی که انتقال وضعیّت ، منحصر به فرد باشد ، جهان به صورت قطعی می باشد . به عنوان مثال ، بازی شطرنج ، بازی ای قطعی می باشد . دنیای جاروبرقی ، قطعی است . رانندگی یک اتومبیل ، تصادفی می باشد .

---

[۱] shippers

[۲] display to user

[۳] observability

نکته : در این جا ما اجتناب می کنیم که سؤالات را با مکانیک کوانتومی (ذره ای) موشکافی نماییم – ممکن است که جهان ، قطعی باشد ، امّا با توجّه به پیچیدگی اش به نظر تصادفی (اتّفاقی) برسد .

دوره ای و ترتیبی بودن : در حالت دوره ای ، هر عملکرد به صورت مستقل می باشد و عامل ، ادراک می کند ، تصمیم می گیرد و عمل می نماید و دوباره [از نو] شروع می نماید و تصمیم گیری بعدی ، وابسته به وضعیّت های قبلی نمی باشد . به عنوان مثال ، عامل فیلترکننده ی اسپم [1] ، دوره ای می باشد . در صورتی که عامل باید یک سری از عملیّات را برای رسیدن به یک عمل یا رسیدن به یک هدف انجام دهد ، محیط ترتیبی می باشد . در محیط ترتیبی ، باید به تصمیم گیری های آینده توجّه شود ؛ به عنوان مثال ، رانندگی یک اتومبیل ، ترتیبی می باشد .

ثابت و پویا (دینامیک) بودن : یک محیط ثابت ، مادامی که عامل در حال تصمیم گیری در مورد یک عملکرد می باشد ، ثابت باقی می ماند و عامل برای رسیدن به یک تصمیم ، تحت محدودیّت یا فشار زمانی نمی باشد ؛ محیط 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 ؛ به طور مثال ، رانندگی یک اتومبیل ، پویا می باشد . در محیط **نیمه پویا** ، محیط تغییر نمی کند ، امّا معیارکارایی در طول زمان تغییر می کند . انجام یک آزمایش در زمان محدود و بازی شطرنج با یک ساعت مثال هایی از محیط های نیمه پویا می باشند .

گسسته و پیوسته بودن : ما می توانیم در مورد گسسته و پیوسته بودن ادراک ها ، عملکردهای عامل یا وضعیّت 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 . محیط گسسته ، شبیه محیط محدود نمی

---

[1] spam ، پیام های نامربوط یا نامناسبی می باشند که در اینترنت برای تعداد زیادی از کاربران یا گروه های خبری فرستاده می شوند .

باشد . یک محیط فیلترکننده ی اسپم ، گسسته می باشد ، حتی در میان تعداد (قابل شمارش) نامحدود از پیام های الکترونیکی . یک محیط پیوسته ، دارای متغیّرهای تغییر کننده به صورت پیوسته است . به طور مثال ، زاویه های فرمان در یک محیط رانندگی اتومبیل . در مورد 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) .

تک عاملی یا چندعاملی بودن : در محیط تک عاملی ، عامل ما در حال عمل برروی خودش می باشد . به طور مثال ، محیط فیلترکردن اسپم ، به صورت تک عاملی می باشد . در محیط چند عاملی ، عملکردها / اهداف / روش های عامل های دیگر باید به حساب آیند . مثل محیط بازی شطرنج . *گرچه یک جهان ممکن است دارای عامل های دیگری باشد ، امّا ما ممکن است به خاطر پیچیدگی استنتاج ، با آن به صورت تک عاملی و اتّفاقی رفتار نماییم* . به عنوان مثال یک عامل کنترل کننده ی علایم ترافیکی .

برخی مثال ها عبارتند از : بازی شطرنج ، ماشینی که در آن پول می ریزند و کالایی را دریافت می کنند [۱] ، روبوتی که در هارنی [۲] به من قهوه می دهد ، مدار مریخ [۳] ، عامل مکالمه ای و عامل تشخیص طبی .

مثال :

	تاکسی	فروشگاه اینترنتی	تخته نرد
قابل مشاهده ؟؟	نه	نه	بله

[۱] slot-machine

[۲] Harney : نام جایی در ایالات متحده ی آمریکا

[۳] Mars orbiter

نتیجه ی فرایند ؟؟	نه	تاحدودی	نه
دوره ای ؟؟	نه	نه	نه
ایستا ؟؟	نه	نیمی	نیمی
مجزا ؟؟	نه	بله	بله
تک مولفه ای ؟؟	نه	بله (به جز حراج ها)	نه

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

ارزیابی عملکرد (معیار کارایی)

در مورد عامل جاروبرقی ، تعدادی از کاشی ها در یک زمان معین تمیز می شوند . انرژی ، اغتشاش [1] ، گم شدن قطعات مفید ، اثاثیه ی خراب ، خراش کف را هم در ارزیابی عملکرد در نظر می گیریم .

عامل های نرم افزاری ، معمولاً به صورت سافتبوت ها شناخته می شوند . در محیط های مصنوعی که براساس کامپیوترها و شبکه ها می باشند وجود دارند . می توانند خیلی پیچیده با نیازمندی های

[1] noise

زیاد به عامل باشند وب جهانی [۱] مثالی از عامل های نرم افزاری می باشند . در این عامل ها ، محیط های طبیعی و مصنوعی ممکن است با هم ادغام شده باشند .

عامل های متحرک [۲] ، در عامل های متحرّک ، برنامه ها می توانند از یک ماشین به ماشین دیگر بروند ، در یک محیط اجرایی مستقل از پایگاه [۳] اجرا می شوند ، به محیط اجرایی عامل نیازمندند ، تغییر پذیری ، لازم یا مناسب وضعیّت برای عامل نمی باشد . این عامل ها ، هزینه ی ارتباط را کاهش می دهند . کاربرد آن ها در بازیابی اطّلاعات توزیع شده و مسیریابی شبکه ی مخابراتی می باشد .

عامل های اطّلاعاتی [۴] ، رشد انفجاری اطّلاعات را مدیریّت می نمایند ، اطّلاعات چند منبع توزیع شده را به کار می گیرند یا مقایسه [۵] می نمایند . عامل های اطّلاعاتی می توانند ثابت یا متحرک باشند مثل ، اطّلاعات درون وب یا یک سند واقعی (مثلا یک سند کاغذی) ، برای صفحات تفسیر کننده ی وب شناخته می شوند و منبع داده ای در داده های بی ساختارند و در آن ها پاسخگویی به سؤال با استفاده از دانش قوی متدهای آماری انجام می شود .

برنامه های محیط ، شبیه سازی کننده های محیط برای آزمایش هایی که توسط عامل ها انجام می شود می باشند ، یک ادراک را به یک عامل ارایه می نمایند ، یک عملکرد را دریافت می نمایند و محیط را به روز می نمایند . اغلب به کلاس های محیط برای عملیّات وابسته یا انواع عامل ها تقسیم می شوند و خیلی از اوقات مکانیزم هایی را برای اندازه گیری عملکرد عامل ها ارایه می نمایند .

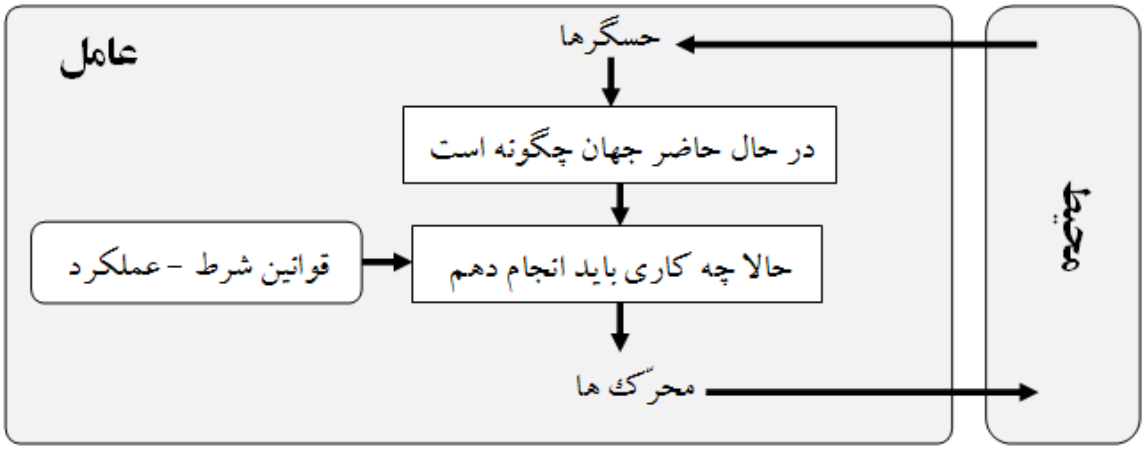
انواع عامل ها

---

[۱] World Wide Web
[۲] mobile agents
[۳] independent-platform
[۴] information agents
[۵] collate

چهار نوع اصلی وجود دارد : عامل های بازتابی ساده [۱] - عامل های بازتابی با حالت یا وضعیّت یا عامل هایی که وضعیّت جهان را حفظ می نمایند [۲] - عامل های براساس هدف یا هدف گرا [۳] و عامل های سودمند [۴] . همه ی این ها می توانند در عامل های آموزشی به کار گرفته شوند.

عامل های بازتابی ساده : دارای جدول جستجوی ساده می باشند و ادراک ها را به عملیّات نگاشت می نمایند (خیلی بزرگ و پرهزینه می باشند) و در آن ها تعدادی از وضعیّت 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 وضعیّت برابر کثیف (Dirty) بود ، مکش (Suck) را برمی گرداند ، در غیر این صورت ،

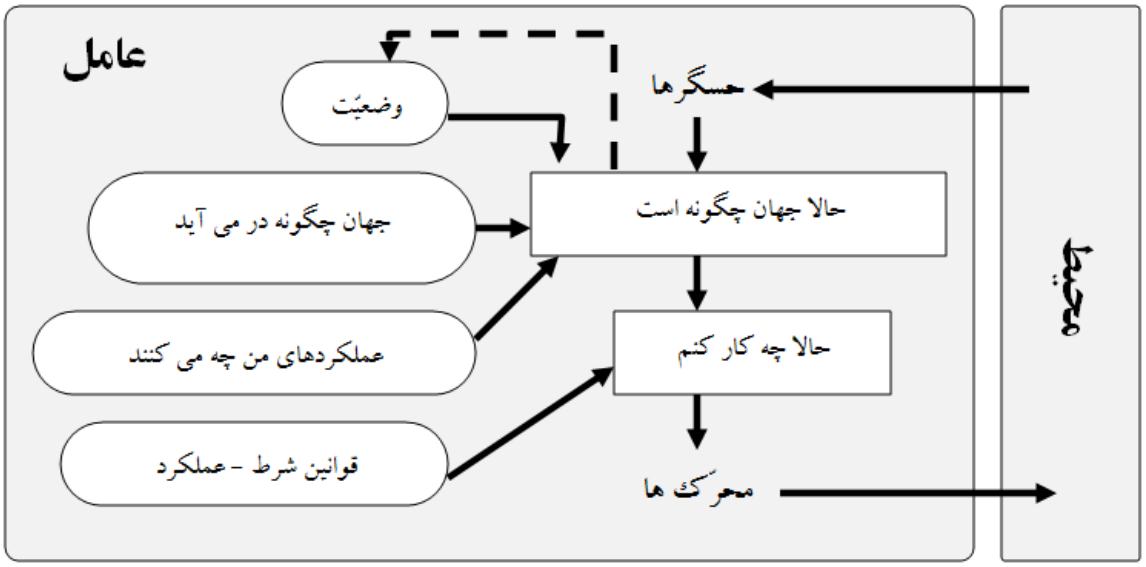
---

[۱] simple reflex agents
[۲] reflex agents with state
[۳] goal-based agents
[۴] utility-based agents

اگر محل [1] برابر A بود ، راست (Right) را برمی گرداند ، در غیر این صورت ، اگر محل برابر B بود ، چپ (Left) را بر می گرداند .

عامل هایی که وضعیّت جهان را حفظ می کنند (عامل های بازتابی براساس مـدل) :

اطّلاعات عامل به تنهایی در مورد مشاهده پذیری جزیی کـافی نمـی باشـند ، لازم اسـت کـه جریـان تغییـرات جهان را نگهداری نماییم و تکامل ها ، به طور مستقل از عامل یا سبب شده توسط عملکرد عامل می باشند .



مثال : تابع Reflex-Vacuum-Agent([location,status]) یک عملکرد را برمی گرداند

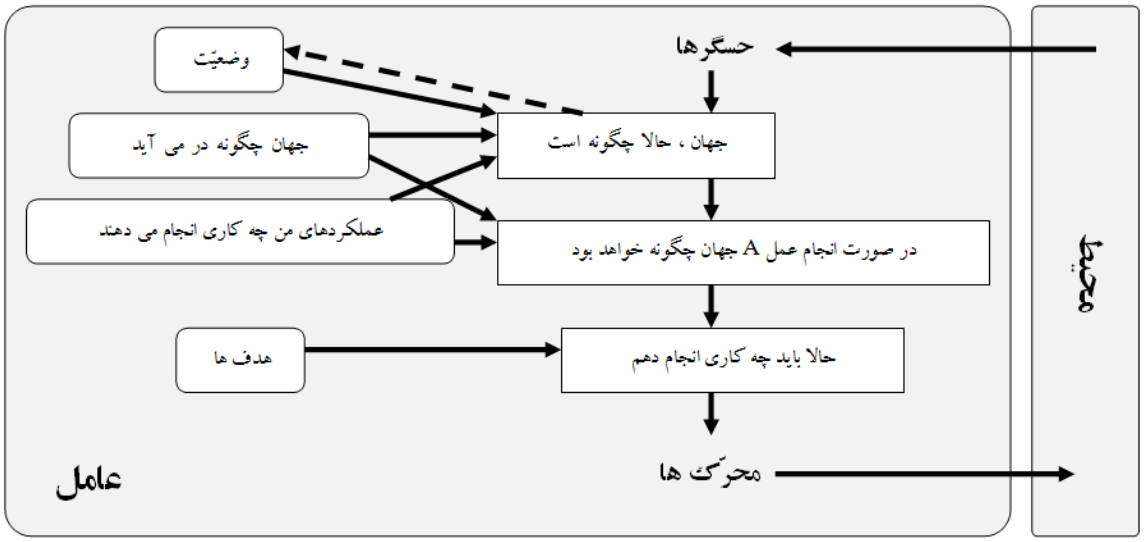
متغیّرهای static : last_A,last_B,numbers دارای مقدار اوّلیه ی ∞

در صورتی که وضعیّت (status) = کثیف (Dirty) باشد ...

---

[1] location

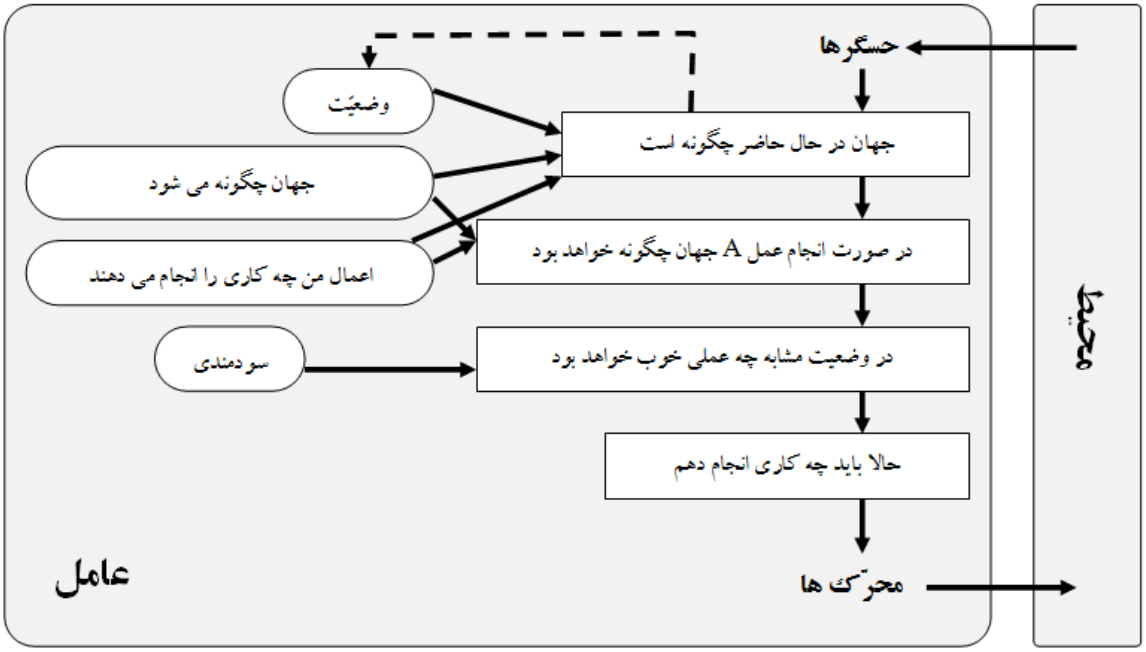
عامل های هدف گرا : در این گونه عامل ها وضعیّت و عملکردها نمی گویند که کجا برویم و به **اهداف** برای ساختن رشته ای از عملکردها (برنامه ریزی) نیازمندیم در هدف گرایی ، از قوانین یکسان برای اهداف مختلف استفاده می نماید و در بازتاب ، به مجموعه ای کامل از قوانین برای هر هدف نیاز خواهیم داشت .



مطالبی در مورد عامل های هدف گرا : دانستن وضعیّت فعلی محیط ، همیشه کافی نمی باشد . عملکرد درست ممکن است همچنین وابسته 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 مسأله 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 وضعیّت جاری آن و چیزی که در حال حاضر در تلاش برای رسیدن به آن است می باشد .

عامل های سودمند : در عامل های سودمند ، دانستن هدف ها ممکن است در محیط های با پیچیدگی بالا ، کافی نباشند ؛ چند رشته از عملکردها برای دسترسی به برخی اهداف لازمند (پردازش

دودویی) ؛ ممکن است نیازمند انتخاب میان عملکردها و رشته ها می باشیم و سودمندی ، وضعیّت را به عددی حقیقی نگاشت می نماید و درجه ی ارضا را بیان می کند و سبک و سنگینی میان موارد متناقض را مشخص می نماید ، از سودمندی برای مقایسه ی شرایط مطلوب وابسته به رشته های عملکرد استفاده می شود و 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 اتّفاقی ، خیلی مفید می باشند . سودمندی ها برخی اوقات یک چیز بحث انگیز در هوش مصنوعی می باشند . فرض می کنیم نتایج می توانند به صورت خطی در یک مقیاس یکسان ، مرتّب شده باشند . مثل : عامل رانندگی تاکسی . در این مورد ، طراح باید نتایج را به طور صریح ارزیابی نماید (به صورت کمّی و کیفی) . سودمندی در دامنه های با احتمال ، خیلی مفید می باشد ، مثل : تجارت برخط ، اکتشاف در محیط های دارای نامعلومی و قماربازی [۱] .



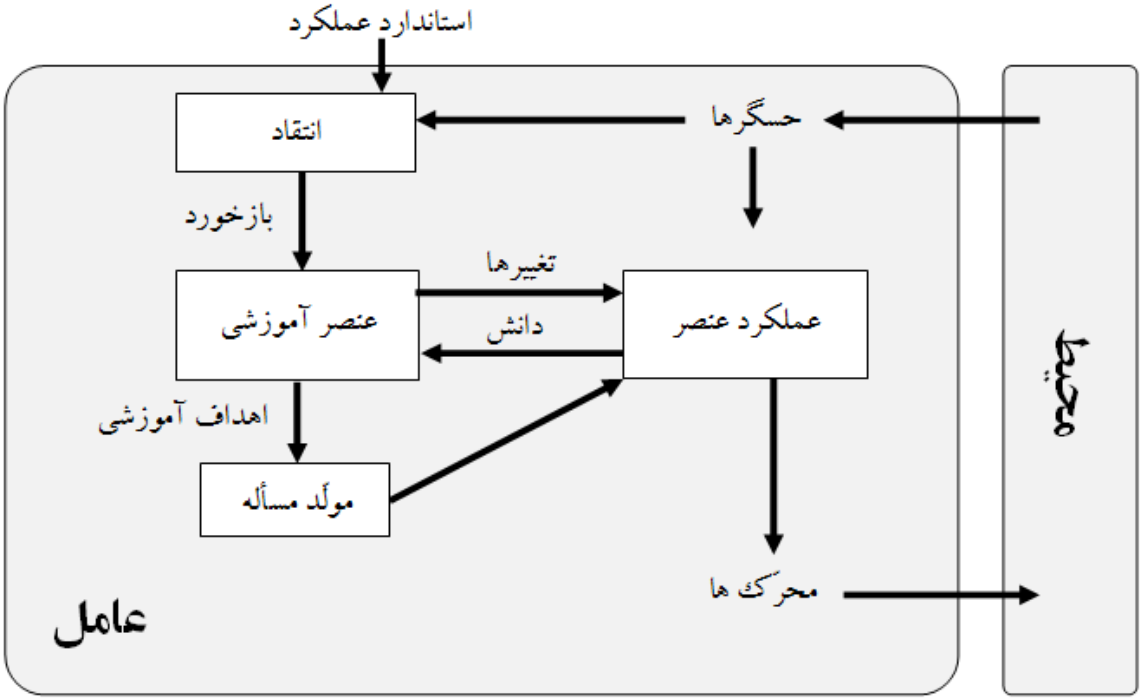
[۱] gambling

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

عامل های آموزشی : در این عامل ها عنصر آموزشی [۱] ، بهبودهایی را به وجود می آورد . عنصر کارایی [۲] ، انتخاب عملکردهای خارجی را ممکن می سازد . انتقاد [۳] ، جمع آوری بازخورد [۴] در مورد چگونگی عمل عامل می باشد . تولید کننده ی مسأله [۵] ، عملکرد (آزمایش) های پیشنهادی (اکتشافی) را در نظر می گیرد . اغلب ، یک عامل ممکن است نیاز به بروزرسانی برنامه ی عامل خود را داشته باشد . برنامه نویسان ممکن است فهم کاملی از محیط نداشته باشند . محیط ، ممکن است در طول زمان ، تغییر نماید در این مورد برنامه نویسی با دست ممکن است خسته کننده باشد . یک عامل آموزشی ، عاملی است که عملکرد خود را با توجّه 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* [۶] و هم به *عنصر آموزشی* [۷] نیازمند می باشد ؛ عنصر کارایی ، عملکرد یا عملکردهای جاری را انتخاب می نماید و عنصر آموزشی ، درستی عنصر کارایی را ارزیابی می نماید .

---

[۱] learning element
[۲] performance element
[۳] critic
[۴] feedback
[۵] problem generator
[۶] performance element
[۷] learning element



آموزش ، می تواند به صورت برخط [۱] یا برون از خط [۲] باشد ؛ آموزش ، ممکن است *غیرفعال* یا *فعال* باشد ؛ آموزش ممکن است نظارت شده یا نظارت نشده باشد . *انتساب اعتبار* [۳] در زمانی که آموزش در محیط های ترتیبی می باشد ، مسأله ای بزرگ می باشد .

نکته : 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

---

[۱] online

[۲] offline

[۳] credit assignment

عامل ها **، روی** ***محیط ها*** **به وسیله ی** ***محرّک ها*** **و** ***حس گرها*** **اثر می گذارند. تابع عامل چگونگی عملکرد عامل را در همه ی شرایط تشریح می کند. ارزیابی عملکرد (معیار کارایی) چگونگی 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 ؟ - قطعی ؟ - دوره ای ؟ - پویا ؟ - گسسته ؟ - تک مولفه ای ؟***

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

فصل سوّم

حل مسأله و جستجو [۱]

[۱] problem solving and search

خلاصه ی ریوس مطالب

- **عامل های حل کننده ی مسأله**
- **انواع مسأله**
- **فرمول بندی [1] مسأله**
- **مثال هایی از مسایل**
- **الگوریتم های اساسی (پایه ای) جستجو**

یک عامل هدف گرا می تواند برّرسی کند 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 مسأله را برّرسی می کنیم .

---

[1] formulation

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

جستجو به عنوان روش حلّ مسأله

بسیاری از مسایل می توانند به صورت رسیدن به یک حالت هدف [۱] ، از یک نقطه ی شروع دیده شوند . فضای حالت [۲] ، مسأله و راه حل های ممکن آن را به یک صورت رسمی تری تعریف می نماید ؛ معمولاً مسایل باید به صورت تجزیه شده [۳] باشند . عملکردهای عامل ، وضعیّت را تغییر می دهد و فضای حالت را برای یک راه حل ، جستجو (ملاقات) می نماید . اطّلاعات ، در مورد مسأله ی معین یا با دامنه ی عمومی می توانند برای بهبود جستجو استفاده شوند . این اطّلاعات عبارتند از : تجربه از موارد قبلی مسأله ؛ روش های بیان شده به صورت اکتشافات ؛ نوع های ساده تر مسأله و محدودیّت های وضعیّت های مشخّص (معیّن) مسأله .

چرا روش های جستجو را برّرسی می نماییم ؟

روش های جستجو ، روش هایی مهم برای حلّ بسیاری از مسایل می باشند . استفاده از جستجو ، به یک فرمول بندی مجزای مسأله و مراحل ممکن برای به وجود آوردن راه حل ها نیازمند می باشد و الگوریتم های جستجو ، پایه ای برای بهینه سازی [۴] و روش های برنامه ریزی می باشند .

عامل های حل کننده ی مسأله

یک عامل حل کننده ی مسأله ، برای پیداکردن یک رشته از عملکردها که به هدف می رسند تلاش می کند . به عنوان مثال ، چه رشته از حرکت ها یک مکعّب رابیک [۱] را حل می کنند ؟ . چگونه من از

---

[۱] goal state
[۲] state space
[۳] abstracted
[۴] optimization

هوش مصنوعی

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

دانشگاه فلوریدای جنوبی [۲] با اتومبیل به سمت لیورمور رانندگی نمایم ؟ . چگونه من می توانم اجزای روی یک تراشه را بچینم ؟ . کدام رشته از عملیّات یک روبوت را در یک فضا حرکت می دهد ؟ . در زیر تصویری از بازی مکعّب رابیک را مشاهده می نمایید :

برای فرمول بندی مسأله این موارد را در نظر می گیریم : وضعیّت های ممکن وابسته به دنیا برای حل مسأله کدامند ؟ . چه اطّلاعاتی برای عامل در دسترس می باشند ؟ . رفتن عامل از یک وضعیّت به وضعیّت دیگر چگونه می تواند باشد ؟ .

برای فرمول بندی هدف مسأله این موارد را در نظر می گیریم : وضعیّت هدف چیست ؟ . ویژگی های مهم وضعیّت هدف چه می باشند ؟ . چگونه عامل می فهمد که به هدف رسیده است ؟ در این مورد برّرسی می کند که آیا چند وضعیّت پایانی ممکن وجود دارد ؟ و آیا آن ها با هم برابرند یا برخی بهترند ؟ .

جستجو

---

[۱] 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]

[۲] University of South Florida = USF

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

تعریف جستجو : پردازه ی رسیدگی کردن به صورت ترتیبی به عملکردها برای پیداکردن رشته ای از عملیّات که ما را از شروع به هدف می رسانند ، جستجو نام دارد . یک الگوریتم جستجو ، یک رشته از عملیّات که برای عامل اجرا می شوند را برمی گرداند و جستجو معمولاً به صورت "برون خطی " انجام می شود . در این مورد نکته ای که باید در نظر بگیریم این است که ، فرض بر این است که محیط به صورت استاتیک (ثابت) می باشد ، همچنین فرض شده که محیط ، گسسته می باشد . محیط (معمولاً) به صورت قطعی می باشد .

برخی از مسایل جستجوی کلاسیک

مسایل سرگرمی : برای مطالعه به صورت مثال یا برای مقایسه ی الگوریتم ها ، مفید می باشد ؛ مثل : پازل ۸ – تایی ، دنیای جارو ، مکعّب رابیک و N – وزیر .

مسایل دنیای واقعی : معمولاً کار زیادی می برند ، امّا جواب معمولاً جالب می باشد . مثل : مسیر یابی ، مسافرت شخص دوره گرد ، طرح VLSI و جستجو در اینترنت

وضعیّت

ما معمولاً در مورد وضعیّتی که یک عامل در آن می باشد صحبت می کنیم . وضعیّت ، به مقادیر متغیّرهای مربوط توصیف کننده ی محیط و عامل ، اشاره می کند . به عنوان مثال در دنیای جارو ، وضعیّت (۰و۰) ، تمیز می باشد . در مسأله ی رومانی ، در t=0 در بخارست هستیم و در مکعّب رابیک ، وضعیّت ، نظم فعلی مکعّب (چگونگی قرار گرفتن مربّع های روی مکعّب در حال حاضر) می باشد .

فرمول بندی مسأله

فرمول بندی مسأله شامل موارد زیر است :

۱- **شناختن نوع مسأله :** این که عامل چه دانشی در مورد وضعیّت جهان و نتیجه ی عملکرد خودش دارد ؛ آیا اجرای عمل نیازمند به روز کردن اطّلاعات دارد ؟ و تعریف وضعیّت های جهان .

۲- **توضیح رسمی برای عملکرد عامل :** توضیح هدف و عملگرها که همان عملکردهای عامل می باشد .

انتخاب وضعیّت ها و عملکردها

مسایل برای حل باید **مجزا** باشند . وضعیّت های قابل شناخت در مدت فرایند حل مسأله باید توضیح داده شوند . عملکردها (عملگرها) عامل را از وضعیّتی به وضعیّت دیگر می برند و وابسته به وضعیّت ها ، توانایی عامل و خصوصیات محیط می باشند و انتخاب وضعیّت ها و عملگرهای مناسب ، می تواند باعث تفاوت میان این که یک مسأله می تواند حل شود یا نمی تواند حل شود بشود .

انواع مسأله

مسایل تک حالته [۱] : در صورتی که مسأله قطعی و کاملاً قابل مشاهده باشد ، در نتیجه ، مسأله ، تک حالته خواهد بود ؛ در این حالت ، عامل دقیقاً می داند در کدام حالت قرار خواهد گرفت و راه حل به صورت ترتیبی است . در این حالت ، جهان در دسترس می باشد و اثر عملکردها را می دانیم و عامل ، وضعیّتی که بعد از یک رشته از عملیّات در آن خواهد بود را می داند .

مسایل چندحالته [۲] : در صورتی که مسأله غیرقابل مشاهده باشد ، در نتیجه مسأله ، تطبیقی (ترکیبی [۳]) خواهد بود . عامل ، ممکن است تصمیمی در مورد کجا قرار گرفتن نداشته باشد ؛ راه حل (در

---

[۱] single-state problems
[۲] multiple-state problems
[۳] conformant

صورت وجود) ترکیبی می باشد . در مسایل چند حالته ، جهان فقط تا حدودی در دسترس می باشد و عامل به چند وضعیّت ممکن توجّه دارد .

مسایل احتمالی [۱] **:** در صورتی که مسأله ، احتمالی (غیرقطعی [۲]) و یا اندکی قابل مشاهده [۳] باشد ، در نتیجه ، مسأله ی احتمال می باشد . در مسایل احتمالی ، در طول اجرا به مشاهده و حس کردن نیازمندیم . همچنین به درخت عملیّات هم نیاز داریم .

مسایل اکتشافی [۴] **:** در صورتی که مسأله دارای فضای حالت ناشناخته باشد ، در نتیجه ، مسأله ، اکتشافی می باشد . در مسایل اکتشافی ، عامل نتایج عملکردش را نمی داند و آزمایش های عامل برای کشف وضعیّت های جهان و اثرات عملکردها می باشد .

مسایل خوب تعریف شده

در مسایل خوب تعریف شده ، وضعیّت اوّلیه ، نقطه ی شروع که عامل از آن شروع می کند می باشد ، مثلا در مورد مسأله ی کشور رومانی ، وضعیّت اوّلیه ، شهر آراد می باشد . فضای حالت ، مجموعه ای از همه ی وضعیّت هایی که از وضعیّت اوّلیه توسط رشته ای از عملکردها قابل دسترسی می باشند است و مسیر ، رشته ای از عملکردها که راهنمایی کننده از یک وضعیّت به وضعیّت دیگر می باشد هستند . عملکردها (عملگرها و توابع جانشین [۵]) می گویند که عامل چه عملکردهایی را می تواند داشته باشد ؟ . عملکردها ،

---

[۱] contingency problems
[۲] nondeterministic
[۳] partially observable
[۴] exploration problems
[۵] successor functions

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 وضعیّت ارایه شده ، یک وضعیّت هدف می باشد یا نه ؟ و راه حل ، مسیری از وضعیّت اوّلیه به یک وضعیّت هدف می باشد .

تابع جانشین

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

(In(0,0))→ ('Left',In(0,0)),('Right',In(0,0)),('Suck',In(0,0),'Clean')

در مسأله ی رومانی :

In(Arad) →
((Go(Timisoara),In(Timisoara),(Go(Sibiu),In(Sibiu)),(Go(Zerind),In(Zerind))

در جهان های اتّفاقی ، یک عملکرد ، شاید با تعدادی وضعیّت ها ، جفت شده باشد .

آزمون هدف

آزمون هدف ، تشخیص می دهد که آیا یک وضعیّت داده شده ، یک وضعیّت هدف است . در این مورد باید توجّه داشته باشیم که شاید یک وضعیّت هدف منحصر به فرد وجود داشته باشد یا تعدادی . مثلا در دنیای جاروبرقی ، هر فضای تمیز در بازی شطرنج ، مات و در مسأله ی رومانی ، در بخارست بودن وضعیّت هدف می باشند .

هزینه ی مسیر

هزینه ی مسیر ، هزینه ی یک عامل برای رفتن از وضعیّت اوّلیه به وضعیّتی که به تازگی برّرسی شده است می باشد و اغلب ، مجموع هزینه برای هر عملکرد می باشد و هزینه ی مرحله [۱] نام دارد . ما فرض می کنیم که هزینه های مراحل ، مثبت (غیرمنفی) هستند .

فضای حالت

ترکیب حالت های مسأله و توابع جانشین (راه های رسیدن به وضعیّت ها) منجر به مفهوم فضای حالت می شود و گرافی است که همه ی وضعیّت 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 مسأله صحبت می کنیم . به عنوان مثال در پازل ۸- تایی فضای حالت برابر است با : $\frac{9!}{2} = 181,000$ حالت که فضای حالتی ساده است . در مورد پازل ۱۵ – تایی ، فضای خالت برابر است با : تقریبا ۱.۳ تریلیون [۲] حالت که تقریبا ساده است . در مورد پازل ۲۴- تایی : تقریبا 10^{25} وضعیّت که فضای حالتی پیچیده است و در مسأله ی مسافرت شخص دوره گرد : !۲۰ = 2.43×10^{18} وضعیّت که خیلی پیچیده می باشد .

عملیّات حل مسأله

هزینه ی مسیر ، هزینه ی عامل را برای اجرای عملیّات در یک مسیر تعیین می نماید و مجموع هزینه های عملیّات فردی در یک مسیر می باشد . هزینه ی جستجو ، زمان و حافظه ی لازم برای محاسبه ی یک راه حل است و داریم مجموع هزینه = هزینه ی مسیر + هزینه ی جستجو . برای مسایل چند حالته ما از **مجموعه ای از وضعیّت ها** استفاده می نماییم و هزینه ها و دیگر تعریف ها تغییر نمی کنند . شکل محدود شده ی کلّی ، دارای الگوریتم زیر می باشد :

---

[۱] step cost

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

تابع Simple-Problem-Solving-Agent(percept) یک عملکرد را بر می گرداند

متغیّر static :

seq ، که یک ترتیب عملکرد است و در ابتدا دارای مقدار تهی می باشد

state ، توصیف وضعیّت فعلی جهان

goal ، یک هدف که به صورت اوّلیه دارای مقدار NULL می باشد

problem ، یک فرمول بندی مسأله

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** [۱] می باشد ؛ حل به صورت " چشم بسته " اجرا می شود. راه حل های **برخط** [۲] شامل عمل کننده ی بدون دانش تمام می شوند .

استراتژی های جستجو - یک استراتژی با چیدن گره ها به صورت مرتّب تعریف می شود . استراتژی ها به وسیله ی موارد زیر ارزیابی می شوند : تمامیّت [۳] – 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 [۴] - این روش ها فقط از داده های قابل دسترس از تعریف مسأله استفاده می کنند .

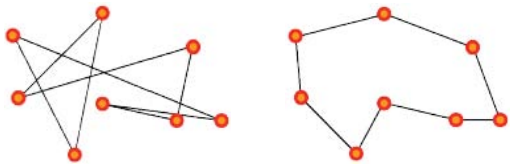
پیچیدگی

در مورد پیچیدگی الگوریتم ها نگران هستیم ، زیرا یک مسأله ممکن است در حالت کلّی قابل حل باشد امّا در حالت عملی خیلی طولانی باشد . با استفاده از تحلیل مجانبی [۵] ، تخمین زمان (یا تعداد عملیّات) لازم برای حل یک نمونه از اندازه ی n یک مسأله در زمانی که n به سمت بی نهایت می رود می توان پیچیدگی الگوریتم ها را ارزیابی نمود .

---

[۱] offline
[۲] online
[۳] completeness
[۴] uninformed search
[۵] asymptotic

مثال : مسأله مسافرت فروشنده ی دوره گرد [۱] – در این مسأله ، n شهر توسط جاده به هم متصل می باشند و مسأله ، پیدا کردن یک مسیر که از تمام شهرها عبور کند و حتی الامکان کوتاه ترین مسیر باشد است .



سختی ما در این مورد این است که الگوریتم های شناخته شده دارای پیچیدگی نمایی هستند و تعداد عملیّات به صورت نمایی رشد می کند .

چرا پیچیدگی نمایی " مشکل " است ؟

این به آن معنی است که تعداد عملیّات لازم برای محاسبه ی راه حل دقیق مسأله به صورت نمایی با اندازه ی مسأله 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 کامپیوترهای سریع ما 10^{12} عملیّات در ثانیه را ما داریم . **بنابراین** ، در حالت کلّی ، مسایل با پیچیدگی نمایی در همه ی موارد قابل حل نمی باشند امّا در اندازه ها ی کوچک نمایی قابل حل می باشند !

---

[۱] Traveling Salesperson Problem (TSP)

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

پیچیدگی

مسایل با زمان چندجمله ای [۱] (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 ، در مرتّب 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 . مسأله در این جا این است که ما نمی توانیم وضعیّت مرتبه ی چند جمله ای را مشخص کنیم ، شاید خیلی بزرگ باشد ! حال سؤال این است که 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 که مسایل غیرقطعی چندجمله ای [۲] جزء این دسته هستند . برای مثال ، مسأله ی مسافرت فروشنده ی دوره گرد این گونه است .

نکته ای در مورد مسایل NP-hard

یک مسأله به صورت چندجمله ای غیرقطعی است اگر یک راه حل (گمان) برای آن مسأله پیدا شده باشد . به عبارت دیگر ، الگوریتمی وجود داشته باشد که دارای زمان چند جمله ای باشد ؛ چه گمان درست باشد و چه نباشد . دشواری ما این جاست که شاید هیچ الگوریتم چندجمله ای که بتواند مسأله را حل کند وجود نداشته باشد و در صورت وجود هم ممکن است نمایش آسان نباشد و معمولاً قابل کاهش به صورت مسایل شناخته شده هم نمی باشد . در عمل ، با فرض این که $P \neq NP$ ، الگوریتم ها زمان بیش تر از چند جمله ای برای حل مسأله دارند .

به دست آوردن پیچیدگی با استفاده تحلیل مجانبی

---

[۱] polynomial-time

[۲] nondeterministic-polynomial-time(NP)

اگر اندازه ی ورودی مسأله برابر n باشد ، تعداد عملیّات برای حل مسأله ، f(n) باشد و g(n) یک تابع ارایه شده باشد . برای برخی از مقادیر k ، تعریف f(n) ، در صورتی که f(n)≤k g(n) باشد ، برابر O(g(n)) می باشد . به عنوان مثال ، $O(n^2)$ بدتر از O(n) می باشد امّا برای n<110 داریم $n^2+1<100n+1000$.

یادآوری : مسایل خوب تعریف شده

در مسایل خوب تعریف شده ، وضعیّت اوّلیه ، نقطه ی شروعی که عامل از آن آغاز می نماید است . فضای حالت ، مجموعه ی همه ی وضعیّت های قابل دسترس از وضعیّت اوّلیه توسط هر رشته از عملیّات می باشد و مسیر ، رشته ای از عملیّات مرتبط از یک وضعیّت به وضعیّت دیگر می باشد . عملیّات (عملگرها ، توابع جانشین) ، تشریح مجموعه ای از عملیّات ممکن است . آزمون هدف ، این مطلب را که آیا وضعیّت ارایه شده یک وضعیّت هدف است را تشخیص می دهد . راه حل ، مسیری از یک وضعیّت اوّلیه به یک وضعیّت هدف می باشد .

یادآوری : جستجو برای راه حل ها

برای پیدا کردن راه حل ها ، فضای جستجو را برّرسی می کنیم و از وضعیّت اوّلیه به وضعیّت نهایی (هدف) حرکت می کنیم که رشته ای مجاز از عملیّات به صورت تعریف شده توسط توابع جانشین (عملگرها) می باشد .

مثال : رومانی [۱] – در یک تعطیلی در کشور رومانی ؛ در حال حاضر در شهر آراد [۲] هستیم و هواپیما فردا آراد را به مقصد شهر بخارست [۱] ترک می کند و ما می خواهیم در شهر بخارست باشیم.

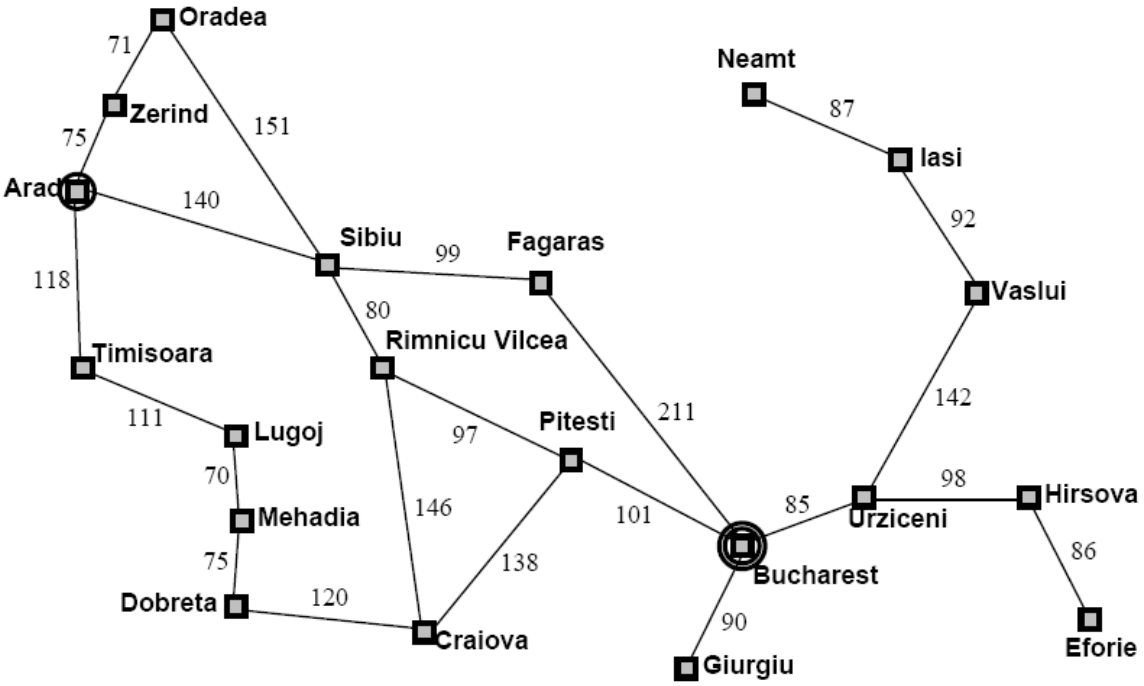
---

[۱] Romania

[۲] Arad

فرمول بندی مسأله [۲] **:** حالت ها ، شهرهای مختلف هستند و عملکردها ، حرکت بین دو شهر است .

پیداکردن راه حل : ترتیب شهرها به صورت Arad و Sibiu , Fagaras , Bucharest می باشد .

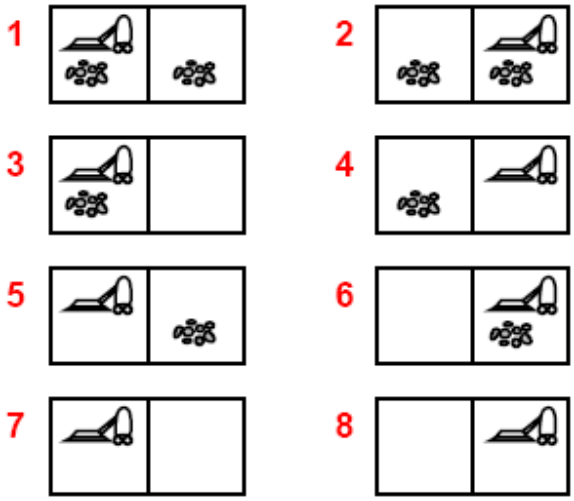


مثال : دنیای جاروبرقی – وضعیّت های عامل جاروبرقی ، تمام ترکیب های دریافت ها (درک ها) هستند و محیط کاملاً قابل مشاهده می باشد . وضعیّت های ۷ و ۸ وضعیّت های هدف می باشند چون کثیف نمی باشند . فرض می کنیم برای هر عمل ، هزینه برابر ۱ باشد .

---

[۱] Bucharest

[۲] Formulate problem



اگر در شماره ی پنج باشیم ، راه حل [Right,Suck] است . اگر در شماره های ۱ ، ۲ ، ۳ ، ۴ ، ۵ ، ۶ ، ۷ یا ۸ باشیم ، راه حل [Right,Suck,Left,Suck] می باشد . در این جا باید به این نکته توجّه کنیم که طبق قانون مرفی [۱] ، مکیدن (عمل مکش جاروبرقی) می تواند یک فرش تمیز را کثیف کند و حس کننده ی محلّی [۲] کار تشخیص کثیف بودن و فقط تعیین محل را انجام می دهد . پس راه حل این است که [اگر کثیف بود مکش کن ، راست] یا [اگر کثیف بود مکش کن ، چپ] .

فرمول بندی مسأله ی تک حالته

یک مسأله توسط چهار عنصر زیر تعریف می شود :

حالت اوّلیه : در شهر Arad هستیم .

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

---

[۱] Murphy's Law

[۲] local sensing

آزمایش هدف : می تواند ، صریح [۱] باشد مثلا ، " در بخارست " =x یا غیرصریح [۲] باشد ، مثلا ، NoDirect(x) باشد .

هزینه ی مسیر : مجموعه ای از فاصله ها ، تعدادی از عملیّات اجرا شده و ..... . c(x,a,y) یک هزینه ی مرحله [۳] است ، طبق پیش فرض باید بزرگ تر یا مساوی صفر باشد یا c(x,a,y)>=0 . یک راه حل ، ترکیبی از عملیّات است . سیر آن ، از حالت اوّلیه به یک حالت هدف است.

انتخاب یک فضای حالت (وضعیّت)

دنیای واقعی خیلی پیچیده است ، در نتیجه فضای حالت باید از راه حل مسأله جدا شود . حالت (مجرد) = مجموعه ای از حالت های واقعی است . عملکرد (مجزا) = مخلوطی پیچیده از عملکردهای واقعی ، مثلا ، "Arad → Zerind" و 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 زریند [۴] 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 مسأله 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 وضعیّت اوّلیه و به کارگیری مکرر تابع جانشین (مولد) ، تولید می نماییم . ایده ی اصلی این است که از یک وضعیّت ، توجّه کنید که چه چیزهایی می تواند انجام شود . سپس توجّه کنید که از هریک

---

[۱] explicit
[۲] implicit
[۳] step cost
[۴] zerind

از وضعیّت های آن ها چه چیزهایی می توانند انجام بگیرند . در اینجا سؤالاتی به وجود می آید که برخی از آن ها عبارتند از ، آیا پیداکردن راه حل بهینه ، ضمانت شده است ؟ تا کی ، جستجو انجام خواهد شد ؟ به چه مقدار از فضا نیاز خواهیم داشت ؟

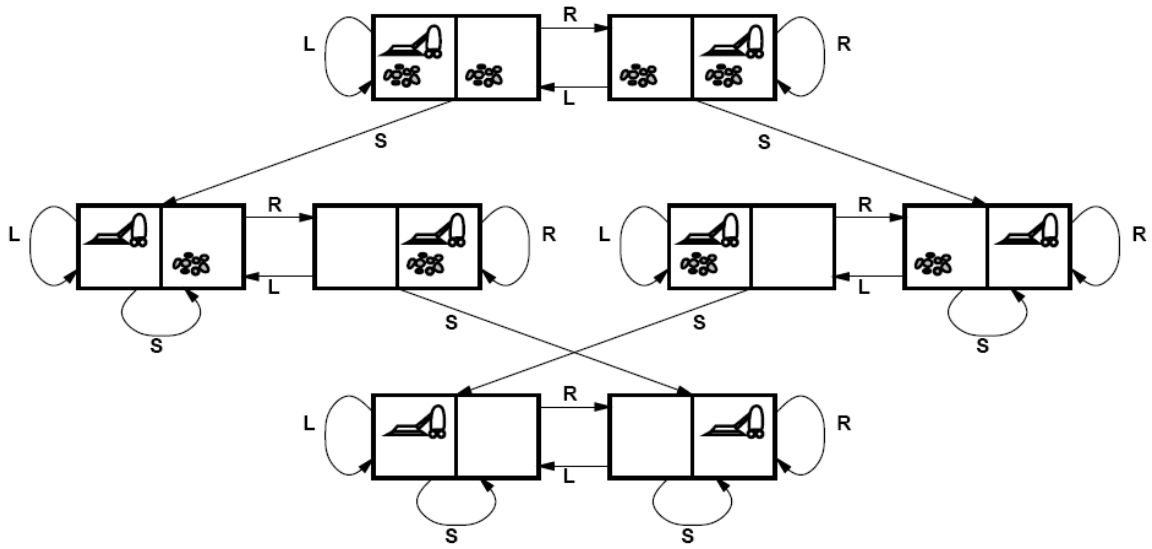
مثال : گراف فضای حالت جاروبرقی

وضعیّت ها ؟؟ کثیفی واقعی و جای روبوت (بدون توجّه به میزان کثیفی و .....)

عملکردها ؟؟ چپ (Left) ، راست (Right) ، مکش (Suck) ، بدون عملکرد (NoOp)

آزمون هدف ؟؟ بدون کثیفی بودن

هزینه ی مسیر ؟؟ در هر عملکرد یک بار (صفر برای بدون عملکرد (NoOp))



مثال : پازل هشت تایی

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

7	2	4
5		6
8	3	1

وضعیّت شروع

1	2	3
4	5	6
7	8	

وضعیّت هدف

حالت ها ؟؟ کاشی ها (بدون توجّه به وضعیّت های میانی)

عملکردها ؟؟ حرکت کردن در کاشی های خالی چپ ، راست ، بالا ، پایین

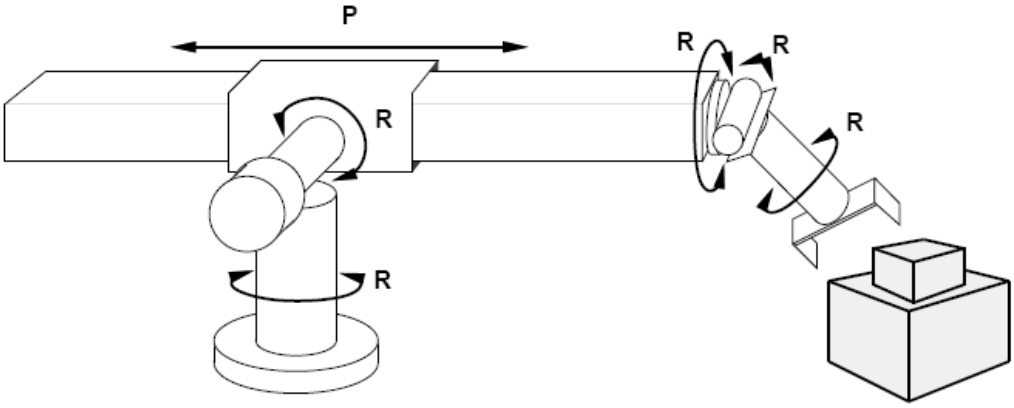
آزمایش هدف ؟؟ = وضعیّت هدف (داده شده)

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 . [۱]

مثال : سرهم بندی روبوتیک

[۱] lorrie.cranor.org/pubs/diss/node1.html



حالت ها ؟؟ مختصات [۱] مقدار دهی صحیح شده ی [۲] تکه های متصل شده ی روبوت برای سرهم بندی

عملکردها ؟؟ حرکات [۳] پیوسته ی [۴] اتصالات [۵] 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 امّا هنوز توسعه داده نشده اند را fringe می نامیم .**

الگوریتم های جستجوی درختی

---

[۱] coordinates
[۲] real-valued
[۳] motions
[۴] continuous
[۵] joints

الگوریتم :

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

در ابتدا درخت جستجو از حالت اوّلیه ی مسأله استفاده می کند .

در حلقه

اگر کاندید یا داوطلبی برای توسعه نبود عدم موفّقیّت را برگردان

یک گره [۱] برگ [۲] را با توجّه 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

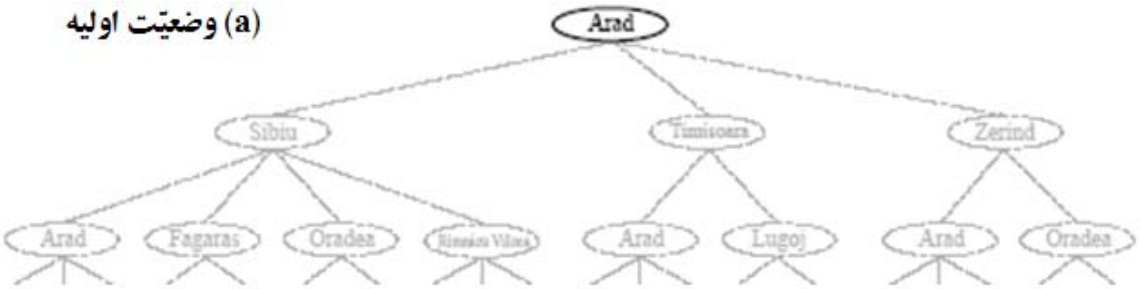
مثال جستجوی درختی

درخت جستجو ؛ در گراف جستجو ، یک وضعیّت می تواند از چند مسیر قابل دسترسی باشد و ریشه یک گره جستجو متناظر با وضعیّت اوّلیه (آراد) می باشد

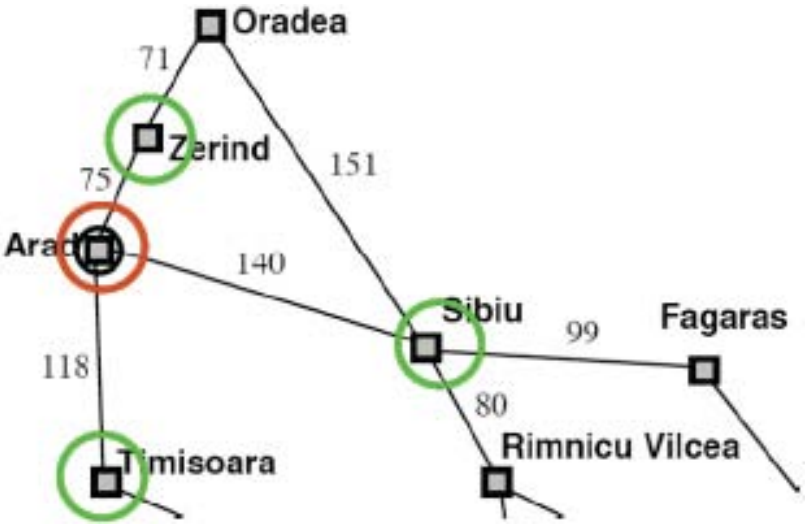
---

[۱] node

[۲] leaf



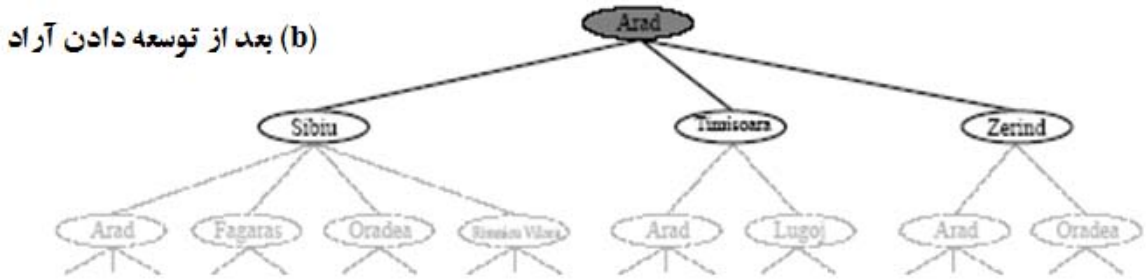
شهرهای متصل شده به وضعیّت اوّلیه عبارتند از : سیبیو [۱] ، تیمیسوارا [۲] و زریند و کاندیدهای حرکت بعدی در شکل زیر نشان داده شده اند :



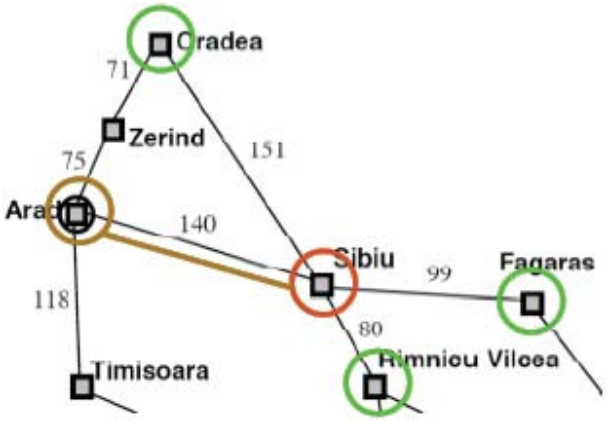
---

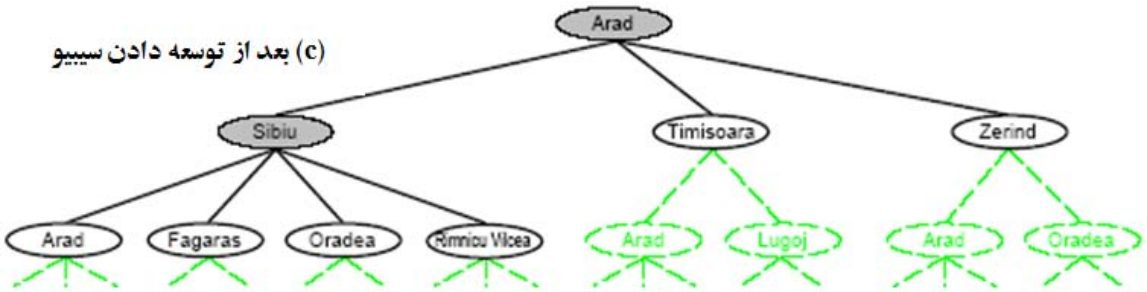
[۱] Sibiu

[۲] Timisoara



کاندیداهای حرکت بعدی عبارتند از : آراد ، فاگاراس [۱] ، اُرادیا [۲] و ریمنیکو ویلک [۳]





---

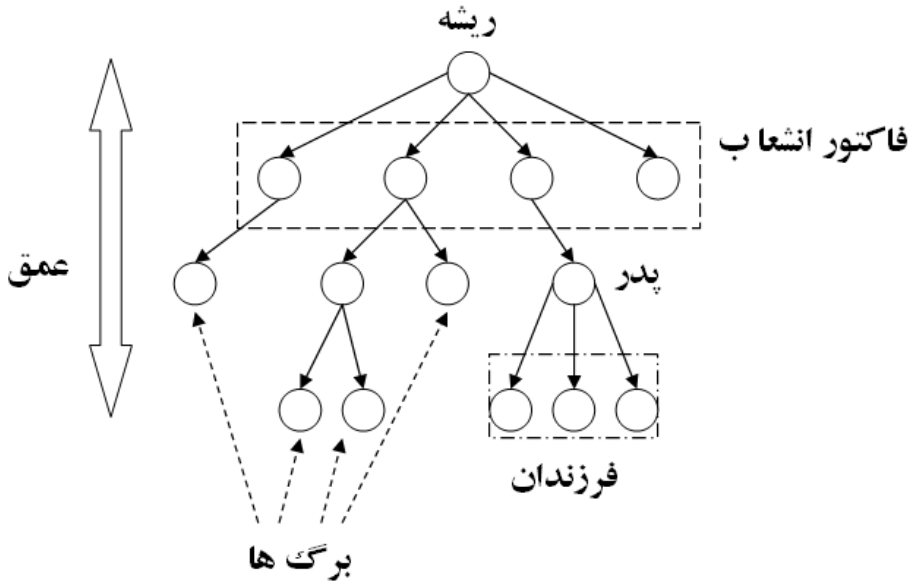
[۱] Fagaras

[۲] Oradea

[۳] Rimnicu Vilc

درخت ها

یک درخت ، یک گراف [۱] بدون حلقه ی [۲] مستقیم [۳] می باشد . گراف مستقیم ، مجموعه ای از گره های متصل شده توسط لبه ها (یال ها) [۴] می باشد . مسیر ، رشته ای از لبه ها (که امکان دارد تکرار هم شده باشند) است . در درخت ، حداکثر یک مسیر متصل کننده ی هر جفت از گره ها وجود دارد (بدون حلقه است) . درخت ها به دلیل این که رفتار الگوریتمی خوبی دارند به صورت گسترده ای در علم کامپیوتر استفاده می شوند و دارای قدرت زیاد برای تعریف الگوریتم های بازگشتی می باشند . در زیر تصویری از یک درخت را مشاهده می نمایید :



---

[۱] graph
[۲] acyclic
[۳] direct
[۴] edges

توجّه کنید که گره ها حداکثر دارای یک پدر هستند و ریشه ، گره ای متمایز است که دارای پدری نمی باشد در ضمن ، عمق هم که برای خاتمه ی الگوریتم ، مهم است می تواند نامحدود باشد .

الگوریتم جستجوی درختی

الگوریتم

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

درخت جستجو را با استفاده از حالت اوّلیه ی مسأله مقداردهی اوّلیه می نماید

در حلقه کارهای زیر را انجام بده

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

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

در صورتی که گره شامل یک وضعیّت 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 براساس ترتیبی که براساس آن گره ها توسعه داده می شوند مشخص می شود.

وضعیّت ها و گره های درخت

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 [۱] ، فرزندان [۲] ، عمق [۳] و ارزیابی مسیر g(x) است . **حالت ها یا وضعیّت ها** ، پدر (والد) ، فرزند ، عمق یا ارزیابی مسیر ندارند . از یک تابع مثل تابع EXPAND برای ساختن گره های جدید می توانیم استفاده می کنیم و از SuccessorFn می توان برای ساختن حالت های مورد نظر در مسأله استفاده کرد .

الگوریتم جستجوی درخت عمومی

الگوریتم

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

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

در حلقه کارهای زیر را انجام بده

در صورتی که fringe خالی است ، عدم موفّقیّت را برگردان

node ← Remove-Front(fringe)

در صورتی که Goal-Test(problem,State(node)) ، گره را برگردان

fringe ← InsertAll(Expand(node,problem),fringe)

---

[۱] parent
[۲] children
[۳] depth

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 موفّقیّت را برمی گرداند

fringe ← make-queue(make-node(initial-state[problem]))

در حلقه کارهای زیر را انجام بده

اگر fringe خالی می باشد عدم موفّقیّت را برگردان

node ← Remove-Front(fringe)

در صورتی که Goal-Test[problem] به کار گرفته شده برای state(node) موفق شد node را برگردان

fringe ← Queuing-Fn(fringe,Expand(node,Operators[problem]))

Queuing-Fn(queue,elements) یک تابع صف بندی است که مجموعه ای از عناصر را به صف وارد می نماید و ترتیب 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 :

تمامیّت : 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 این موارد ارزیابی می شوند : بیشینه ی فاکتور انشعاب درخت جستجو ، عمق کم هزینه ترین راه حل ، بیشینه ی عمق درخت جستجو که شاید بی نهایت باشد .

جستجوی نا آگاهانه

ساده ترین الگوریتم های جستجو ، آن هایی هستند که هیچ اطّلاعات اضافی که در شرح مسأله 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 هم نامیده می شوند . در صورتی که ما دارای اطّلاعات اضافی در مورد چگونگی یک وضعیّت غیر هدف باشیم ، در این صورت ما می توانیم جستجوی *مکاشفه ای* را انجام دهیم .

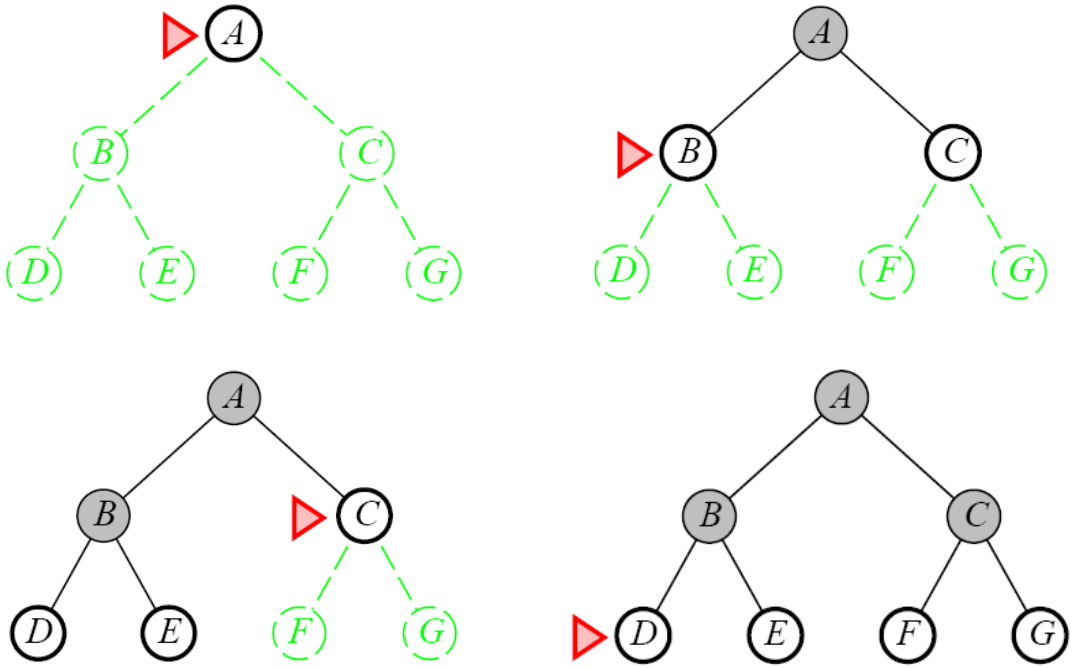
روش های جستجوی ناآگاهانه– چنان که گفته شد ، فقط از اطّلاعات قابل دسترسی از تعریف مسأله استفاده می نمایند . شامل جستجوی اوّل سطح [۱] | جستجوی با هزینه ی یکسان [۲] | جستجوی اوّل عمق [۳] | جستجوی با عمق محدود شده [۴] | جستجوی عمیق شونده ی تکراری [۵] می باشند .

جستجوی اوّل 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 ... . در این روش ، همه ی گره های در عمق n ، قبل از یک گره در عمق n+1 ملاقات می شوند . ما می توانیم جستجوی اوّل سطح را با استفاده از یک صف ، پیاده سازی نماییم . به عبارت دیگر ، جستجوی اوّل سطح ، کم عمق ترین گره توسعه داده نشده را توسعه می دهد. در **پیاده سازی** ، ریشه [۶] یک صف FIFO [۷] می باشد و جانشینان جدید به انتها می روند. مثال :

---

[۱] Breadth-first
[۲] Uniform-cost
[۳] Depth-first
[۴] Depth-limited
[۵] Iterative deeping
[۶] fringe
[۷] 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

• ...

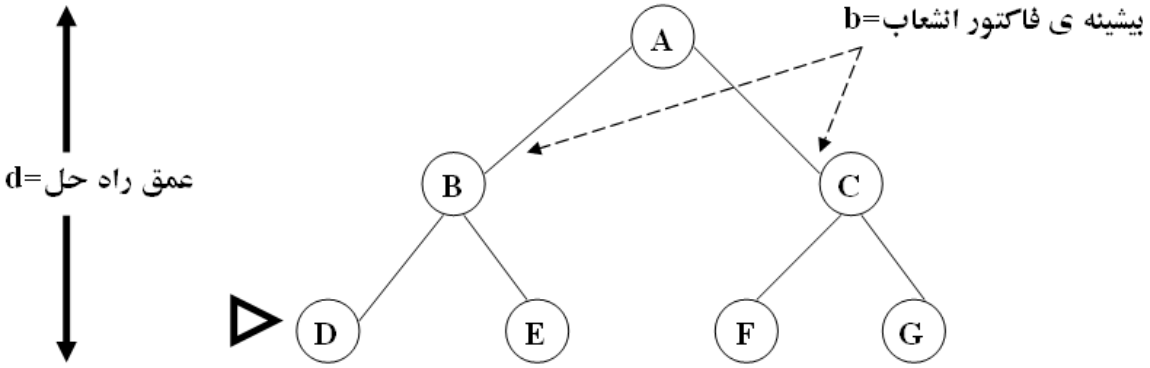
برخی نکات ریز

چگونه ما از ملاقات مجدد آراد جلوگیری نماییم ؟ با استفاده از لیست closed ، یک لیست از وضعیّت های توسعه داده شده را نگهداری نماییم . توجّه کنید که 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 وضعیّت هایی که تولید شده اند امّا ، 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 .

خصوصیّات جستجوی اوّل سطح



کامل ؟؟ آیا جستجوی اوّل سطح ، راه حل را پیدا می کند ؟ بله (در صورتی که b محدود باشد)

توضیح : تصوّر 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 اوّل سطح ، به زمان اجرای $O(b^{d+1})$ نیاز دارد . که در آن b ، فاکتور انشعاب ، میانگین تعداد فرزندان می باشد . جستجوی اوّل 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 توجّه نمایید)

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

توضیح : جستجوی اوّل 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 اوّل سطح ، بهترین راه حل را پیدا می کند ؟ بله

توضیح : جستجوی اوّل 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 ، لزوماً 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 ، مسأله 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 [۱] 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 اوّل سطح ، مسأله ای بزرگ تر از زمان مورد نیاز می باشد .

---

[۱] 100MB/sec

جستجوی با هزینه ی یکسان

به یاد بیاورید که جستجوی اوّل سطح در زمانی که هزینه ی مراحل ، غیر یکسان باشد ، غیربهینه می باشد . ما می توانیم این اشکال را رفع نماییم برای این کار اوّل کوتاه ترین مسیرها را توسعه می دهیم و یک هزینه ی مسیر را به گره های توسعه داده شده ، اضافه می کنیم . از یک صف اولویّت هم برای منظم کردن آن ها به صورت افزایشی بر اساس هزینه ی مسیر ، استفاده می نماییم . این روش ، کوتاه ترین مسیر را پیدا خواهد کرد . در صورتی که هزینه ها ، یکسان باشند ، این روش با جستجوی اوّل سطح ، برابر می باشد . پس ، جستجوی با هزینه ی یکسان ، کم هزینه ترین گره توسعه داده نشده را توسعه می دهد . در پیاده سازی ، ریشه ، صفی با ارزیابی مسیر است که کم ترین در اوّل صف قرار دارد .

خصوصیات جستجوی با هزینه ی یکسان

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

زمان ؟؟ تعداد گره ها در صورتی که g کوچک تر یا مساوی هزینه ی 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.

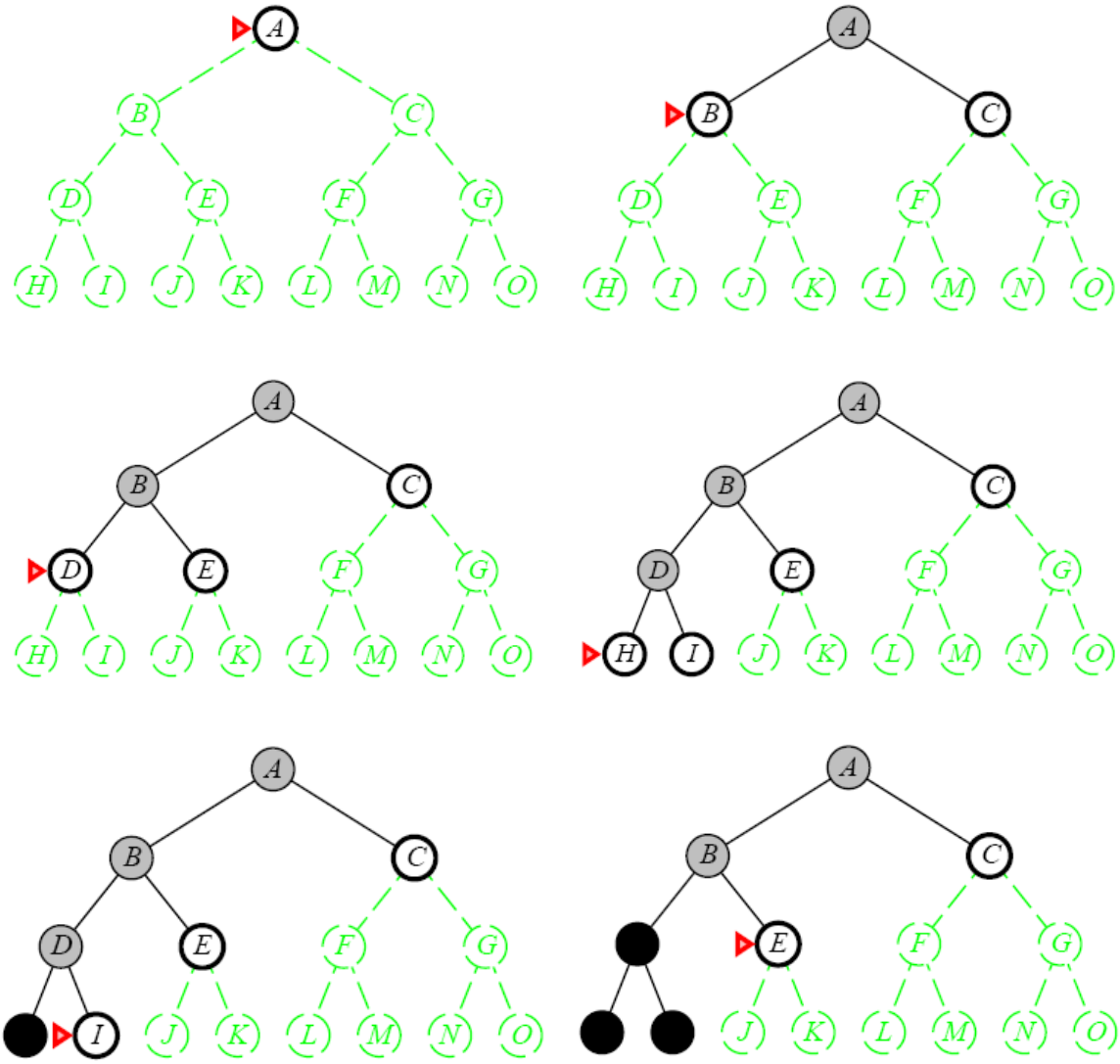
برای پیاده سازی ، ریشه ، صفی LIFO است و جانشین ها را [به ترتیب] در جلو قرار می دهد.

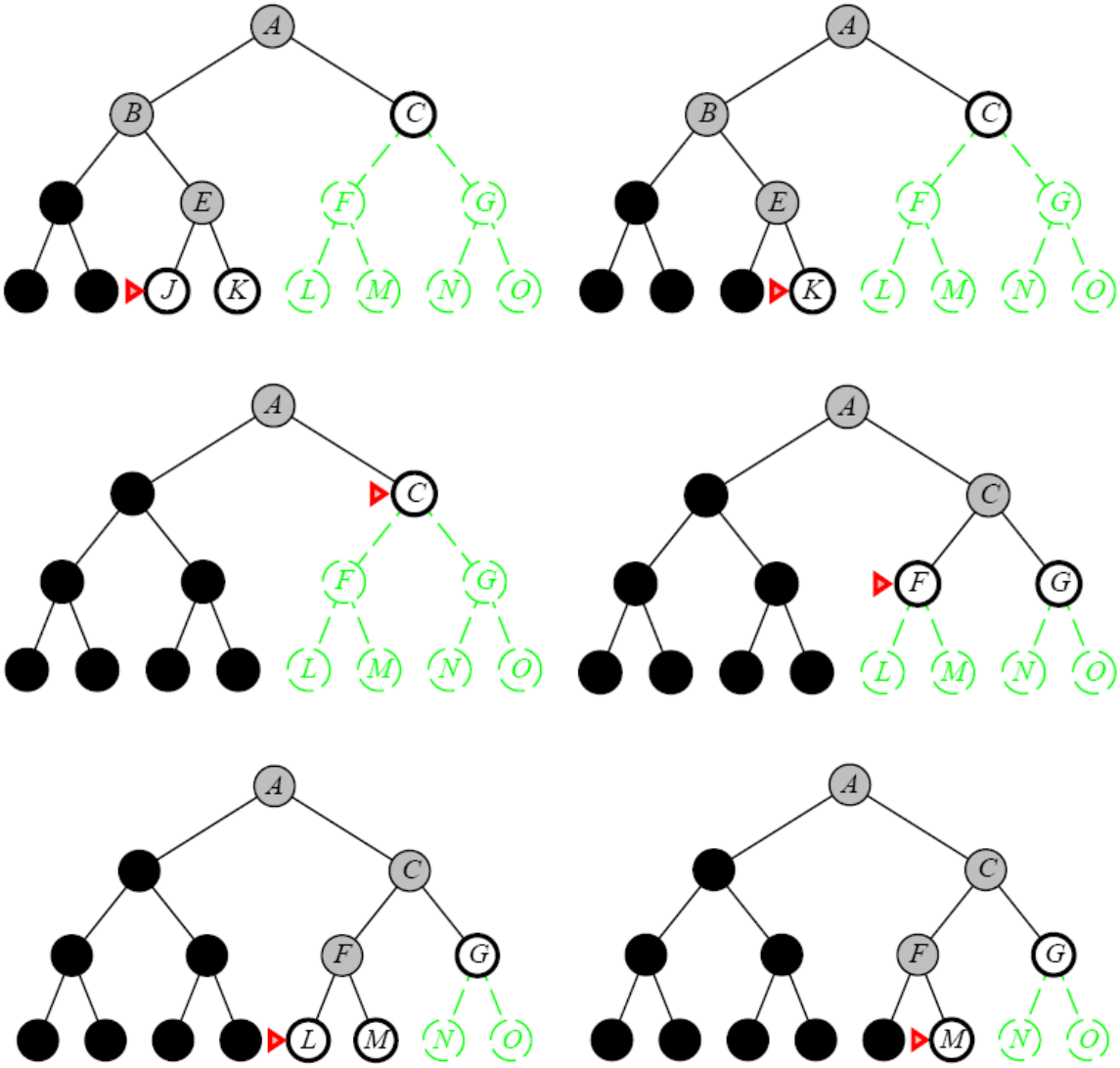
جستجوی اوّل عمق

دارای روش متفاوتی نسبت به جستجوی اوّل سطح می باشد و همیشه ، عمیق ترین گره را توسعه می دهد . ابتدا 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 اوّل 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 اوّل عمق ، عمیق ترین گره ی توسعه داده نشده را توسعه می دهد . **پیاده سازی دیگر :** ریشه ، صفی LIFO است و جانشین ها را [به ترتیب] در جلو قرار می دهد.


A
B C
D E F G
H I J K L M N O



مثال دیگر : آراد به بخارست (به صورت پشته)

- آراد را از پشته بردارید (pop)
- سیبیو ، تیمیسوارا و زریند 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
- ...

خصوصیّات جستجوی اوّل 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 اوّل سطح می باشد. توجّه 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 و این حسن بزرگ جستجوی اوّل 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 اوّل ریمنیکو ویلسیا را از پشته برمی داشتیم چه اتّفاقی می افتاد ؟)

اوّل ، یک سؤال

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

جستجوی اوّل سطح

اگر جستجوی اوّل سطح را با استفاده از یک صف پیاده سازی کنیم ، تمام گره های در سطح n ، قبل از هر گره در سطح n+1 ارزیابی می شوند و این دارای این حسن است که ، کامل می باشد و در زمانی که هزینه ی مرحله به صورت یکسان می باشد ، راه حل بهینه را پیدا خواهد نمود و دارای این ضعف است که به فضایی به صورت نمایی نیازمندیم .

جستجوی اوّل عمق

اگر جستجوی اوّل سطح را با استفاده از پشته پیاده سازی نماییم ؛ ابتدا یک گره توسعه داده می شود ، سپس فرزندانش توسعه داده می شوند و ... و در هر لحظه ، فقط یک شاخه از درخت جستجو در حافظه نگهداری می شود . حسن این روش این است که به فضایی به صورت خطی نیازمندیم و عیب آن این است که نا کامل و غیربهینه می باشد .

جلوگیری از جستجوی نامحدود

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

جستجوی با عمق محدود شده

با محدود نکردن عمق l با جستجوی اوّل عمق برابر است و گره های در عمق l هیچ جانشینی ندارند. **در واقع** ، جستجوی با عمق محدود شده ، همان جستجوی اوّل 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) و عملیّات جستجو در این عمق ، متوقف می شود . در این جا مشکل دیگری به وجود می آید و آن این است که اگر راه حل در عمقی بیش تر از l باشد ، چطور ؟ چگونه ما یک l خوب را به دست آوریم ؟ . در مسأله 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 روش جستجوی با عمق محدود شده 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 :

تابع function Depth-Limited-Search(problem,limit) مقادیر soln ، fail و یا 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 شونده ی تکراری

بر مبنای ایده ی جستجوی با عمق محدود شده ، توسعه می بابد . 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 این روش این است که 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 و ... در نتیجه در سطح d ، تعداد b^d بار تولید می شوند . در این روش ، مجموع زمان اجرا برابر ($O(b^d)$ می باشد . که اندکی کم تر از تعداد گره های تولید شده در جستجوی اوّل سطح می باشد و هنوز به حافظه ای به صورت خطی نیازداریم . **پس** ، عمق محدودشده را با افزایش محدودیّت ها به کار می گیرد . الگوریتم این روش به صورت زیر است :

تابع function Iterative-Deepening-Search(problem) یک راه حل را برمی گرداند .

ورودی ها problem که یک مسأله 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

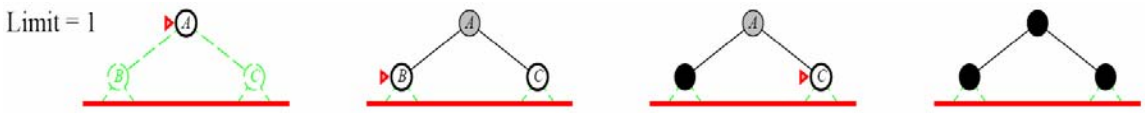
شبه کدی به صورت دیگر برای جستجوی عمیق شونده ی تکراری :

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

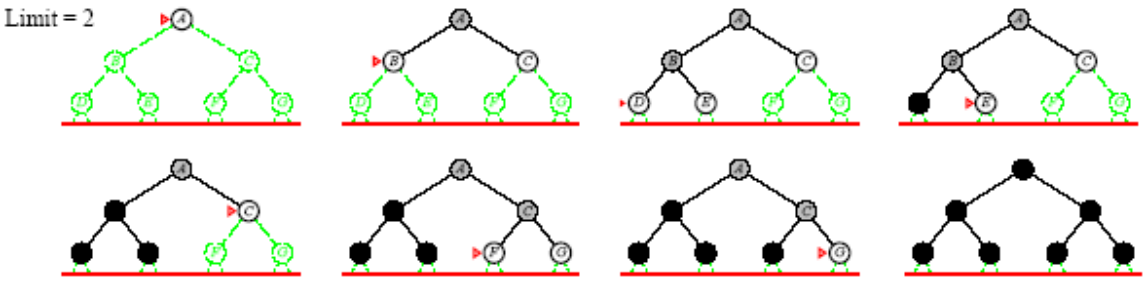
مثال :

جستجوی عمیق شونده ی تکراری با l=0

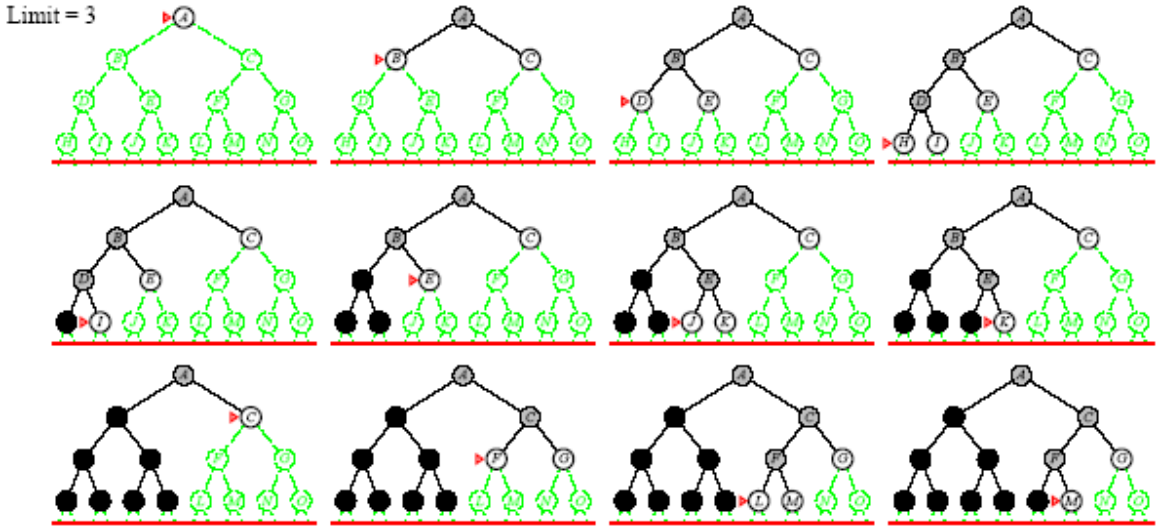
جستجوی عمیق شونده ی تکراری با l=1



جستجوی عمیق شونده ی تکراری با l=2



جستجوی عمیق شونده ی تکراری با l=3



جستجوی عمیق شونده ی تکراری ، جستجوی اوّل سطح و اوّل عمق را با هم ترکیب می کند و لایه ها را شبیه جستجوی اوّل سطح به وجود می آورد ، امّا در هر اجرا یک اوّل عمق را انجام می دهد که باعث صرفه جویی در فضا می شود . در این روش ، وضعیّت ها چند بار توسعه داده می شوند .

در واقع ، جستجوی عمیق شونده ی تکراری ، شبیه جستجوی اوّل سطح می باشد که در آن همه ی گره های در عمق n قبل از هر گره در عمق n+1 برّرسی می شوند . نظیر جستجوی اوّل سطح ، ما می توانیم بهینگی را در جهان های با هزینه ی غیریکسان با توسعه دادن با توجّه به هزینه ی مسیر و ترجیحا با استفاده از

عمق ، به دست آوریم . این روش ، *جستجوی دراز کننده ی طول* [۱] نـام دارد . همـه ی مسـیرها را در هزینـه 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 [۲]

در زمانی که جستجوی اوّل عمق و همتایانش به وضعیّت عدم موفّقیّت می رسند ، چه اتّفاقی می افتـد ؟ به بالای گره ی پدر (والد) می روند و برادر بعدی را امتحان می کنند . و فرض را بر این قرار می دهند کـه جدیدترین عملکرد انتخاب شده ، عملکردی است که باعـث عـدم موفّقیّـت شـده اسـت . ایـن روش ، *جریـان معکوس به ترتیب وقوع* [۳] نام دارد و جدیدترین کاری را کـه انجـام داده ایـد ، بـی اثـر نماییـد . مسـأله ای کـه وجود دارد این است که عدم موفّقیّت ، شاید یک نتیجه از یک تصمیم گیری قبلی باشـد ؛ مثـال ، ۴ – وزیـر . محدودیّت ها می توانند به شما برای محدود کردن اندازه ی فضای جستجو کمـک نماینـد . *جریـان معکـوس هوشمند* [۴] برای تحلیل دلیلی برای عدم موفّقیّت و غیرفعّال کردن جستجو برای آن نقطه تلاش می نماید و مـی تواند جدیدترین متغیّر ناسازگار را غیرفعال نماید (جریان معکوس) ، همچنـین مـی توانـد برّرسـی مسـتقیم [۵] را انجام دهد ؛ آیا یک انتساب ممکن مقدارها برای متغیّرهایی در این نقطه وجود دارد ؟

خصوصیات جستجوی عمیق شونده ی تکراری

کامل بودن ؟؟ بله

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

---

[۱] iterative lengthening search

[۲] backtracking

[۳] chronological backtracking

[۴] intelligent backtracking

[۵] forward checking

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 :

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

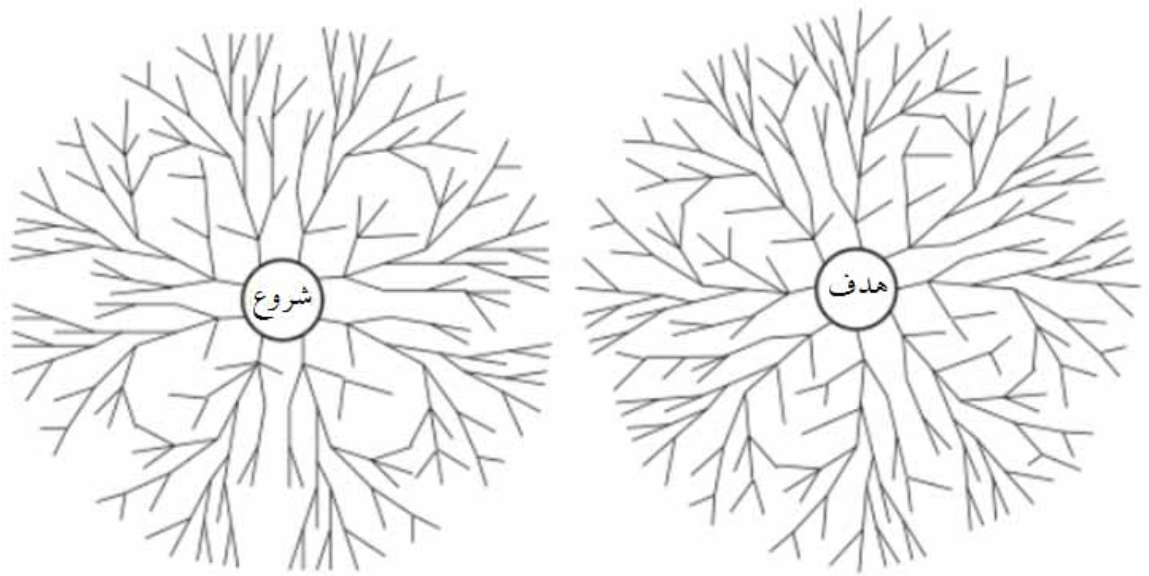
$$N(BFS) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100$$

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

جستجوی دوطرفه [1]

دو جستجوی شبیه را اجرا می نماید ، یکی در جهت وضعیّت اوّلیه و دیگری در خلاف جهت و از طرف وضعیّت هدف و در زمانی که دو جستجو به هم می رسند جستجو خاتمه می یابد .

[1] bidirectional search



خصوصیّات جستجوی دو طرفه

- در صورتی که هر دو جستجو اوّل سطح باشند کامل است .
- زمان : $O(b^{d/2})$
- فضا : $O(b^{d/2})$
- در صورتی که هر دو روش اوّل سطح باشند بهینه می باشد

اشکالات [1] این روش : تولید قبلی ها (معکوس کردن عملگرها) و هماهنگی میان دو جستجو و این که جستجو باید گره ها را در حافظه نگهداری نماید (برای برّرسی این که آیا گره ها قبلا ملاقات شده اند یا نه)

---

[1] drawbacks

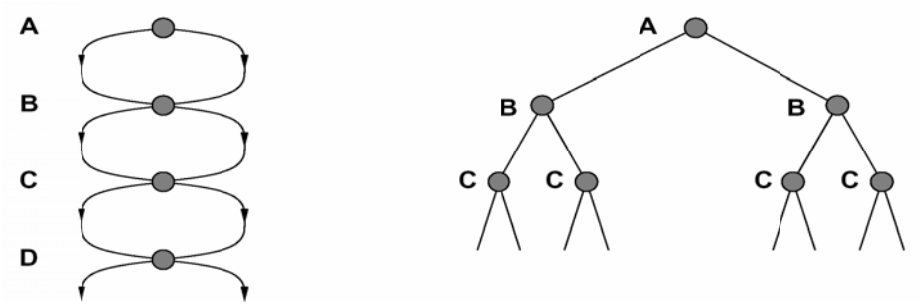
خلاصه ی الگوریتم ها

مقیاس [1]	اوّل سطح	با هزینه ی یکسان	اوّل 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	$b^{\lceil C^*/\in \rceil}$	b^m	b^l	b^d
فضا ؟؟	b^d+1	$b^{\lceil C^*/\in \rceil}$	bm	bl	bd
بهینگی ؟؟	بله *	بله	نه	نه	بله *

در جدول بالا ، b : فاکتور انشعاب ، d : عمق کم عمق ترین راه حل ، m : بیشینه ی عمق درخت جستجو و l : محدودیّت عمق می باشند .

حالت (وضعیّت) های تکرار شده – *عدم موفّقیّت در پیداکردن حالت های تکرار شده می تواند یک مسأله ی خطی را تبدیل به یک مسأله ی نمایی کند .*

[1] Criterion



راه حل این مشکل ، نگهداری تاریخچه است ؛ گره های ملاقات شده (توسعه داده شده) در closed list نگهداری می شوند و گره جاری اگر به تازگی در closed list بوده ، حذف می شود . مشکل این روش این است که بهینگی ممکن است از بین برود چون همه ی گره ها در حافظه نگهداری می شوند .

الگوریتم جستجوی گراف

تابع function Graoh-Search(problem,fringe) یک راه حل یا عدم موفّقیّت 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 موفّقیّت را بر گردان

node ← Remove-Front(fribge)

در صورتی که Goal-Test(problem,State[node]) موفّقیّت 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 مسأله ، معمولاً 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 مسأله ی جستجو با استفاده از وضعیّت اوّلیه ، آزمون هدف ، عملیّاتی که باید انجام شوند ، تابع جانشین (مولد) و هزینه ی مسیر ، منجر به جستجو در یک فضای حالت با استفاده از یک درخت جستجو می شود . الگوریتم ها عبارتند از : جستجوی اوّل سطح ، جستجوی اوّل عمق ، جستجوی با هزینه ی یکسان ، جستجوی با عمق محدود شده و جستجوی عمیق کننده ی تکراری

[1] iterative deeping search

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل چهارم

جستجوی آگاهانه

(مکاشفه ای)

خلاصه ی ریوس مطالب

- **جستجوی اوّل - بهترین** [۱]
- **جستجوی A^***
- **ابتکارات** [۲] **(اکتشافات)**

آگاهانه کردن جستجو

الگوریتم های قبلی ، می توانستند راه حل ها را پیدا کنند ، امّا ناکارامد بودند ؛ گره ها به صورت نمایی به وجود می آمدند . با استفاده از آگاهی هایی که در مورد ساختار مسأله داریم می توانیم کارایی را بهتر کنیم . البتّه باید توجّه کنیم که این دانش را باید از جایی دیگر بگیریم و این دانش باید درست باشد .

چرا از دانش در یک مسأله استفاده می نماییم ؟

---

[۱] Best-first

[۲] Heuristics

روش های ناآگاهانه فقط از اطّلاعات قابل دسترس از تعریف مسأله استفاده می نمایند . *بدترین حالت پیچیدگی زمانی وقتی است که پیچیدگی زمانی به صورت نمایی باشد* ؛ در اکثر موارد روش های ناآگاهانه ناکارآمد هستند . ممکن در صورتی که چیزهایی در مورد مسأله بدانیم بهتر عمل کنیم . ایده ی اصلی این است که برای به وجود آوردن گره ها از یک تابع ارزیابی [۱] استفاده نماییم .

تابع ارزیابی

یک تابع ارزیابی f(n) ، شایستگی یک وضعیّت هدف را تخمین می زند ؛ یک وضعیّت با هر گره درخت جستجو پیوند برقرار می کند و 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 ؛ توجّه کنید که ما به دنبال ارزان ترین راه حل هستیم .

در پیاده سازی این روش ، از fringe که یک صف مرتب شده توسط تابع ارزیابی است استفاه می نماییم . جستجوی با هزینه ی یکسان را به یاد بیاورید در این روش گره ها براساس مجموع هزینه ی مسیر ، توسعه داده می شوند و از یک صف اولویّت ، برای پیاده سازی جستجوی با هزینه ی یکسان استفاده می شود ؛ هزینه ی مسیر ، مثالی از یک تابع ارزیابی می باشد .

روش **جستجوی اوّل – بهترین** [۲]

اوّل ، بهترین گره را برای رسیدن به هدف توسعه دهید . 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) در ادامه گره ای را که به نظر می رسد بهترین باشد را توسعه دهید .

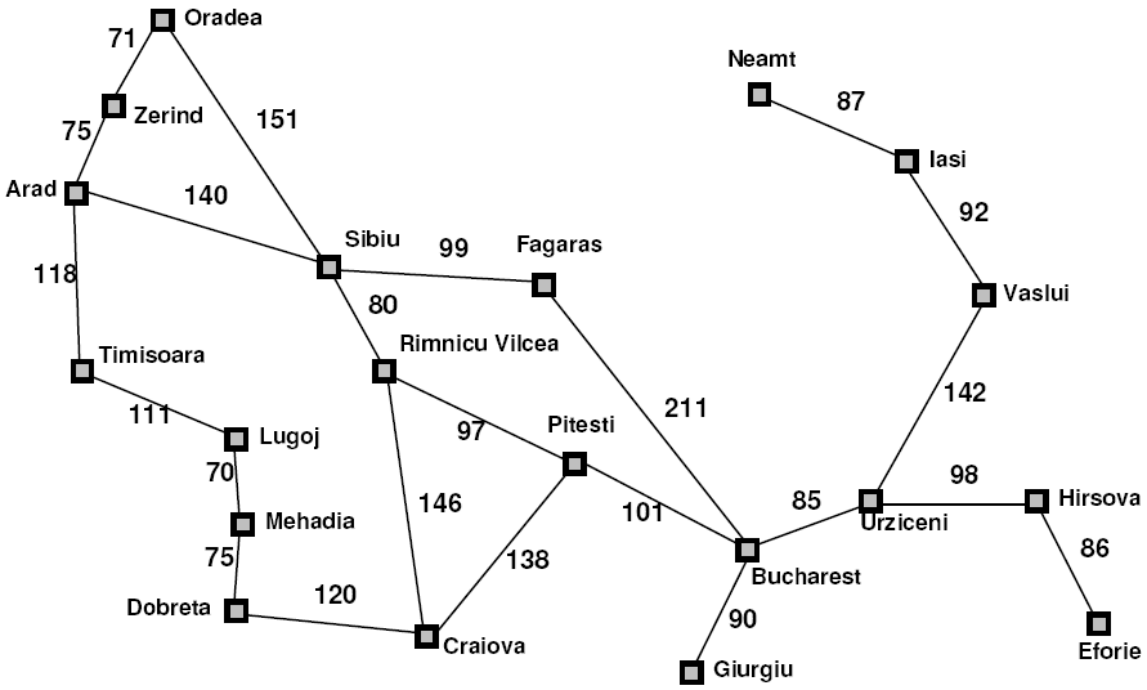
---

[۱] evaluation function

[۲] best-first search

پس ایده ی جستجوی اوّل – بهترین استفاده از یک تابع ارزیابی می باشد .

مسأله ی کشور رومانی با ارزیابی مراحل به صورت کیلومتر



آراد	۳۶۶	فاگاراس	۱۷۸	مهادیا [۱]	۲۴۱	سیبیو	۲۵۳

[۱] Mehadia

۳۲۹	تیمیسوارا [۱]	۲۳۴	نمت [۲]	۷۷	گیورگیو [۳]	۰	بخارست
۸۰	ارزیسنی [٤]	۳۸۰	ارادیا [٥]	۱۵۱	هیرسوا [٦]	۱۶۰	کرایوا [۷]
۱۹۹	واسلوی [۸]	۹۸	پیتستی [۹]	۲۲۶	یاسی	۲۴۲	دوبرتا [۱۰]
۳۷۴	زریند	۱۹۳	ریمنیکو ویلسیا	۲۴۴	لوگوج [۱۱]	۱۶۱	افری [۱۲]

توجّه کنید که اکتشاف (که به صورت جدولی در بالا داده شده است) جزیی از تعریف اوّلیّه ی مسأله نمی باشد و در این مورد به صورت یک جدول خارجی داده شده است .

---

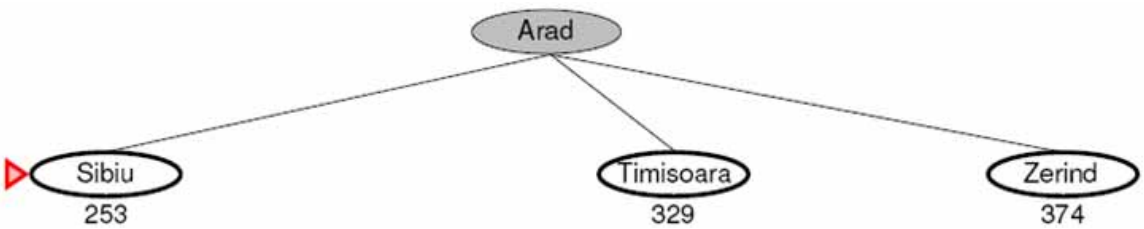
[۱] Timisoara
[۲] Neamt
[۳] Giurgiu
[۴] Urziceni
[۵] Oradea
[۶] Hirsova
[۷] Craiova
[۸] Vaslui
[۹] Pitesti
[۱۰] Dobreta
[۱۱] Lugoj
[۱۲] Eforie

مسافت خطی مستقیم به شهر بخارست

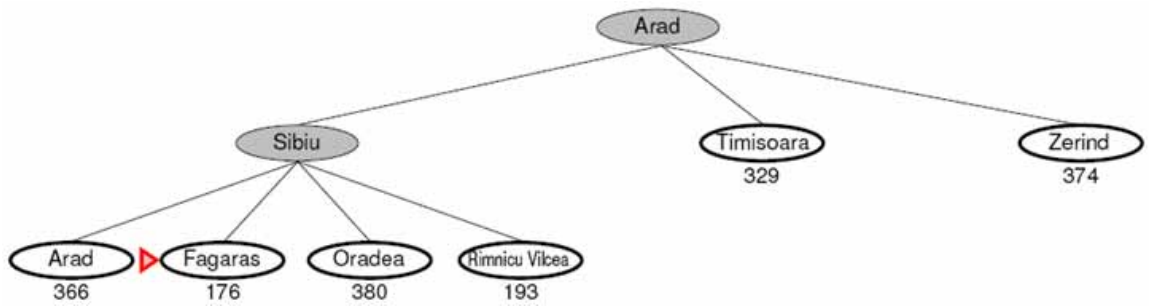
جستجوی حریصانه [۱]

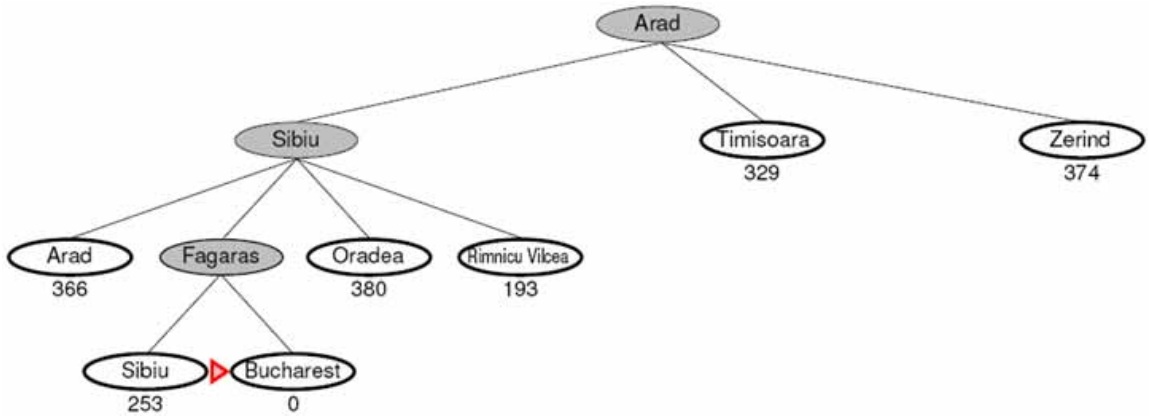
در این روش ، در ارزیابی ، فقط از مکاشفه استفاده می شود f(n)=h(n) و الگوریتم ، گرهی را که به نظر می رسد به هدف نزدیک تر است را توسعه می دهد . و تابع ارزیابی h(n) (ابتکاری) برابر است با تخمین ارزش از n تا نزدیک ترین هدف و $h_{SLD}(n)$ برابر است با فاصله ی خطی مستقیم از n تا شهر بخارست .

مثالی از جستجوی Greedy



[۱] Greedy search





خصوصیّات جستجوی حریصانه

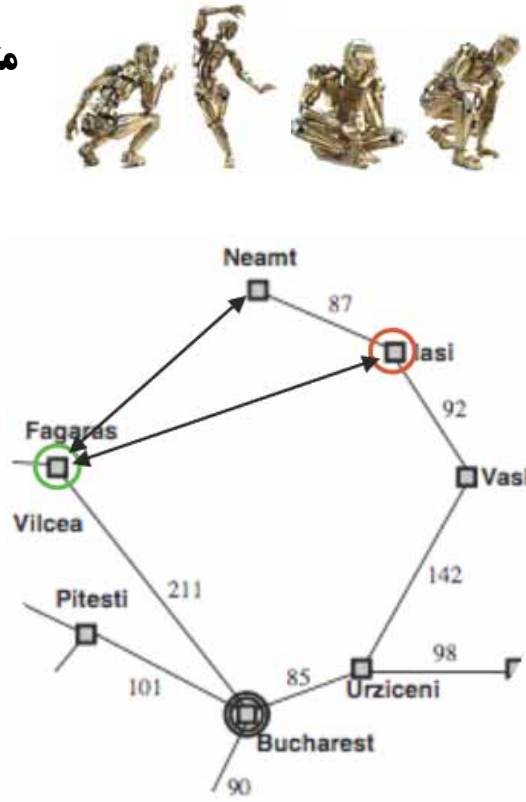
کامل بودن ؟؟ نه ممکن است در حلقه ها گیر کند ، مثلا ، اگر مقصد ما شهر Oradea باشد ، داریم :

lasi → Neamt → lasi → Neamt →

به عنوان مثالی دیگر ، اگر بخواهیم از یاسی [۱] به فاگاراس برویم ؛ میان یاسی و نمت [۱] حلقه به وجود می آید که برای رفع این مشکل باید از وضعیّت های ملاقات شده خودداری نماییم .

---

[۱] Iasi



این روش در فضاهای جستجوی محدود با تست وضعیّت تکرار شده ، **کامل** است .

زمان ؟؟ : $O(b^m)$، امّا با یک ابتکار صحیح می توان بهبودی قابل توجّهی 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 جستجوی اوّل – عمق می باشد .

جستجوی A^*

مسأله ای که در مورد روش های جستجوی اوّل – بهترین و حریصانه وجود دارد ، این است که هزینه ای که مسیر تا کنون داشته است مورد توجّه قرار نگرفته است . روش جستجوی A^* از به وجود آوردن (توسعه دادن) مسیرهایی که پرهزینه هستند اجتناب می کند .

---

[1] Neamt

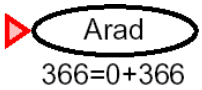
مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

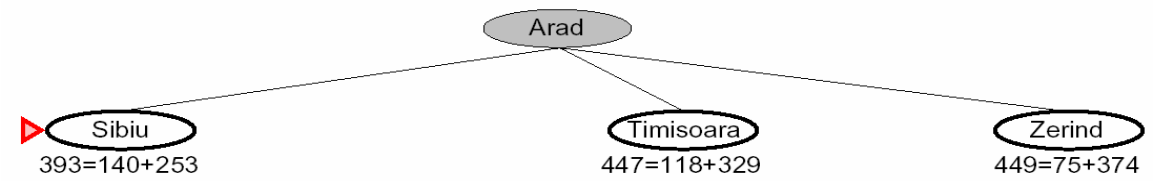
در این روش ، تابع ارزیابی $f(n)=g(n)+h(n)$ می باشد . که $g(n)$ = هزینه ای که تا کنون برای رسیدن به n صرف شده است . $h(n)$ = هزینه ی تخمین زده شده برای n و $f(n)$ = کل هزینه ی تخمین زده شده ی مسیر از n تا هدف .

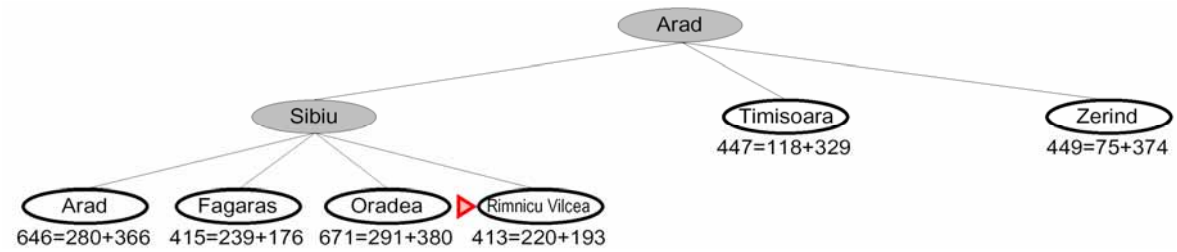
روش جستجوی A^* از یک روش ابتکاری مجاز برای جستجو استفاده می کند . در واقع ، جستجوی A^* ترکیب جستجوی با هزینه ی یکسان و جستجوی حریصانه می باشد .

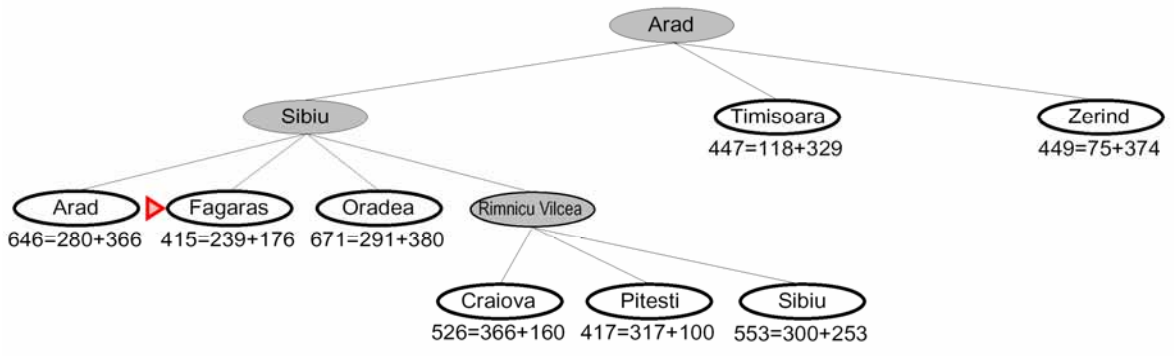
**** قضیه : جستجوی A^*، جستجویی بهینه یا مطلوب است .***

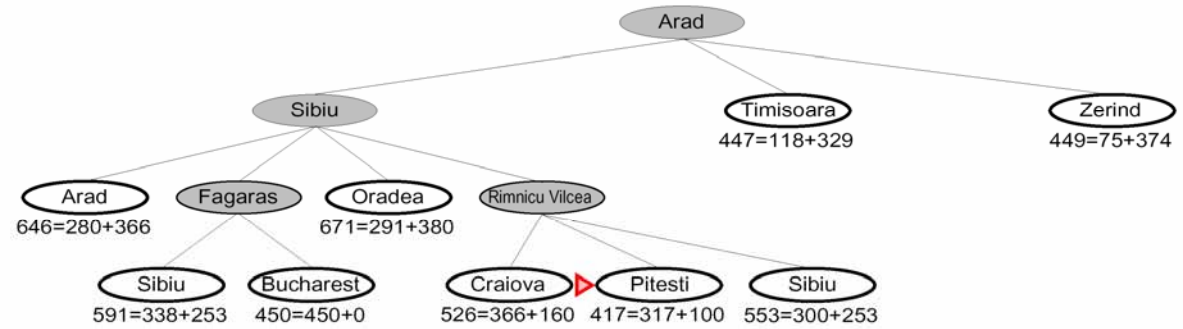
مثالی برای جستجوی A^*

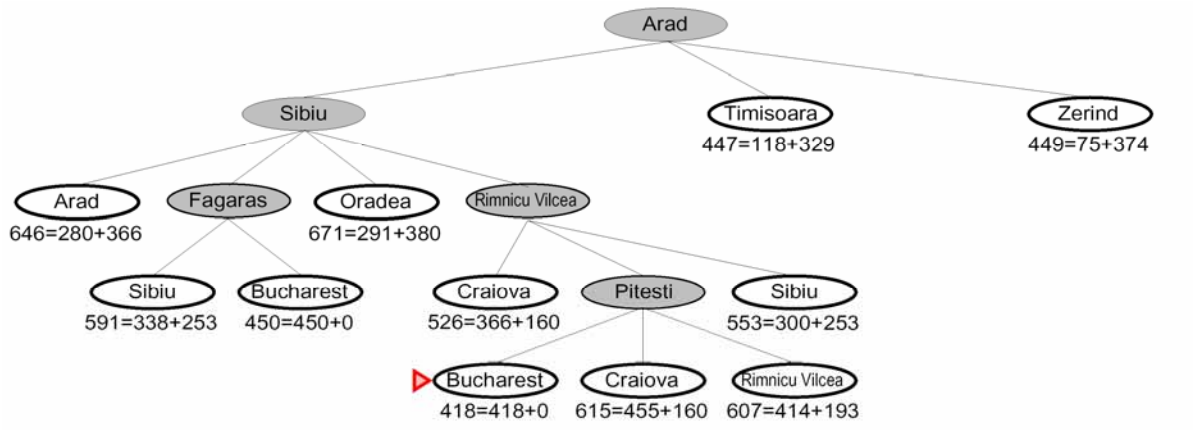







Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253



بهینگی جستجوی A^*

برای تضمین بهینگی مکاشفه نباید هزینه ی رسیدن به هدف را زیاد برآورد نماییم در مکاشفه ی قابل قبول داریم ؛ $h(n) \le h^*(n)$ که $h^*(n)$ هزینه ی واقعی رسیدن به هدف می باشد . در مسیریابی ، فاصله ی خطی مستقیم قابل قبول می باشد . این روش قابلیت قبول بهینگی را فقط برای الگوریتم جستجوی کلّی تضمین می کند .

الگوریتم

تابع General-Search(problem,Queuing-Fn) یک راه حل یا عدم موفّقیّت 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 موفّقیّت را برگردان

node ← Remove-Front(fringe)

در صورتی که Goal-Test[problem] به کار گرفته شده برای وضعیّت (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 موفّقیّت را برگردان

node ← Remove-Front(fringe)

در صورتی که Goal-Test[problem] به کار رفته برای State(node) موفّقیّت آمیز بود node را برگردان

در صورتی که $node \notin closed$ آن گاه

node را به closed اضافه نما

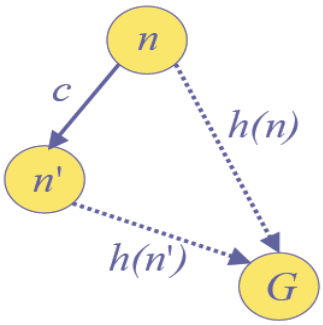
fringe ← Queuing-Fn(fringe,Expand(node,Operators[problem]))

پایان الگوریتم

سازگاری (یکنوایی) اکتشافات

در این مورد ، مطمین شوید که مسیر بهینه ی وضعیّت های تکرار شده اوّلین باری است که توسعه داده می شود (به وجود می آید) . این کار می تواند با انتخاب اکتشاف صحیح انجام شود .

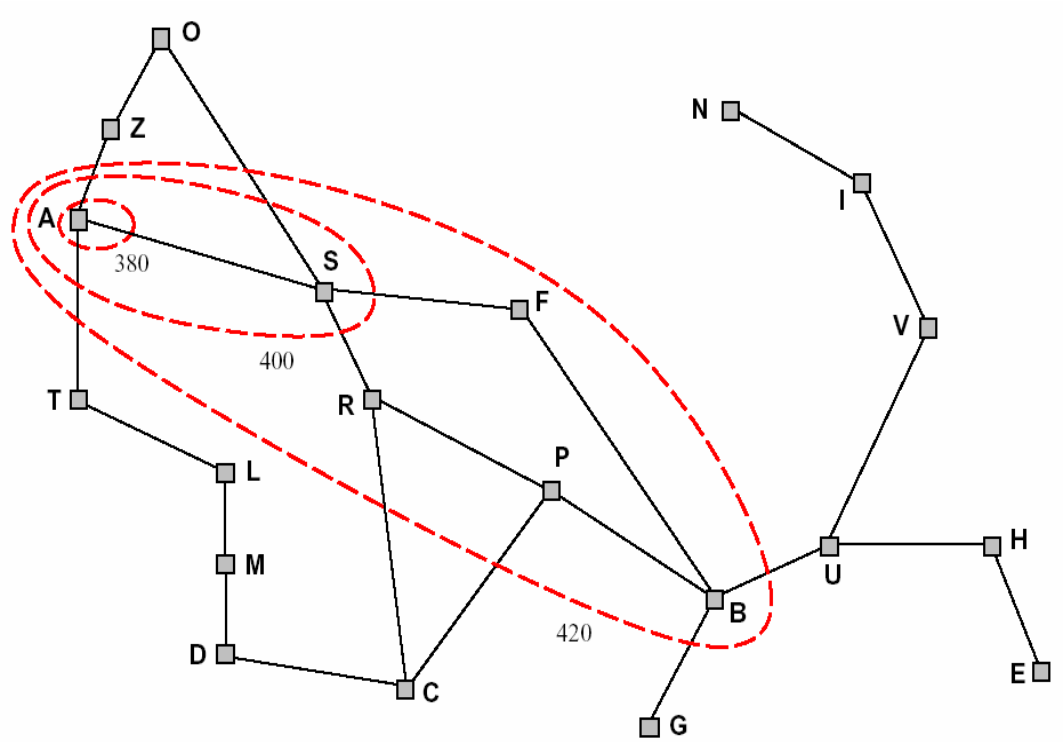
با توجّه به نامساوی مثلثی داریم : $h(n) \leq c + h(n')$



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

بهینه سازی A^* (مفیدتر)

روش A^* ، گره ها را براساس ترتیب افزایش مقدار f به وجود می آورد .



خصوصیّات 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.

اگر 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 .

روش A^* به طور موثری برای هر تابع ارزیابی بهینه مـی باشـد . یـک الگـوریتم ممکـن است که راه حل بهینه را در صورتی که همه ی گره های با $f(n) < C^*$ را توسعه ندهد ، گم کند .

بهتر کردن A^*

A^* یکی از الگوریتم های کلیدی در هوش مصنوعی می باشد ، امّا دارای اشکال در حافظـه ی مـورد نیاز می باشد چون که تمام گره های ملاقات شده را در حافظه نگه می دارد و برای مسایل با سـاختار بـزرگ ، عملی نمی باشد ؛ برای رفع این مشکل از روش عمیق شونده ی تکراری استفاده می کنیم .

روش A^* عمیق شونده ی تکراری

در این روش ، به جای محدود کردن عمق ، به صورت تدریجی محدودیّت تابع ارزیابی افزایش داده می شود و جستجوی اوّل عمق با هزینه ی محدود شده را برای تابع ارزیابی انجام می دهد . مشکل این روش ، مقدار افزایش تدریجی محدوده ی ارزیابی است . برای رفع ایـن مشـکل یـک راه ایـن اسـت کـه مقـدار را در محدوده ی مقدار مرحله ی قبلی در نظر بگیریم و روش دیگر این است که محدوده را به وسیله ی یک مقـدار ثابت به نام ε افزایش دهیم .[1]

اکتشافات (ابتکارات)

پیدا کردن یک تابع مکاشفه ای خوب برای جستجو کاری بسیار مهم است . تا کنـون مـا از فاصـله ی اقلیدسی استفاده می کردیم که برای مسایل مسیریابی ، خوب بود . امّا مکاشفات را برای مسایل جدید چگونـه

[1] برگرفته از مطالب Milos Hauskrecht استاد دانشگاه ایالت پیتسبورگ کشور ایالات متحده ی آمریکا

پیدا کنیم ؟ جواب این است که راه استانداردی وجود ندارد ، در بیش تر موارد باید مکاشفات به صورت دستی پیدا شوند .

ابتکارات قابل قبول [۱] – برای پازل ۸- تایی : $h_1(n)$ = تعداد کاشی های در جای نادرست گذاشته شده . $h_2(n)$ = مجموع فاصله تا جزیره ی مانهاتان [۲] (به تعداد کاشی (خانه) های در جای درست قرار داده شده توجّه نمایید)

1	2	3
4	5	6
7	8	

حالت نهایی

7	2	4
5		6
8	3	1

حالت اوّلیّه

$$h_1(S) = ??6$$
$$h_2(S) = ??4+0+3+3+1+0+2+1=14$$

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 اوّل بهترین و حریصانه ی اوّل – بهترین پایین ترین h را توسعه می دهد و کامل نیست و همیشه هم بهینه نیست . جستجوی A^* پایین ترین g+h را

---

[۱] admissible heuristics

[۲] Manhattan

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.

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل پنجم

الگوریتم های جستجوی محلّی[۱]

[۱] local search algorithms

رئوس مطالب

- تپّه نوردی (بالا رونده از تپّه) [۱]
- گرم و سرد کردن شبیه سازی شده [۲]
- الگوریتم های ژنتیکی ، پیدایشی یا تکوینی [۳] (به طور خلاصه [۴])
- جستجوی محلّی در فضاهای پیوسته (خیلی به طور خلاصه)

مسایل بهینه سازی

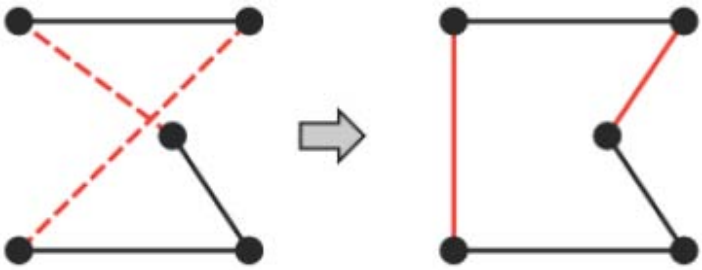
---

[۱] Hill-climbing
[۲] Simulated annealing
[۳] Genetic algorithms(GA)
[۴] briefly

در برخی از ساختارهای ترکیبی که ما در تلاش برای بهینه سازی آن ها هستیم ، محدودیّت ها باید در نظر گرفته شوند و تابع هزینه باید بهینه شود . اغلب پیدا کردن راه حل آسان است ، امّا پیداکردن بهترین راه حل سخت می باشد و پیداکردن همه ی راه حل های ممکن هم غیر ممکن می باشد و ما دست کم ، یک راه حل خوب را می خواهیم .

الگوریتم های بهبود یافته ی تکراری [۱] - در تعدادی از مسایل بهینه سازی ، مسیر ، نامربوط است ؛ وضعیّت (حالت) هدف ، خودش راه حل است . بنابراین ، برای فضای حالت ، مجموعه ای از پیکربندی های کامل ؛ پیکربندی بهینه ، یا ارضای محدودیّت 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 تک را نگهداری نمایید و برای بهبود آن تلاش نمایید. در فضای ثابت ، جستجوی برخط به اندازه ی جستجوی برون خط مناسب است .

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



مثال : وزیر های شطرنج n – تایی [۲] ، n وزیر را در یک صفحه ی شطرنج n×n بدون این که هیچ یک از دو وزیر در سطر [۱] ، ستون [۲] همانند قرار گیرند یا به طور مورب [۳] ، همانند قرار گیرند

---

[۱] iterative improvement algorithms

[۲] n-queens

، قرار دهید . برای کاهش تعداد 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 مسأله ی بهینه

دو نوع روش وجود دارد : یکی روش **سازنده** [۴] که از یک وضعیّت ابتدایی شروع کنید و به طرف یک راه حل حرکت نمایید ؛ به عنوان مثال ، در مورد مسأله ی مسافرت شخص دوره گرد ، از یک شهر شروع نمایید و یک مسیر را با ملاقات همه ی شهرها به دست آورید . و دیگری روش **تکراری** که در این روش ، با یک راه حل شروع نمایید که می تواند غلط یا غیربهینه باشد و آن را به طرف یک راه حل بهینه ببرید ؛ به عنوان مثال ، در مورد مسافرت شخص دوره گرد ، با یک مسیر شروع کنید و هزینه ی آن را با تعویض شهرها بهبود دهید .

از روش سازنده استفاده نمی کنیم ، چون که هزینه فقط به راه حل بستگی دارد ، نه به چگونگی آن ؛ روش جستجوی آگاهانه (مثل A^*) وقتی که ما یک راه حل برای همه ی مسیرها داریم خوب کار می کند . از جستجوی ناآگاهانه هم استفاده نمی کنیم (چون فضای حالت در آن خیلی بزرگ می باشد) .

جستجوی محلّی

الگوریتم هایی که تا حالا ما برّرسی کرده ایم ، تمام مسیر را از وضعیّت اوّلیه به وضعیّت نهایی نگهداری می کنند و این باعث مصرف حافظه ی زیاد یا فضای زیاد در مورد مسایل بزرگ می شود . برای

---

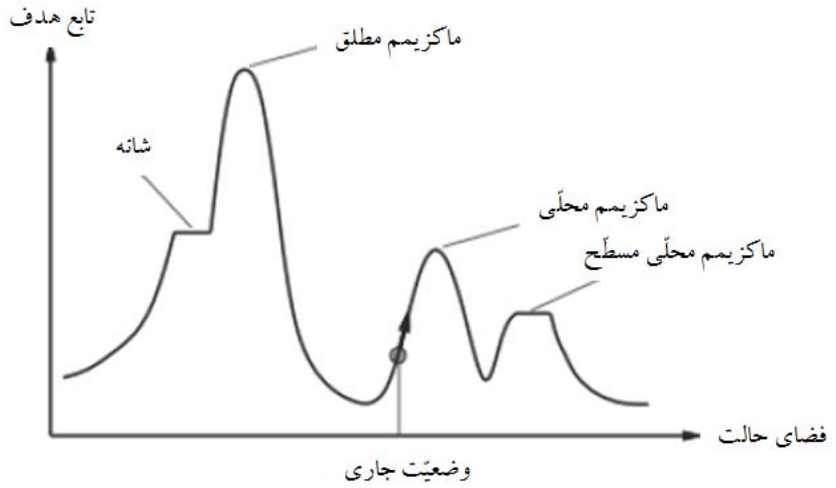
[۱] row
[۲] column
[۳] diagonal
[۴] constructive

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

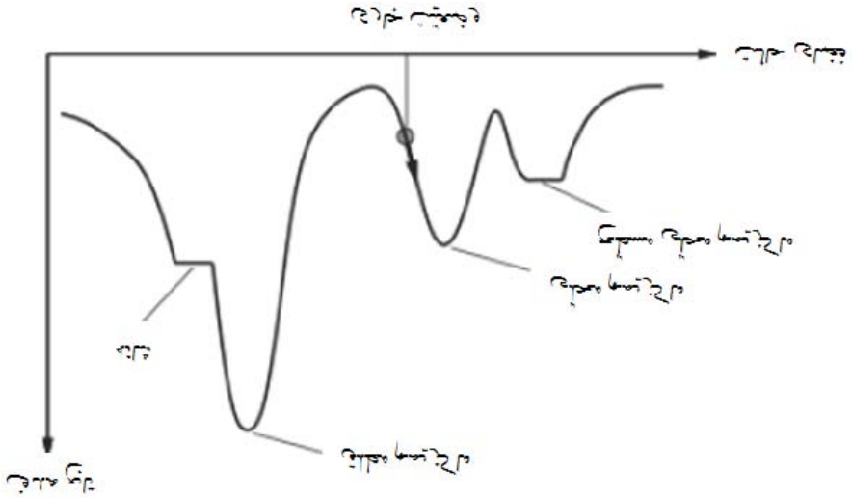
برخی از مسایل ، اطّلاعات مسیر ، ضروری (حیاتی) می باشد ؛ مثل ، مسیر یابی و پازل ۸ – تایی . در این مسایل ما هدف را می دانیم ، امّا این که چگونه به آن برسیم را نمی دانیم . برای دسته ای دیگر از مسایل ، ممکن است ما نگران چگونگی رشته ی عملکردها نباشیم و پیدا کردن راه حل بهینه ، چیزی است که مهم می باشد مثل : زمانبندی ، طرح VLSI و پنهان شناسی (رمزنویسی) [۱] . در این موارد ، ما می توانیم با اطمینان ، برخی از اطّلاعات مسیر را حذف نماییم . ***یک الگوریتم جستجو که فقط از وضعیّت جاری استفاده می کند ، یک الگوریتم جستجوی محلّی نام دارد .*** مزایای این روش عبارتند از : به حافظه ی ثابت ، نیازمندیم و می توانیم راه حل هایی باورنکردنی را در مورد فضاهای بزرگ پیدا نماییم . و معایب این روش عبارتند از : معمولاً ، بهینه نمی باشد – فقط بهینه های محلّی ، پیدا خواهند شد و می تواند به علت کمبود حافظه ، مردد باشد . در این روش ، فضای حالت شبیه روی زمین دارای تپّه ها و دره ها می باشد و ارزیابی توسط تابع هزینه (ارزیابی) داده شده است .

[۱] cryptography

فضای حالت یک بعدی [۱] برای مثال ها ، آسان تر نمایش داده می شود .



در این مورد ، ماکزیمم و مینیمم قابل تعویض اند ، برای این کار فقط مقدار را با استفاده از منفی کردن معکوس نمایید .

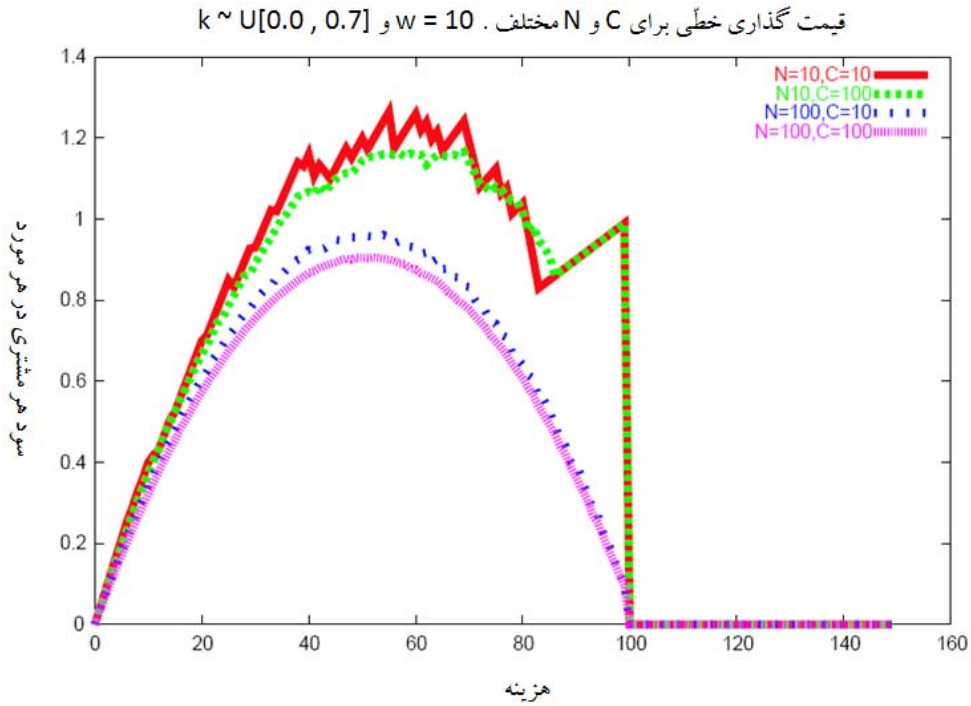


[۱] one-dimensional

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

دورنما (چشم انداز [۱]) جستجو

جستجوی محلّی ، معمولاً برای مسایل بهینه سازی ، مفید می باشد ؛ "پارامترهایی را پیدا نمایید که o(x) بیشینه / کمینه باشد " ، این روش زمانی که فضای حالت ، ترکیبی از پارامترها می باشد ، مشکل است . در صورتی که n پارامتر وجود داشته باشد ، ما می توانیم یک فضای n+1 بعدی را تصوّر نماییم ، که n بعد اوّل ، پارامترهای تابع می باشند و n+1 امین بعد ، *تابع هدف* [۲] می باشد که ما این فضا را یک چشم انداز جستجو می نامیم . توجّه کنید نقاط بهینه روی تپّه ها هستند و گودی ها ، راه حل های ضعیف هستند . یک چشم انداز یک بعدی را در شکل زیر مشاهده می نمایید :

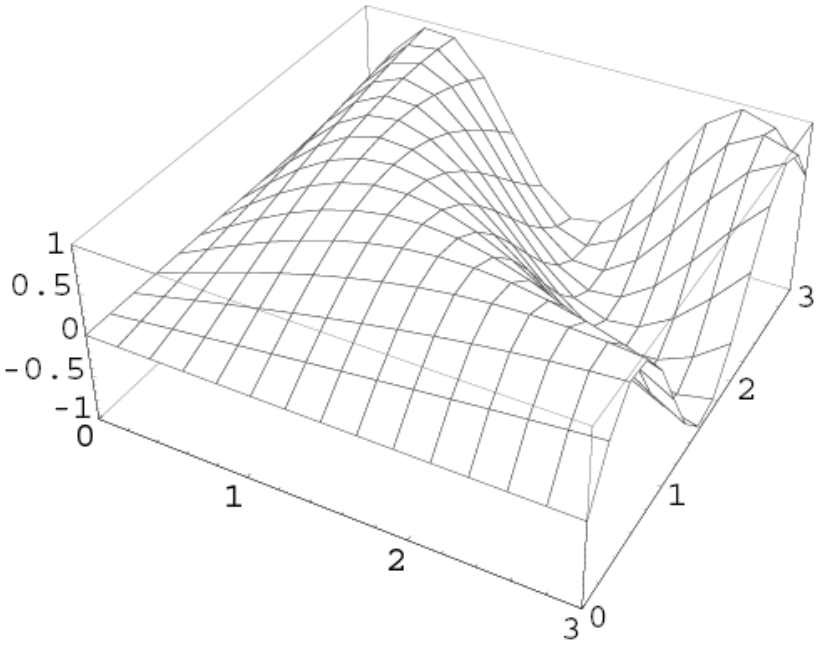


---

[۱] landscape

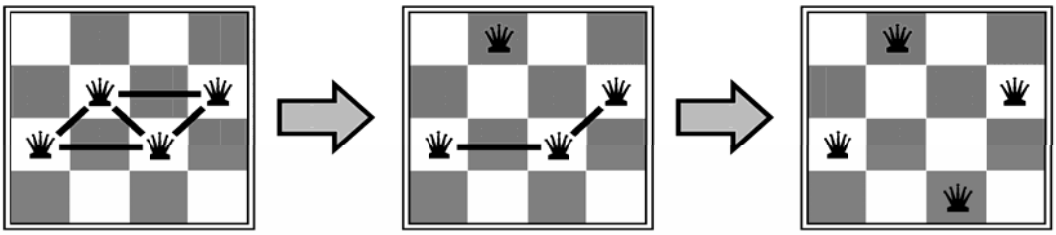
[۲] objective function

در شکل زیر یک چشم انداز دوبعدی نشان داده شده است :



دورنماها ، یک روش نمایش خوب برای الگوریتم های جستجوی محلّی هستند . بالا رفتن از یک تپّه را در نظر مجسم کنید ؛ این مورد یک راه را برای تشخیص مسایل آسان از مسایل سخت ، ارایه می کند . حالت های آسان ؛ تعداد قلّه های کم ، سطوح صاف ، بدون برآمدگی (خط الرّاس) / مسطّح (فلات) هستند و حالت های سخت : تعداد قلّه های زیاد ، سطوح دندانه دار یا ناپیوسته (گسسته) هستند . در شکل زیر که شکل مسأله ی ۴ – وزیر است وقتی که $h = 0$ می باشد ما هیچ حالت نادرستی را نداریم که در این مورد کاملاً راحت هستیم ولی وقتی که $h = 2$ است ما دارای دو حالت نادرست هستیم (چون که دو وزیر در یک ردیف قرار گرفته اند و دو وزیر دیگر به صورت مورّب قرار گرفته اند) و کار ما نسبت به حالتی که $h = 0$ بود سخت تر است و در حالت $h = 5$ چون که پنج مورد نادرست داریم کار ما نسبت به دو مورد قبلی که h

0 = و h = 2 بود سخت تر است . در حالتی که h = 0 است می توانیم تصوّر کنیم که در جایی مسطح قرار گرفته ایم و در حالتی که h = 5 است می توان تصوّر کرد که در شیب یک تپّه قرار گرفته ایم .[۱]



الگوریتم تپّه نوردی

ساده ترین شکل جستجوی محلّی ، جستجوی تپّه نوردی می باشد و دارای روشی ساده است به این ترتیب که در هر نقطه ، به جانشینان (همسایه ها) نگاه کنید و در جهت بهتر ، حرکت نمایید . این روش خیلی شبیه به جستجوی حریصانه می باشد و به حافظه ی خیلی کمی نیاز دارد . فلات می تواند سبب این شود که الگوریتم ، بی هدف ، سرگردان باشد .

جستجوی محلّی در حساب دیفرانسیل و انتگرال [۲]

ابتدا 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= $f'(x_1)$) . و برای سایر نقطه ها از فرمول $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_1)}$ استفاده نمایید ؛ این کاری که انجام دادید یک جستجوی تپّه نوردی می باشد و برای موارد مسطح ، خیلی خوب می باشد .

---

[۱] مترجم
[۲] calculus

الگوریتم

تابع Hill-Climbing(problem) یک وضعیّت 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 تپّه نوردی (به بیان دیگر)

تابع Hill-Climbing(problem) یک حالت که یک ماکزیمم محلّی است را برمی گرداند

ورودی ها ، problem که یک مسأله است می باشد

متغیّرهای محلّی current و neighbor که هر کدام یک گره هستند ، می باشند.

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

کارهای زیر را انجام بده

بالاترین مقداردهی successor از current ← neighbor (مقدار بالا ترین جانشین current را در neighbor قرار می دهد)

اگر Value[neighbor] ≤ Value[current] می باشد State[current] را برگردان

current ← neighbor

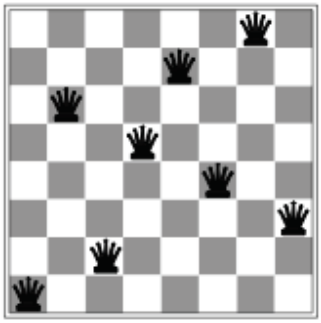
پایان حلقه

پایان الگوریتم

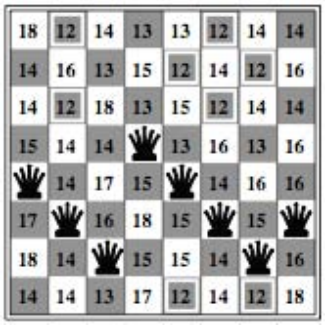
شروع مجدّد تصادفی الگوریتم بالا رونده از تپّه بر ماکزیمم محلّی غلبه می کند .

مثال جستجوی تپّه نوردی

فرمول بندی کامل برای ۸ – وزیر ، تابع جانشین : یک وزیر را به مربّع دیگر در همان ستون ببرید (۸*۷=۵۶ جانشین) که هزینه برابر است با تعداد جفت هایی که در جای نامناسب قرار دارند .



هزینه ی کمینه ی محلّی = ۱



هزینه ی فعلی =۱۷

خصوصیّات تپّه نوردی

پیاده سازی آن آسان می باشد ؛ مقدار کمی از حافظه را استفاده می نماید ولی تپّه نوردی ، زیاد تحت تأثیر شکل فضای حالت است در ماکزیمم محلّی کم خوب می باشد ولی برای فضاهای جوجه تیغی (دندانه دار) مانند بد است . در شانه ها ، ماکزیمم محلّی و خط الراس [1] ها دچار مشکل می شود ؛ در ماکزیمم محلّی نمی تواند قله ی بالاتر را ببیند . در شانه ، راهی به طرف بیرون را نمی تواند پیدا نماید و در خط الرّاس ها برای حرکت ، شما باید به طرف پایین حرکت نمایید و حرکت های بد را مجاز می داند .

تنوعات تپّه نوردی

حرکت های غیرمستقیم [2] : روی کف [3] (جای مسطح) حرکت می نماید **و** در حلقه نمی افتد (یک تعداد بیشینه از حرکات را ثبت می کند)

تپّه نوردی اتّفاقی : در میان حرکت های به طرف سربالایی یکی را به تصادف انتخاب می نماید .

اوّلین انتخاب تپّه نوردی : اوّلین حرکت سربالایی را انتخاب می کند .

تپّه نوردی با شروع مجدد تصادفی : چند تپّه نوردی از وضعیّت های اوّلیه ی انتخاب شده به صورت تصادفی است و تولید وضعیّت های تصادفی می تواند غیر جزیی باشد .

بهبود روش تپّه نوردی

---

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

تپّه نوردی ، می تواند خوشایند باشد ، برنامه نویسی آن آسان می باشد ، به فضای کمی نیازمند می باشد و در ضمن ، ممکن است ما ، روشی بهتر را نداشته باشیم . حال این سؤال مطرح می شود که چگونه این روش ، بهبود می یابد ؟ با استفاده از موارد زیر این روش بهبود می یابد :

- تپّه نوردی تصادفی – به صورت تصادفی حرکت های بالای تپّه را انتخاب نمایید
- تپّه نوردی با شروع مجدّد تصادفی
- تا زمانی که به یک بهینه برسید ، ادامه دهید
- یک وضعیّت اوّلیه را به صورت تصادفی ، انتخاب نمایید
- مجدّداً ، اجرا نمایید .
- بعد از n تکرار ، بهترین راه حل را نگهداری نمایید .

روش شبیه سازی گرم و سرد کردن

ضعف روش تپّه نوردی ، این می باشد که هرگز به طرف پایین تپه حرکت نمی کند و شبیه به جستجوی حریصانه ، نمی تواند "پشتیبان [1]" داشته باشد ؛ گرم و سرد کردن شبیه سازی شده ، تلاشی برای برطرف نمودن این موارد می باشد . ***ایده ی این روش ، گریختن از ماکزیمم محلّی با اجازه دادن به بعضی از حرکت های بد یا نادرست می باشد . امّا به تدریج تعداد و فراوانی [2] حرکت های بد کاهش می یابد .*** در واقع این روش تپه نوردی را با حرکت تصادفی ترکیب می نماید .

---

[1] back up

[2] frequency

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

*اگر یک توپ پینگ پنگ را به عمیق ترین جای یک سطح دارای گودی (کمینه یا مینیمم مطلق) بیندازید ، خارج کردن آن **سخت تر** از وقتی است که آن را در یک کمینه ی محلّی می اندازید .*

شبیه سازی گرم و سرد کردن

وقتی که یک فلز مثل آهن را خیلی گرم می کنید دارای ارتعاش های زیادی است ولی وقتی که به تدریج دما کم می شود ارتعاش های آن هم به تدریج کم تر می شود در تپه نوردی این کار مثل این است که از کمینه ی محلّی با اجازه دادن به برخی از حرکت های بد بگریزید ، امّا به تدریج تعداد و فراوانی آن ها را کاهش دهید .

الگوریتم شبیه سازی گرم و سرد کردن :

تابع Simulated-Annealing(problem,schedule) یک راه حل را برمی گرداند

ورودی ها ، یک مسأله و schedule که یک نگاشت از زمان به دما می باشد ، هستند .

متغیّرهای محلّی 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

$\Delta E \leftarrow$ 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

روش شبیه سازی گرم و سرد کردن ، همیشه حرکت های خوب را می پذیرد ولی در آن احتمال پذیرش یک حرکت بد وجود دارد .

خصوصیّات شبیه سازی گرم و سرد کردن – در دمای ثابت T ، حالت کاری ممکن است به توزیع بولتزمان [۱] برسد .

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

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 ؟؟

---

[۱] Boltzman

از روش شبیه سازی گرم و سرد کردن در بسیاری از موارد مثل قالب بندی VLSI [۱] ، زمانبندی خطوط هوایی و غیره استفاده می شود . شبیه سازی گرم و سرد کردن 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 پرتومحلّی [۲]

ایده : به جای یک حالت k ، حالت های k را نگه می دارد و بالاترین مقدار k را در بین همه ی آن ها (جانشینان یا نامزدها) 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 . این روش اغلب کامل است ، ولی تمام حالت های k در نهایت در بالای تپه ی محلّی قرار می گیرند ؛ در این مورد راه حل این است که جانشینان k را به طور تصادفی انتخاب نماییم . این روش ، یک روش برمبنای مسیر است که به چند مسیر " اطراف " مسیر جاری نگاه می کند و k وضعیّت را به جای یک وضعیّت در حافظه نگهداری می نماید و با k گره ی به طور تصادفی انتخاب شده شروع می کند ؛ تمام جانشینان تولید می شوند ، در صورتی که هدف پیدا شود متوقّف می شود و در غیر این صورت تعداد k تا از بهترین جانشینان انتخاب می شوند . در این روش اطّلاعات میان وضعیّت ها می توانند مشترک شوند و به وعده داده شده ترین ناحیّه ها حرکت می نماید . روش جستجوی پرتو محلّی اتّفاقی تعداد k جانشین را به صورت تصادفی انتخاب می نماید که این کار به وسیله ی احتمالی که توسط تابع ارزیابی تشخیص داده می شود انجام می شود .

الگوریتم های ژنتیکی

---

[۱] مخفف Very Large Scale Integration می باشد ؛ تعداد زیادی ترانزیستور که روی یک صفحه یا تراشه (chip) ی کوچک قرار گرفته باشند (مترجم) .

[۲] local beam search

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

الگوریتم های ژنتیکی می توانند به صورت یک شکل از جستجوی موازی تپه نوردی ، تصوّر شوند . ایده ی اساسی این است که برخی از راه حل ها را به صورت تصادفی ، انتخاب نمایید و برای به دست آوردن راه حل های جدید ، بهترین تکه های راه حل ها را با هم ترکیب نمایید و این کار را تکرار نمایید . در این روش جانشین (کاندید یا نامزد) ها ، تابعی از دو وضعیّت می باشند .

الگوریتم های ژنتیکی *= جستجوی پرتوی محلّی اتّفاقی + تولید جایگزین هایی از جفت حالت ها*

جستجوی پرتو محلّی به شکل " انتخاب طبیعی " : جانشینان (موجود تولید شده توسط والدین [۱]) یک وضعیّت (زیستمند [۲]) نسل بعدی را با توجّه به مقدارشان (سازگاری [۳] شان) تولید می کنند .

الگوریتم های ژنتیکی (GA ها) : الگوریتم های نمونه بعد از تکامل زیستی هستند .

در روش جستجوی پرتو اتّفاقی ، وضعیّت های جانشین به صورت تنوعاتی از دو وضعیّت والد تولید می شوند ، نه فقط یکی .

اصطلاحات الگوریتم ژنتیکی

جمعیّت [۴] ، به صورت اوّلیه مجموعه ای از k وضعیّت تولید شده به صورت تصادفی است و مجموعه ای از وضعیّت های مشتق شده است .

---

[۱] offspring
[۲] organism
[۳] fitness
[۴] population

نسل [۱] ، جمعیّت در یک نقطه از زمان است و معمولاً ، انتشار [۲] برای تمام جمعیّت همزمان [۳] می شود .

فرد [٤] ، یک عنصر از جمعیّت است و به صورت یک رشته با الفبای محدود توصیف می شود .

تابع مناسب [٥] ، هر چقدر که تابع مناسب تری انتخاب کنیم منجر به شانس های بهتر برای تولید مجدد می شود و واژه ی مناسب بودن بیان کننده ی خوبی یک راه حل می باشد .

ژنوم [٦] ، یک راه حل یا وضعیّت است .

ویژگی / ژن [۷] - یک پارامتر یا متغیّر وضعیّت است .

یک الگوریتم ژنتیکی پایه

یک جمعیّت 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

---

[۱] generation
[۲] propagation
[۳] synchronize
[۴] individual
[۵] fitness function
[۶] genome ؛ تمامی ترکیب ژنتیکی یک فرد یا جمعیتی که بوسیله کروموزومها به ارث می رسد .
[۷] trait/gene

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

رمزکردن یک مسأله

فرض کنید که ما می توانیم مسأله امان را به صورت یک رشته ی بیتی ، رمز نماییم . مثلا مسأله ی ۸ – وزیر را در نظر بگیرید . 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 توجّه به تابع مناسب (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 که این کار ، crossover نام دارد . برای انجام عمل crosspver دو راه حل را به صورت تصادفی انتخاب نمایید . برای انتخاب شایستگی مناسب ؛ تمام شایستگی ها را جمع نمایید ؛ داریم احتمال انتخاب s = شایستگی s ، تقسیم بر مجموع شایستگی . سپس یک نقطه (مکان هندسی) تصادفی را در رشته های بیتی انتخاب نمایید و تکه ی اوّل 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 جمعیّت جدید از همان اندازه به صورت جمعیّت قدیمی ، جمعیّت 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 جمعیّت اعمال می کند . در صورتی که یک ژن از جمعیّت اوّلیه گم شده بود ، crossover نمی تواند آن را تولید کند .

اصول الگوریتم ژنتیکی

تولید مجدد توسط crossover : دو قسمت کردن [1] والدین در یک نقطه ی crossover داده شده و ترکیب شده ؛ این کار دو وضعیّت جدید را تولید می نماید .

نقطه ی crossover : برای جلو بردن کار استفاده می شود و اغلب به صورت تصادفی انتخاب می شود ؛ برای هر دو والد باید یکسان باشد و باید نسل های قابل قبول را تولید نماید .

---

[1] split

جهش [۱] **:** تولید مشخصه های جدید که در والدها وجود ندارد ؛ تغییرات تصادفی اجزای رمزکننده در نسل که توسط احتمال کنترل شده است .

الگو [۲] : اجزای مفید یک راه حل می توانند در میان نسل ها باقی بمانند .

انتخاب مناسب ترین فرضیه [۳] **ها**

انتخاب شایستگی مناسب با استفاده از فرمو زیر می تواند منجر به تولید چند کپی منتشرشده شود .

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

انتخاب مسابقه ای [۴] : دو فرد را به صورت تصادفی و با احتمال یکسان انتخاب نمایید سپس با یک احتمال ثابت ، مورد مناسب تر را انتخاب نمایید .

انتخاب ردیف [۵] **:** همه ی فرضیه ها را براساس شایستگی مرتب نمایید ؛ احتمال انتخاب با ردیف متناسب است .

crossover

مرکب از تکه های ترکیب کننده ی افراد برای ساختن افراد جدید است .

---

[۱] mutation
[۲] schema
[۳] hypothese
[۴] tournament
[۵] rank

Crossover تک نقطه ای : یک نقطه ی Crossover را انتخاب نمایید و Crossover ها را از آنجا بردارید و قطعه ها را تعویض نمایید ؛ به عنوان مثال :

0111110 101|1110

crossover

1010101 011|0101

گام بعد :

0111110 101|1110

crossover

1010101 011|0101

گام بعدی :

0111110 101|1110

crossover

1010101 011|0101

گام بعدی :

0111110 101|1110

crossover

1010101 011|0101

گام بعدی :

0111110 101|1110

crossover

1010101 011|0101

گام بعدی :

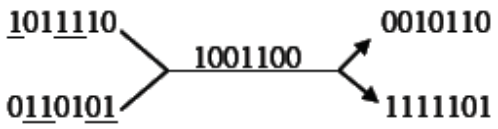
0111110 101|1110

crossover

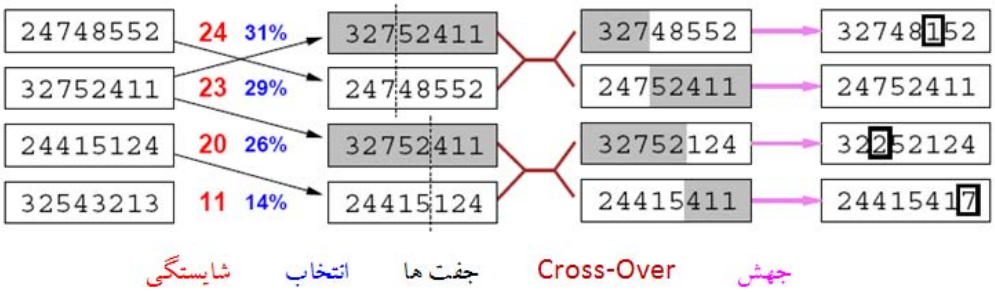
1010101 011|0101

پیاده سازی : از یک ماسک **crossover** (رشته ی دودویی) استفاده نمایید که در مثال 1110000 می باشد . دو والد h_1 و h_2 ارایه شده : $(h_1 \wedge m) \vee (h_2 \wedge \neg m), (h_1 \wedge \neg m) \vee (h_2 \wedge m)$ می باشند . Crossover با ماسک های اختیاری به صورت زیر است :

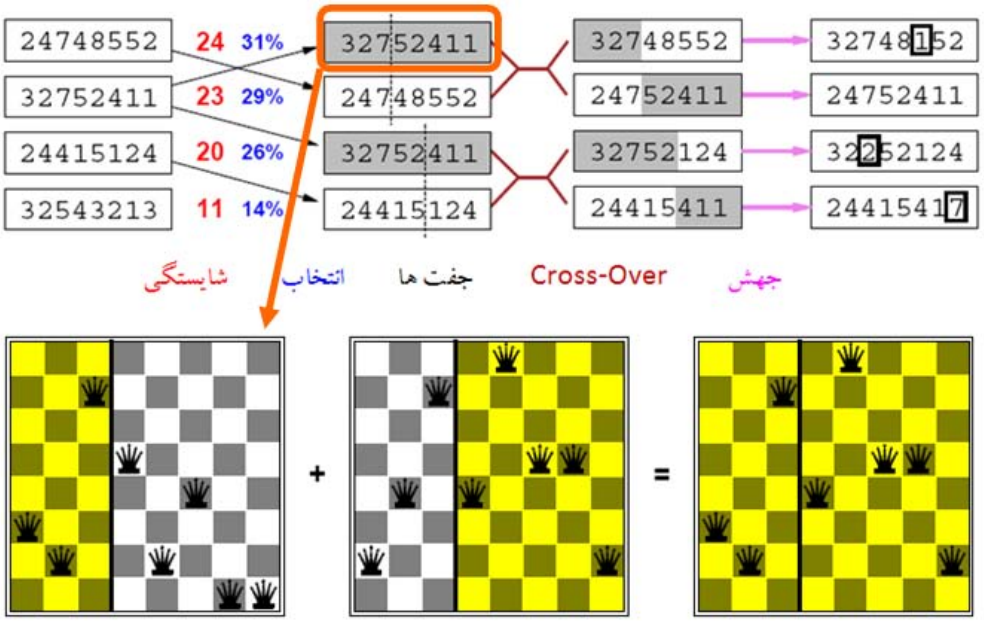
	اوّلیه	ماسک	نسل
تک نقطه	1011110 0110101	1110000	0111110 1010101
دو نقطه	1011110 0110101	0011100	1010110 0111101

یکسان	

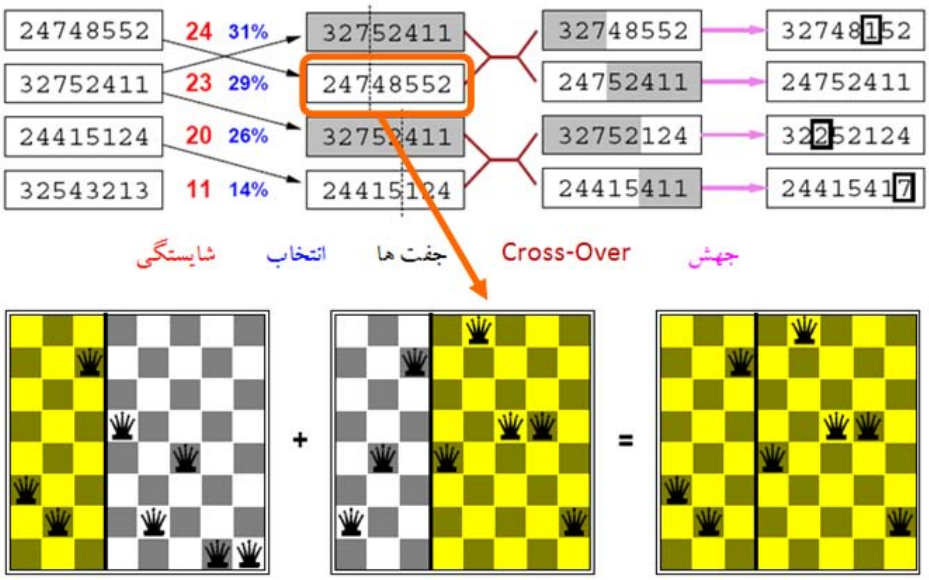
مثال : ۸ – وزیر



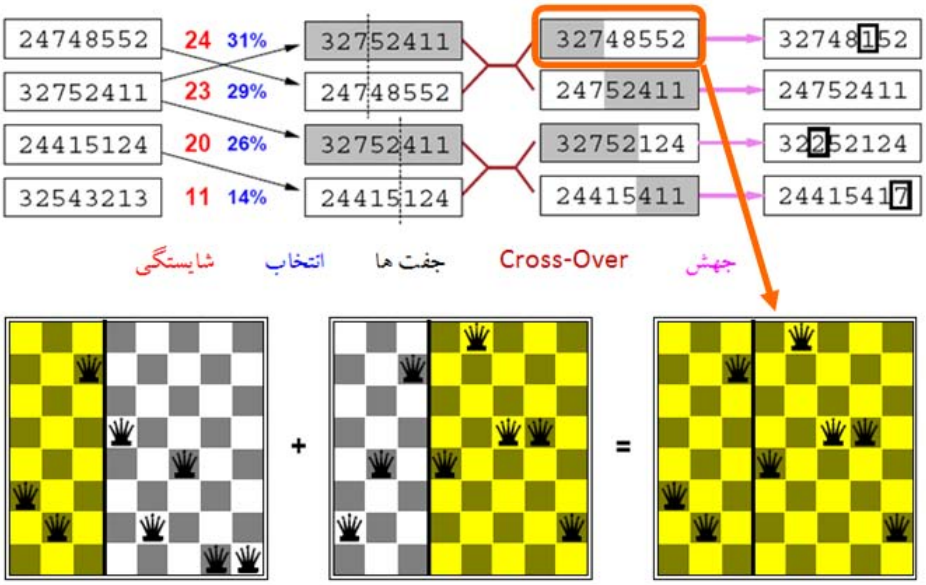
گام بعدی :



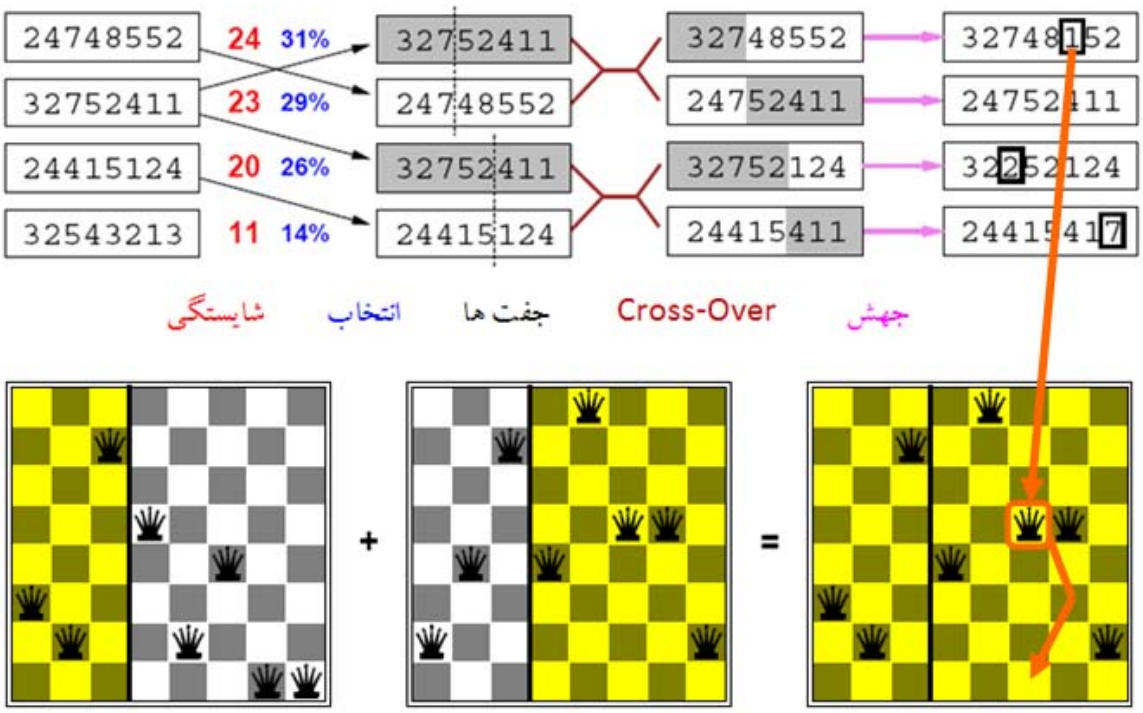
گام بعدی :



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) . crossover به زیر رشته های iff ، که اجزایی معنی دار هستند کمک می کند. الگوریتم های ژنتیکی با تکامل مخالفند ؛ عوامل تکامل [1] واقعی ، تشکیلات [2] تکراری را رمز گذاری [3] می کند .

الگوریتم های ژنتیکی به صورت جستجو

---

[1] genes
[2] machinery
[3] encode

وضعیّت ها ، راه حل های ممکن هستند . عملگرهای جستجو : جهش ، دورگه (crossover) و انتخاب هستند . از آنجایی که برای به وجود آوردن جمعیّت ها چند راه حل به صورت موازی به کار گرفته می شوند . با انتخاب تابع مناسب ، مثل روش تپّه نوردی ، بدون سراشیبی می باشد . جهش و دورگه باید به ما اجازه دهند که از کمینه ی (مینیمم) محلّی خارج شویم . وابسته به روش جستجوی شبیه سازی گرم و سرد کردن می باشد .

نکته : برنامه نویسی ژنتیکی وابسته به الگوریتم ژنتیکی می باشد و در آن ها فردها (رشته ها) همان برنامه ها هستند .

کاربردهای الگوریتم ژنتیکی

اغلب برای مسایل بهینه سازی مورد استفاده قرار می گیرد مثل : آرایش مدار ، طراحی سیستم و زمانبندی . به طور خلاصه باید بگوییم که به اندازه ی کافی برای پیداکردن راه حل ، مناسب می باشد ولی بهبود قابل توجّهی در مورد چند نسل ندارد و دارای محدودیّت زمانی هستند .

فضاهای حالت پیوسته – تصوّر 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 مربّعی 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 وضعیّت متناسب است . گرادیان از فرمول زیر محاسبه می شود:

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

[1] discretization methods

با توجّه 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 [۱] (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:

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

[۱] Newton-Raphson

فصل ششم

مسایل ارضای محدودیّت [۱]

[۱] Constraint Satisfaction Problems (CSPs)

رئوس مطالب

- مثال های مسایل ارضای محدودیّت
- جستجوی معکوس لیست [۱] برای مسایل ارضای محدودیّت
- ساختار مسأله و تجزیه [۲] ی مسأله
- جستجوی محلّی برای مسایل ارضای محدودیّت

مسایل ارضای محدودیّت

مسأله ی جستجوی استاندارد : حالت ، یک " جعبه ی سیاه [۱] " است – هر ساختمان داده ای قدیمی که از تست اهداف ، ارزیابی و جانشین پشتیبانی می کند .

---

[۱] backtracking

[۲] decomposition

مسأله ی ارضای محدودیّت : حالت ، توسط **متغیّرهای** X_i و با **مقدارهای** درون **دامنه ی** D_i تعریف می شود. تست هدف ، مجموعه ای از محدودیّت های معین کننده ی محدودیّت های مجاز از مقادیر زیرمجموعه ی متغیّر ها می باشد. یک **زبان ارایه ی رسمی** [۲] مثال ساده ای از آن می باشد ، الگوریتم های **همه منظوره** [۳] ی مفید را با توانایی بیش تر از الگوریتم های جستجوی استاندارد سبب می شود .

مثال نقشه ی رنگی : متغیّرها عبارتند از : WA,NT,Q,NSW,V,SA,T . دامنه ها :

$$D_i = \{red, green, blue\}$$

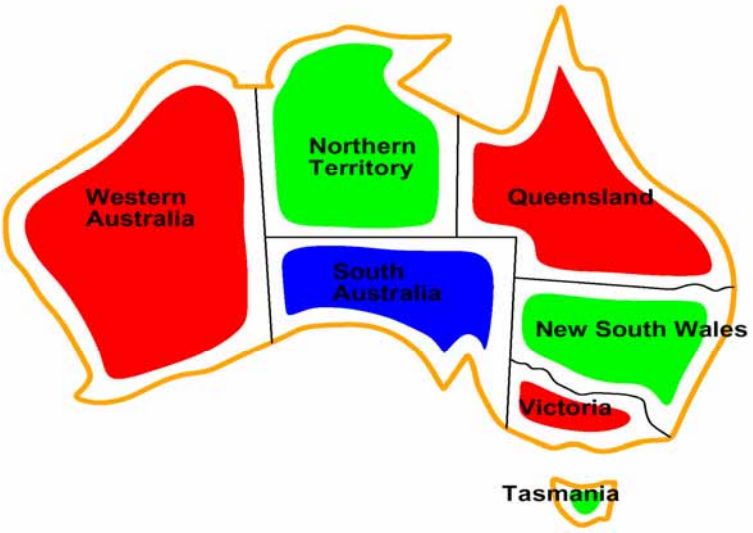
[۱] black box : نوعی از مهندسی معکوس است که هدف اصلی آن بررسی رفتار برنامه بدون توجه به کد برنامه ی آن است . مهندسی معکوس ، از حالت کامپایل در آوردن یا تکه تکه کردن برنامه ی کاربردی بدون توجه به زبان برنامه نویسی ای که توسط آن به وجود آمده است می باشد بنابراین ممکن است به کد اصلی برنامه دسترسی پیدا نماید . (Babylon / Code Analysis)

[۲] formal representation language

[۳] general-purpose

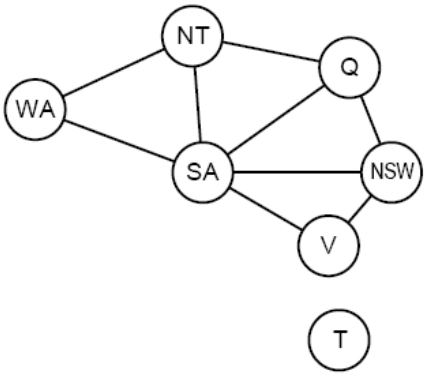
محدودیّت ها : نواحی مجاور ، باید رنگ های متفاوتی داشته باشند. به عنوان مثال $WA \neq NT$ (اگر زبان این مطلب را اجازه بدهد) ، یا $(WA,NT) \in \{(red,green),(red,blue),(green,red),(green,blue),\ldots\}$

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}



مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

گراف محدودیّت [۱]

مسأله ی ارضای محدودیّت دودویی [۲] : هر محدودیّت حداکثر دو متغیّر را تشریح می نماید.

گراف محدودیّت : در این گراف ، گره ها ، متغیّرها هستند ، کمان ها [۳] ، محدودیّت ها را نشان می دهند. الگوریتم های همه منظوره ی مسأله ی ارضای محدودیّت از ساختار درختی برای بالا بردن سرعت جستجو استفاده می کنند . توجّه نمایید که شهر تاسمانیا یک زیر مسأله ی مستقل است !

انواع مسایل ارضای محدودیّت

مسایل ارضای محدودیّت با متغیّرهای مجزا (گسسته) ، در دامنه های محدود ؛ اندازه d است ← (در نتیجه) $O(d^n)$ و انتساب ها کامل است . به عنوان مثال ، مسایل ارضای محدودیّت بولین [۴] (NP – کامل) . در دامنه های نامحدود (integer ها ، string ها (رشته ها) و .....) به عنوان مثال در زمانبند کار [۵] ، متغیّرها روزهای شروع / پایان برای هر کار هستند و به یک زبان محدود [۶] نیازمندند ، به عنوان مثال ، $StartJob_1 + 5 \le StartJob_3$.

مسایل ارضای محدودیّت با متغیّرهای پیوسته [۱] : به عنوان مثال ، زمان های شروع / پایان برای مشاهدات تلسکوپ [۲] هابل [۳]

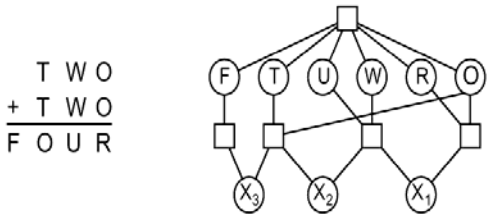
---

[۱] Constraint graph
[۲] Binary CSP
[۳] arcs
[۴] Boolean CSPs
[۵] job scheduling
[۶] constraint language
[۱] continuous variables

تنوع محدودیّت ها

محدودیّت های منحصر به فرد [۳] ، شامل یک متغیّر منفرد می شوند ، به عنوان مثال ، SA≠WA ؛ محدودیّت های دودویی ، شامل جفت هایی از متغیّرها می باشند ، به عنوان مثال ، SA≠WA و محدودیّت های مرتبه ی بالاتر ، شامل تعداد سه یا بیش تر متغیّر می باشند ، به عنوان مثال ، محدودیّت های با ستون های پنهان (رمزی) [۴] .

مثال : مسایل رمزی ، ما باید هر حرف را طوری تغییر دهیم که حاصل جمع به دست آید .



متغیّرها : $X_3\ X_2\ X_1$ F T U W R O . دامنه ها : {۰،۱،۲،۳،۴،۵،۶،۷،۸،۹} . محدودیّت ها : alldiff(F,T,U,W,R,O) و X_1 O+O=R+10. و غیره .

مسایل ارضای محدودیّت در دنیای واقعی

مسایل انتساب ، به عنوان مثال ، چه کسی درس می دهد و در چه کلاسی ؛ مسایل برنامه ی زمانی ، کدام کلاس ارایه می شود ، چه زمانی و کجا ؟ ؛ پیکربندی سخت افزاری - صفحات گسترده [۱]

---

[۱] Telescope
[۲] Hubble
[۳] unary
[۴] cryptarithmetic column constraints
[۱] برنامه ای که امکان اجرای محاسبات روی چندین ستون از اعداد را برقرار کند ، spreadsheet نام دارد

– زمان بند کننده ی انتقال – زمان بند کننده ی مرکز تولید . توجّه کنید که تعدادی از مسایل دنیای واقعی شامل متغیّرهای 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 نسبت داده شده تعریف می شدند : حالت اوّلیه را برابر با تهی در نظر می گیریم ؛ تابع جایگزین ، یک مقدار را به یک متغیّر فاقد مقدار شده نسبت می دهد طوری که با مقدار فعلی تعارض یا ناسازگاری نداشته باشد . در نتیجه در صورتی که مقداردهی های مجاز وجود نداشته باشد ، رد می شود . آزمایش یا تست هدف : برّرسی می کند که انتساب جاری کامل باشد .

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

الگوریتم

تابع Backtracking-Search(csp) ، راه حل یا عدم موفّقیّت را برمی گرداند

Recursive-Backtracking({},csp) را برگردان

تابع Recursive-Backtracking(assignment,csp) ، soln یا عدم موفّقیّت 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

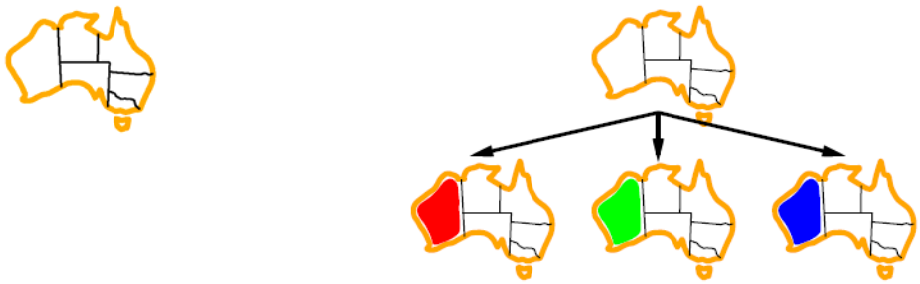
result ← Recursive-Backtracking(assignment,csp)

در صورتی که result ≠ failure است ، result را برگردان

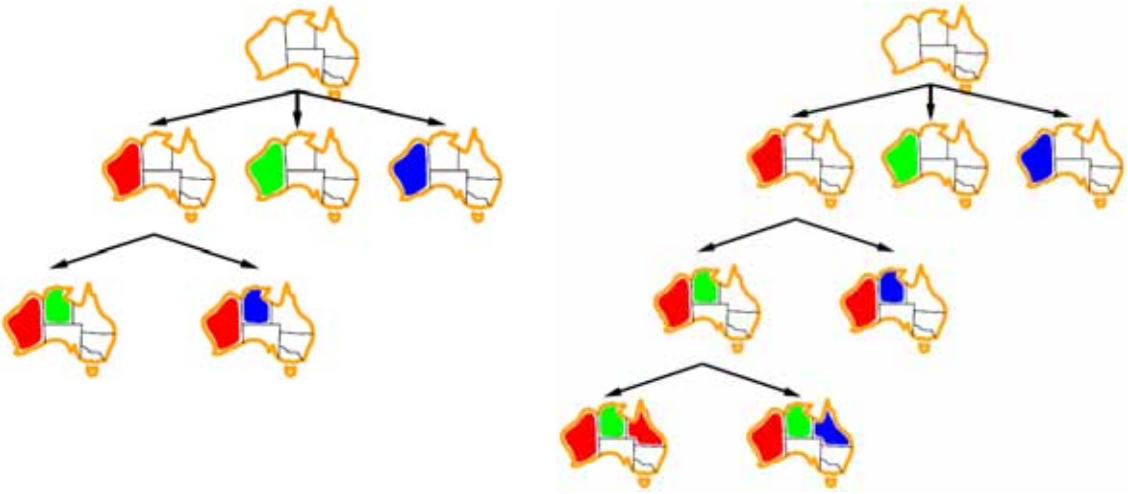
{var = value} را از انتساب [1] بردار

عدم موفّقیّت را برگردان

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



[1] assignment



بهبود بخشیدن به کارایی لیست معکوس

متدهای همه منظوره می توانند افزایش زیادی در سرعت داشته باشند :

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

a. قرمز دارای محدودترین مقدار می باشد

b. متغیّری با کم ترین مقادیر مجاز را انتخاب نمایید

[1] Minimum Remaining Values (MRV)

پیداکردن درجه – متغیّرها را در طول MRV به طور تصادفی بشکنید . کشف درجه : متغیّری با بیشترین محدودیّت در متغیّرهای باقی مانده را انتخاب نمایید .

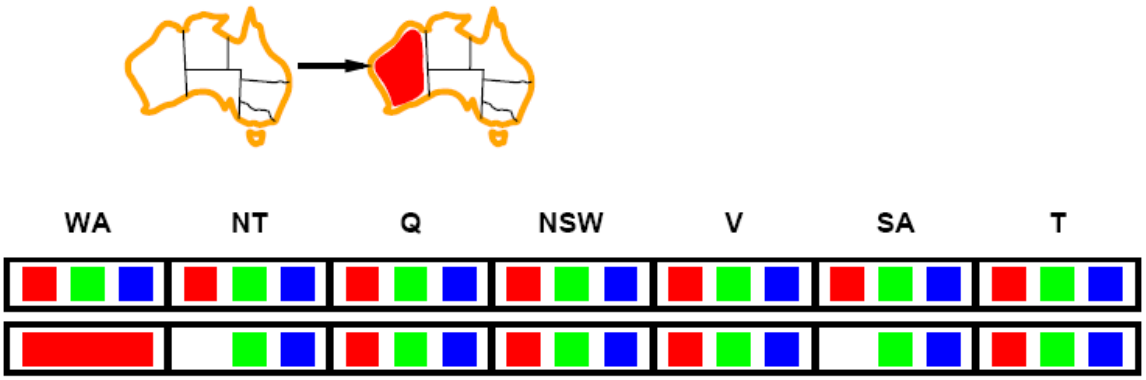
کم ترین مقدار محدود کننده – یک متغیّر ارایه شده ، کم ترین مقدار محدود کننده را انتخاب می نماید

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

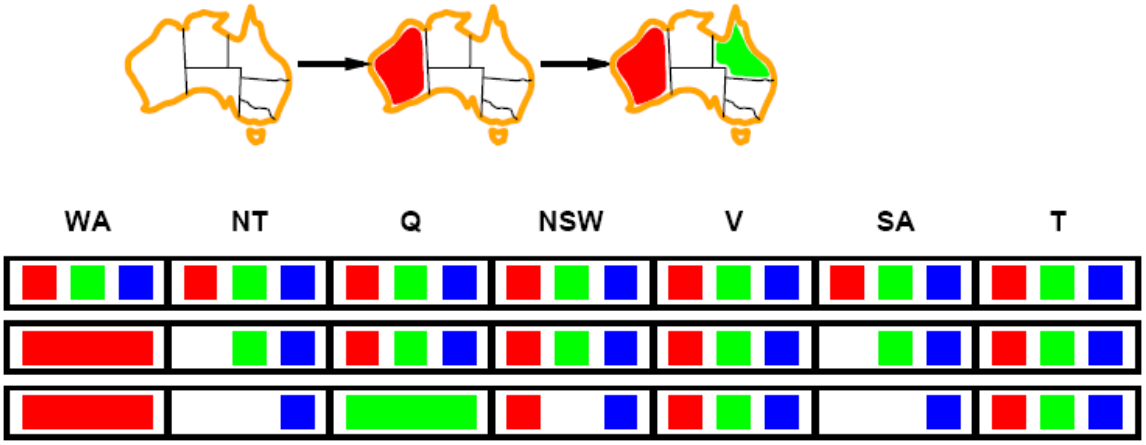
برّسی مستقیم [1] – مسیر باقی مانده با مقدارهای مجاز را برای متغیّرهای انتساب داده نشده نگهداری نمایید و جستجو را هنگامی که هر متغیّر مقدارهای مجاز ندارد خاتمه دهید .

[1] forward checking

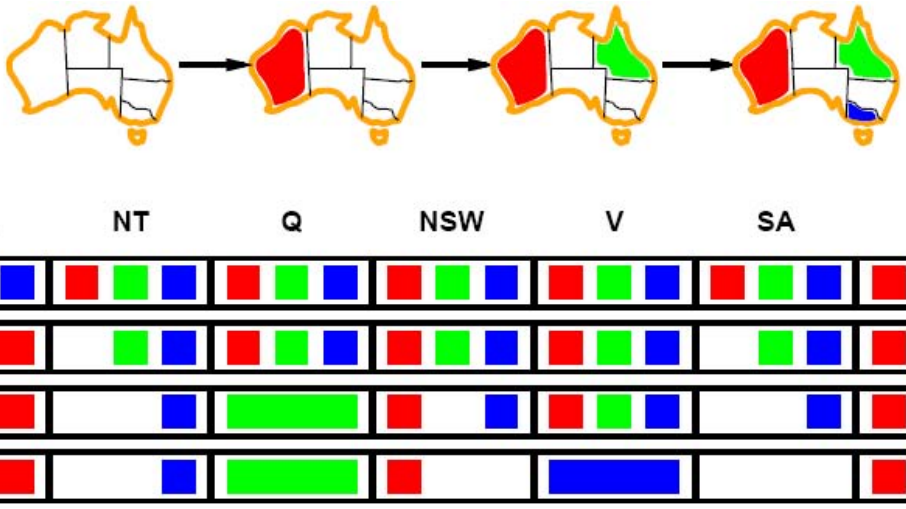
گام بعدی :



گام بعدی :



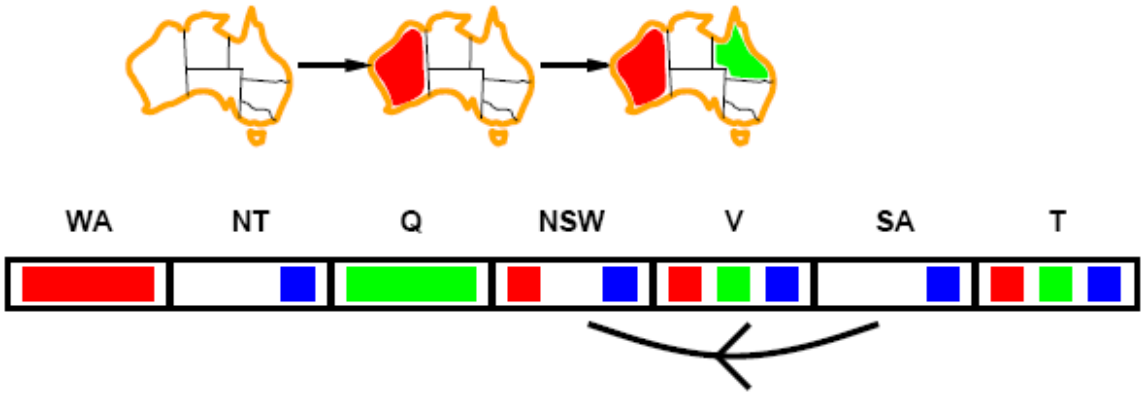
گام بعدی :



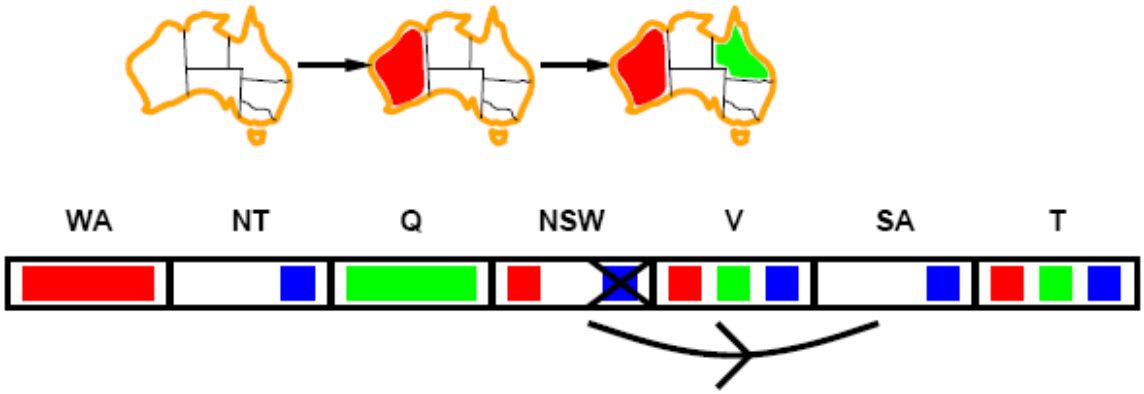
توزیع محدود – برّرسی اطّلاعات توزیع های مستقیم از متغیّرهای منتسب به متغیّرهای منتسب نشده ، به صورت سریع برای همه ی عدم موفّقیّت ها عمل نمی کند :

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

سازگاری قوس [۱] – 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 .

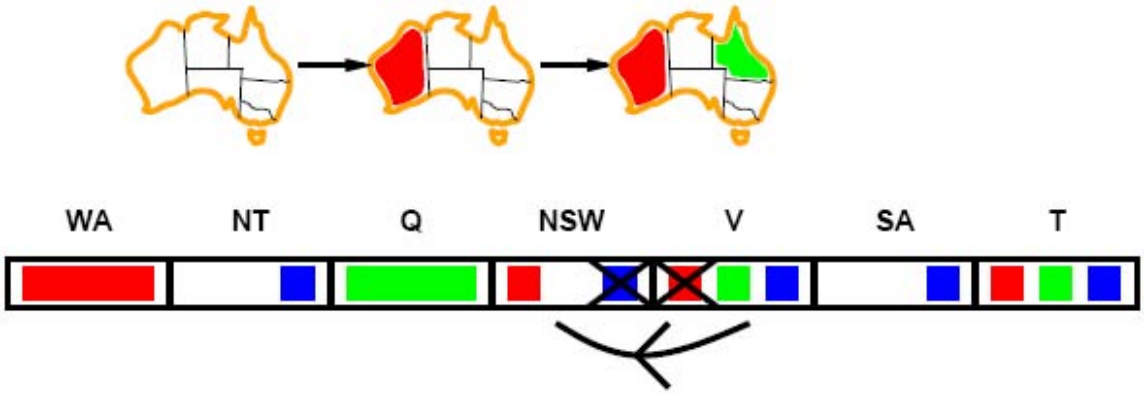


گام بعدی :

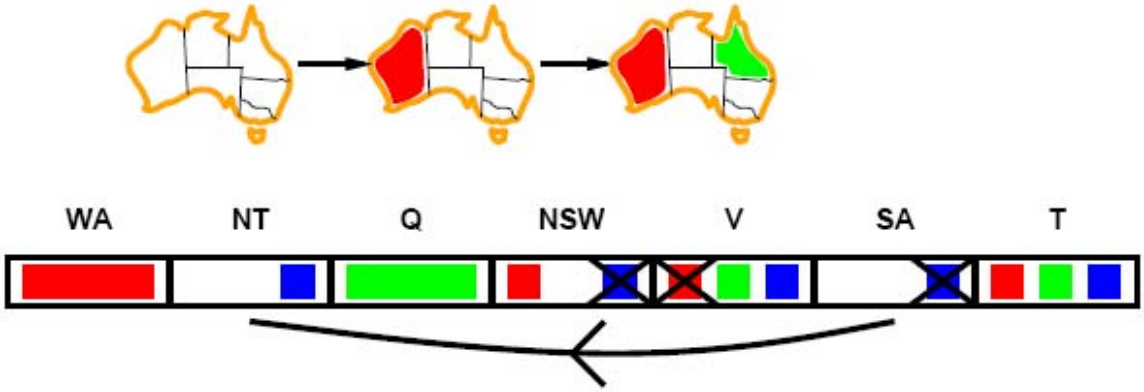


گام بعدی :

[۱] Arc consistency (AC)



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



اگر X یک مقدار را از دست داد ، همسایه های X باید دوباره برّرسی شوند. سازگاری قوس زودتر از بررسی مستقیم ، عدم موفّقیّت پیدا می کند و 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 مسأله ی ارضای محدودیّت دودویی با متغیّرهای $\{X_1, X_2, ..., X_n\}$ می باشد

متغیّرهای محلّی ، queue ، که یک صف از قوس ها که همه به صورت اوّلیه 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 X_iNeighbours( ) کارهای زیر را انجام بده

(X_k, X_i) را به queue اضافه کن

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

remove $\leftarrow$ 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) محدودیّت $X_i \leftrightarrow X_j$ را ارضا کند آن گاه x را از $Domain[X_i]$ حذف کن ؛ removed $\leftarrow$ true

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 و با این کار فضای جستجو را هرس می کند.

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

سازگاری قوس هدایتی [۱]

سازگاری قوس ، تجدید نظر را تکرار می کند (تعداد دفعات وابسته به قوس ها و دامنه ها می باشد) ولی سازگاری قوس هدایتی ، فقط یک بار بازنگری می کند و یک ترتیب متغیّر را بازبینی می کند (<) ؛ مسأله ی ارضای محدودیّت در صورتی یک سازگاری قوس هدایتی است اگر هر قوس (X_i, X_j) که $X_i < X_j$ سازگاری قوس باشد . توجّه کنید که سازگاری قوس ، کلّی تر از سازگاری قوس هدایتی است و برای درخت ها مناسب می باشد .

سازگاری مسیر [۲]

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 محدودیّت های x_1 و x_m باشد ؛ یک انتساب برای $X_2,...,X_{m-1}$ که راضی کننده ی محدودیّت های (X_i, X_{i+1})باشد وجود دارد . مسأله ی ارضای محدودیّت 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 جفت های مقادیر را برمی دارد و محدودیّت ها را صریح می سازد (به عنوان مثال ، X<Y و X+1<Z ⇐ Y<Z

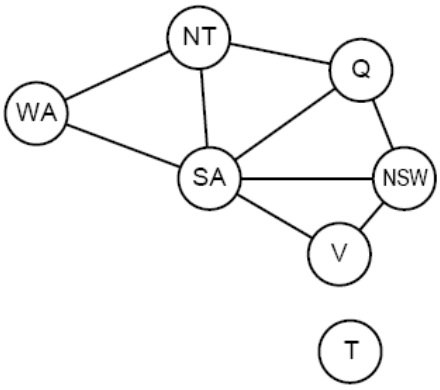
K – سازگاری [۱]

---

[۱] Directional Arc Consistency (DAC)

[۲] Path Consistency (PC)

k – سازگاری ، اَشکال سازگاری را تعمیم می دهد . مسأله ی ارضای محدودیّت ، k- سازگار است اگر برای هر مجموعه ی k-1 متغیّری با انتساب سازگار و یک مقدار سازگار بتواند به هر k انتساب داده شود . در صورتی که هر متغیّر با خودش سازگار باشد ، سازگار می باشد و دارای سازگاری قوس و سازگاری مسیر هم هست .



K – سازگار قوی - یک مسأله ی ارضای محدودیّت k – سازگار قوی است در صورتی که k – سازگار و k-1 – سازگار و ... و ۱- سازگار باشد .

ساختار مسأله [۲]

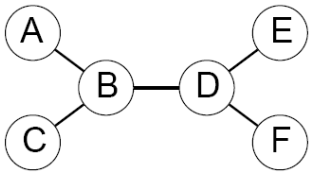
تاسمانیا و مین لند [۳] زیر مسأله هایی مستقل هستند و به صورت اجزای متصل شده به گراف محدودیّت قابل تعریف اند حال تصوّر کنید که هر زیر مسأله دارای متغیّرهای c تا حد مجموع n باشد . در بد ترین حالت ، هزینه ی راه حل برابر است با $n/c.d^c$ و به صورت خطی بر حسب مقدار n می

---

[۱] K-consistency
[۲] problem structure
[۳] mainland

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 .

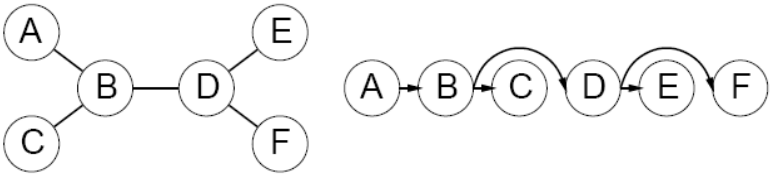
مسایل ارضای محدودیّت درختی ساختیافته [۱]



قضیّه : *در صورتی که گراف محدودیّت هیچ حلقه ای نداشته باشد ، مسأله ی ارضای محدودیّت 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 محدودیّت ها و پیچیدگی استدلال است.

الگوریتمی برای مسایل ارضای محدودیّت درختی ساختیافته

۱ . متغیّری را به صورت ریشه انتخاب نمایید ، متغیّرها را از ریشه به برگ ها مرتّب 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 RemoveInconsistent $(Parent(X_j), X_j)$ را به کار ببرید

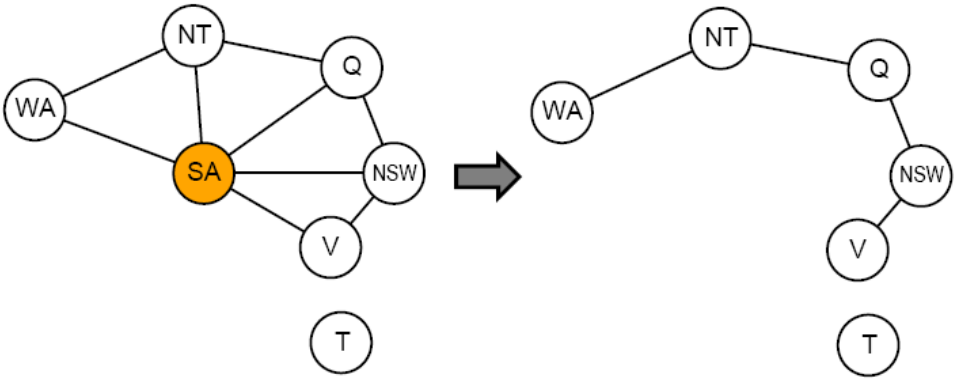
---

[۱] Tree-structured CSPs

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

مسایل ارضای محدودیّت درختی ساختیافته ی تقریبی [۱]

شایسته سازی [۲] : معرّفی یک متغیّر و هرس کردن دامنه ی همسایه های آن .



شایسته سازی برش [۳] : معرّفی (در همه ی موارد) یک مجموعه از متغیّرها تا این که گراف محدودیّت ، 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 تجزیّه [۴]

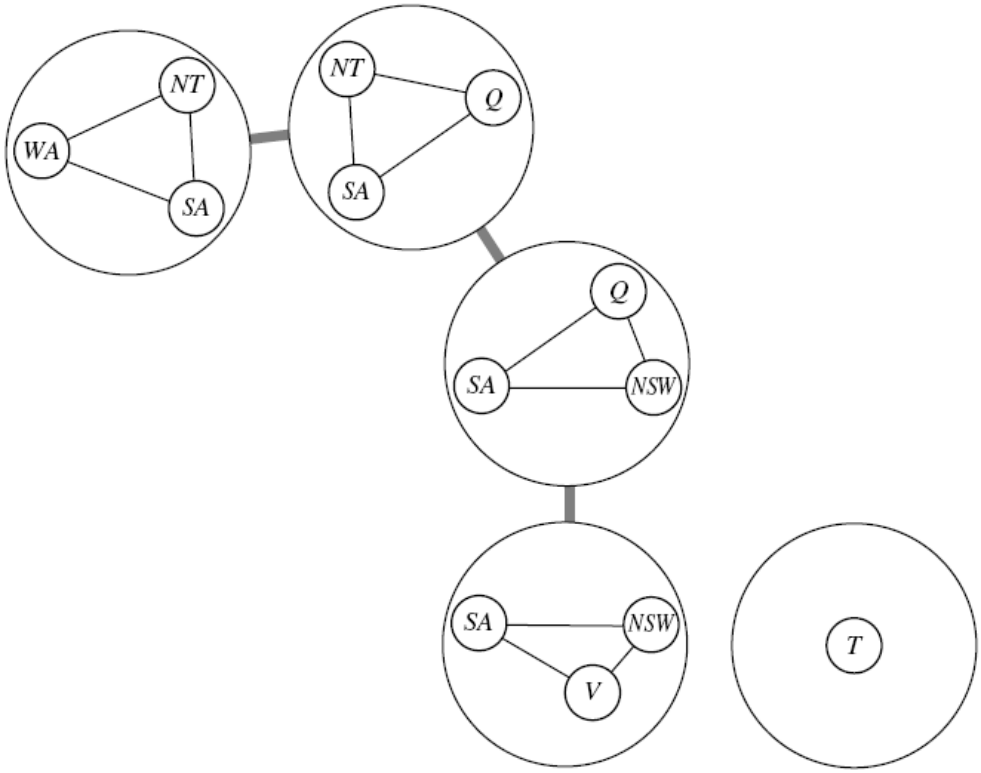
مسأله را به زیر مسأله های پیوسته تجزیّه می نماید که زیر مسأله ها همان گره ها هستند . احتیاجات (نیازهای) ما عبارتند از : برای هر متغیّر ، لااقل یک گره باید داشته باشیم ؛ متغیّرهای پیوسته باید لااقل در یک

---

[۱] Nearly tree-structured CSPs
[۲] Conditioning
[۳] Cutset conditioning
[۴] Tree decomposition

گره با هم به نظر برسند و متغیّری که به نظر می رسد در دو گره است باید در هر گره دیگر در مسیر وجود داشته باشد .

مثال درخت تجزیّه



توجّه کنید که محدودیّت میان گره ها ، سازگاری متغیّرها را سبب می شود و تجزیه منحصر به فرد نمی باشد در ضمن زیر مسأله ها باید تا حد امکان کوچک باشند . زیرمسأله ها به صورت مستقل حل می شوند و سپس به وسیله ی متغیّرهای مشترک به هم چسبانده [1] می شوند . *در صورتی که یک زیرمسأله دارای*

[1] glued

هیچ راه حلی نباشد آن گاه کل مسأله دارای هیچ راه حلی نمی باشد . در صورتی که اندازه ی بزرگ ترین مسأله ، w باشد ، آن گاه مسأله می تواند در $O(nd^{w})$ حل شود .

الگوریتم های تکراری برای مسایل ارضای محدودیّت – تپّه نوردی و شبیه سازی گرم و سرد کردن معمولی با حالت های کامل کار می باشد ، توجّه کنید که همه ی متغیّرها نسبت داده شده اند . برای به کاربردن با مسایل ارضای محدودیّت ، باید حالت های مجاز با محدودیّت های عملگرهای ناخوشایند ، دوباره به مقدار متغیّرها نسبت داده شوند .

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

ابتکار انتخاب مقدار با کم ترین ناسازگاری : مقداری را که از کم ترین محدودیّت ها تخلف می کند را انتخاب نمایید . توجّه نمایید ، در تپّه نوردی ، h(n) برابر با تعداد مجموع محدودیّت های متخلف می باشد .

مثال : چهار وزیر

تصوّر نمایید که یک وزیر در هر ستون وجود دارد .

متغیّرها : Q_1 و Q_2 و Q_3 و Q_4

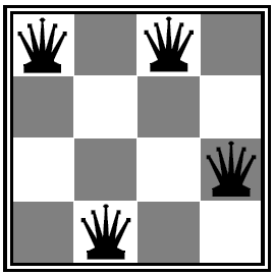
دامنه ها : $D_i = \{1,2,3,4\}$

محدودیّت 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)

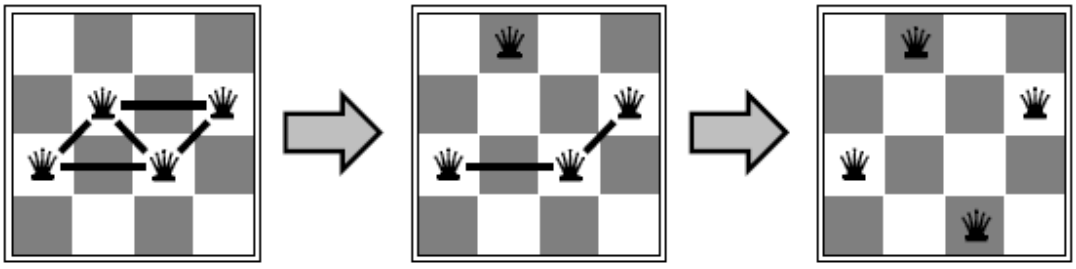
هر محدودیّت را به مجموعه ای از مقادیر مجاز برای متغیّرشان ترجمه نمایید :



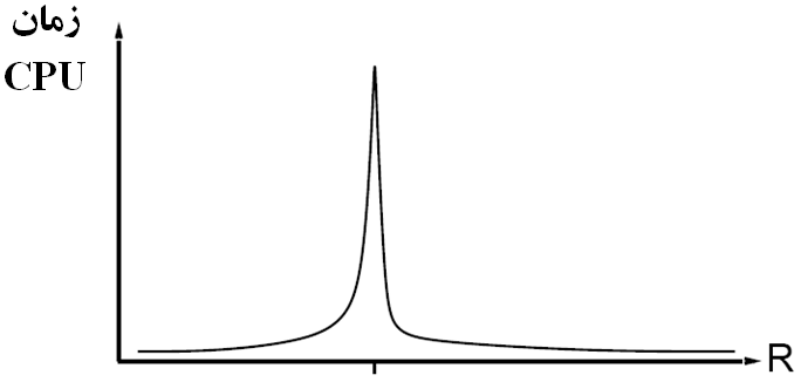
به عنوان مثال ، برای (Q_1, Q_2) مقادیر عبارتند از (۳و۱) و (۴و۱) و (۴و۲) و (۱و۳) و (۱و۴) و (۲و۴)

.

حالت ها (وضعیّت 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 اوّلیه ی تصادفی ارایه شده ، می تواند n – وزیر را در ثابت زمانی تقریبی دلخواه n با احتمال زیاد حل کند ($n = 10{,}000{,}000$) . بیان های همانند می توانند برای هر مسأله ی ارضای محدودیت به صورت تصادفی صحیح باشند **به جز** در یک محدوده ی باریک از نسبت زیر : **R** = **تعداد محدودیت ها / تعداد متغیّرها**



محدودیّت های دودویی

بیش تر تکنیک های گفته شده فقط با محدودیّت های دودویی کار می کنند . تعـدادی از مسـایل بـه طور طبیعی به صورت محدودیّت های n تایی [۱] ، رمزگذاری می شوند ، بـه عنـوان مثـال : X+Y=Z . توجّـه نمایید که گره با سازگاری زوج ، دودویی نمی باشد ؛ بـه عنـوان مثـال $X \leq 12$ و محـدودیت هـای یکـانی بـه سادگی می توانند با کاهش دامنه به کار گرفته شوند :

$$D'_X = \{1,5\} \leftarrow X \leq 12 \text{ و } D_X = \{1,5,24\}$$

کاهش محدودیّت های n تایی

هر مسأله ی ارضای محدودیت می تواند به یک مسأله ی ارضای محدودیّت دودویـی معـادل تبـدیل شود . برای این کار چند روش وجود دارد ؛ در این جا ما روش **رمزکننده ی متغیّر پنهان** [۲] را بررسـی مـی کنیم ؛ در این روش محدودیّت های n تایی با متغیّرهای جدید ارایـه مـی شـوند ؛ دامنـه ی متغیّرهـای جدیـد ،

---

[۱] n-ary

[۲] hidden variable encoding

چندتایی [۱] (n تایی) هستند ؛ محدودیّت های دودویی متغیّرهای " اصلی " را به اجزای چندتایی ها بچسبانید . مثال : X+Y=Z ، با $X,Y,Z \in N$. در این صورت متغیّر جدید U با دامنه ی $D_U = \{(x_1, x_2, x_3) \in N^3 \mid x_1 + x_2 = x_3\}$ خواهد بود . 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 مسأله ی ارضای محدودیّت

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

برنامه نویسی براساس محدودیّت

محدودیّت ها ، یک راه طبیعی برای بیان مسایل می باشند . مسأله ی ارضای محدودیّت ، با یک زبان برنامه نویسی حل می شود . برنامه نویسی با زبان های منطقی نظیر پرولوگ انجام می شود که دارای محیط اختصاصی می باشد (مثل موزارت [۲]) . پرولوگ به صورت های تجاری و رایگان ارایه شده است (به عنوان مثال SICStus یا GNU-Prolog) . برای اطّلاعات بیش تر نگاه کنید به راهنمای برخط برای برنامه

---

[۱] tuple

[۲] Mozart نامی برای نسخه ی ۷.۵ سیستم عامل شرکت Apple برای کامپیوترهای MAC و PowerPC . نسخه های جدید دارای خصوصیات چند وظیفه ای ، بهبود شبکه ، پشتیبانی چند رسانه ای بهتر و امکانات سیستم عامل های Dos و Windows می باشند . (Babylon / Jensen's Technology Glossary)

نویسی محدود با آدرس اینترنتی http://kti.mff.cuni.cz/~Ebartak/constraints/ و آرشیو محدودیت ها با آدرس اینترنتی http://www.cs.unh.edu/ccc/archive .

خلاصه ی فصل

مسایل ارضای محدودیّت ، نوع خاصی از مسایل هستند که در آن ها حالت ها توسط مقدارهای یک مجموعه ی ثابت از متغیّر ها تعریف می شوند . و هدف آزمایش ، توسط محدودیت هایی بر مقدار متغیّرها تعریف می شود . پیمایش معکوس ، جستجوی اوّل – عمق با یک متغیّر انتساب داده شده به هر گره می باشد . مرتب کردن متغیّر و انتخاب ابتکاری مقدارها به طور بسیار مطلوبی به ما کمک می کند . بررسی مستقیم ، از مقداردهی 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 . بیان مسأله به صورت مسأله ی ارضای محدودیت ، تحلیل ساختار مسأله را اجازه می دهد . درخت ساختیافته ی مسایل ارضای محدودیت می تواند در زمانی که به صورت خطی می باشد ، حل شود .

فصل هفتم

مسایل ارضای محدودیّت و برنامه نویسی ارضای محدودیّت [1]

[1] Constraint Satisfaction Programming

مسأله ی بهینه سازی محدودیّت [۱]

مسأله ی ارضای محدودیت استاندارد ، مجموعه ای از متغیّرهای مربوط به هم و یک مجموعه از محدودیت هاست . مسأله ی بهینه سازی محدودیّت همان مسأله ی ارضای محدودیّت استاندارد است به اضافه ی تابع هدف نسبت دهنده (نگاشت کننده) ی هر راه حل به یک مقدار عددی . راه حلّ این نوع مسایل ، انتساب کامل محدودیت های ارضا کننده و بیشینه سازی یا کمینه سازی مسأله است . به عنوان مثال در رنگ آمیزی نقشه ، هزینه ، وابسته به رنگ های استفاده شده است و چاپ با چند رنگ دارای هزینه ی بیش تری می باشد .

الگوریتم های مسایل بهینه سازی محدودیّت

الگوریتم ساده ، همه ی راه حل ها را به حساب می آورد و از تمام روش هایی که تا کنون توضیح داده شده است استفاده می کند . الگوریتم در زمانی که تعداد زیادی راه حل وجود داشته باشد بد است (به عنوان مثال ، n – وزیر) . بهبودهای الگوریتم مسأله ی ارضای محدودیّت برای اجتناب از جستجو در شاخه

---

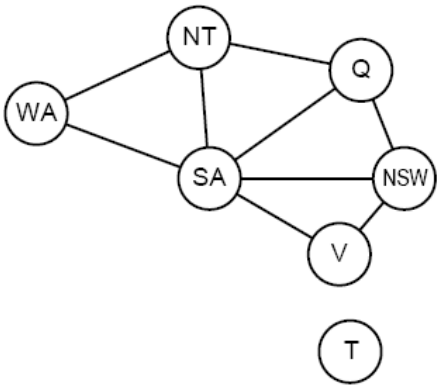
[۱] Constraint Optimisation Problem (COP)

های بدون راه حل طراحی می شود . ایده ی کلیدی این است که شاخه ها را با زیر راه حل های بهینه هرس نمایید و از مکاشفه برای هرس فضای جستجو استفاده نمایید .

ابتکارها (مکاشفه ها)

تابع هدف ، مقداردهی های کامل را ارزیابی می کند و تابع مکاشفه ای ، انتساب های جزیی را ارزیابی می نماید و به مقادیر عددی ، نسبت می دهد . در تخمین تابع هدف ، به همه ی راه حل های توسعه دهنده ی یک انتساب جزیی توجّه نمایید باید بهترین مقدار این راه حل ها را تخمین بزنند . توجّه کنید که مقداردهی های با مقادیر ابتکاری (مکاشفه ای) بد را توسعه ندهید . هرس اضافی و هرس استاندارد مسأله ی ارضای محدودیت هم می تواند به کار گرفته شود .

مسأله 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,S
A\=V.
```

هر مسأله ی ارضای محدودیّت با دامنه ی محدود می تواند به صورت یک شرط صریح تکی با برخی واقعیّات اضافه بیان شود . هر اتصال [۱] ، یک محدودیت در متغیّرهای شامل شده می باشد . بنابراین ، مسأله ی ارضای محدودیت را به زبان پرولوگ بنویسید و اجازه بدهید که مفسر [۲] آن را حل کند . مسأله این است که روش اوّل – عمق پرولوگ در بیش تر موارد ، بهترین نمی باشد .

برنامه نویسی منطقی محدود

الگوریتم هایی از الگوریتم های مسأله ی ارضای محدودیّت به پرولوگ استاندارد اضافه شده اند که معمولاً دارای نام گستره یا پسوند یا دامنه ی CLP می باشند (به صورت فایل های CLP.* هستند . [۳]) . CLP(FD) در دامنه های محدود ؛ CLP(B) در مورد بولین ها و CLP(Q,R) در مورد عقلانیّات و واقعیّات می باشند . با استفاده از مکانیزم محلی [۴] پرولوگ برای توسعه ی زبان پیاده سازی شده اند .

مثال : مسأله ی رمزی ؛ SEND+MORE=MONEY

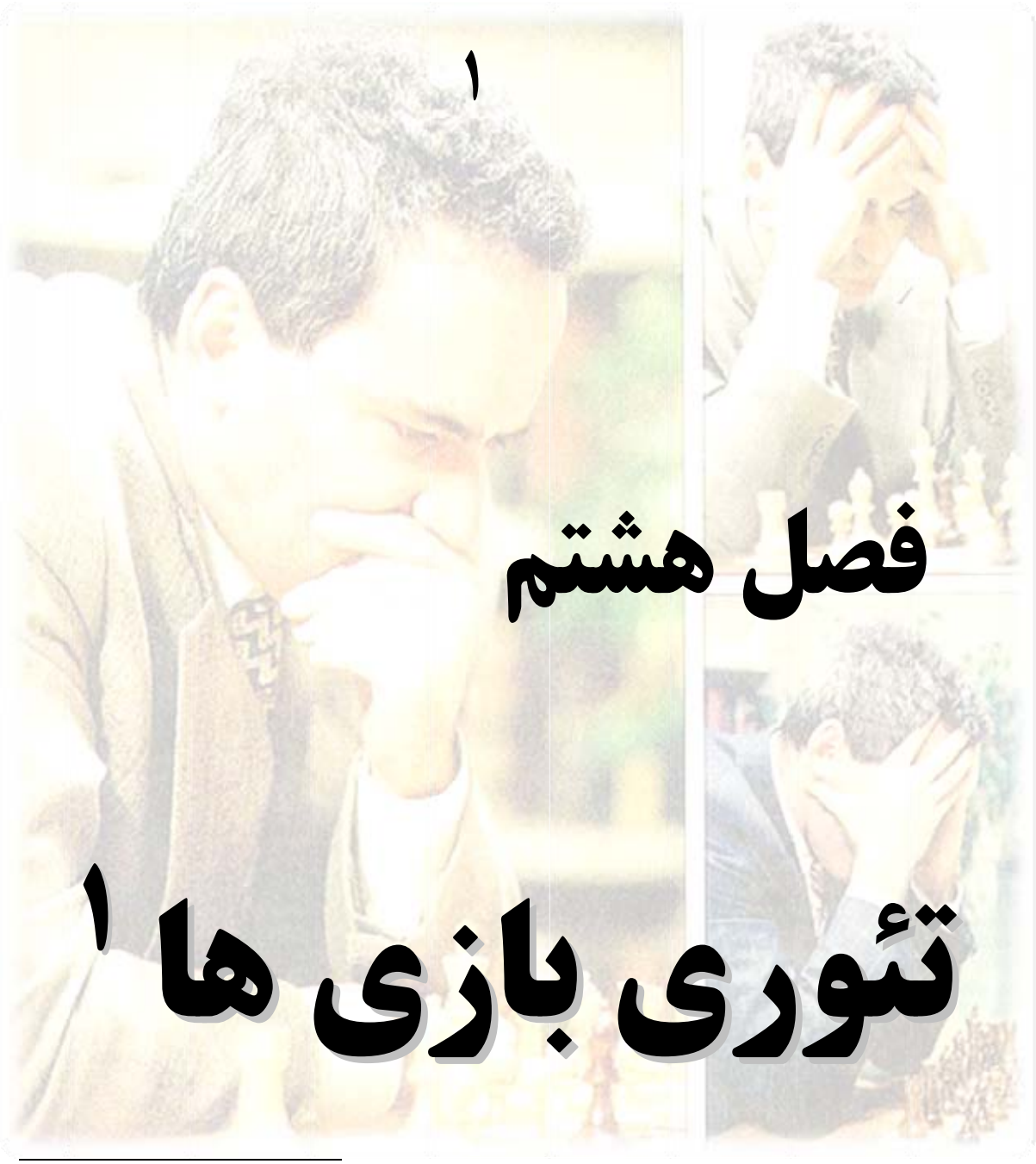
```
:- use_module(library('clp/bounds')).

cryptarithmetic(S,E,N,D,M,O,R,Y) :-
```

---

[۱] conjunct
[۲] interpreter
[۳] مترجم
[۴] native

```
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).
```



۱

فصل هشتم

تئوری بازی ها [۱]

[۱] – تصویرهای بالا متعلق به گری کاسپاروف (Gary Kasparov) ، قهرمان شطرنج روسی است .

[۱] Game playing

رئوس مطالب

- بازی ها
- بازی کامل [۱]

– تصمیم گیری های مینیماکس [۲]

– هرس [۳] α-β

- محدودیت منابع و ارزیابی تقریبی [٤]
- بازی های تصادفی [٥]
- بازی های با اطّلاعات غیر کامل [٦]

---

[۱] Perfect play

[۲] Minimax decisions

[۳] pruning

[۴] Resource limits and approximate evaluation

[۵] Games of chance

[۶] Games of imperfect information

محیط های چندعاملی

الگوریتم های جستجویی که ما تا حالا بررسی کرده ایم ، دارای یک محیط تک عاملی بودند . و عامل در حال کار در یک جهان بی اثر [۱] می باشد به عبارت دیگر ، دیگر عامل ها بر روی آن اثری نداشتند ؛ امّا در مواردی که یک عامل باید دیگر عامل ها را به حساب بیاورد چه طور ؟ مثل محیط های رقابتی نظیر بازی های شطرنج ، چکرز ، go و ... ؛ یا حراجی ها [۲] ، فروشگاه های برخط و محیط های شراکتی [۳] مثل عامل های یک تیم فوتبال .

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

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

بازی های با اطّلاعات 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 :

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

---

[۱] neutral

[۲] auctions

[۳] cooperative

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 در مورد این که چه کار باید بکند تصمیم گیری نماید ؟

روش های بهینه

بیایید با ساده ترین نوع بازی ها شروع کنیم ؛ مثل ، بازی های دو نفره ، بازی های با اطّلاعات کامل و بازی های رقابتی به صورت کامل – که در آن ها یکی می برد و دیگری می بازد . ما می توانیم این بازی ها را به صورت یک مسأله 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 : اوّل ، افراد بازی ها را دوست دارند ! و خوب می توانند با بازی ها کار کنند . دوّم ، اغلب ، بازی ها به صورت یک نماینده ی هوش دیده می شوند . سوم ، بازی ها دارای یک توضیح واضح از محیط می باشند ، ولی در آن ها فضاهای حالت ، خیلی بزرگ و پیچیده اند ، بنابراین برخی اوقات اطّلاعات اتّفاقی و ناکار آمد می باشند و این باعث به وجود آمدن چالش در مسأله 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 روش مشخص کننده ی یک حرکت ، برای هر جواب ممکن حریف است .

---

[1] opposite

محدودیت های زمانی موجود برای پیداکردن هدف ، نا مطلوب یا نـا خوشـایند هسـتند ، در نتیجـه ، بایـد تخمین زده شوند .

تاریخچــه : کـامپیوتر بـه مـوارد ممکـن در بـازی توجّـه مـی کنـد (Babbage,1846) | الگـوریتمی بـرای بـازی کامـل (Zermelo,1912;Von Neumann,1944) | وسـعت محـدود ، ارزیابی تقریبی (Zuse , 1945 ; Wiener , 1948 ; Shannon , 1950) | برنامه ی شطرنج اوّلیـه (Turing , 1951) | آموزش ماشینی [۱] برای بهبود بخشـیدن بـه درسـتی ارزیـابی (Samuel , 1952-57) | بازبینی یا هرس برای اجازه دادن به جستجوی عمیق تر (McCarthy , 1956)

انواع بازی ها

	قطعی	شانسی
اطّلاعات کامل	شطرنج[۲]،بازی چکرز[۳]،go	تخته نرد[۴]
اطّلاعات ناقص	تیک تا تو کور[۵]	پل[۶]،پوکر[۷]

[۱] machine learning
[۲] chess
[۳] checkers
[۴] backgammon
[۵] blind tictactoe
[۶] bridge
[۷] poker

شطرنج : بازی ای است که بر روی یک صفحه انجام می شود و برای دو بازیگر است که شانزده مهره را با توجّه 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 . [۱]

تیک – تاک – تو کور : بازی تیک – تاک – تو دارای دو بازیگر می باشد . برای هر دو بازیگر هدف این است که اوّلین 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 . [۲]

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 توجّه به عدد ظاهر شده روی دو تاس حرکت می دهد . [۳]

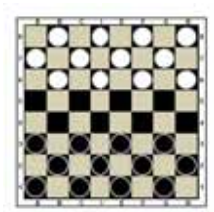
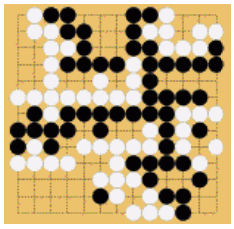
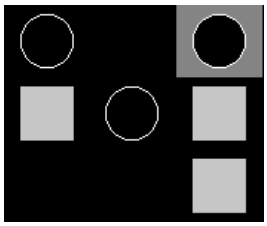
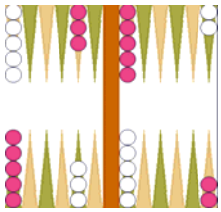
پل : یک بازی کارتی حقّه ای برای چهار نفر می باشد که این چهار نفر به صورت دو گروه دو نفری با هم بازی می کنند و در هر طرف هر گروه مقابل هم می نشیند . بازی پل دارای دو مرحله می باشد : پیشنهاد و بازی . [۴]

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 کاملاً پنهان یا اندکی پنهان روی یک چیزی شرط

---

[۱] wordnet.princeton.edu/perl/webwn
[۲] http://www.seeingwithsound.com/tictactoe.htm
[۳] wordnet.princeton.edu/perl/webwn
[۴] en.wikipedia.org/wiki/Bridge_game

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 .[1] در زیر تصویر صفحه ی چند بازی ای که هم اکنون معرّفی نمودیم مشاهده می نمایید :

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

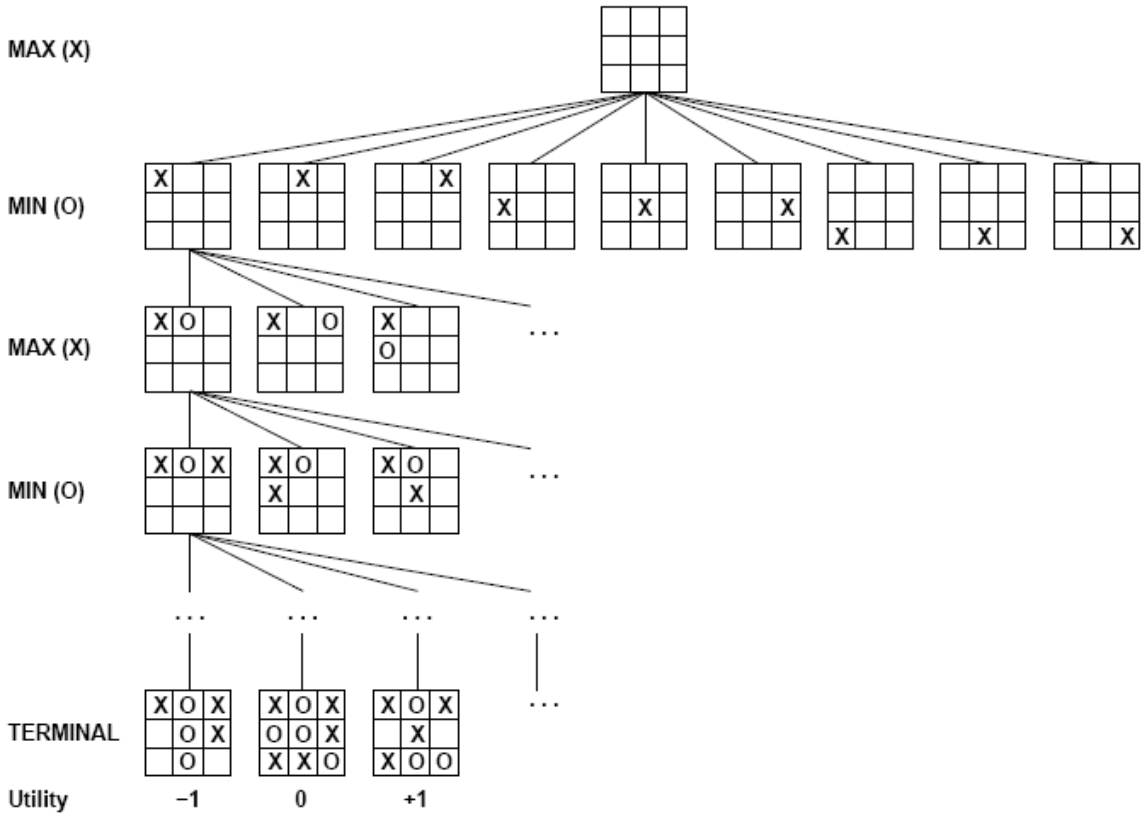
تئوری بازی به صورت جستجو – یک بازی دو نفره با اطّلاعات کامل را در نظر بگیرید ؛ آیا ما می توانیم این بازی را به صورت مسایل جستجو فرمول بندی نماییم ؟ ؛ وضعیّت صفحه ی بازی ، وضعیّت جستجوی مسأله می باشد ؛ حرکت های مجاز ، عملگرها هستند و وضعیّت های پایانی ، وضعیّت هایی هستند که درآن ها بازی را برده ایم یا 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 .

در این مورد ، یک حریف وجود دارد . که بداندیش است ؛ یعنی ، 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 . توجّه

[1] en.wikipedia.org/wiki/Poker

کنید که ، 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 .

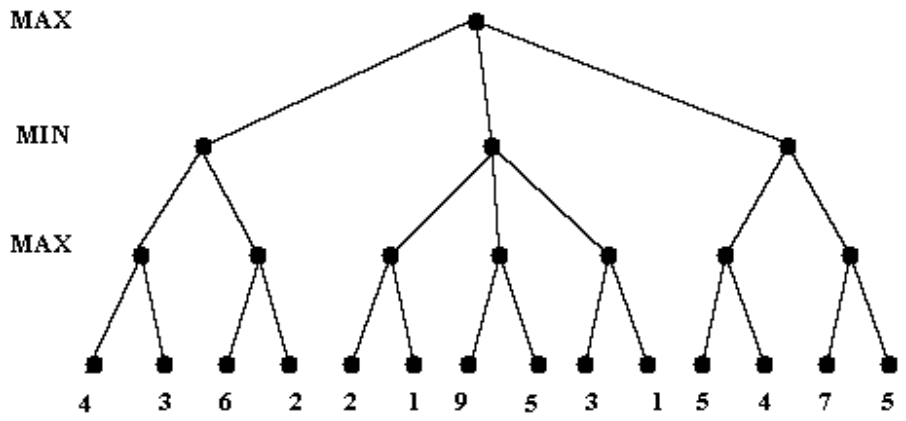
درخت بازی – در حالت کلّی ، ما می توانیم درخت بازی را برای هر بازی به صورت زیر رسم نماییم .



مینیماکس – [۱] MIN و MAX ، دو بازیگر هستند که MAX می خواهد بازی را ببـرد و MIN هم می خواهد بازی را ببرد و در واقع MAX و MIN ، رقیب هم هسـتند . و هـر دو بهتـرین بـازی ممکـن را انجام می دهند .

بازی کامل قطعی [۲] ، بازی های با اطّلاعات کامل می باشند .

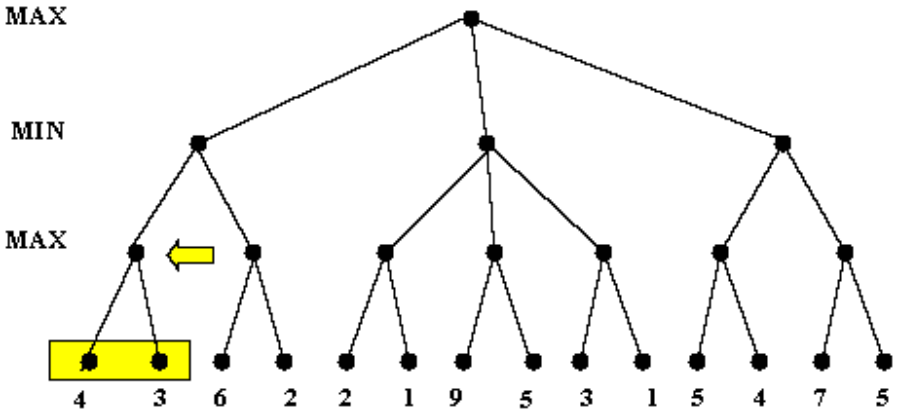
مثال زیر گویای این روش است :

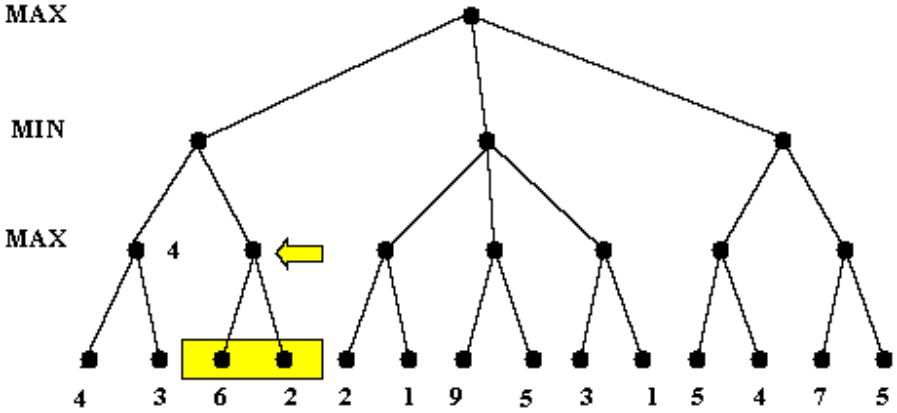


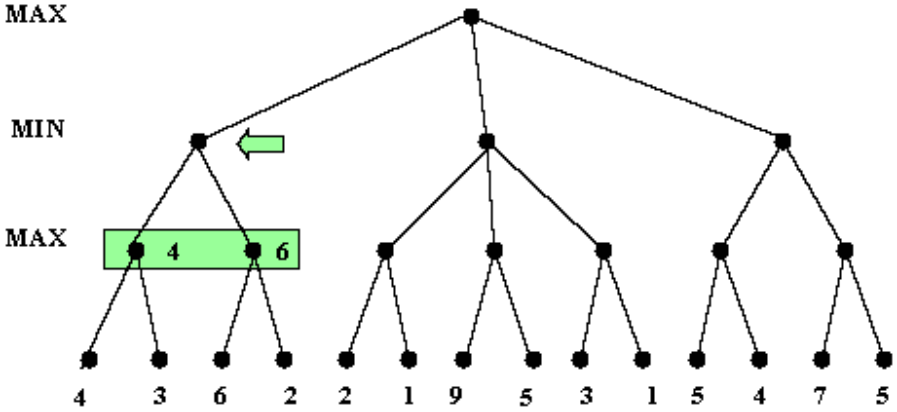
داریم :

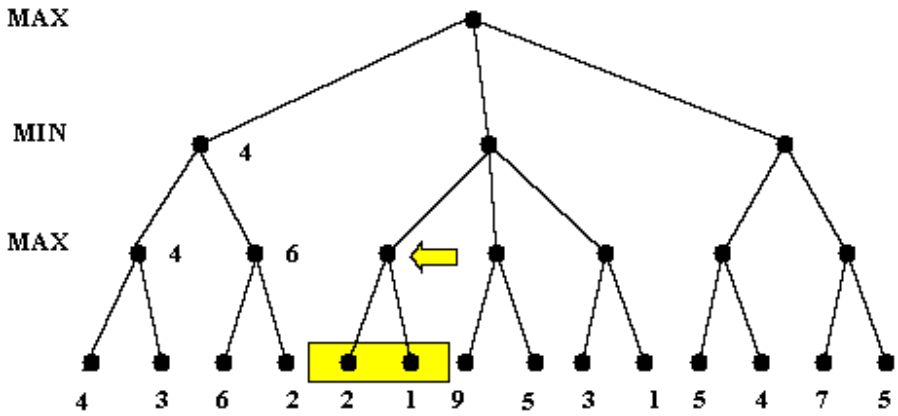
[۱] www.informatics.susx.ac.uk/books/computers-and-thought/gloss/node1.html

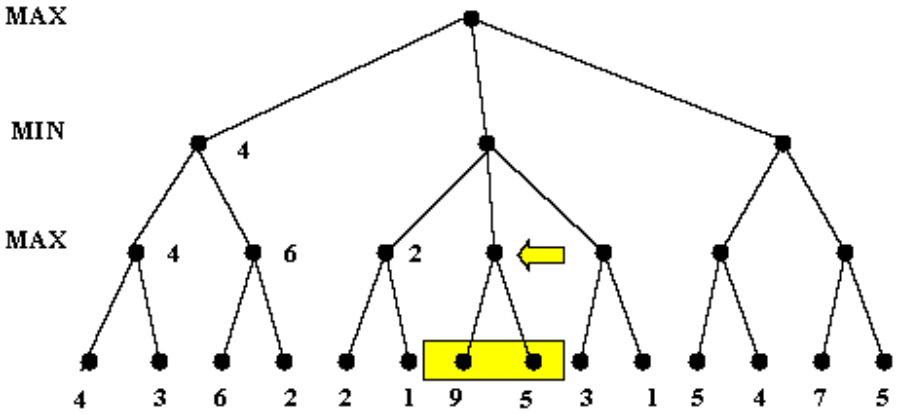
۱ Deterministic در کامپیوتر ، نتیجه ی فرایندی که به موقعیّت های اولیّه ورودی ها بستگی دارد

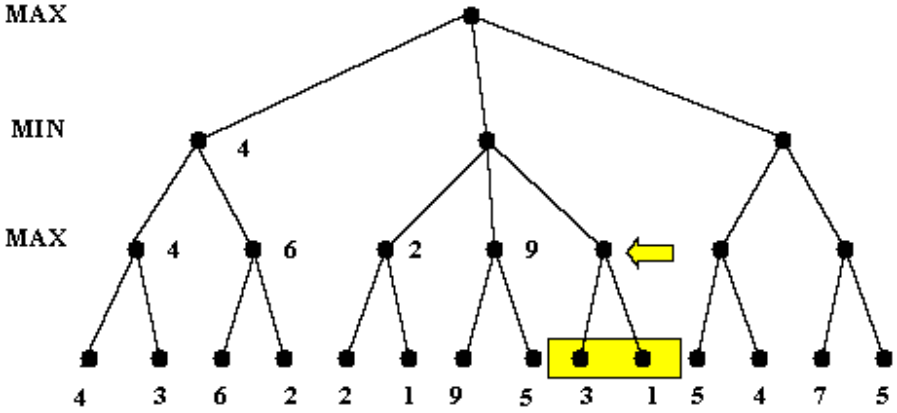

MAX
MIN
MAX
4 3 6 2 2 1 9 5 3 1 5 4 7 5

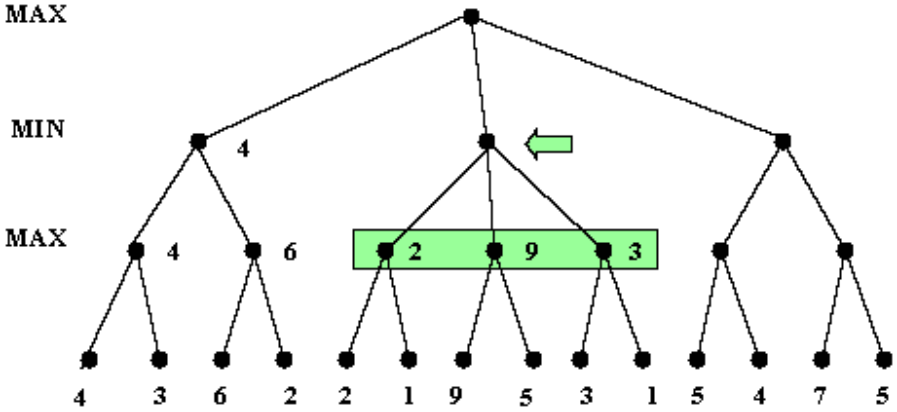

MAX
MIN
MAX
4
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
4 6
4 3 6 2 2 1 9 5 3 1 5 4 7 5

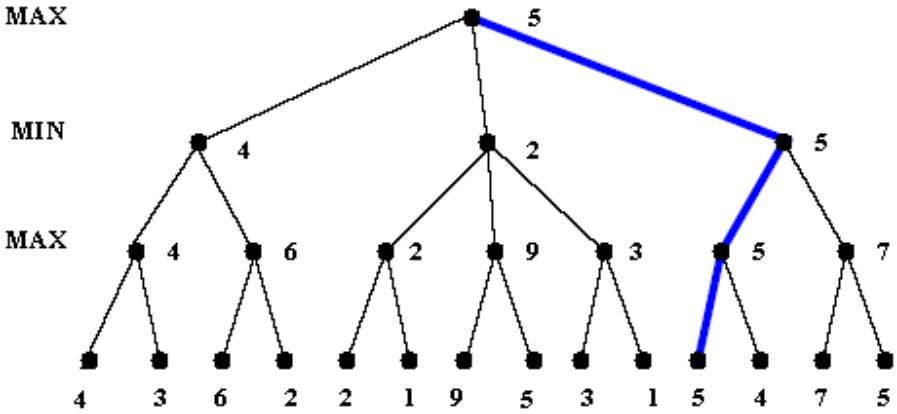

MAX
MIN
MAX
4
4
6
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
4
4
6
2
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
4
4
6
2
9
4 3 6 2 2 1 9 5 3 1 5 4 7 5

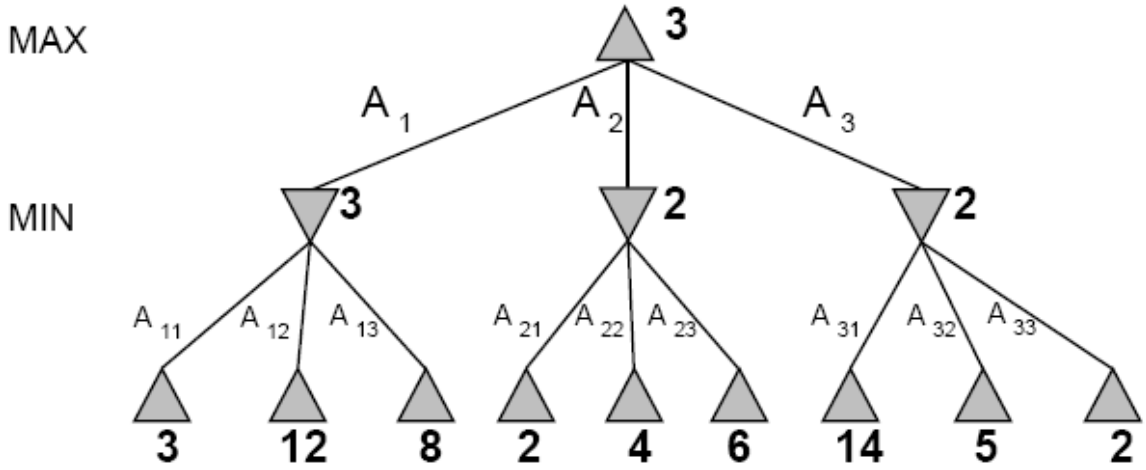


و به همین ترتیب موارد دیگر را هم به دست می آوریم تا در نهایت به شکل زیر برسیم :



پس جواب این مثال عدد ۵ است .

به عنوان مثال دیگر ، برای بازی دو نفره داریم :



که در نتیجه حاصل برابر با ۳ خواهد بود .

الگوریتم مینیماکس

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

ورودی : state ، که حالت جاری بازی می باشد

a ای را کـه در Actions(state) بیشـینه کننـده ی Min-Value(Result(a ,state)) مـی باشـد را برگردان

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

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

$v \leftarrow -\infty$

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

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

خصوصیّات مینیماکس

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

بهینه بودن ؟؟ بله

پیچیدگی زمانی ؟؟ $O(b^m)$

پیچیدگی فضا ؟؟ O(bm) (مثل جستجوی اوّل عمق)

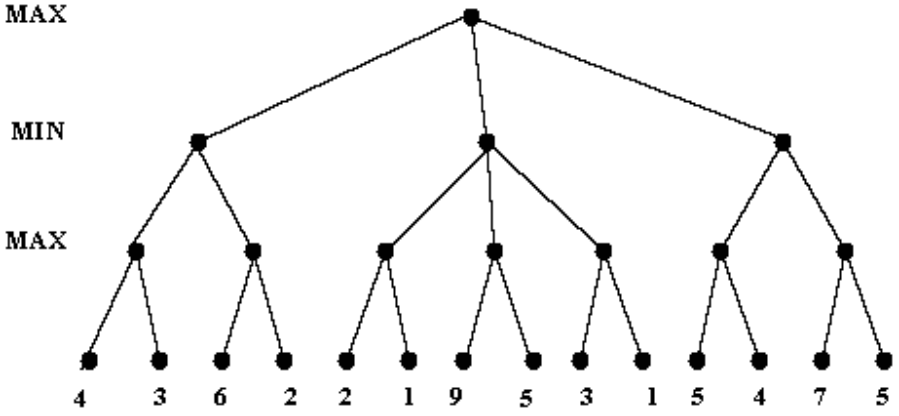
برای شطرنج ، $m \approx 100, b \approx 35$ ، (برای بازی های مستدل) ← راه حل دقیق کاملاً اجرا نشدنی است .

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

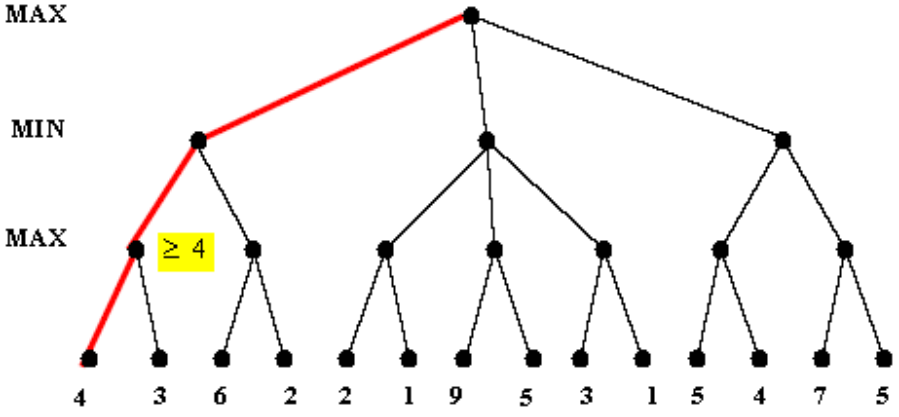
هرس α - β

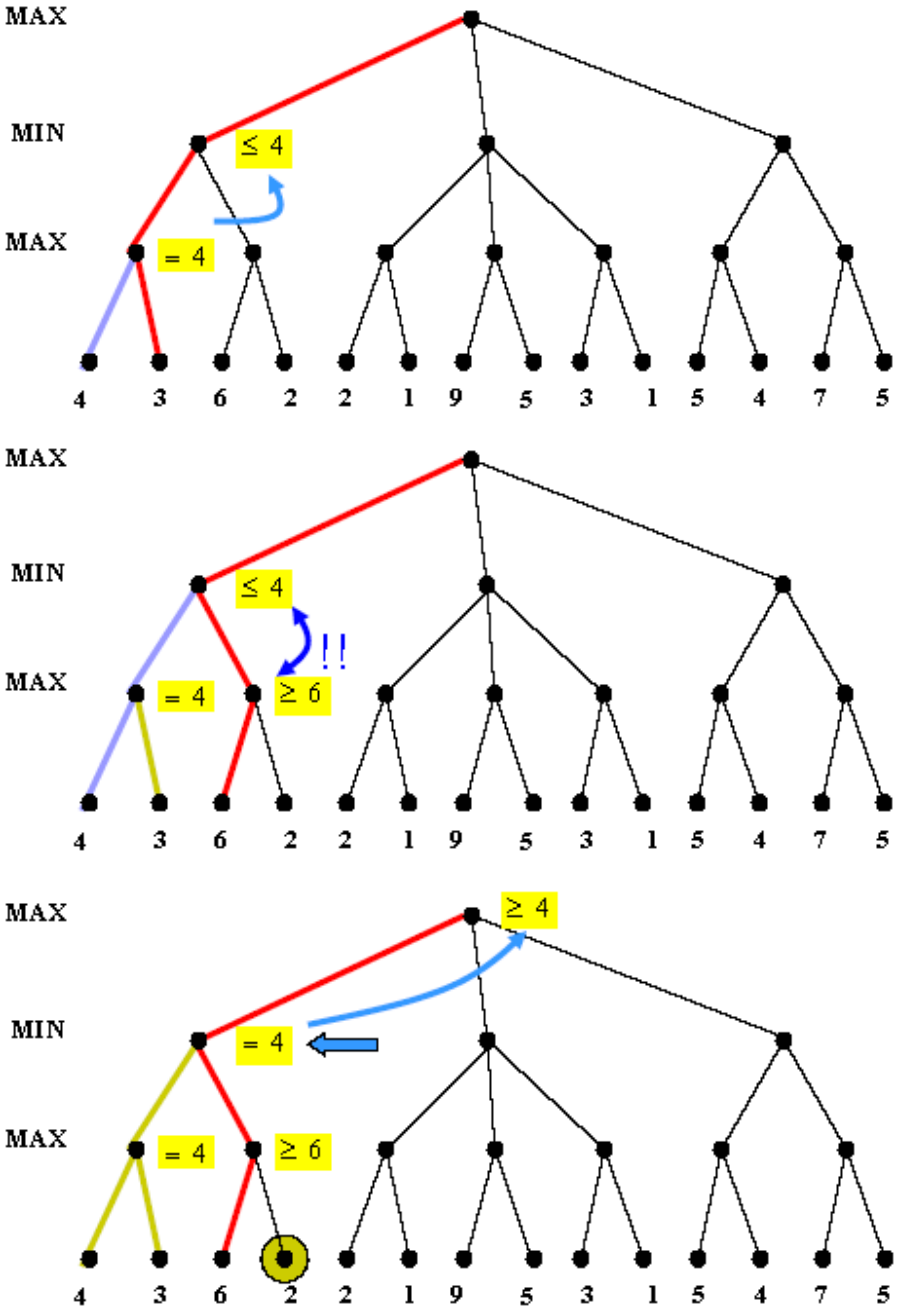
روشی استاندارد برای بازی های قطعی و با اطّلاعات کامل می باشد ؛ در این روش ، شاخه ای که هیچ وقت مورد استفاده قرار نمی گیرد را از درخت جستجو حذف می کنیم . توجّه کنید که این شاخه ها هیچ وقت توسط یک بازیکن عاقل مورد استفاده قرار نمی گیرند ، در ضمن ، توسط بازیکن حریف هم مورد استفاده قرار نمی گیرند .

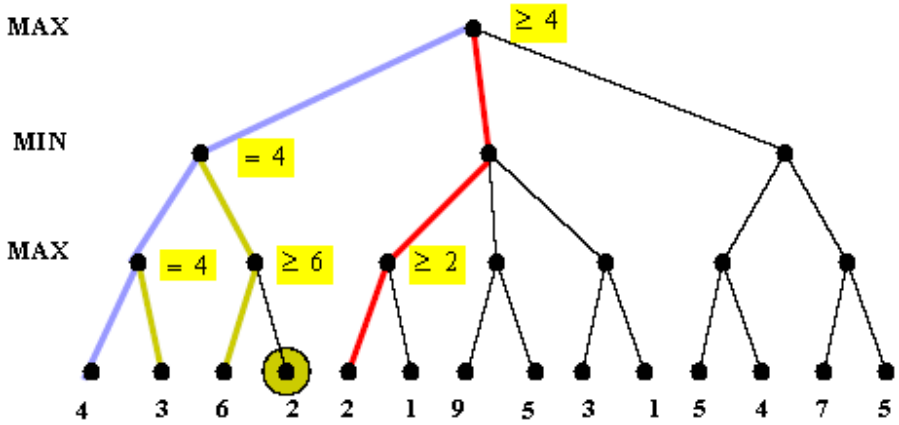
مثال هرس α - β : به درخت زیر توجّه کنید .

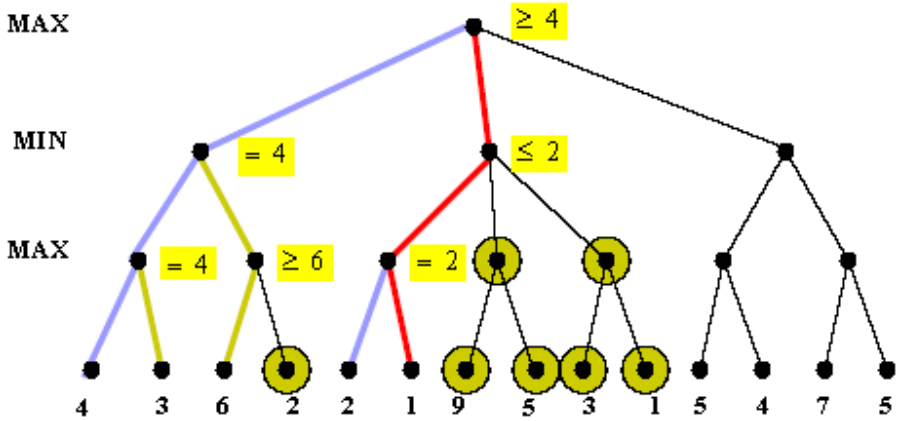


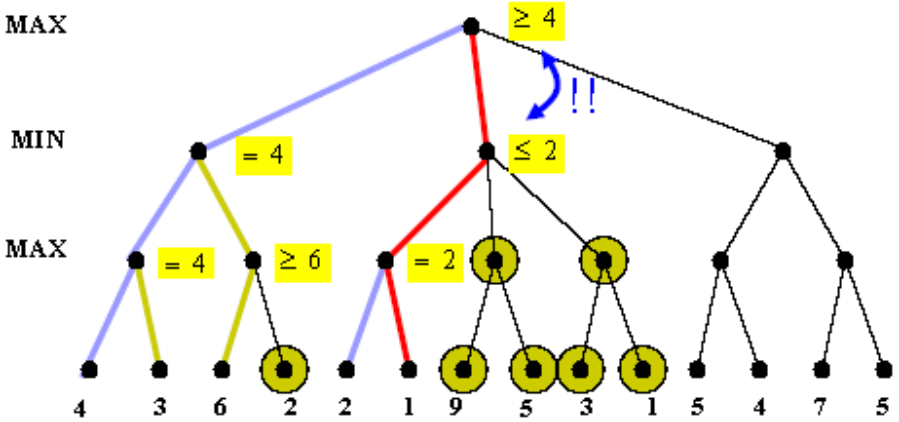
داریم (توجّه کنید که گره های به شکل ⊙ هیچوقت مورد استفاده قرار نمی گیرند) :

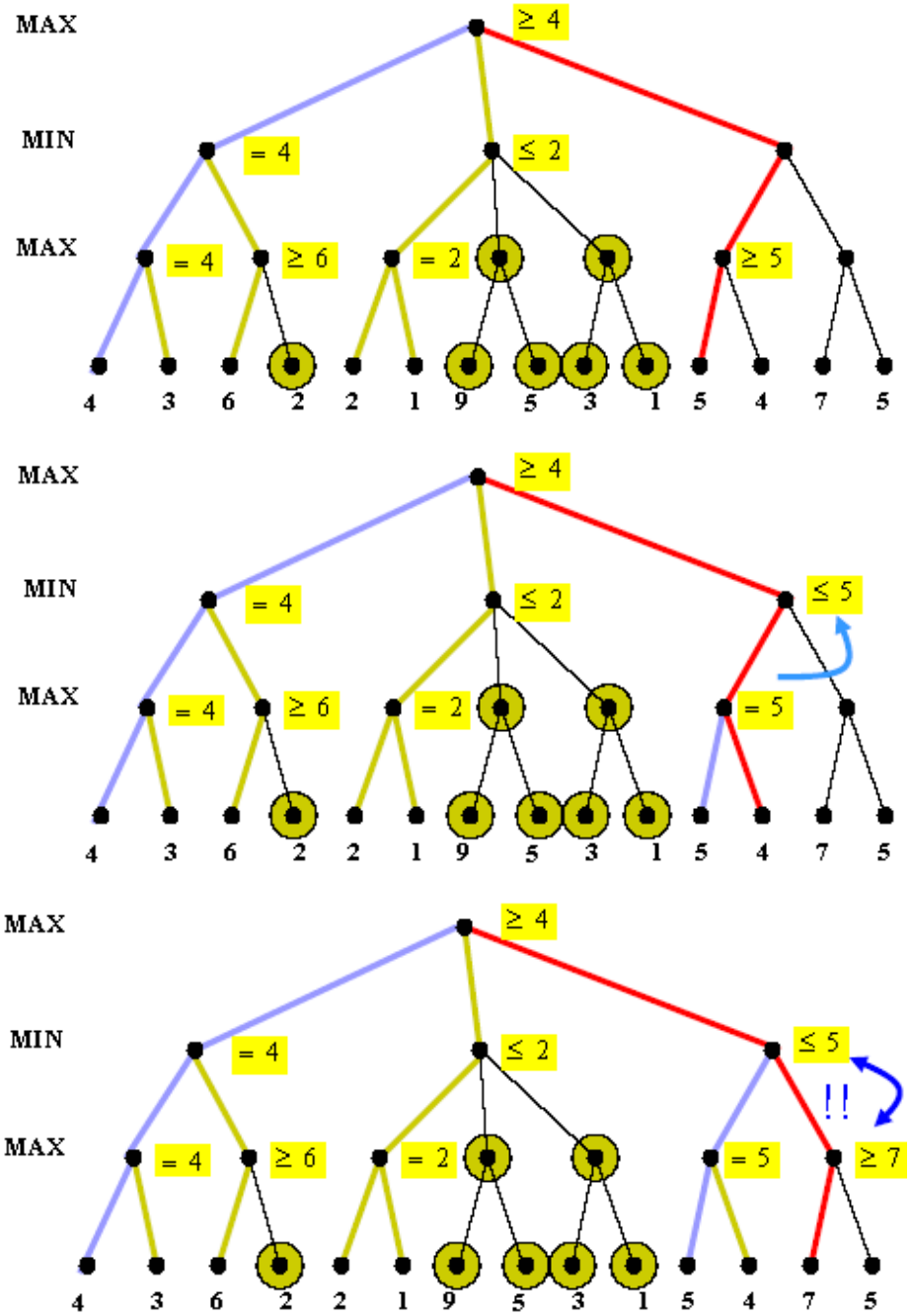


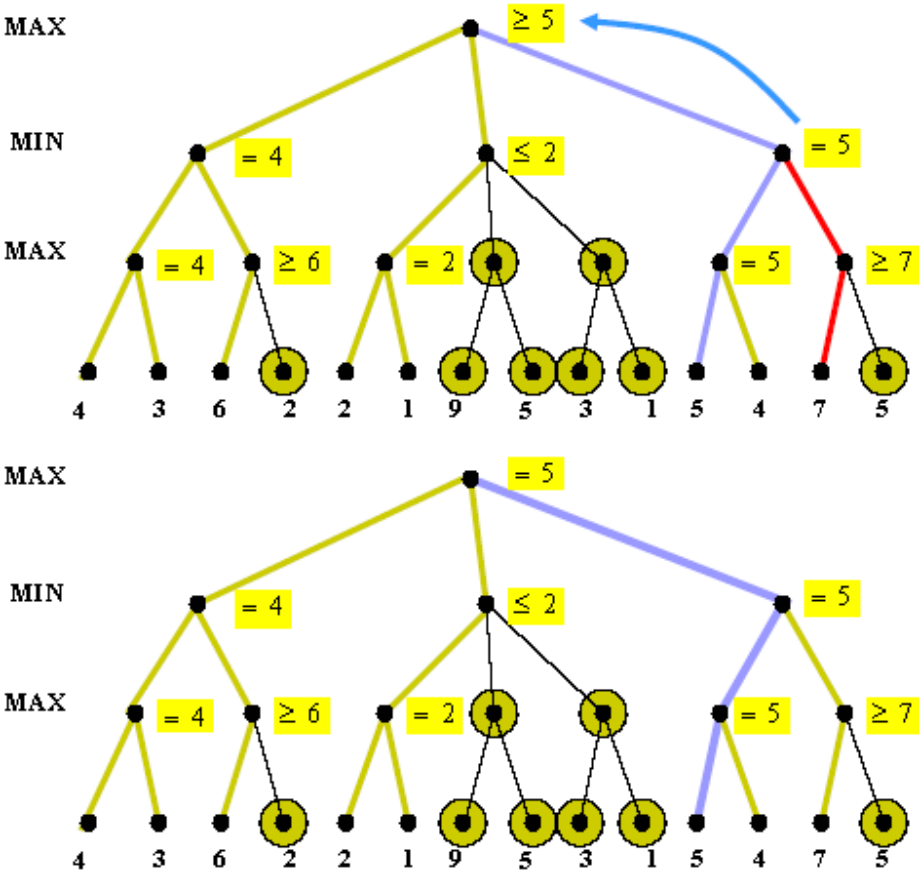

MAX
MIN
MAX
≤ 4
= 4
4 3 6 2 2 1 9 5 3 1 5 4 7 5
MAX
MIN
MAX
≤ 4
!!
= 4
≥ 6
4 3 6 2 2 1 9 5 3 1 5 4 7 5
MAX
MIN
MAX
≥ 4
= 4
= 4
≥ 6
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 4
= 4
= 4
≥ 6
≥ 2
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 4
= 4
≤ 2
= 4
≥ 6
= 2
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 4
!!
= 4
≤ 2
= 4
≥ 6
= 2
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 4
= 4
≤ 2
= 4
≥ 6
= 2
≥ 5
4 3 6 2 2 1 9 5 3 1 5 4 7 5
MAX
MIN
MAX
≥ 4
= 4
≤ 2
≤ 5
= 4
≥ 6
= 2
= 5
4 3 6 2 2 1 9 5 3 1 5 4 7 5
MAX
MIN
MAX
≥ 4
= 4
≤ 2
≤ 5
!!
= 4
≥ 6
= 2
= 5
≥ 7
4 3 6 2 2 1 9 5 3 1 5 4 7 5



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

$V \leftarrow -\infty$

برای a,s در تابع Successors(state) کارهای زیر را انجام بده

$v \leftarrow Max(v,Min\text{-}Value(s,\alpha,\beta))$

در صورتی که $v \geq \beta$ باشد ، v را برگردان

$\alpha \leftarrow Max(\alpha,v)$

انتهای حلقه

v را برگردان

تابع Min-Value(state, α,β) یک مقدار مجاز را بر می گرداند

شبیه مقدار Max-Value ، امّا با قوانین رزرو شده ی α و β

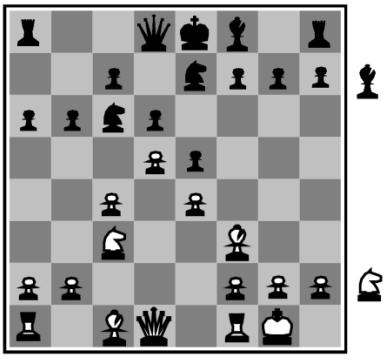
خصوصیّات α-β – بازبینی یا هرس بر روی نتیجه ی نهایی اثری ندارد . حرکت خوب و منظم اثر هرس را بهبود می بخشد . با " منظّم کننده ی کامل " ، پیچیدگی زمانی برابر با $O(b^{m/2})$ است ، درنتیجه عمق جستجو را مضاعف می کند . بنابراین ، به سادگی می تواند به عمق هشت برسد و بازی شطرنج خوبی را ارایه نماید . متأسّفانه 35^{50} هنوز غیر ممکن است !

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

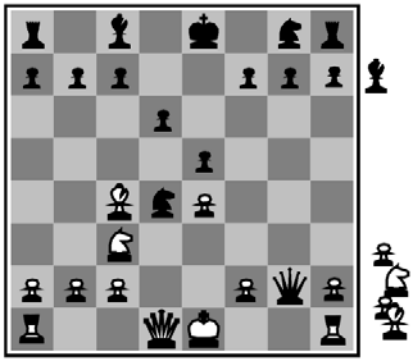
محدودیّت منابع – فرض کنید ما صد ثانیه زمان داریم و می توانیم 10^4 گره در ثانیه را ملاقات کنیم درنتیجه در صد ثانیه 10^6 گره را می توانیم ملاقات نماییم .

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

$w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$ ، که s وضعیّت صفحه ی بازی می باشد و 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) . توجّه کنید که برنامه های پیشرفته از ترکیبات غیرخطّی هم استفاده می نمایند ، در ضمن ، ترکیبات و توزیع ها جزیی از قوانین شطرنج نمی باشند .



با حرکت سیاه ، وضعیّت سفید بهتر می شود



با حرکت سفید ، سیاه برنده می شود

بازی های قطعی در عمل [۱]

– بازی چکرز : چینوک [۲] به چهل سال قهرمانی قهرمان جهان ماریون تینزلی [۳] در سال ۱۹۹۴ خاتمه داد. از پایگاه داده ی مشخص کننده ی پایان بازی شطرنج برای تعریف تمام بازی و برای وضعیّت های شامل هشت یا کم تر مهره در صفحه استفاده شد ، یک مجموعه از ۲۴۷و۴۰۱و۷۴۸و۴۴۳ حالت . Deep Blue قهرمان شطرنج جهان ، گری کاسپاروف [۴] 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 [۵] : قهرمانانی که خیلی خوب هستند نمی پذیرند که با کامپیوتر رقابت نمایند .

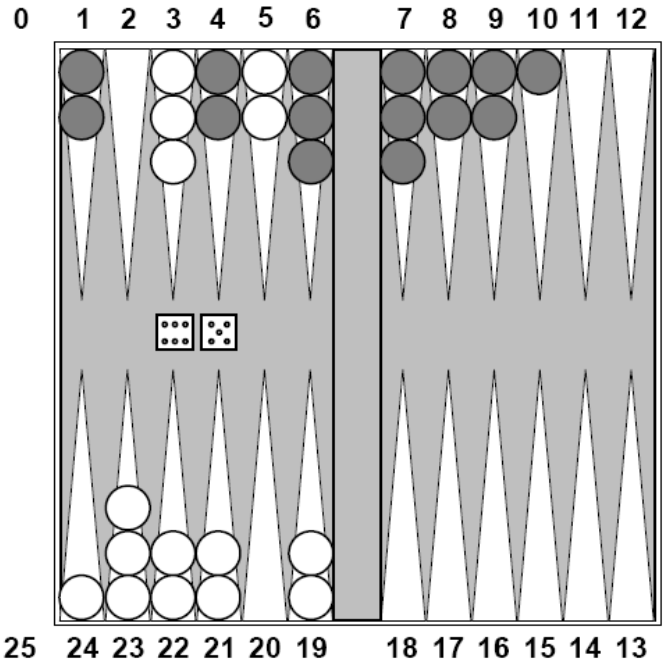
Go : قهرمانان نمی پذیرند که با کامپیوتر رقابت کنند . در این بازی ، b>300 ، بنابراین بیش تر برنامه ها از الگوی براساس دانش برای پیشنهاد حرکت های ممکن استفاده می نمایند .

بازی های غیرقطعی در حالت کلّی

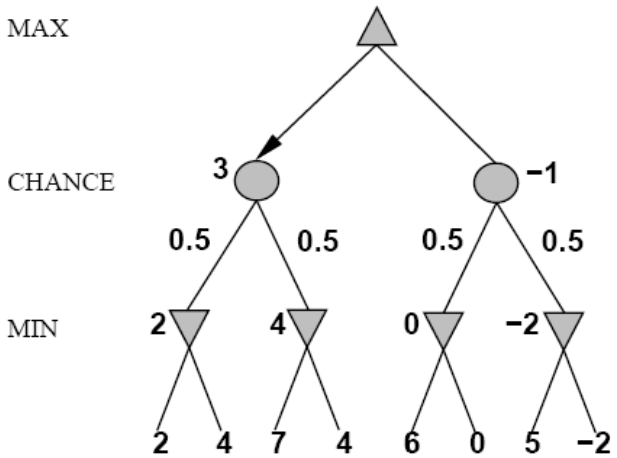
در بازی های غیر قطعی مثل تخته نرد که تصویری از صفحه ی این بازی را در شکل زیر مشاهده می کنید ، شانس به وسیله ی تاس [۶] و برهم زدن کارت [۱] معرّفی می شود.

---

[۱] deterministic games in practice
[۲] Chinook
[۳] Marion Tinsley
[۴] Gary Kasparov
[۵] Othello
[۶] dice



مثال ساده شده با سکه ی دورویه [۲] :



---

[۱] card-shuffling

[۲] coin-flipping

الگوریتمی برای بازی های غیرقطعی

از آمار و احتمالات داریم ؛ مقدار مورد انتظار ار یک متغیّر 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 با Minimax در این است که از مقدارهای مورد انتظار در گره های تصادفی هم استفاده می نماید . EXpectminimax بازی کامل را ارایه می نماید ؛ مینیماکس هم بازی کامل را ارایه می کند در ضمن ما باید از گره های تصادفی هم 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

...

توضیحی در مورد Deep Blue

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

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 آلفا – بتا استفاده می نمود و دارای سخت افزار مخصوص محاسبه کننده ی توابع ارزیابی بود . تعداد ارزیابی ها ، دویست میلیون حالت در ثانیه بود که ۱۰۰ – ۲۰۰ بیلیون وضعیّت را در سه دقیقه ، ارزیابی می کند . در ضمن میانگین فاکتور انشعاب در شطرنج ۳۵ می باشد و فضای حالت در آن 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 و از پایگاه داده های استادان بزرگ شطرنج در بازی های قبلی و همچنین از تعداد زیادی کتاب های پیشرفته استفاده شده بود .

فصل نهم

عامل های منطقی [1]

[1] Logical Agents

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

ریوس مطالب

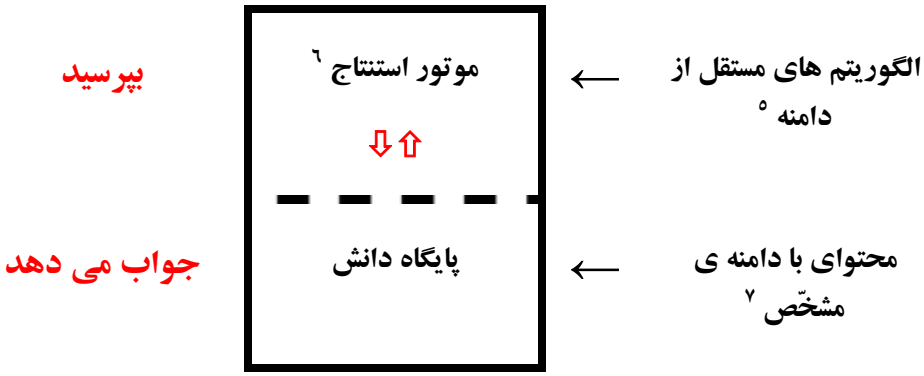
- عامل ها براساس دانش [۱]
- دنیای Wumpus
- منطق در حالت کلّی – مدل ها و دارایی ها [۲]
- منطق گزاره ای [۳](بولین)
- تساوی [٤]، صحت [٥]، توانایی راضی کردن [٦]
- قوانین استنتاج [۷] و اثبات قضیه [۸]
  - زنجیره ی مستقیم [۹]
  - زنجیره ی معکوس [۱۰]
  - تحلیل [۱]

---

[۱] Knowledge-based agents
[۲] entailment
[۳] Propositional
[۴] Equivalence
[۵] validity
[۶] satisfiability
[۷] Inference rules
[۸] theorem proving
[۹] forward chaining
[۱۰] backward chaining

پایگاه های دانش

یک پایگاه دانش [۲] ، تشکیل شده از جملاتی که واقعیّاتی را در مورد جهان [خود] ارایه می کنند . به بیان دیگر ، پایگاه دانش ، مجموعه ای از عبارت ها است که به وسیله ی یک زبان رسمی [۳] بیان شده اند . حال این سؤال مطرح می شود که تفاوت میان پایگاه دانش و پایگاه داده [۴] چیست ؟ : به طور کلّی ، در طرز بیان و کاربرد آن هاست و در عمل ، یک پایگاه دانش ممکن است از یک پایگاه داده استفاده کند . جملات ، در مورد چیزهایی که در جهان وجود دارد مثل Wumpus ها ، طلا ، چاله ها ، فضاها و ارتباط میان عامل ها توضیح می دهند .



---

[۱] resolution
[۲] Knowledge Base
[۳] formal
[۴] Database
[۵] domain-independent algorithms
[۶] inference engine
[۷] domain-specific content

قانون استنتاج : وقتی کسی از پایگاه دانش می پرسد ، جواب باید از آن چه که قبلا به پایگاه دانش گفته ایم بیاید .

شیوه ی [۱] اعلانی یا اظهاری برای ساختن یک عامل (یا یک سیستم دیگر) :

به پایگاه دانش **بگویید ( )** [۲] که می خواهید چه چیزی را بدانید . سپس از خودش **سؤال می کند ( )** [۳] که چه کاری باید انجام دهد ؛ جواب هایی که می دهد باید از KB یا پایگاه دانش گرفته شوند . عامل ها می توانند در **سطح دانش** [٤] 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** [٥] به ساختمان های داده ای [٦] 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

متغیّر KB که یک متغیّر static است ، یک پایگاه دانش می باشد

t ، یک شمارنده با مقدار اوّلیه ی صفر و نشان دهنده ی زمان می باشد

---

[۱] approach
[۲] Tell ( )
[۳] Ask ( )
[۴] knowledge level
[۵] implementation level
[۶] data structures

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 را ارایه نماید ؛ دریافت های ادراکی [۱] جدید را یکی کند [۲] ؛ ارایه های درونی جهان را به روز نماید ؛ خصوصیات پنهان جهان خود را استنباط [۳] نماید و عملکردهای مناسب را استنباط نماید .

توضیح : عامل پایگاه دانش ، شبیه عامل های با وضعیّت درونی می باشد . عامل ها می توانند در سطح های مختلف ، بیان شوند :

۱- **سطح دانش :** چه می دانند ، بدون توجه به پیاده سازی واقعی .

۲- **سطح پیاده سازی :** ساختمان های داده در پایگاه دانش و الگوریتم هایی که آن ها به کار می گیرند ؛ به عنوان مثال ، منطق گزاره ای و نتیجه [۴] .

انواع دانش

---

[۱] percepts

[۲] Incorporate

[۳] deduce

[۴] resolution

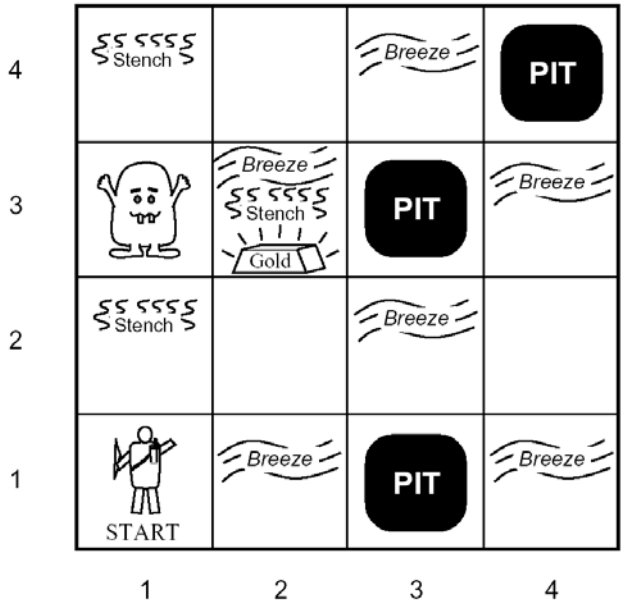
مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

۱- **رویه ای :** به عنوان مثال ، تابع ها که مثل دانش می توانند فقط به یک روش و آن هم در زمان اجرا مورد استفاده قرار گیرند .

۲- **بیانی :** محدودیّت ها و قانون ها که می توانند برای انجام انواع زیادی از استنباط ها مورد استفاده قرار گیرند .

دنیای Wumpus

Wumpus ، نام یک بازی کامپیوتری است ؛ در این بازی ، عامل یک غار را که از اتاق های متصل شده توسط راهرو تشکیل شده کشف می کند . *کمین گاه ، جایی است که در آن Wumpus قرار دارد و Wumpus ، جانوری است که هر عاملی را که به اتاقش وارد شود می خورد . همچنین ، برخی از اتاق ها دارای چاله های بدون کف هستند که هر عاملی را که در اتاق سرگردان است را به تله می اندازند . گاهی از وقت ها کپه ای از طلا هم در اتاق هست .* ***هدف ما در بازی این است که طلا ها را جمع کنیم و بدون اینکه خورده شویم خارج شویم .***



توضیح PEAS دنیای wumpus

اندازه گیری عملکرد یا معیارکارایی : طلا (۱۰۰۰+) ، مرگ (۱۰۰۰-) ، (منفی یک) در هر گام ، (۱۰-) برای استفاده از پیکان [۱]

محیط : مربّع های نزدیک به wumpus بدبو [۲] هستند ؛ مربّع های نزدیک به چاله [۳] خوش هوا [۴] هستند و اگر طلا در همان خانه ی خوش هوا باشد ، درخشان تر است ؛ اگر به دیوار بخورید ، ضربه خواهید خورد ؛ اگر Wumpus ، کشته شد بدبخت می شوید ؛ تیراندازی [۵] ، wumpus را در صورتی که شما روبه روی آن باشید می کشد ؛ تیراندازی فقط پیکان را مصرف [۶] می نماید ؛ قلاب [۷] ، فقط طلا را در صورتی که در همان مربع باشد انتخاب می نماید . آزاد کردن ، طلا را در همان خانه باقی می گذارد .

حسگرها : بوی بد [۸] ، بوی خوب [۹] ، درخشش [۱۰] ، ضربه [۱۱] و جیغ زدن [۱۲]

محرّک ها : چپ گرد [۱] ، راست گرد [۲] ، حرکت مستقیم [۳] ، قلّاب [۴] ، آزاد کردن [۵] ، تیراندازی [۶]

---

[۱] arrow
[۲] smelly
[۳] pit
[۴] breezy
[۵] glitter
[۶] use up
[۷] grabbing
[۸] stench
[۹] breeze
[۱۰] glitter
[۱۱] bump
[۱۲] scream

دنیای Wumpus معمولی

عامل ، معمولاً از خانه ی [۱و۱] شروع می کند . کار عامل این است که طلا را پیدا کند ، بـه خانـه ی [۱و۱] برگردد و از غار خارج شود .

خصوصیات دنیای wumpus

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

قطعیّت ؟؟ بله – نتایج دقیقا مشخص است .

دوره ای نه – دارای ترتیب در سطح عملکردها است .

ایستایی ؟؟ بله – wumpus و چاله ها نمی توانند حرکت نمایند .

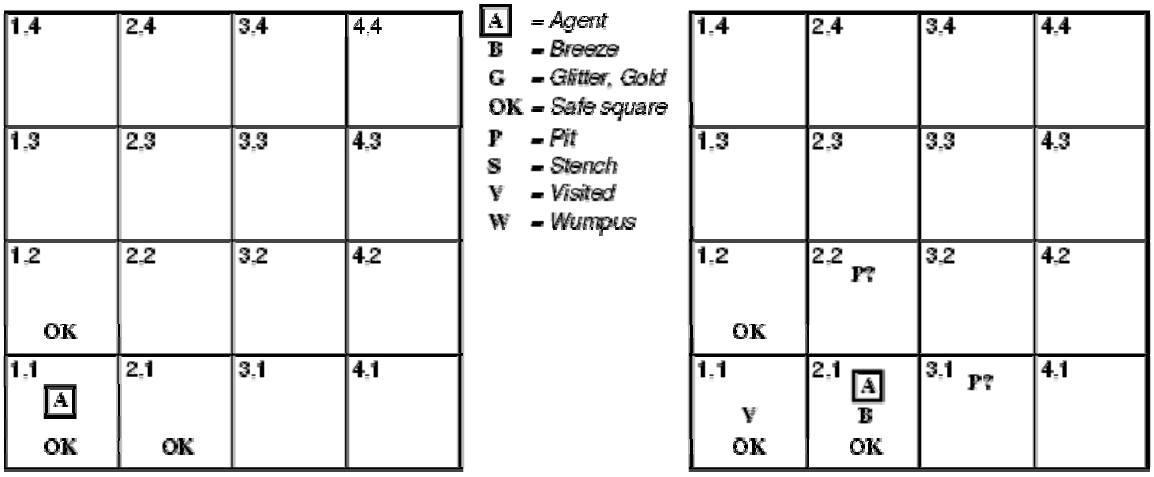
گسسته؟؟ بله .

تک عاملی ؟؟ بله wumpus در اصل یک طرح طبیعی می باشد .

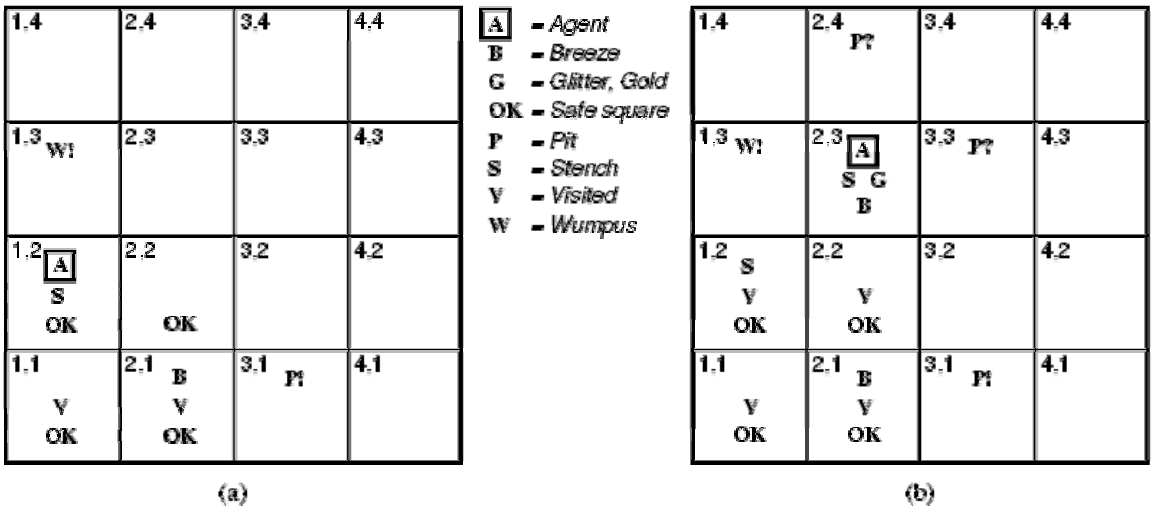
حرکت عامل در دنیای Wumpus

---

[۱] Left turn
[۲] Right turn
[۳] Forward
[۴] Grab
[۵] Release
[۶] Shoot

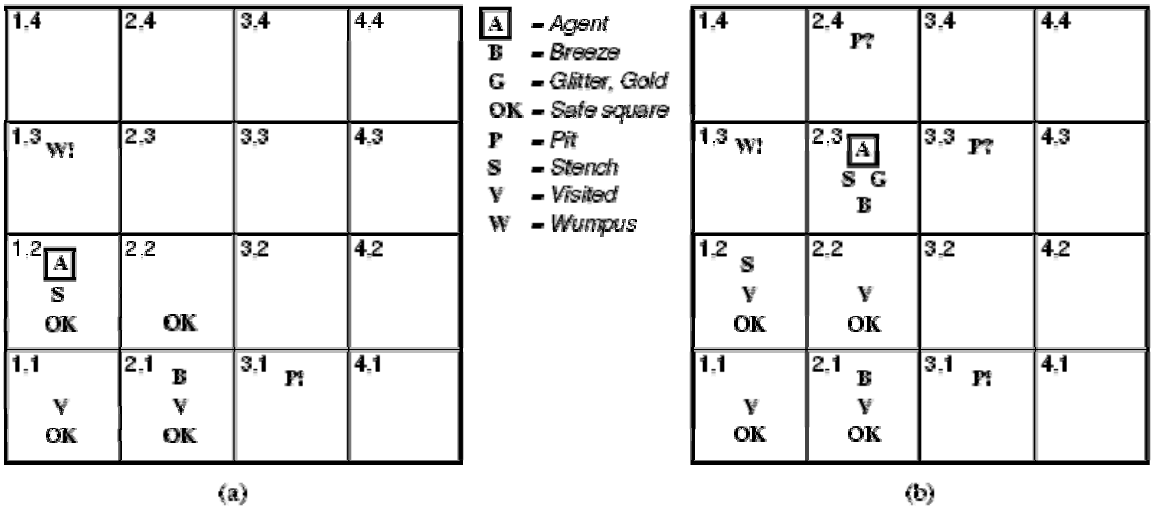


در خانه ی [۱و۱] ، پایگاه دانش دارای قانون های محیط است . اوّلین دریافت یا ادراک [هیچ و هیچ و هیچ و هیچ] است ، به یک خانه ی امن [1] مثل [۱و۲] می رویم . خانه ی [۱و۲] چون خوش بو است نتیجه می گیریم که چاله در [۲و۲] یا [۱و۳] است ؛ در نتیجه به خانه ی [۱و۱] برمی گردیم و خانه ی امن بعدی را امتحان می نماییم .



[1] safe square

در خانه ی [۲و۱] ، از بوی تعفن خانه می فهمیم کـه Wumpus در خانـه ی [۳و۱] یـا [۲و۲] اسـت . Wumpus در خانه ی [۱و۱] یا [۲و۲] یا [۱و۲] هم نیست . در نتیجه Wumpus در خانه ی [۳و۱] اسـت . بنابراین ، خانه ی [۲و۲] ، امن است چون بوی خـوش در خانـه ی [۲و۱] نیسـت ؛ بنـابراین ، چالـه در خانـه ی [۱و۳] می باشد . بنابراین ، به خانه ی [۲و۲] می رویم .



(a) (b)

از خانه ی [۲و۲] به خانه ی [۳و۲] می رویم . در خانه ی [۳و۲] درخشش طلا ، بوی بد و بوی خوش را داریم ، بنابراین طلا را بر می داریم و از بوی خوش موجود در این خانه نتیجه می گیریم که چاله در خانه ی [۳و۳] یا [۴و۲] می باشد . سپس از مسیر امنی که قبلا آمده ایم برمی گردیم و در نهایت ، بازی خاتمه می یابد .

منطق چیست ؟

منطق ، روشی کلاسیک برای ارایه ی دانش عامل می باشد و به ما اجازه می دهد که عامل ها را به صورت اعلانی (اظهاری) ، برنامه ریزی و توصیف کنیم . به بیان دیگر ، منطق ، یک زبان رسمی است که

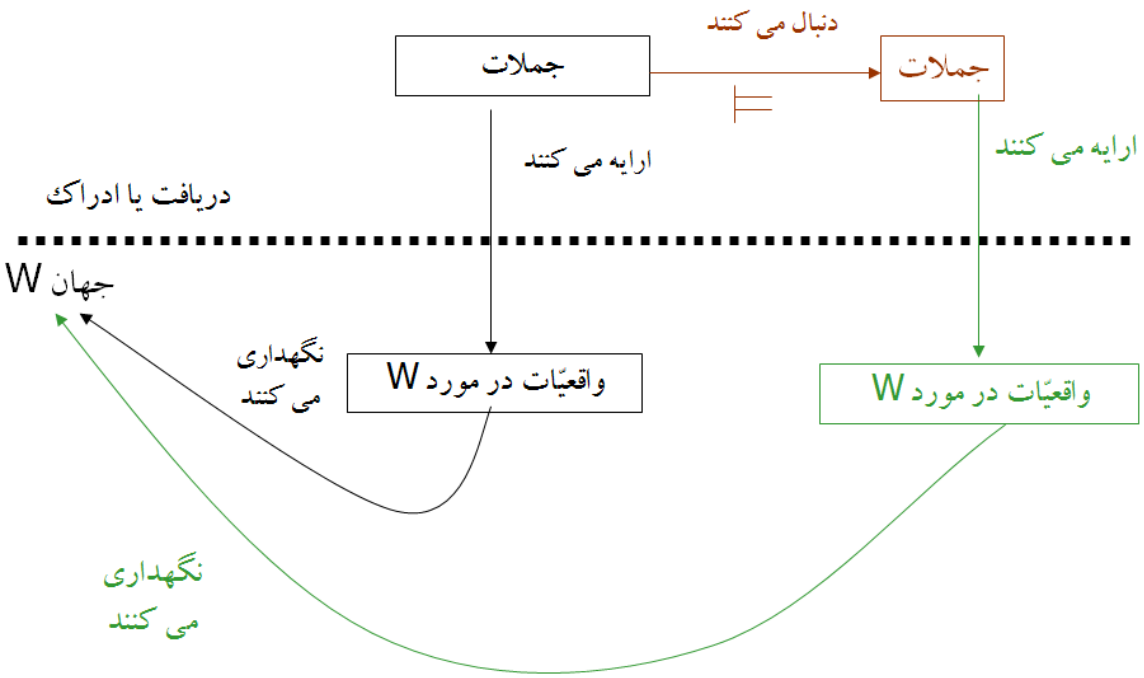
دارای ظاهر [۱] و سمانتیک [۲] می باشد ، ظاهر (نحو یا گرامر) ، می گوید که چه عبارت هایی مجازند مثلا در ریاضیّات ، عبارت $x+2=5$ به صورت گرامری درست می باشد ، امّا $x+=3$ به صورت گرامری درست نمی باشد . به عنوان مثال دیگر ، در زبان ریاضی ، $x+2\geq y$ یک جمله است ولی $x2+y>$ یک جمله نمی باشد . و معنا یا سمانتیک ، مشخص می نماید که یک عبارت در چه موقعی درست یا در چه موقعی ، غلط می باشد ؛ معنای $x+2=5$ در زمانی درست است که $x=3$ باشد و در غیر این صورت غلط می باشد . به عنوان مثال دیگر ، $x+2\geq y$ 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 . *توجه کنید که ، عبارت های منطقی ، باید یا درست باشند و یا غلط باشند ؛ به بیان دیگر ، "درجه ی درستی ، وجود ندارد"* .

ارتباطات موجود در جهان

---

[۱] Syntax

[۲] Semantics

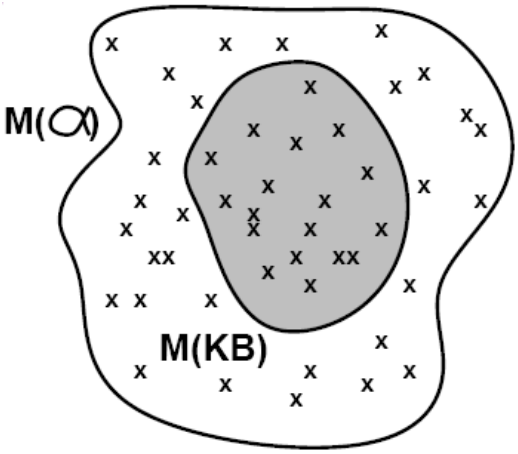


دنبال کردن [1] – معنای دنبال کردن این است که چیزی توسط دیگری دنبال شود : KB ⊨α . **پایگاه دانش KB جمله ی α را دنبال می کند اگر و فقط اگر α در همه ی جهان ها در جایی که KB درست است ، درست باشد .** به عنوان مثال ، x+y=4 ، 4=x+y را دنبال می نماید . دنبال کردن ، ارتباطی میان جملات است که براساس **سمانتیک 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 " . به صورت تکنیکی (مطابق اصول فنی) ، دنبال کردن ، می گوید که : برای همه ی مدل هایی که a درست است ، b هم درست می باشد ؛ مثلا ، a+2=5 |= a=3 . دنبال کردن به ما اجازه خواهد داد که استدلال نماییم و واقعیّات جدیدی را به پایگاه دانش عامل مان اضافه نماییم .

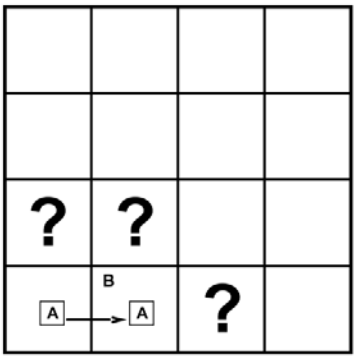
---

[1] Entailment : استلزام یا ایجاب کردن

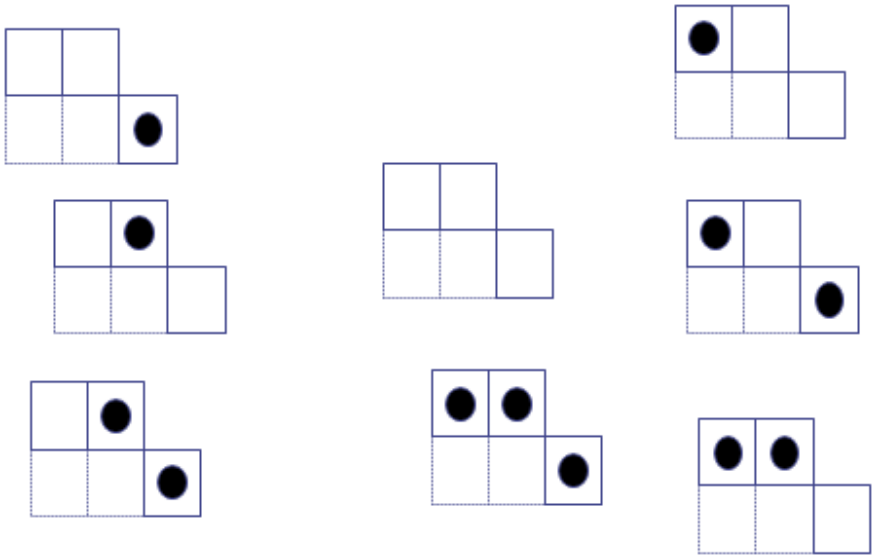
مدل ها – مدل ها ، تعریف های رسمی وضعیّت های ممکن جهان هستند . گوییم **m مدلی از** یک جمله ی **α** است در صورتی که **α** در **m** درست باشد . اگر $M(\alpha)$ مجموعه ای از تمام مدل های α باشد در این صورت $KB \models \alpha$ اگر و فقط اگر $M(KB)$ زیر مجموعه یا مساوی $M(\alpha)$ باشد .



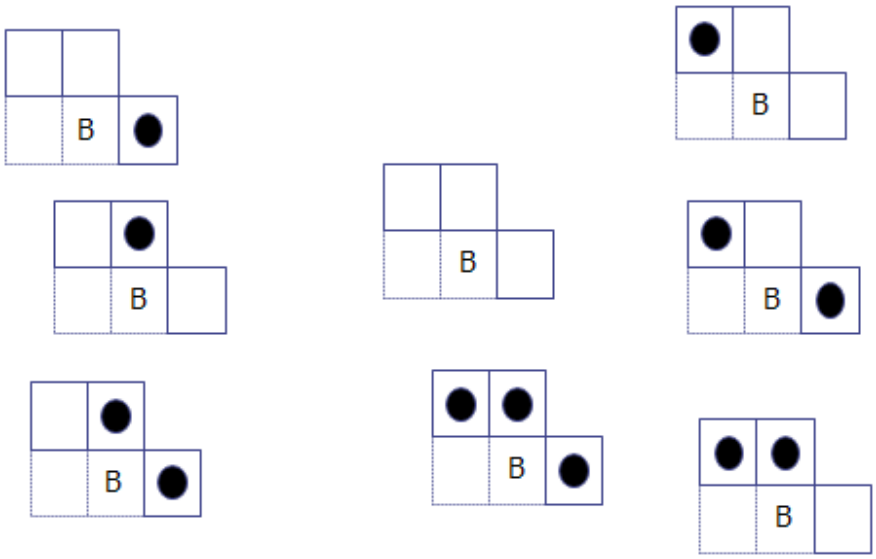
دنبال کردن در دنیای wumpus – بعد از اینکه در خانه ی [۱و۱] هیچ چیزی را پیدا نکردیم حرکت بعدی ما ، حرکت به راست در خانه ی [۱و۲] خوش هواست . مدل های ممکن برای ؟ برای احتمال وجود چاله یا عدم وجود چاله چیستند ؟ . سه انتخاب بولین وجود دارد و در نتیجه ۸ مدل ممکن است .



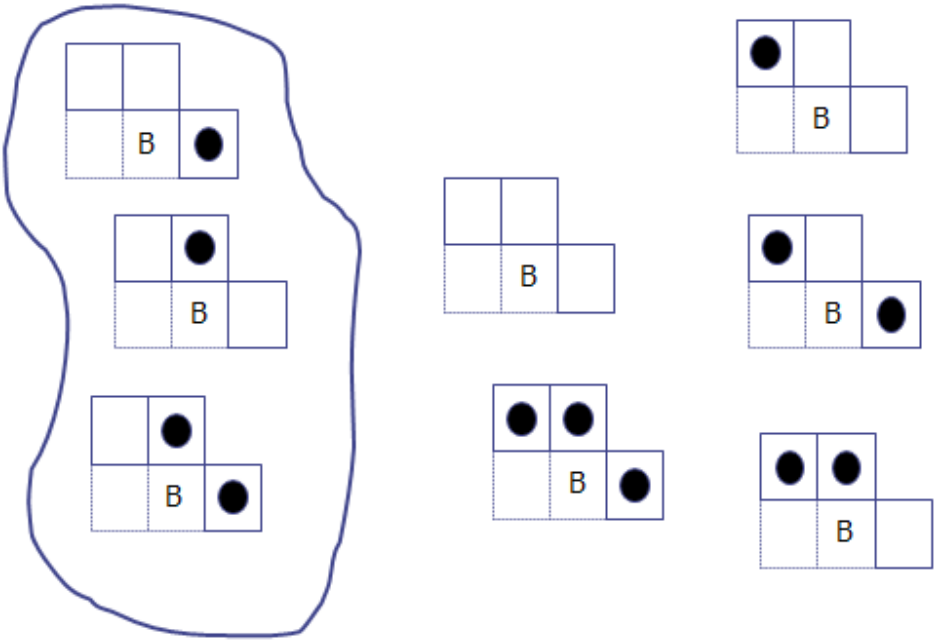
مدل های Wumpus



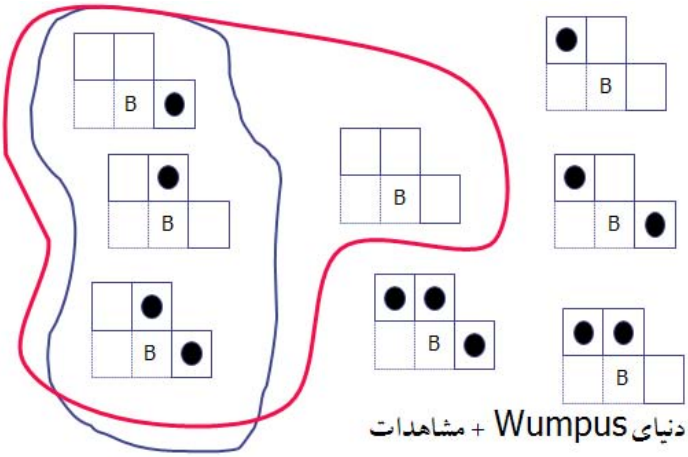
مدل های Wumpus



مدل های Wumpus



مدل های Wumpus

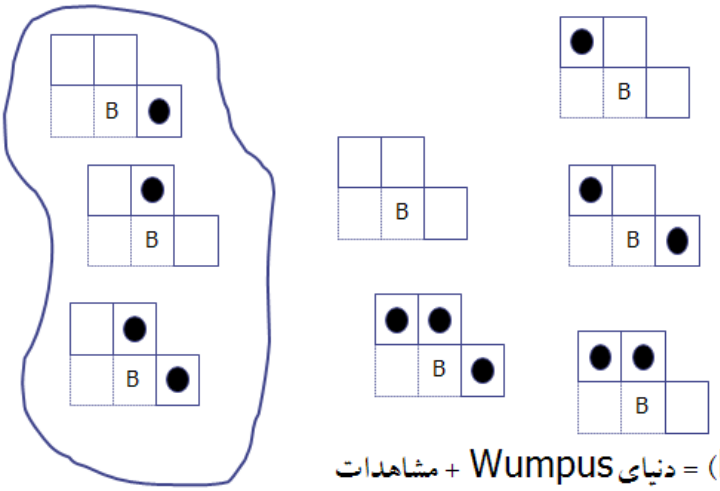


پایگاه دانش (KB) = دنیای Wumpus + مشاهدات

α_1 = "ایمن است [۲و۱]"

KB |= α_1

مدل های Wumpus

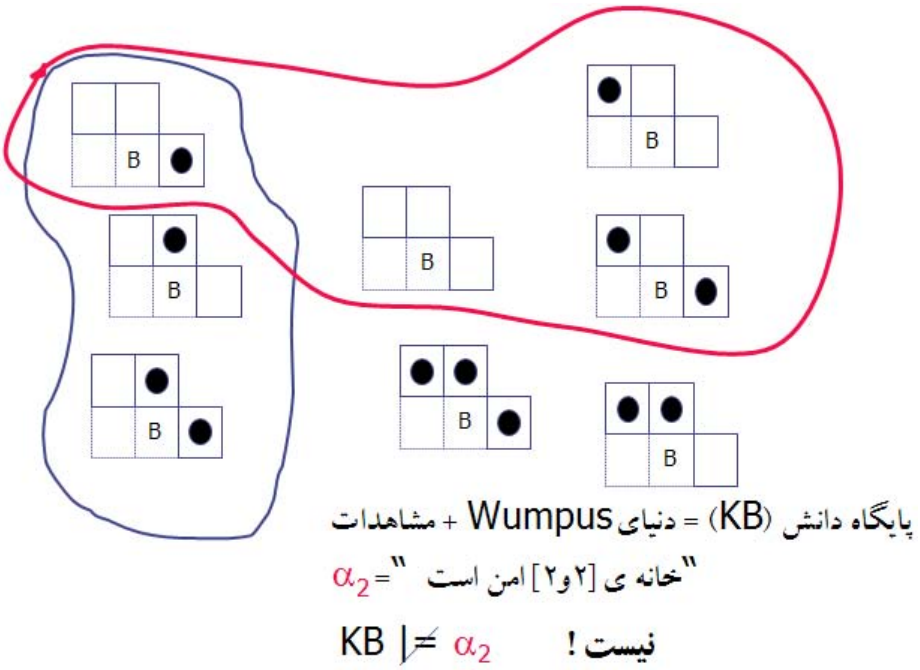


پایگاه دانش (KB) = دنیای Wumpus + مشاهدات

"خانه ی [۲و۲] امن است" = α_2

KB |= α_2 ??

مدل های Wumpus



استنتاج منطقی

نظریّه ی دنبال کردن ، که قبلا آن را بررسی کردیم ، می تواند برای استنتاج منطقی مورد استفاده قرار گیرد . بررسی مدل [۱] (مثال Wumpus را ببینید) همه ی مدل های ممکن را بررسی می کند و چک می کند که آیا α درست است .

در صورتی که یک الگوریتم فقط جمله های دنبال شده را به وجود می آورد ، ***نگهدارنده ی درستی*** [۲] **یا کامل** [۱] نام دارد ، در غیر این صورت فقط چیزها را به وجود می آورد . *i در صورتی کامل است که در جایی که KB |-i α ، این هم درست باشد که KB|= α .*

---

[۱] model checking

[۲] truth preserving

کمال یا تمامیّت [۲] **:** الگوریتم می تواند هر عبارتی که دنبال می شود را به وجود آورد . i در صورتی دارای تمامیّت یا کمال است که در هر جایی که $KB \models \alpha$ ، این هم درست باشد که $KB \vdash_i \alpha$.

استنتاج [۳]

$KB \vdash_i \alpha$ ؛ یعنی ، جمله ی α می تواند از KB توسط روال [۴] i مشتق شود (به وجود آید) .

کامل بودن [۵] **:** *i در صورتی کامل است که در جایی که $KB \vdash_i \alpha$ ، این هم درست باشد که $KB \models \alpha$.*

کمال یا تمامیّت [۶] ***:*** *i در صورتی دارای تمامیّت یا کمال است که در هر جایی که $KB \models \alpha$ ، این هم درست باشد که $KB \vdash_i \alpha$.*

ما می خواهیم منطقی را که به اندازه ی کافی رسا می باشد برای گفتن تقریباً 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 .

مثال هایی از منطق

منطق گزاره ای : $A \wedge B \Rightarrow C$

---

[۱] sound
[۲] completeness
[۳] inference
[۴] procedure
[۵] soundness
[۶] completeness

منطق مرتبه ی اوّل : $(\forall x)(\exists y)$ Mother(y,x)

منطق عقیده ای یا زود باور [1] : B(John,Father(Zeus,Cronus))

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 ؛ $\neg S$ ، عبارت نقیض [2] آن می باشد .

برای عبارات S_1 و S_2 ؛

$S_1 \wedge S_2$ ؛ AND (ترکیب فصلی [3] یا جدایی ؛ در صورتی که ورودی (ها) صحیح باشند ، نتیجه درست خواهد بود) این دو خواهد بود .

$S_1 \vee S_2$ ، OR (ترکیب عطفی [4] ؛ در صورتی که از S_1 و S_2 یکی درست باشد ، نتیجه درست خواهد بود) این دو خواهد بود .

$S_1 \Rightarrow S_2$ نتیجه گیری (استلزام یا استنباط [1]) خواهد بود و در صورتی درست است که A درست باشد و B هم درست باشد ؛ مثلا ، اگر الان هوا بارانی باشد ، آن گاه الان هوا ابری می باشد . استلزام به ما امکان استنتاج را می دهد .

---

[1] Logic of Belief
[2] negation
[3] conjunction
[4] disjunction

نکته : $S_1 \Rightarrow S_2$ معادل است با $\neg S_1 \vee S_2$.

$S_1 \Leftrightarrow S_2$ استنباط دو شرطی (اگر و فقط اگر [۲]، S_1 اگر و فقط اگر S_2) خواهد بود . به عنوان مثال ، $(P \wedge Q) \Rightarrow R$ و $(A \Rightarrow B) \vee (\neg C)$.

ترتیب اولویّت ها [۳]

به ترتیب از چپ به راست عبارتند از $\Rightarrow \quad \vee \quad \wedge \quad \neg$ ؛ به عنوان مثال ، $\neg A \vee B \Rightarrow C$ برابر است با $((\neg A) \vee B) \Rightarrow C$.

مدل های موجود در منطق گزاره ای

مدل ، نسبت دادن درست یا غلط به هر عبارت اتمیک (تجزّیه ناپذیر) است ؛ اگر A ، B ، C و D سمبل های گزاره ای باشند ؛ $m = \{A=true, B=false, C=false, D=true\}$ ، یک مدل است ؛ همچنین ، $m' = \{A=true, B=false, C=false\}$ هم یک مدل است . چند مدل می توانند برای n سمبل گزاره ای تعریف شوند ؟ ۲^n

سمانتیک ها یا معناهای منطق گزاره ای ، هر مدل درست [۴] / غلط [۵] را برای هر نشان [۶] مشخص می نماید .

---

[۱] implication
[۲] biconditional
[۳] Order of Precedence
[۴] true
[۵] false
[۶] symbol

$P_{۳و۱}$	$P_{۲و۲}$	$P_{۱و۲}$
↓	↓	↓
غلط	درست	درست

با این سه 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 . مثال :

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

جدول صحّت برای رابطه ها

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔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

عبارت **A ⇒ B** را به این صورت بخوانید ؛ *اگر A درست باشد آن گاه B درست خواهد بود* و در غیر این صورت ادعایی ندارم .

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 داشته باشد ، درست می باشد .

$$\neg P_{1,1}$$
$$\neg B_{1,1}$$
$$B_{2,1}$$

" 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 صحّت 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 برای 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 ؛ مسأله 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 :

α≡β اگر و فقط اگر β ⊨ α و α ⊨ β باشد .

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$: جابه جایی پذیری[۱] ∧

$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$: جا به جایی پذیری ∨

$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$: شرکت پذیری[۲] ∧

$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$: شرکت پذیری ∨

$\neg(\neg\alpha) \equiv \alpha$: حذف نقیض دوگانه[۳]

$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$: مفهوم مخالف[۱]

---

[۱] commutativity

[۲] associativity

[۳] double-negation elimination

$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$: حذف استنتاج[۲]

$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$: حذف شرط دو طرفه[۳]

$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$: قانون دمورگان[۴]

$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$: قانون دمورگان

$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$: توزیع پذیری[۵] $\wedge$ روی $\vee$

$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$: توزیع پذیری $\vee$ روی $\wedge$

صحّت و قابلیّت ارضا – یک عبارت در صورتی **درست** است که در **همه ی** مدل ها درست باشد . مثلاً ، $True, A \vee \neg A, A \Rightarrow A, (A \wedge (A \Rightarrow B)) \Rightarrow B$. صحّت مربوط می شود به نتیجه گیری از راه قضیّه ی قیاس[۶] : $KB \models \alpha$ اگر و فقط اگر $(KB \Rightarrow \alpha)$ مجاز باشد . یک عبارت *راضی کننده* است اگر در برخی از مدل ها درست باشد ؛ مثل ، $A \vee B, C$ و یک جمله *ناراضی کننده* است اگر در هیچ مدلی درست نباشد ؛ مثل ، $A \wedge \neg A$

قابلیّت 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[۷] اثبات نمایید .

---

[۱] contraposition

[۲] implication

[۳] biconditional elimination

[۴] De Morgan

[5] distributivity

[۶] Deduction Theorem

[7] reductio ad absurdum

متدهای اثبات [۱]

به طور کلّی متدهای اثبات به دو دسته تقسیم می شوند:

- ✓ **استفاده از قوانین استنتاج :** در این روش ، معمولاً به ترجمه ی عبارات به صورت یک فرم نرمال نیازمندیم و برای اثبات ، یک سری از قوانین استنتاج را به کار می بریم .
- ✓ **بررسی مدل** [۲] : که در این روش ، از جدول صحت استفاده می نماییم .

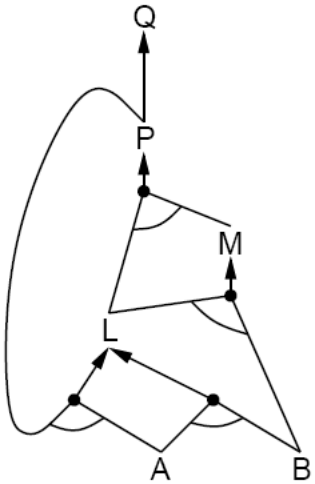
زنجیره ی مستقیم و معکوس

زنجیره ی مستقیم : از قوانین و چیزهای موجود استفاده می کنیم تا چیزهای اضافی را به دست آوریم و اگر به عبارت مورد نظر رسیدیم ، دیگر این کار را ادامه نمی دهیم . این روش ، استنتاج مشتق شده از داده ها [۳] هم نام دارد و برای به دست آوردن واقعیّات یا قوانین جدید ، بسیار مفید می باشد .

$$P \Rightarrow Q, L \wedge M \Rightarrow P, B \wedge L \Rightarrow M, A \wedge P \Rightarrow L, A \wedge B \Rightarrow L, A, B$$

---

[۱] Proof methods
[۲] Model checking
[۳] data-driven reasoning



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 [۱]

q ، صف ، که یک سمبل گزاره ای می باشد

متغیّرهای محلی [۲] : count ، یک جدول شاخص گذاری شده توسط شرط ، به صورت اوّلیه حاوی تعداد permis ها می باشد

---

[۱] یک عبارت شیپوری ، عبارت هایی " یایی " هستند که در آن ها حداکثر یک لفظ مثبت وجود دارد ؛ مثل ، $\neg A \vee \neg B \vee \neg C \vee D$ و $\neg A \vee \neg B$ ؛ عبارت های شیپوری می توانند به صورت استنتاج هایی با یک نتیجه (تالی) نوشته شوند ؛ مثلا ، $\neg A \vee \neg B \vee \neg C \vee D$ می تواند به صورت $A \wedge B \wedge C \Rightarrow D$ نوشته شود . عبارت های شیپوری ، مبنا یا پایه ی برنامه نویسی منطقی هستند .

[۲] local variables

inferred یک جدول شاخص گذاری شده توسط سمبل ، هر ورودی دارای مقدار اوّلیه ی false می باشد

agenda ، لیستی از سمبل ها ، به صورت اوّلیه 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 قضیّه 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

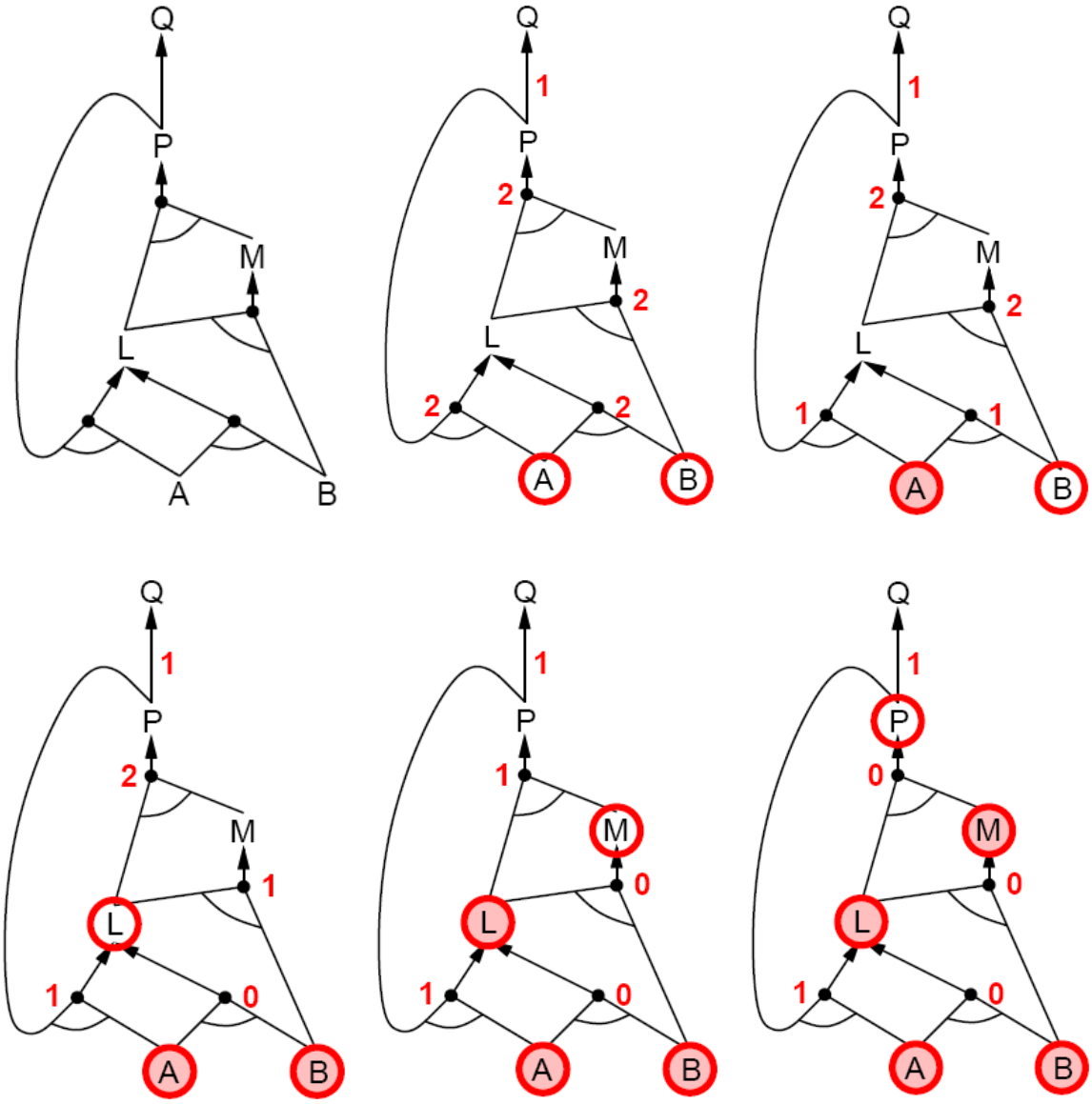
در صورتی که Head[c]=q می باشد true را برگردان

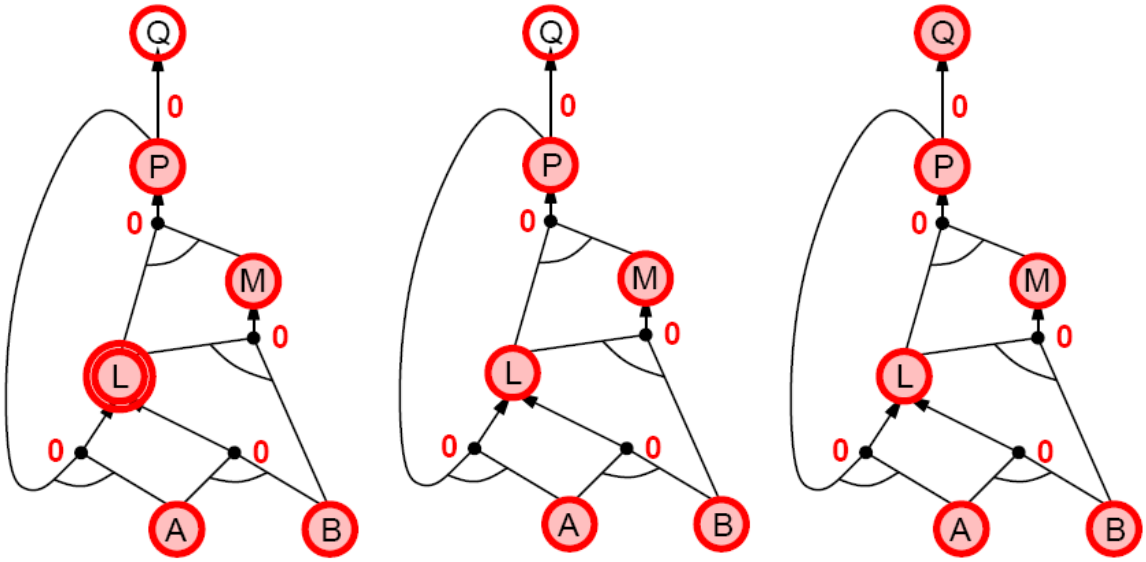
Push(Head[c],agenda)

false را برگردان

مثال زنجیره ی مستقیم

$$P \Rightarrow Q, L \wedge M \Rightarrow P, B \wedge L \Rightarrow M, A \wedge P \Rightarrow L, A \wedge B \Rightarrow L, A, B$$


Q
P
M
L
A
B



اثبات تمامیّت

FC یا زنجیره ی مستقیم ، هر عبارت اتمیک که توسط KB به وجود آمده است را نتیجه می دهد ؛ FC در جایی که هیچ عبارت اتمیک جدیدی مشتق نمی شود به یک **نقطه ی ثابت** [1] می رسد ؛ توجه کنید که حالت نهایی به صورت یک مدل m ، نسبت دهنده ی درست / غلط به سمبل ها می باشد ؛ هر شرط در KB اصلی ، در m درست است ؛ از این رو m مدلی از KB می باشد ؛ در صورتی که KB|=q ، q در هر مدل KB ، شامل m درست می باشد .

زنجیره ی معکوس

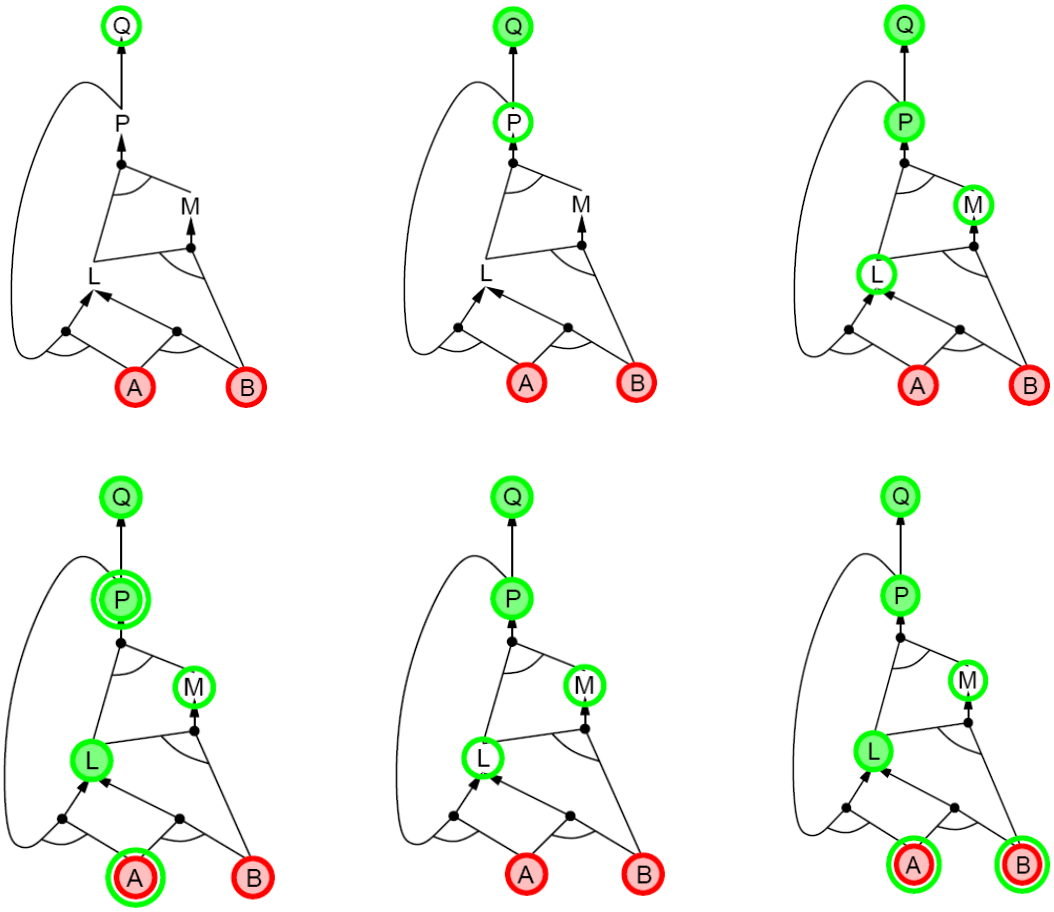
از نتیجه ، با پیدا کردن قانون هایی که می توانند نتیجه را به وجود بیاورند به عقب برمی گردد ؛ سپس تلاش می کند تا مقدمات قانون ها را به دست آورد . این روش برای حل مسأله مناسب است و برخی اوقات ،

---

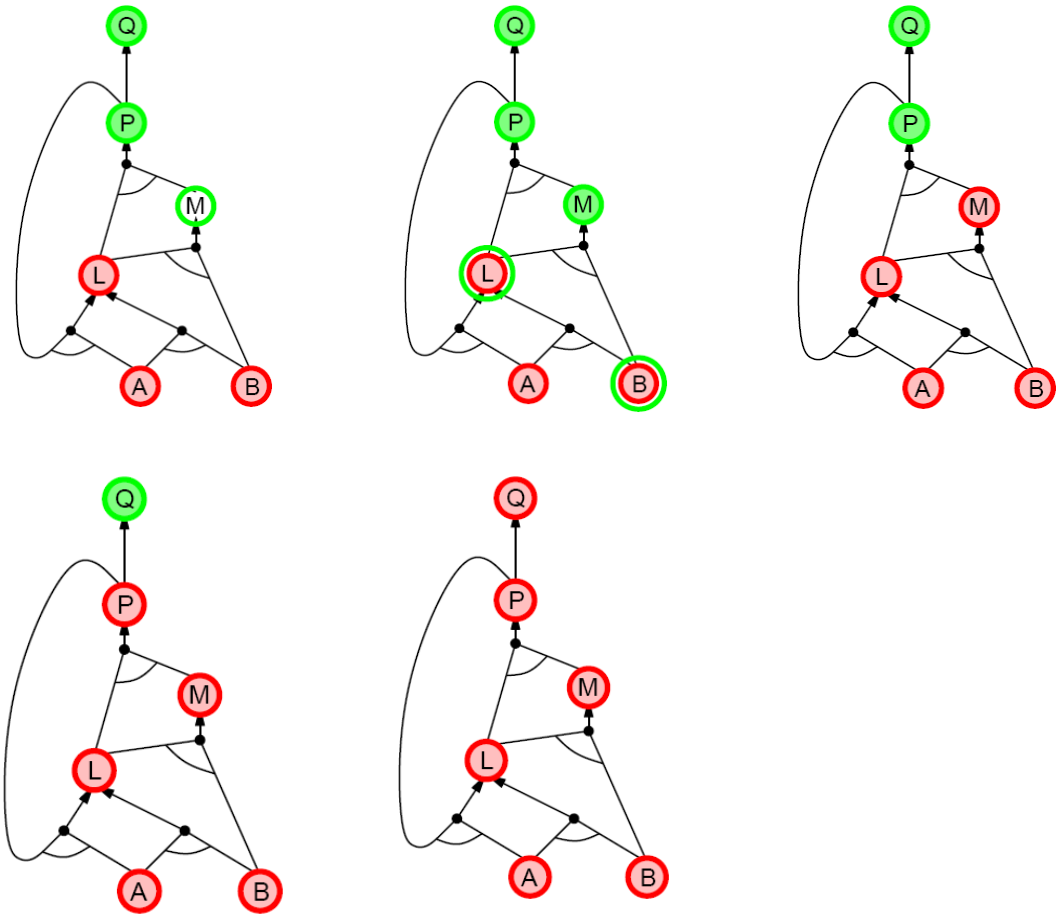
[1] fixed point

استنتاج مشتق شده از پرس و جو [۱] نامیده می شود . در این روش ، احتمال به وجود آوردن اطّلاعات جدید و ناشناخته ، کم تر می باشد . زبان برنامه نویسی پرولوگ هم از زنجیره ی معکوس استفاده می نماید .

مثال زنجیره ی معکوس (شکل ها را از چپ به راست دنبال نمایید .)



[۱] query-driven reasoning



زنجیره ی مستقیم (FC) و معکوس (BC)

در FC از داده ها استفاده می کنیم و ممکن است تعدادی کار را که به هدف نامربوطند انجام دهیم . در BC از نتیجه ها شروع می کنیم و برای حلّ مسأله مناسب می باشد .

تحلیل

قوانین قبلی ، درست هستند ولی الزاماً کامل نمی باشند . خوشبختانه ، قانون کاملی برای استنتاج وجود دارد که این قانون ، *تحلیل* نام دارد . قانون تحلیل پایه شبیه این می باشد ، $A \vee B$ و $\neg A \vee C$ به ما اجازه می دهند که $B \vee C$ را استنتاج کنیم . در موقع تحلیل ، باید پایگاه دانش ما به صورت CNF باشد .

صورت نرمال ربط دهنده [۱] (CNF)

اغلب ، مفید است که فرمول ها را به صورت های نرمال بنویسیم . صورت های نرمال با آنچه که قبل از نرمال کردن بودند فرقی ندارند و تفاوت آن ها در ظاهر آن هاست و برای استدلال ، مناسب ترند . یک حرف [۲] ، یک چیز تجزیه ناپذیر [۳] یا نقیض آن است مثل ، P یا نقیض آن که $\neg P$ است . یک فرمول به صورت CNF است اگر به صورت $A_1 \wedge A_2 \wedge \ldots A_k$ باشد که A_i تشکیل شده از یا (OR) گزاره ها یا نقیض آن ها . به عنوان مثال ؛

$(p \vee q) \wedge r \wedge (\neg p \vee \neg r \vee s)$ ، به صورت CNF است .

$\neg(p \vee q) \wedge r \wedge (\neg p \vee \neg r \vee s)$ ، به صورت CNF نمی باشد .

$(p \vee q) \wedge r \wedge (p \Rightarrow (\neg r \vee s))$ ، به صورت CNF نمی باشد .

نکته : *هر عبارت منطق گزاره ای می تواند به فرم نرمال ربط دهنده تبدیل شود .*

تبدیل یک عبارت به CNF

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

---

[۱] Conjunctive Normal Form یا CNF

[۲] literal

[۳] atom

۱. حذف کردن $\Leftrightarrow$ ؛ جایگزین کردن $\alpha \Leftrightarrow \beta$ با $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

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

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 به درون عبارات ، 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:

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

مثال :

فرمول نهایی (به صورت CNF)	فرمول اوّلیه
$(A \vee B) \wedge (A \vee C)$	$A \vee (B \wedge C)$
$(B \vee A) \wedge (C \vee A)$	$(B \wedge C) \vee A$

مزیّت های منطق گزاره ای

- اعلانی یا اظهاری بودن این روش – دانش می تواند از استنتاج ، مجزّا باشد .

- می تواند اطّلاعات 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 ، کارآمد می باشد)

معایب منطق گزاره ای

- افزایش تشریحی در تعداد لفظ ها
- راهی برای تشریح ارتباط های میان اشیا وجود ندارد .
- راهی برای بیان کیفیّت ، در مورد اشیا وجود ندارد .

منطق مرتبه ی اوّل ، روشی برای رسیدگی کردن به این مسایل می باشد .

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل دهم

منطق مرتبه ی اوّل[1] به بیان ساده

[1] First-Order Logic(FOL)

منطق مرتبه ی اوّل ، تمام خصوصیّات منطق گزاره ای را داراست . منطق مرتبه ی اوّل براساس این ایده است که جهان مرکب از دو نوع موجودیت [۱] می باشد : اشیا [۲] و ارتباطات [۳] ؛ اشیا ، معمولاً به نام ها و چیزها اشاره می نمایند ، به عنوان مثال ، جورج بوش [۴] و خورشید و ارتباطات معمولاً به خصوصیات [۵] و ویژگی ها [۶] اشاره می کنند ، به عنوان مثال ، جورج بوش زنده است . خورشید ، داغ است . لیلی پاتر [۷] ، مادر هری پاتر [۸] است .

در تفسیر زبان طبیعی از " هر ... " یا " همه ... " استفاده می کنیم ؛ 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 از " برخی ... " هم استفاده می نماییم ؛ برای مثال ، " برخی از پرندگان

---

[۱] entity
[۲] objects
[۳] relations
[۴] George Bush
[۵] properties
[۶] attributes
[۷] Lily Potter
[۸] Harry Potter

نمی توانند پرواز نمایند " ، " برخی از افراد از شطرنج لذت می برند " ؛ این مطالب اظهار می کنند که لااقل ، یک شی دارای یک خصوصیت معین می باشد .

یک زبان مرتبه ی اوّل شامل موارد زیر است :

۱- یک مجموعه از ثابت ها [۱] ؛ که با حروف کوچک مشخص می شوند می باشد ، نظیر a ، b ، c و ... ؛ به طور حسّی ، یک ثابت نام یک شی می باشد .

۲- یک مجموعه از متغیّرها ، معمولاً با حروف کوچک مشخص می شوند ، نظیر x ، y ، z و ...

۳- یک مجموعه از سمبل های تابعی ، که معمولاً با حروف کوچک نمایش داده می شوند ، نظیر f ، g ، h و ... ؛ هر سمبل تابعی دارای یک مقدار صحیح و مثبت می باشد .

مجموعه ای از واژگان با قوانین [۲] زیر تعریف می شود :

۱. یک ثابت ، یک واژه می باشد

۲. یک متغیّر ، یک واژه می باشد

۳. یک عبارت [۳] $f(t_1,...,t_n)$ در صورتی یک واژه است که یک سمبل تابعی و هر t_i واژه باشند

۴. هیچ چیز دیگری واژه نمی باشد

گزاره ها [۱]

---

[۱] constants

[۲] rules

[۳] expression

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

یک زبان مرتبه ی اوّل همچنین شامل یک مجموعه از گزاره ها که معمولاً با حروف بزرگ مشخص می شوند است ، نظیر P ، Q ، R می باشد که گزاره ها خصوصیات اشیا را نمایش می دهند .

رابط ها [۲] و کمّیّت سنج ها (سورها) [۳]

رابط های " استاندارد " منطق گزاره ای عبارتند از : $\vee,\wedge,\neg,\rightarrow$ و ... . دو کمّیّت سنج عبارتند از : یکی $\forall$ ، بخوانید " برای همه " ؛ که سور عمومی [۴] نامیده می شود و دیگری $\exists$ ، بخوانید " وجود دارد " ؛ که سور وجودی [۵] نام دارد . به طور معمول ، سورها با متغیّر ها و متغیّرها با سورها به کار می روند .

فرمول مرتبه ی اوّل

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 اوّل می باشد ؛ در صورتی که φ و ψ فرمول باشند ، آن گاه $\varphi \vee \psi$ ، $\varphi \wedge \psi$ ، $\neg \varphi$ و $\varphi \rightarrow \psi$ هم فرمول هستند ؛ در صورتی که φ یک فرمول باشد و x یک متغیّر باشد ، آن گاه $\forall x(\varphi)$ و $\exists x(\varphi)$ هم فرمول هستند و *فرمول دیگری به غیر از موارد گفته شده وجود ندارد .*

متغیّرهای آزاد و محدود [۶]

ساختار فرمول مرتبه ی اوّل ، کامل نمی باشد زیرا فرمول هایی نظیر $\forall x : P(a)$ و P(x) که x یک متغیّر آزاد نامیده می شود را مجاز می داند .

---

[۱] predicates
[۲] Connectives
[۳] Quantifiers
[۴] universal quantifier
[۵] existential quantifier
[۶] Free and Bound variables

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

معنی یک فرمول منطق مرتبه ی اوّل

در منطق مرتبه ی اوّل ، معنی $\neg, \vee, \wedge$ و $\rightarrow$ مثل معنی آن ها در منطق گزاره ای می باشد ، 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 معمولاً با حرف D نمایش داده می شود فقط یک مجموعه می باشد ، یک مجموعه می تواند مثل اعداد طبیعی {۱،۲،۳،...} یا مجموعه ای از افراد باشد . 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 ، که $\varphi(x \mid d)$ فرمولی گرفته شده از φ بعد از جایگزین نمودن هر وقوع x در φ با d می باشد . به عنوان مثال ، 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 و همین طور این مورد برای هر دانشجوی درون OMB 31 هم درست باشد . حال 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 که $\varphi(x \mid d)$ فرمولی گرفته شده از φ بعد از جایگزین نمودن هر وقوع x در φ با d می باشد به عنوان مثال ، به فرمول $\exists x : standing(x)$ و یک دامنه ی D مرکب از افرادی که در حال حاضر در OMB 31 هستند توجه نمایید ؛ فرمول $\exists x : standing(x)$ 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 : $\exists x : standing(x)$ درست می باشد . توجه کنید که یک فرمول ممکن است در برخی از دامنه ها درست باشد امّا در برخی دیگر نادرست باشد ؛ به عنوان مثال ، به فرمول $\forall x \exists y : (x = y^2)$ توجه نمایید ؛ این فرمول در دامنه ی اعداد حقیقی مثبت درست می باشد ولی 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 .

تعریف – یک فرمول که در همه ی دامنه ها درست است یک *همانگویی* [۱] *(همیشه درست)* می باشد .

تعریف – یک فرمول که در همه ی دامنه ها غلط است یک *خلاف گویی* [۲] *(تناقض)* است .

بیان نسبت خانوادگی

منطق مرتبه ی اوّل 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(_ , _) [۳]
- Child(_ , _) [۴]
- Grandparent(_ , _) [۵]
- Sibling(_ , _) [۶]
- Male(_) [۷]
- Female(_) [۱]

---

[۱] tautology
[۲] contradiction
[۳] پدر یا مادر
[۴] بچه ، فرزند
[۵] پدر بزرگ یا مادر بزرگ
[۶] برادر یا خواهر
[۷] مرد

- o - = (_ , _)

راهی کلّی برای تبدیل عبارت های انگلیسی به منطق مرتبه ی اوّل وجود ندارد ؛ در این مورد نظریّه های [۲] غیردقیق [۳] سادگی [۴] و اختصار [۵] وجود دارد . در زیر جملاتی بیان شده و معادل آن ها در منطق مرتبه ی اوّل هم آورده شده است :

- o " James پدر Harry است "
  - $Parent(James, Harry) \wedge Male(James)$
- o " Lily مادر Harry است "
  - $Parent(Lily, Harry) \wedge Female(Lily)$

ارتباطات خانوادگی

- " مردان [۶] و زنان [۷] کاملاً متمایز هستند
  - o $\forall x : (Male(x) \leftrightarrow \neg Female(x))$
  - o $p \leftrightarrow q$ خوانده می شود " p معادل با q است و کوتاه نویسی برای $((p \rightarrow q) \wedge (q \rightarrow p))$ می باشد

---

[۱] زن
[۲] notions
[۳] imprecise
[۴] simplicity
[۵] conciseness
[۶] males
[۷] females

- برادر و خواهر [1] افرادی دقیقا متمایز هستند که دارای یک والد مشترک می باشند

o $\forall x \forall y (Sibling(x,y) \leftrightarrow (\neg = (x,y) \wedge \exists p (Parent(p,x) \wedge Parent(p,y)))$

منطق مرتبه ی اوّل و منطق گزاره ای

در منطق مرتبه ی اوّل $\forall$ لازم است ؛ 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 : $\forall x : P(x) \equiv P(a_1) \wedge ... \wedge P(a_n)$ و 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 : $\forall x : P(x) \equiv P(a_1) \wedge ... \wedge P(a_n) \wedge ...$ ، امّا یک فرمول نمی تواند دارای یک طول بی نهایت باشد .

در منطق مرتبه ی اوّل ، $\exists$ هم لازم است ؛ 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 : $\exists x : P(x) \equiv P(a_1) \vee ... \vee P(a_n) \vee ...$ ، امّا یک فرمول نمی تواند دارای یک طول بی نهایت باشد .

نقیض و سورها

داریم :

o $\neg \forall x : P(x) \equiv \exists x : \neg P(x)$

o $\forall x : \neg P(x) \equiv \neg \exists x : P(x)$

---

[1] siblings

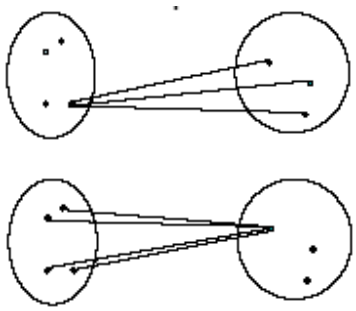
- $\neg\forall x : \neg P(x) \equiv \exists x : P(x)$
- $\forall x : P(x) \equiv \neg\exists x : \neg P(x)$

مثال برای نقیض و سورها

به گزاره ی male(_) در دامنه ی دانش آموزان درون OMB 31 توجه نمایید . معنای $\neg\forall x : male(x)$ این است که موردی وجود ندارد که در آن هر کس در OMB 31 ، مرد باشد ؛ این مطلب معادل با این عبارت است که کسی در OMB مرد نمی باشد ، ظاهرا معادل $\exists x : \neg male(x)$ می باشد . سایر معادل ها عبارتند از : $\exists x \exists y : P(x, y) \equiv \exists y \exists x : P(x, y)$ و $\forall x \forall y : P(x, y) \equiv \forall y \forall x : P(x, y)$.

به عنوان مثال دیگر ، 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(_ , _) هم یک گزاره ی دودویی نشان دهنده ی این که دو نفر همدیگر را می شناسند می باشد :

- $\forall x \forall y (Group1(x) \wedge Group2(y) \rightarrow Know(x, y))$



$\forall y \forall x (Group1(x) \wedge Group2(y) \rightarrow Know(x, y))$

فصل یازدهم

منطق مرتبه ی اوّل

ریوس مطالب

- چرا منطق مرتبه ی اوّل ؟
- ظاهر ومعنای منطق مرتبه ی اوّل
- بازی با عبارات
- دنیای wumpus در منطق مرتبه ی اوّل

موافقین و مخالفین منطق گزاره ای

موافقین

☺ منطق گزاره ای ، اظهاری (اعلانی) است : اجزای گرامری با واقعیّات برابرند . ☺ منطق گزاره ای ، به اطّلاعات جزیی / فصلی / منفی شده اجازه می دهد (برخلاف بیش تر ساختارهای داده ای

و پایگاه های داده ای) .☺ منطق گزاره ای ، **ترکیبی** [۱] می باشد ؛ به عنوان مثال ، مفهوم $B_{1,1} \wedge P_{1,2}$ از $B_{1,1}$ و از $P_{1,2}$ به وجود می آید . ☺ مفهوم در منطق گزاره ای ، **مستقل از متن** [۲] است . (برخلاف زبان طبیعی ، که مفهوم وابسته به متن می باشد) .

مخالفین

☹ منطق گزاره ای محدودیّت 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 اوّل – 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 [۳] ، رنگ ها ، بازی های بیسبال [۴] ، جنگ ها ، قرن ها و ..... می باشند .

- **وابستگی ها (ارتباطات یا روابط) :** قرمز ، گرد [۵] ، ساختگی [۶](جعلی یا قلابی) ، اوّل [۷] (عمده) ، ساختمان چند طبقه [۱] و ..... می باشند .

---

[۱] compositional
[۲] context-independent
[۳] Ronald McDonald
[۴] baseball
[۵] round
[۶] bogus
[۷] prime

- برادر ... ، بزرگ تر از ، درون ... ، قطعه ای از ... ، به رنگ ، اتفاق افتاده بعد از ، داشتن ، آمدن میان و ..... می باشند .

- **عملکردها [۲](تابع ها) :** پدر ... ، بهترین دوست ، تصدی ... ، یکی بیش تر از ، پایان ..... می باشند .

منطق در حالت کلّی

زبان	هستی شناسی [۳]	معرفت شناسی [٤]
منطق گزاره ای	واقعیّات	**درست / غلط / ناشناخته**
منطق مرتبه ی اوّل	واقعیّات ، اشیاء ، رابطه ها	**درست / غلط / ناشناخته**
منطق زمانی [٥]	واقعیّات ، اشیاء ، رابطه ها ، زمان ها	**درست / غلط / ناشناخته**
تیوری احتمال [٦]	واقعیّات	**اندازه ی اعتقاد [۷]**
منطق فازی [۸]	**واقعیّات + درجه ی درستی [۹]**	**مقدار فاصله ی شناخته شده [۱۰]**

---

[۱] multistoried
[۲] functions
[۳] Ontological Commitment
[۴] Epistemological Commitment
[۵] Temporal logic
[۶] Probability logic
[۷] degree of belief
[۸] Fuzzy logic
[۹] degree of truth
[۱۰] known interval value

ظاهر منطق مرتبه ی اوّل - عناصر اساسی عبارتند از :

- ثابت ها مثل : KingJohn,2,UCB,Koblenz,C,…
- گزاره ها مثل : Brother,>,=,...
- توابع مثل : Sqrt,LeftLegOf,…
- متغیّرها مثل : x,y,a,b,...
- رابط ها مثل : $\wedge,\vee,\neg,\Rightarrow,\Leftrightarrow$
- تساوی : =
- کمّیّت سنج ها (سورها) : $\forall,\exists$

عبارات اتمیک [1]

- عبارت اتمیک = $predicate(term_1,...,term_n)$ یا $term_1 = term_2$

واژه = $function(term_1,...,term_n)$ ، که یا ثابت است و یا متغیّر می باشد .

مثال :

Brother(KingJohn,RichardTheLionheart) >
(Length(LeftLegOf(Richard)),Length(LeftLegOf(KingJohn)))

---

[1] Atomic

در عبـارت Brother(KingJohn,RichardTheLionheart) ، Brother گـزاره ، KingJohn ثابـــــــــت ، RichardTheLionheart ثابـــــــــت ، 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 ، همانند منطق گزاره ای ، از عبارات اتمیک با اسـتفاده از رابط ها ساخته می شوند ، مثل :

$$\neg S, S_1 \wedge S_2, S_1 \vee S_2, S_1 \Rightarrow S_2, S_1 \Leftrightarrow S_2$$

مثال :

$Sibling(KingJohn, Richard) \Rightarrow Sibling(Richard, KingJohn)$
$>(1,2) \vee \leq (1,2)$
$>(1,2) \wedge \neg >(1,2)$

در عبارت فوق ، Sibling ، گـزاره مـی باشـد . KingJohn واژه اسـت . Richard واژه است . Sibling طـرف راسـت گـزاره اسـت . Richard طـرف راسـت واژه اسـت . KingJohn طـــرف راســـت واژه اســـت . Sibling(KingJohn,Richard) ، عبـــارت اتمیـــک اســـت . Sibling(Richard,KingJohn) عبارت اتمیک است و کل عبارت ، عبارت پیچیده می باشد .

صحّت در منطق مرتبه ی اوّل – عبارت ها با رعایت یک مدل و یک تفسیر [۱] ، درست می باشند . مدل ، شامل اشیای بزرگ تر یا مساوی یک (عناصر دامنه) و رابط های میان آن ها می باشد ($contains \geq 1$) . تفسیر موارد مراجعه را برای موارد زیر تایین می نماید :

سمبل های ثابت [۲] ⟵ اشیاء

سمبل های گزاره ای [۳] ⟵ رابط ها

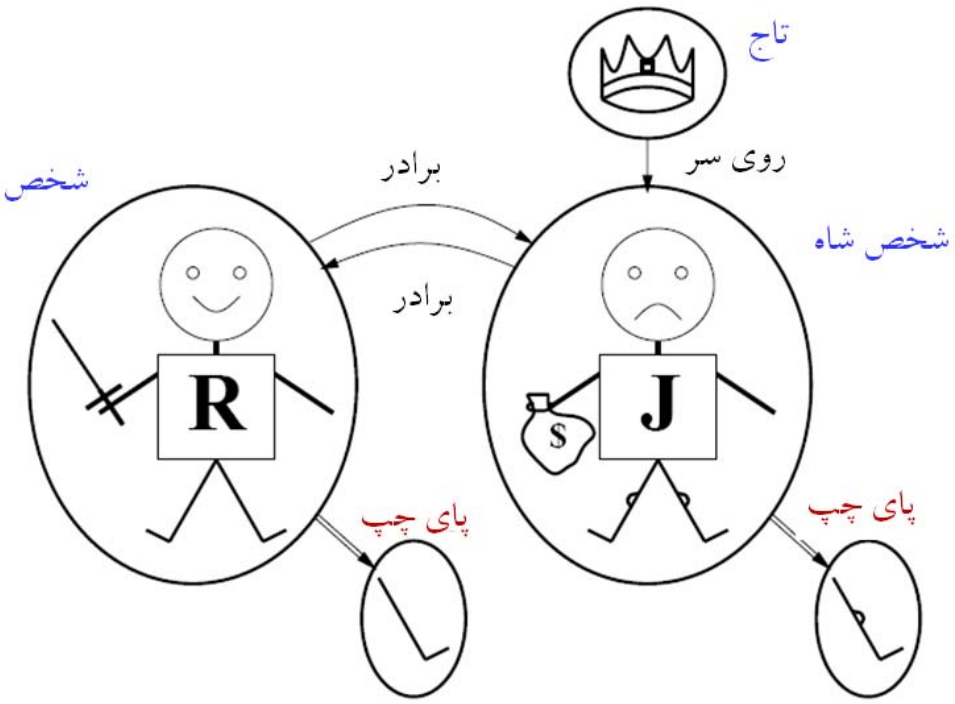
سمبل های تابعی [۴] ⟵ رابط های تابعی [۵]

یک عبارت اتمیک $predicate(term_1,...,term_n)$ در صورتی درست است که اشیاء ارجاع 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 اوّل

---

[۱] interpretation
[۲] constant symbols
[۳] predicate symbols
[۴] function symbols
[۵] functional relations



مثال صحت – به تفسیری به صورت زیر توجّه نمایید :

ریچارد [۱] ← فردی دلیر [۲] است

جان [۳] ← پادشاهی بد

برادر ← ارتباط برادری [۴]

---

[۱] Richard

[۲] Lionheart

[۳] John

[۴] brotherhood relation

تحت این تفسیر ، Brother(Richard,John) فقط در مورد **ریچارد فردی دلیر است** درست است و **جان پادشاهی بد است** درست است ، که در این مدل دارای ارتباط برادری هستند .

سور عمومی – همه در برکلی [۱] زرنگ [۲] هستند : $\forall x : At(x, Berkeley) \Rightarrow Smart(x)$

$\forall x\ P$ ، در یک مدل m درست است در صورتی که P به ازای هر x ممکن در مدل درست باشد .

$$(At(KingJohn, Berkeley) \Rightarrow Smart(KingJohn)) \wedge (At(Richard, Berkeley) \Rightarrow Smart(Richard)) \wedge (At(Berkeley, Berkeley) \Rightarrow Smart(Berkeley)) \wedge \ldots$$

$\forall x : At(x, Berkeley) \wedge Smart(x)$ معنای این عبارت این است که " همه در برکلی هستند و همه باهوش هستند " .

تعیین خواص وجودی [۳]

شخصی در Stanford باهوش است را به این صورت می توانیم بیان نماییم : $\exists x At(x, Stanford) \wedge Smart(x)$. که با مثال های زیر برابر است :

$$(At(KingJohn, Stanford) \wedge Smart(KingJohn)) \vee (At(Richard, Stanford) \wedge Smart(Richard)) \vee (At(Stanford, Stanford) \wedge Smart(Stanford))$$

خصوصیّات کمّیّت سنج ها (سورها)

---

[۱] Berkely

[۲] smart

[۳] Existential quantification

$\forall x \forall y$ مثل $\forall y \forall x$ می باشد . $\exists x \exists y$ مثل $\exists y \exists x$ می باشد . $\exists x \forall y$ 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 کمّیّت سنج ها [۱] – 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 [۲] هستند را به این صورت بیان می نماییم :

$$\forall x,y \ Brother(x,y) \Rightarrow Sibling(x,y)$$

" برادری (هم نژادی) " [رابطه ای] [۳] متقارن [۴] است ؛ یعنی :

$$\forall x, y : Sibling(x, y) \Leftrightarrow Sibling(y, x)$$

مادر یکی والده ی یکی دیگر است :

$$\forall x, y : Mother(x, y) \Leftrightarrow (Female(x) \wedge Parent(x, y))$$

---

[۱] Quantifier duality
[۲] Sibling
[۳] مترجم
[۴] symmetric

اوّلین عموزاده بچه ای هم نژاد با والد است :

$$\forall x, y : FirstCou\sin(x, y) \Leftrightarrow \exists p, psParent(p, x) \wedge Sibling(ps, p) \wedge Parent(ps, y)$$

فصل دوازدهم

استنتاج در منطق مرتبه ی اوّل [۱]

[۱] Inference in First-order Logic

ریوس مطالب

- کاهنده ی استنتاج منطق مرتبه ی اوّل به استنتاج گزاره ای
- یک شکلی [۱]
- طرز عمل کلّی [۲]
- زنجیره ی مستقیم و معکوس
- برنامه نویسی منطقی [۳]
- تحلیل [٤]

---

[۱] unification
[۲] Generalized Modus Ponens
[۳] Logic programming
[۴] Resolution

تاریخچه ی مختصری از استدلال

در سال ۴۵۰ قبل از میلاد مسیح ، پیروان فلسفه ی رواقیون [۱] ، منطق گزاره ای و(شاید) استدلال را بنیان نهادند . | در سال ۳۲۲ قبل از میلاد مسیح ، ارسطو ، قیاس منطقی [۲] (قوانین استدلال) و سورها را بیان کرد . | در سال ۱۵۶۵ ، کاردانو [۳] ، تیوری احتمال [۴] (منطق گزاره ای [۵] + نامعلومی [۶]) را به وجود آورد . | در سال ۱۸۴۷ ، بول [۷] ، دوباره منطق گزاره ای را بررسی کرد . | سال ۱۸۷۹ ، فرگ [۸] ، منطق مرتبه ی اوّل را به وجود آورد . | در سال ۱۹۲۲ ، ویتجنستین [۹] ، اثبات توسط جدول صحت را بررسی کرد . | در سال ۱۹۳۰ ، گدل [۱۰] ، گفت : الگوریتم کاملی برای منطق مرتبه ی اوّل وجود دارد . | در سال ۱۹۳۰ ، هربرند [۱۱] ، الگوریتم کاملی برای منطق مرتبه ی اوّل به وجود آورد . | در سال ۱۹۳۱ ، گدل گفت : الگوریتم کاملی برای محاسبه وجود

---

[۱] Stoics
[۲] syllogisms
[۳] Cardano
[۴] probability theory
[۵] propositional logic
[۶] uncertainty
[۷] Boole
[۸] Frege
[۹] Wittgenstein
[۱۰] Godel
[۱۱] Herbrand

ندارد . | در سال ۱۹۶۰ ، دیویس- پوتنام [۱] ، الگوریتمی کاربردی برای منطق گزاره ای ارایه کرد . | در سال ۱۹۶۵ ، رابینسن [۲] الگوریتمی کاربردی برای منطق مرتبه ی اوّل – تحلیل به وجود آورد .

استنتاج در منطق مرتبه ی اوّل

می توانیم انواع استنتاج را با منطق مرتبه ی اوّل داشته باشیم .

برداشتن سورها

نمونه سازی (معرفی) عمومی [۳]

ما همیشه می توانیم یک واژه ی زمینه را با یک متغیّر جانشین نماییم مثلا در مثال زیر ؛ $\forall x LivesIn(x, Springfield) \Rightarrow knows(x, Homer)$ ، 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 : $\frac{\forall \upsilon : \alpha}{Subs(\{\upsilon / g\}, \alpha)}$ برای هر متغیّر 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 :

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$
$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$$
$$\vdots$$

---

[۱] Davis/Putnam

[۲] Rabinson

[۳] Universal instantiation (UI)

نمونه سازی (معرفی) وجودی [۱]

در مورد $\exists x : LivesIn(x, Springfield) \wedge knows(x, Homer)$ ، ما می دانیم که این باید برای لااقل یک شی ، درست باشد . بیایید این شی را k بنامیم ؛ داریم ، $LivesIn(k, Springfield) \wedge knows(k, Homer)$ که در این مورد ، k یک ثابت اِسکَلِم نامیده می شود . پس ، برای هر عبارت α و متغیّر v و سمبل ثابت k *که در جای دیگری در پایگاه دانش وجود ندارد* داریم :

$$\frac{\exists \upsilon \alpha}{Subst(\{\upsilon / k\}, \alpha)}$$

مثال : نتیجه ی $\exists x Crown(x) \wedge OnHead(x, John)$ به صورت $Crown(C_1) \wedge OnHead(C_1, John)$ می تواند باشد که C_1 به وجود آمده از یک سمبل ثابت جدید به نام ثابت اسکلم [۲] 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) .

گزاره بندی [۳]

ما می توانیم هر عبارت دارای سور وجودی را با یک نوع اسکلمایز شده جایگزین نماییم . برای عبارت های با سور عمومی ، می توانید هر جایگزین ممکن را جایگزین نمایید ؛ که این کار به ما اجازه می

---

[۱] Existential instantiation (EI)
[۲] Skolem constant
[۳] propositionalization

دهد که از قانون های استنتاج گزاره ای استفاده نماییم . البتّه این کار خیلی ناکارآمد می باشد ؛ تا سال ۱۹۶۰ از همین روش استفاده می کردند .

تبدیل به استدلال گزاره ای

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)

که پایگاه دانش جدید ، گزاره ای می باشد ، سمبل های گزاره ای عبارتند از :

King(John) , Greedy(John) , Evil(John) , King(Richard)

و .....

تبدیل

– **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 .

– **ادعا :** هر پایگاه دانش منطق مرتبه ی اوّل می تواند برای حفظ موجودیت ، به صورت گزاره ای بیان شود .

– **ایده :** پایگاه دانش را به صورت گزاره ای و پرس و جو بیان نمایید ، تحلیل را به کار ببرید و نتایج را برگردانید .

– **مسأله :** با سمبل های توابع ، واژگان زمینه به صورت نامحدود وجود خواهند داشت .

مثال : Father(Father(Father(John)))

قضیّه ی هربرند (۱۹۳۰) : در صورتی که عبارت α توسط یک پایگاه دانش منطق مرتبه ی اوّل 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 (پایگاه دانش) گزاره ای با عمق n بسازید

بررسی نمایید که آیا α توسط این KB تولید می شود

قضیّه ی تورینگ و چورچ [۱](۱۹۳۶) : به وجود آمدن [۲] در منطق مرتبه ی اوّل تقریبا قابل تصمیم گیری [۳] می باشد .

ایرادهای گزاره ها

گزاره ، برای تولید تعداد زیادی عبارات که به هم نامربوط هم هستند به کار می رود .

مثال :

---

[۱] Turing,Church

[۲] entailment

[۳] semidecidable

$\forall$ 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 واقعیّات 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 ، معرّفی های با مقادیر به مراتب بدتری وجود خواهد داشت !

یک شکلی

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 :

$$\forall x : LivesIn(x, Springfield) \wedge WorksAt(x, PowerPlant) \Rightarrow know(x, Homer)$$

$$\forall y : LivesIn(y, Springfield)$$

$$WorksAt(MrSmithers, PowerPlant)$$

آن گاه ، ما باید قادر باشیم مستقیما نتیجه بگیریم knows(MrSmithers,Homer) با 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 .

$\alpha\theta = \beta\theta$ اگر θ = {x/John,y/John} , Unify(α,β)= θ

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 [۱] متغیّرها جلوگیری می کند ، مثال : $Knows(z_{17}, OJ)$

طرز عمل کلّی [۲]

ایده ی اساسی این است که فرض کنیم یک استنتاج به شکل $P_1 \wedge P_2 \wedge ... \wedge P_i \Rightarrow Q$ را داریم و عبارات به صورت $P_1', P_2', ..., P_i'$ می باشند ، اگر مجموعه ای از جانشین ها به صورت $P_1 = P_1', P_2 = P_2', ..., P_n = P_n'$ وجود داشته باشند ، سپس ما می توانیم جانشین را به کار بگیریم و روش طرز عمل کلّی 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 پیداکردن جانشین هایی که به ما اجازه خواهند داد عبارات جدید را به وجود آوریم می باشد .

الگوریتم یگانی : دو عبارت را می پذیرد و مجموعه ای از جانشین هایی که عبارات را به صورت یکتا می سازند برمی گرداند .

---

[۱] overlap

[۲] Generalized Modus Ponens (GMP)

WorksAt(x,PowerPlant),WorksAt(Homer,PowerPlant) ، {x/Homer} را تولید می نماید .

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 .

آخرین عبارت یک اشتباه است ؛ فقط به خاطر این که ما خواسته ایم x را در هر دو عبارت استفاده نماییم . ما می توانیم x را با یک متغیّر منحصر به فرد (x_{21}) که یک عبارت است جایگزین نماییم که 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 ؟ ؛ مثلا :

Sibling(Bart,x) و Sibling(y,z) می تواند {Bart/y,x/z} یا {x/Bart,y/Bart,z/Bart} را تولید نماید که اوّلین یکسان سازی ، کلّی تر از دومی است – چون الزام های کم تری را سبب می شود . پس ، ***ما می خواهیم کلّی 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 متغیّر را به یک ثابت متصل می کند را پیدا نمایید . در غیر این صورت یکانگی درون واژه ی اوّل را ، که توسط باقی مانده ی هر عبارت دنبال شده است را صدا بزنید .

مثال :

$Sibling(x, Bart) \wedge PlaysSax(x)$ و Sibling(Lisa,y) که 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 که وضعیت ، لیستی از جانشین ها می باشد و تابع جانشین ، لیستی از یکسان سازی های بالقوه و لیست های جانشین جدید است . پس در مورد یکسان سازی داریم :

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

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 طرز عمل کلّی ، با شرط های نامحدود پایگاه دانش استفاده می شود (**دقیقا** با یک لفظ مثبت) . تمام متغیّرها ، کمّیّت های عمومی را به خود می گیرند .

اثبات روش طرز عمل کلّی – باید نشان بدهیم که : $p_1', \ldots, p_n', (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\theta$

مشروط بر این که برای همه ی i ها ی موجود ، $p_i'\theta = p_i\theta$ باشد .

اثبات : با استفاده از نمونه سازی عمومی (UI) برای هر شرط نامحدود p داریم $p \models p\theta$

$1.(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$

$2. p_1', ..., p_n' \models p_1' \wedge ... \wedge p_n' \models p_1'\theta \wedge ... \wedge p_n'\theta$

۳. از ۱ و ۲ نتیجه می شود که $q\theta$ توسط روش های معمولی دنبال می شود .

مثال : پایگاه دانش – قانون می گوید که این یک جنایت است که یک آمریکایی به ملّت های دشمن ، اسلحه بفروشد . کشور نونو [۱] یک دشمن آمریکا می باشد و دارای تعدادی موشک می باشد و تمام این موشک ها توسط کلنل وست [۲] 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 : $Weapon(x) \Rightarrow Missle(x)$

$$Enemy(x,America) \Rightarrow Hostile(x)$$

وست آمریکایی است ... American(West).

---

[۱] Nono

[۲] Colonel West

کشور نونو یک دشمن آمریکا است : Enemy(Nono,America)

زنجیره ی مستقیم

ایده ی اساسی این است که ، با واقعیّات و قوانین شروع کنید (استنباط ها) و دایما روش طرز عمل کلّی (GMP) را به کار ببرید تا واقعیّات جدید بتوانند به وجود بیایند .

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 ←{}

برای هر عبارت r در KB موارد زیر را انجام بده

$(p_1 \wedge ... \wedge p_n \Rightarrow q) \leftarrow S\tan dardize - Apart(r)$

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

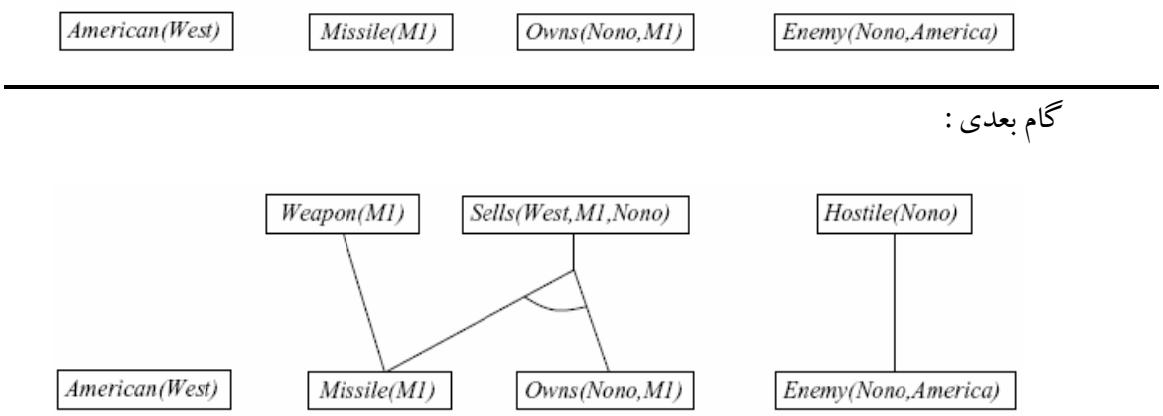
$\phi \leftarrow Unify(q', \alpha)$

در صورتی که ϕ ، fail نمی باشد ، ϕ رابرگردان

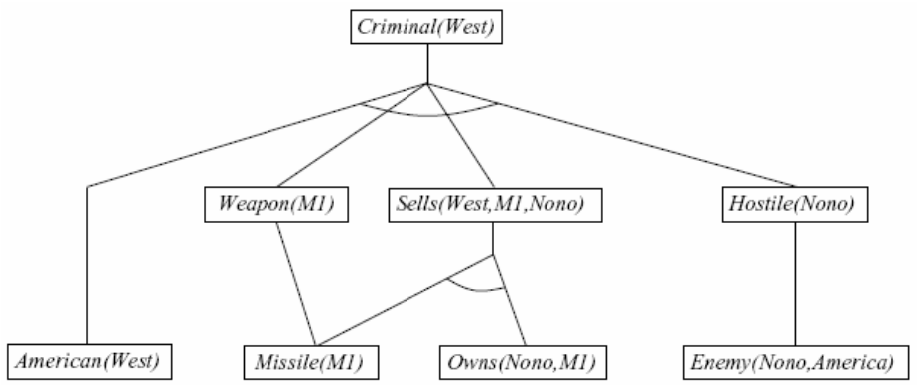
new را به KB اضافه کن

false را برگردان

اثبات به روش زنجیره ی مستقیم



گام بعدی :

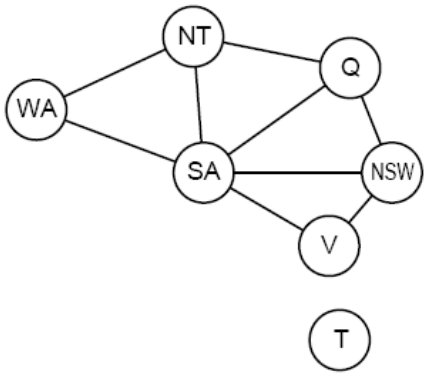


خصوصیّات زنجیره ی مستقیم

زنجیره ی مستقیم ، بی عیب می باشد ، چون که از روش طرز عمل کلّی (GMP) استفاده می کند . برای شرط های محدود ، زنجیره ی مستقیم ، کامل می باشد . خیلی شبیه به جستجوی اوّل – بهترین عمل می کند . این الگوریتم اساسی ، خیلی هم کارآمد نیست ، چون : اوّل ، همه ی یکی کننده های ممکن را پیدا می کند ولی گران است . دوم ، هر قانون ، در هر تکرار ، مجددا چک می شود و سوم ، این که واقعیّاتی که به هدف منجر نمی شوند ، تولید می شوند .

زنجیره ی مستقیم به طور گسترده ای در ***پایگاه داده های استقرایی یا استنتاجی یا قیاسی*** [۱] استفاده می شود . یک سیستم پایگاه داده ی قیاسی ، سیستمی پایگاه داده ای است که می تواند قیاس را براساس قوانین و واقعیّات نگهداری شده پایگاه دانش قیاسی انجام دهد . سیستم های پایگاه دانش قیاسی ، اساسا به قوانین و واقعیّات رسیدگی می کنند ، از یک زبان اعلانی (نظیر پرولوگ) برای مشخص کردن این قوانین و واقعیّات استفاده نمایید و از یک موتور استنتاج که می تواند قوانین و واقعیّات جدید را از آن هایی که ارایه شده اند قیاس نماید استفاده نمایید . [۲]

مثال تطبیق دهنده ی سخت [۳]



---

[۱] deductive databases

[۲] en.wikipedia.org/wiki/Deductive_database

[۳] hard matching

$$Diff(wa,nt) \wedge Diff(wa,sa) \wedge Diff(nt,q) Diff(nt,sa) \wedge Diff(q,nsw) \wedge Diff(q,sa) \wedge Diff(nsw,v)$$
$$\wedge Diff(nsw,sa) \wedge Diff(v,sa) \Rightarrow Colorable()$$

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

این روش ، به طرف معکوس ، از هدف به طرف واقعیّاتی که باید برای هدف نشان داده شوند حرکت می کند . از روش طرز عمل کلّی ، 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 و جستجو به روش اوّل – 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 اوّلیه ، تعویض خالی می باشد { }

متغیّرهای محلی : answers ، که مجموعه ای از تعویض ها می باشد ، و به صورت اوّلیه تهی ({ }) می باشد .

در صورتی که goals تهی است مجموعه ی $\{\theta\}$ 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 .

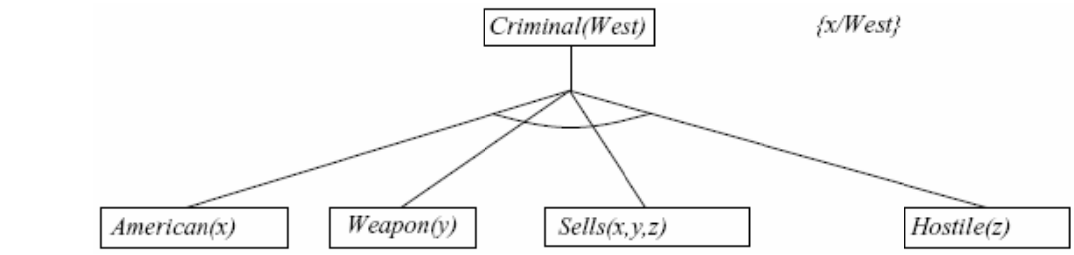
$$new-goals \leftarrow [p_1, ..., p_n \mid \mathrm{Re}st(goals)]$$

$$answers \leftarrow FOL-BC-Ask(KB, new-goals, Compose(\theta', \theta)) \cup answers$$

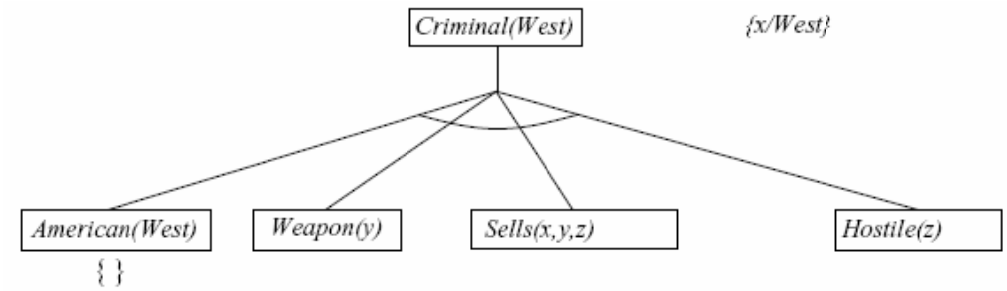
answers را برگردان

مثال زنجیره ی معکوس

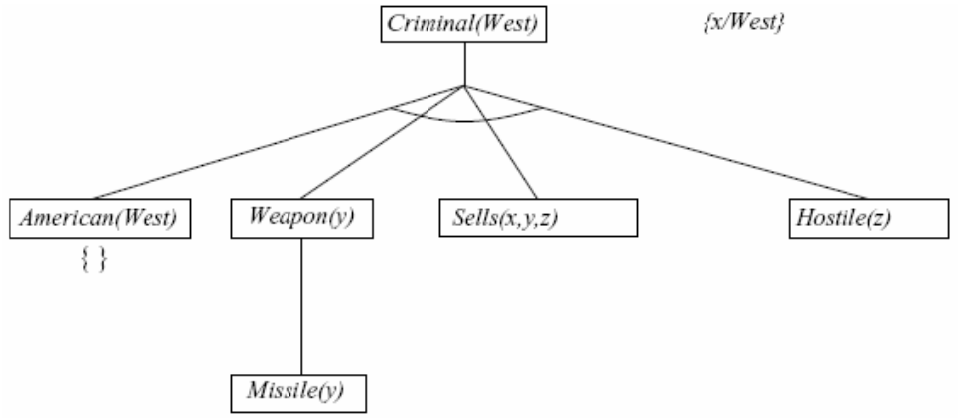
گام بعدی :



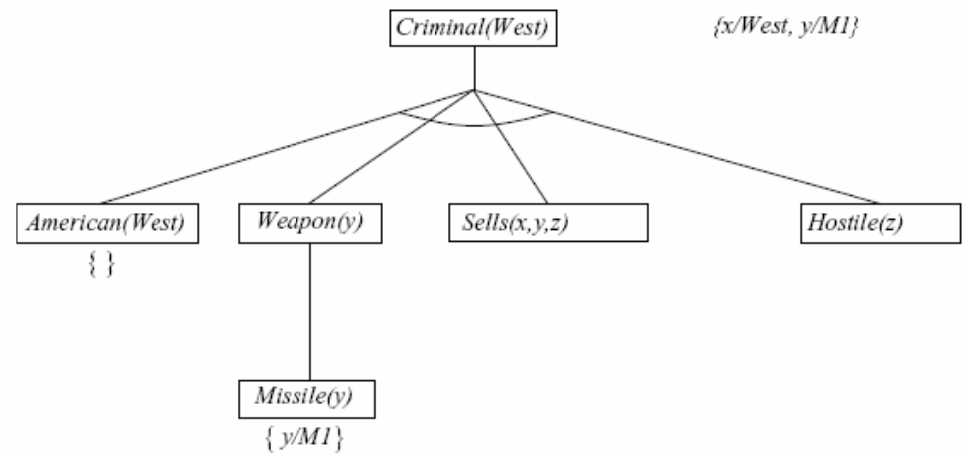
گام بعدی :



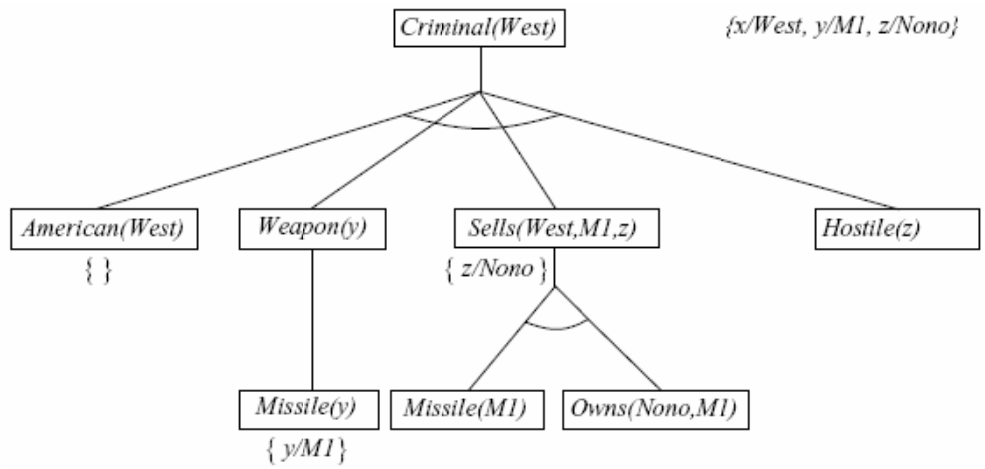
گام بعدی :



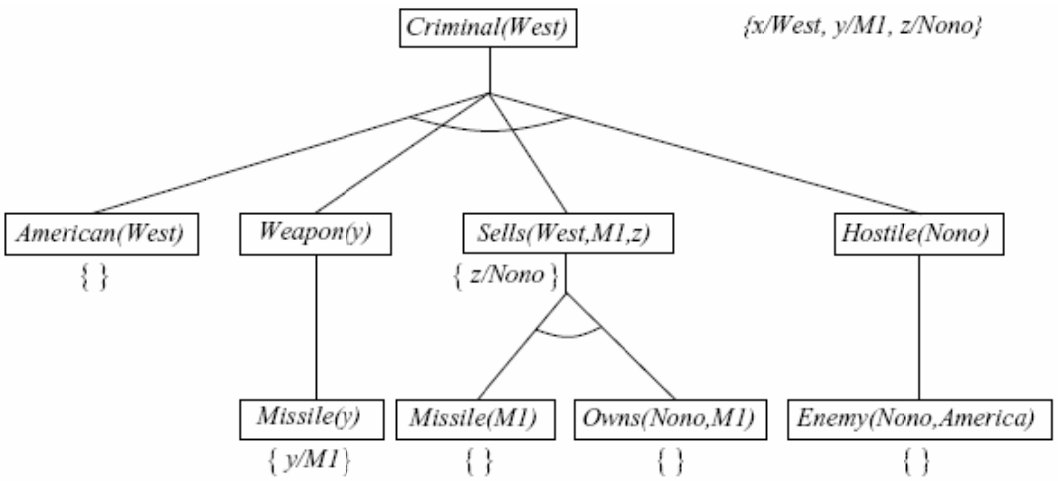
گام بعدی :



گام بعدی :



گام بعدی :



خصوصیّات زنجیره ی معکوس

زنجیره ی معکوس از روش اوّل – عمق استفاده می کند ؛ این به این معنی است که از وضعیت های تکرار شده رنج می برد . همچنین این روش کامل نمی باشد و می تواند برای سیستم های برمبنای پرس و جو [۱] خیلی موثر باشد در ضمن ، این روش برای برنامه نویسی منطقی [۲] به طور گسترده ای استفاده می شود .

تحلیل : خلاصه

تحلیل را در منطق گزاره ای به یاد بیاورید : $(A \vee C) \wedge (\neg A \vee B) \Rightarrow (B \vee C)$ ، تحلیل در منطق مرتبه ی اوّل به طریقی مشابه عمل می کند . *توجه کنید که لازم است عبارات به فرم CNF باشند .*

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[۳] 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 :

---

[۱] query – based systems

[۲] logic programming

[3] implication

در کامپیوتر ، عملی منطقی با ساختار
IF-THEN
اگر A,B درست باشند ، آن گاه تابع AND این دو هم درست است

$\neg\forall x, p \equiv \exists x \neg p, \neg\exists x, p \equiv \forall x \neg p$:

$\forall x[\exists y \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y Loves(y,x)]$

$\forall x[\exists y \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y Loves(y,x)]$

$\forall x[\exists y Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y Loves(y,x)]$

۳. متغیّرهای استاندارد : هر کمّیّت ، باید از یک متغیّر 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

$\forall x[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

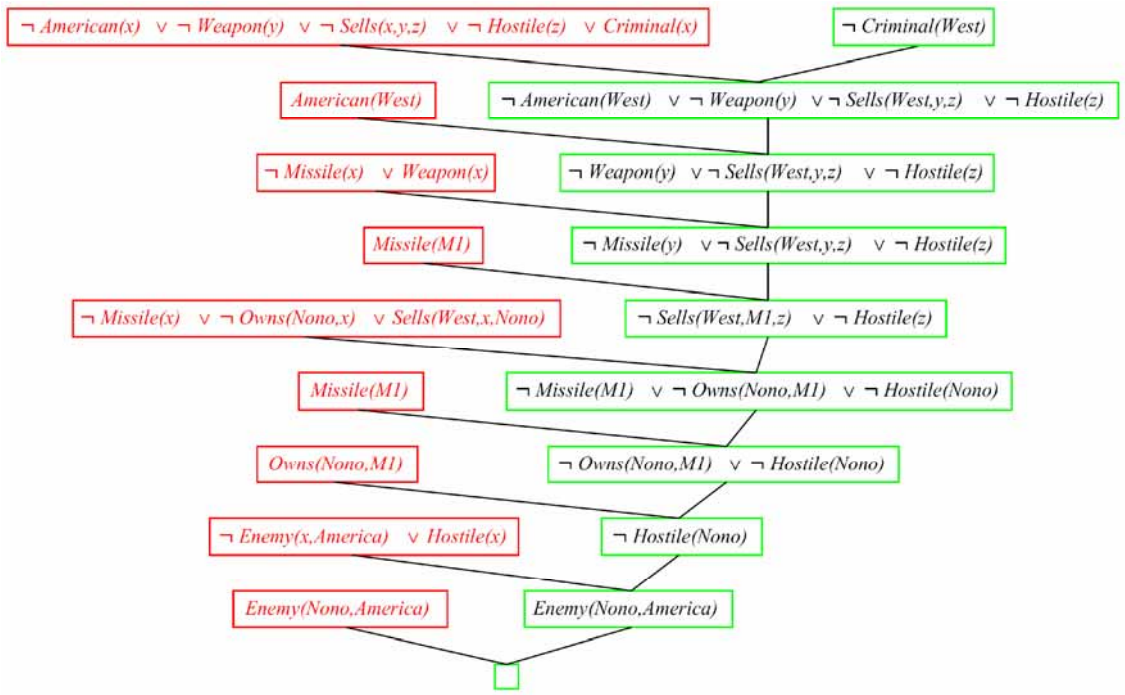
۵. سورهای عمومی را رها نمایید :

$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

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


¬ American(x) ∨ ¬ Weapon(y) ∨ ¬ Sells(x,y,z) ∨ ¬ Hostile(z) ∨ Criminal(x)
¬ Criminal(West)
American(West)
¬ American(West) ∨ ¬ Weapon(y) ∨ ¬ Sells(West,y,z) ∨ ¬ Hostile(z)
¬ Missile(x) ∨ Weapon(x)
¬ Weapon(y) ∨ ¬ Sells(West,y,z) ∨ ¬ Hostile(z)
Missile(M1)
¬ Missile(y) ∨ ¬ Sells(West,y,z) ∨ ¬ Hostile(z)
¬ Missile(x) ∨ ¬ Owns(Nono,x) ∨ Sells(West,x,Nono)
¬ Sells(West,M1,z) ∨ ¬ Hostile(z)
Missile(M1)
¬ Missile(M1) ∨ ¬ Owns(Nono,M1) ∨ ¬ Hostile(Nono)
Owns(Nono,M1)
¬ Owns(Nono,M1) ∨ ¬ Hostile(Nono)
¬ Enemy(x,America) ∨ Hostile(x)
¬ Hostile(Nono)
Enemy(Nono,America)
Enemy(Nono,America)

فصل سیزدهم

نامعلومی[۱]

(عدم قطعیّت)

[۱] uncertainty یا ابهام یا عدم قطعیّت

ریوس مطالب

- نامعلومی
- احتمال [۱]
- ظاهر و معناها
- استدلال (استنتاج)
- استقلال [۲] و قانون بیز [۳]

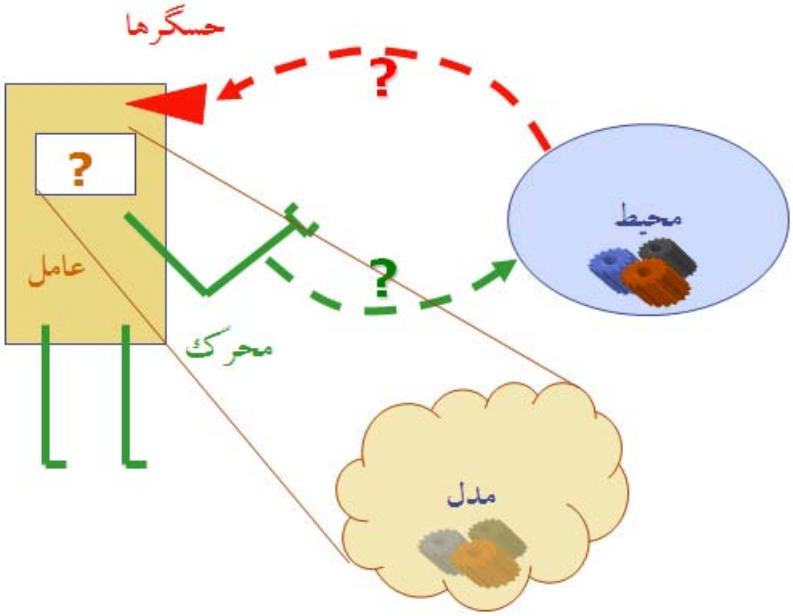
---

[۱] Probability
[۲] Independence
[۳] Bayes' Rule

نامعلومی

در تعدادی از محیط های جالب عامل ها ، نا معلوم بودن یا عدم قطعیّت دارای یک نقش زیاد می باشد ؛ به عبارت دیگر ؛ عملکردها ممکن است دارای اثرات نامعلومی باشند ؛ مثل ، پرتاب یک پیکان به هدف . عامل ها ممکن است وضعیّت درست جهان را ندانند . در ضمن ارتباط های میان واقعیّت ها ممکن است به صورت معلوم نباشند . مثلا گاهی اوقات ، در زمانی که هوا ابری می باشد ، باران می بارد ؛ به عبارت دیگر ، هر وقت که هوا ابری باشد ، باران نمی بارد ! . عامل های هوشمند باید نامعلوم بودن یا ابهام را بررسی نمایند .

عامل نامعلوم [1]



انواع نامعلومی

[1] uncertain agent

نامعلومی در دانش قبلی

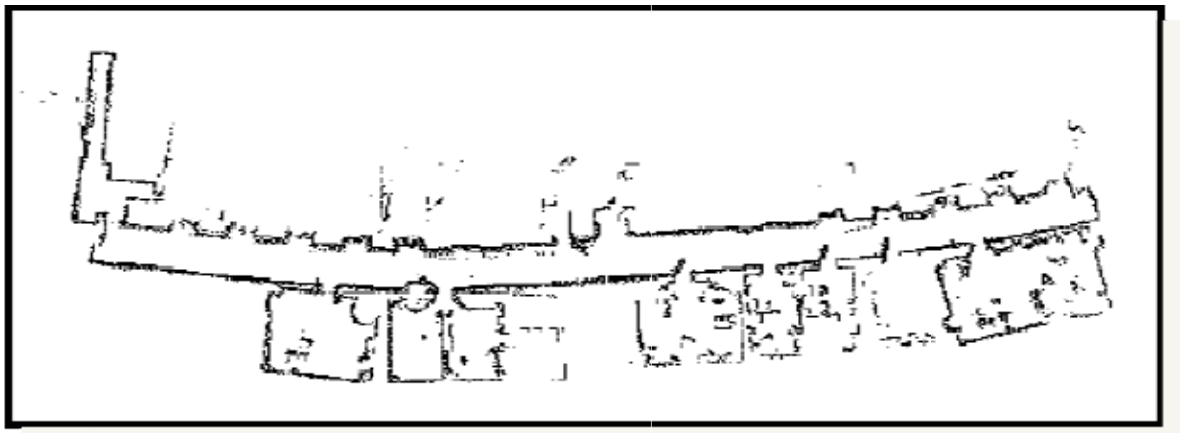
به عنوان مثال ، بعضی از دلایل بیماری ناشناخته اند و در دانش یک عامل دستیار پزشک بیان نشده اند .

نامعلومی در عملکردها

مثلا ، کارهایی که باید برای انجام آن ها پیش شرط هایی برقرار باشند ؛ شاید هم این پیش شرط ها طولانی هم باشند ؛ به عنوان مثال ، برای رانندگی کردن با اتومبیل خودم در صبح باید اتومبیل من در طول شب دزدیده نشده باشد ؛ تایر یا تایرهای آن پنچر نباشند ؛ باید در مخزن (باک) آن ، گاز وجود داشته باشد ؛ باتری آن از کار نیفتاده باشد ؛ باید موتور آن روشن شود ؛ کلیدهای آن را گم نکرده باشم ؛ کامیون مسیر رانندگی را مسدود نکرده باشد ؛ نباید به طور ناگهانی ، من کور یا فلج شوم و ... . البتّه نمی توانیم همه ی موارد را لیست کنیم .

نامعلومی در ادراک

مثلا ممکن است حسگرها اطّلاعات کامل یا دقیق را در مورد جهان برنگردانند ؛ مثلا ممکن است یک ربات مسیر درست را پیدا نکند :



چگونه با نامعلومی ، برخورد کنیم ؟

به صورت ناآشکار (ضمنی یا تلویحی)

به مواردی که نامعلوم هستند ، در زمانی که شما می توانید کار خود را انجام دهید ، بی اعتنا باشید و عملیّاتی را به وجود بیاورید که در مقابل نامعلومی ، مقاوم هستند .

به صورت آشکار

مدلی از جهان را بسازید که نامعلومی را در وضعیت ها ، حرکت ها و مشاهده هایش در نظر می گیرد و در مورد اثر عملیّاتی که مدل انجام می دهد دلیل بیاورید .

منطق و نامعلومی

ما به زودی خواهیم دید که چگونه منطق را با نامعلومی به کار ببریم ؛ مثال هایی از منطق را در زیر مشاهده می نمایید :

$$Studies(Bart) \vee WatchesTV(Bart)$$

$$Hungry(Homer) \Rightarrow Eats(Homer, HotDog) \vee Eats(Homer, Pie)$$

$$\exists x : Hungry(x)$$

متاسفانه ، نگرش منطقی دارای برخی اشکالات می باشد .

ضعف های منطق [۱]

نمی توانیم همه ی نتیجه های ممکن را تایین کنیم ؛ مثلا ، نمی دانیم که اگر الان از خانه خارج شویم ، آیا به موقع در محل مورد نظر خواهیم بود یا نه ؟ . یا اگر پدال گاز اتومبیل خود را زیاد تر فشار دهیم آیا باعث زودتر رسیدن ما به مقصد می شود یا اینکه باعث می شود که ما تصادف کنیم ؟! .

ما ممکن است همه ی نتیجه های ممکن را ندانیم . مثلا ، " در صورتی که یک بیمار دارای دندان درد باشد ، وی ممکن است پوسیدگی [۲] در دندان خود داشته باشد ، یا ممکن است دارای بیماری لثه [۳] 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 . مثلا ، " امروز ممکن است که من با تیر چراغ برق برخورد کنم . "

کمّیّت و کیفیّت

منطق ، یک نگرش کمّی به نامعلومی را ارایه می دهد ؛ ما می توانیم بگوییم که یک اتّفاق ، بیش تر از دیگری روی می دهد یا این که ممکن است یک اتّفاق روی دهد . این روش در مواردی که ما آماری نداریم ، مفید می باشد . احتمال به ما اجازه می دهد که به صورت کیفی بحث کنیم ؛ ما احتمال رخ دادن یک اتفاق را با یک عدد بیان می کنیم .

نامعلومی و عقلانیّت (هوشمند بودن)

---

[۱] weaknesses with logic

[۲] cavity

[۳] gum

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

تعریف ما از عقلانیّت را به یاد بیاورید ؛ یک عامل هوشمند (عقلانی) ، عاملی است که با بیشینه نمودن معیارکارایی عمل می نماید . ما چگونه این عامل را در یک جهان دارای نامعلومی تعریف نماییم ؟ . می گوییم ، عملکرد عامل دارای نتیجه هایی است و هر نتیجه را با یک عدد که نشان دهنده ی احتمال آن است بیان می نماییم . سپس یک عامل می تواند به هر کدام از نتایج ممکن و عدد 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 با احتما ل رخداد یک نتیجه ، *تیوری تصمیم گیری* [۱] نام دارد .

مثال : بیایید عملکرد A_t را ترک کردن به مقصد فرودگاه ، t دقیقه قبل از پرواز قرار دهیم . آیا در زمان A_t به موقع آن جا خواهم بود ؟

مشکلات :

۱) مشاهده پذیری جزیی [۲] ؛ مثل ، حالت جاده ، برنامه ی رانندگان دیگر اتومبیل ها و .....

۲) حس گرهای پر پارازیت [۳](اغتشاش) ؛ مثلا ، گزارش های ترافیک ممکن است درست نباشند .

۳) نامعلومی در نتایج عملکرد [۴] ؛ مثلا ممکن است تایر اتومبیل مان پنچر شود و .....

۴) پیچیدگی بیش از حد [۵] طرح ریزی [۱] و پیش بینی ترافیک [۲]

---

[۱] decision theory
[۲] partial observability
[۳] noisy sensors
[۴] uncertainty in action outcomes
[۵] immense

بنابراین یک شیوه ی منطقی یا با ریسک های دروغ [۳] است ؛ مثلا ، [این که بگوییم] " با A_{25} به موقع خواهم رسید " یا به نتایجی منجر می شود 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 ! .....)

روش هایی برای به کارگیری نامعلومی

منطق پیش فرض یا غیر یکنواخت [۴]*:* فرض کنید که اتومبیل من دارای تایر پنچر نباشد . یا فرض کنید که A_{25} کار می کند مگر این که با تناقض [۵] ثابت شود که کار نمی کند .

احتمال

یک چارچوب مشهور و قابل فهم برای نامعلومی می باشد که دارای معنی های واضح است و برای ترکیب شواهد ، استدلال در مورد پیشگویی و تشخیص و ترکیب شواهد جدید ، جواب های اساسی را مهیّا می کند . احتمال ، برخی از سطح های انسان های ماهر (خبره های انسانی) را شناسایی می کند .

اساس احتمال

به عنوان مثال ، احتمال های P(BartStudied) = 0.01 و P(Hungry(Homer)) = 0.99 را در نظر بگیرید . توجه کنید که خود گزاره ممکن است درست یا غلط باشد و فرق دارد با اینکه بگوییم ، عبارت به صورت جزیی ، درست است .

---

[۱] modelling
[۲] predicting traffic
[۳] risks falsehood
[۴] nonmonotonic
[۵] contradiction

مقادیر تصادفی [۱]

یک متغیّر تصادفی ، متغیّر یا گزاره ای است که مقدارش ناشناخته است و دارای دامنه ای از مقادیر است که می تواند آن ها را به خود بگیرد ؛ این متغیّرها می توانند از موارد زیر باشند :

- بولین (درست و غلط) ؛ مثل : Hungry(Homer) و isRaining

- گسسته ؛ مقادیر از یک دامنه ی *قابل شمارش* گرفته می شوند ؛ مثل ، دما [۲] : < گرم [۳] ، خنک [۴] ، ملایم [۵] > و پیش بینی وضع هوا : < آفتابی [۶] ، ابری [۷] ، بارانی >

- پیوسته ؛ مقادیر ، می توانند از یک *فاصله (دامنه ی غیرقابل شمارش)* گرفته شوند ، نظیر : [۱و۰] ؛ مثل ، سرعت [۸] ، زمان ، مکان [۹]

در اینجا بیش تر تمرکز ما بر روی مورد گسسته خواهد بود .

پس ، یک متغیّر تصادفی ، یک تابع از فضای نمونه برای برای بعضی از محدوده ها می باشد ، مثال : real ها یا Boolean ها . مثل : Odd(1) = true

---

[۱] random values
[۲] temperature
[۳] hot
[۴] cool
[۵] mild
[۶] sunny
[۷] overcast
[۸] velocity
[۹] position

رویدادها (رخدادها) ی تجزیه ناپذیر یا اتمیک [۱]

ما می توانیم گزاره ها را با استفاده از رابط های منطقی استاندارد و به کارگیری اجتماع و اشتراک با هم ترکیب نماییم :

$P(Hungry(Homer) \wedge \neg Study(Bart))$

$P(Brother(Lisa,Bart) \vee Sister(Lisa,Bart))$

یک عبارت که یک مقدار ممکن را برای هر متغیّر نامعین مشخص می کند یک *رویداد اتمیک* [۲] نام دارد ؛ رویدادهای اتمیک دو به دو ناسازگار [۳] (مستقل از هم) هستند . مجموعه ای از همه ی رویدادهای اتمیک ، شامل تمام جزییات [۴] هستند . یک رویداد اتمیک ، درستی یا نادرستی هر گزاره را پیش بینی می کند . توجه کنید که رویدادهای اتمیک در تشخیص درستی موارد دارای چند متغیّر نامعین ، مفید می باشند .

نکته : منطق فازی [۵] از درجه ی قطعیّت (درجه ی درستی) [۶] استفاده می کند نه نامعلومی [۷] .

مثال : $P(Cavity^{1}=true) = 0.1$ و $P(Weather=sunny) = 0.72$ نمونه هایی از احتمال غیرشرطی هستند . توزیع احتمال ، مقادیر را برای همه ی انتساب های ممکن ، ارایه می دهد :

---

[۱] Atomic Events

[۲] atomic event

[۳] mutually exclusive

[۴] exhaustive

[۵] Fuzzy logic : یک شکل از منطق ریاضی است که در آن مقادیر بین صفر و یک دارای ارزش درست هستند . (WordNet 2.0)

[۶] degree of truth

[۷] uncertainty

P(Weather) = <0.72,0.1,0.08,0.1>

توجّه 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 متغیّرهای تصادفی ، احتمال هر رویداد اتمیک را در متغیّرهای تصادفی ارایه می کند . به عنوان مثال ، P(Weather,Cavity) برابر است با یک ماتریس ۲*۴ 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 .

اگر داشته باشیم ، P(Rain)=0.1 ، P(Overcast[۲])=0.4 ، P(Sunny)=0.5 ، در این صورت ، می توانیم احتمالات ترکیب های متغیّرها 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 احتمال توام* [۳] نام دارد . برای متغیّرهای پیوسته ، ما نمی توانیم مقادیر را بشماریم ؛ در عوض ، ما از یک تابع پارامتری به نام توزیع نرمال [۱] استفاده می نماییم :

---

[۱] کرم خوردگی دندان
[۲] ابری
[۳] joint probability distribution

$$P(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{-x^2}{2}} dz$$ (توزیع نرمال)

تصمیم گیری تحت شرایط نامعلومی – در مثال رفتن به فرودگاه که قبلا در مورد آن صحبت کردیم ، 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 مورد فوق را باید انتخاب نماییم ؟ انتخاب ، وابسته به صلاح دید های ما ، برای از دست ندادن پرواز ، وضعیت فرودگاه و ..... می باشد .

قضیّه ی سودمندی [۲] ، برای ارایه و استدلال در مورد صلاح دیدها استفاده می شود .

قضیّه ی تصمیم گیری [۳] برابر است با ، قضیّه ی سودمندی + قضیّه ی احتمال .

مبانی احتمال – با یک مجموعه از فضای ساده ی Ω شروع می کنیم ؛ مثال ، شش حالت ممکن یک تاس . اگر $\omega \in \Omega$ باشد .

---

[۱] normal distribution

[۲] Utility theory

[۳] Decision theory

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

در یک بار پرتاب یک تاس ، احتمال آمدن هر کدام از اعداد ۱ تا ۶ روی شش وجه تاس ، بین ۰ تا ۱ است ؛ پس داریم : $۰ \leq P(\omega) \leq ۱$. در ضمن مجموع احتمال های آمدن هر وجه تاس ، برابر ۱ است ؛ یعنی ، $\Sigma_{\omega}P(\omega) = ۱$.

مثال : $P(1)=P(2)=P(3)=P(4)=P(5)=P(6)=1/6$ ؛ می بینید که احتمال آمدن هر وجه تاس برابر با $\frac{1}{6}$ می باشد .

یک رویداد A زیرمجموعه ای از Ω می باشد :

$$P(A) = \sum_{\{\omega \in A\}} P(\omega)$$

مثلا ، احتمال این که عددی که در یک بار پرتاب یک تاس ، ظاهر می شود کم تر از ۴ باشد ، برابر است با : احتمال ۱ آمدن + احتمال ۲ آمدن + احتمال ۳ آمدن = ۱/۶+۱/۶+۱/۶ = 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 داشته باشیم . به هر متغیّر مستقل نگاه کنید و احتمال وابسته را پیدا نمایید .

	Humidity[1]=High	Humidity=High	Humidity=Normal	Humidity=Normal

[1] رطوبت

	Outlook[1]=Overcast	Outlook=Sunny	Outlook=Overcast	Outlook=Sunny
Rain	0.1	0.05	0.15	0.05
¬ Rain	0.2	0.15	0.1	0.2

می توانیم توزیع پیوسته ی احتمال را برای تشخیص *احتمال نهایی* [2] متغیّر وابسته ، با جمع کردن همه ی موارد وابسته به متغیّر که می توانند درست باشند به دست آوریم :

P(Rain)=0.1+0.05+0.15+0.05=0.35

اگر **P** ، یک **توزیع تصادفی** را برای هر متغیّر تصادفی نظیر **X** ، بیان نماید ؛ داریم :

$$P(X = x_i) = \sum_{\{\omega : X(\omega) = x_i\}} P(\omega)$$

مثال : در پرتاب یکبار یک تاس ، احتمال این که عدد فرد ، ظاهر شود برابر است با :

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 . برای متغیّرهای تصادفی بولین A و B ، اگر رویداد a ، مجموعه ای از فضاهای نمونه ای به شرط آن که A(ω)=true باشد ، آن گاه رویداد ¬a ، مجموعه ای از فضاهای نمونه ای به شرط آن که A(ω) = false خواهد بود و رویداد a∧b = مواردی هستند که A(ω) و B(ω) [هر دو] ، True می باشند .

---

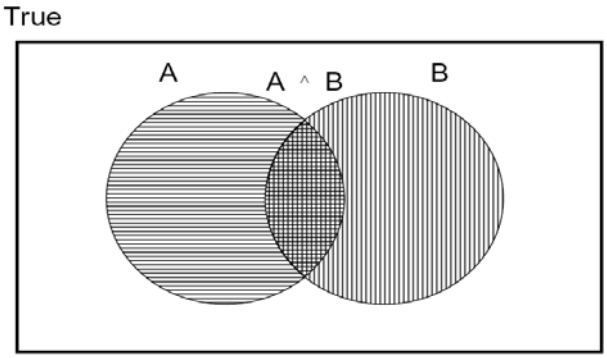
[1] پیش بینی
[2] marginal probability

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

اغلب در هوش مصنوعی ، *فضای نمونه ای ، به وسیله ی مقادیر یک مجموعه از متغیّرهای تصادفی تعریف می شود* . توجه نمایید که فضای نمونه ای حاصلضرب کارتزین محدوده هایی از متغیّرها می باشد . با متغیّرهای بولین ، فضای نمونه با مدل منطق گزاره ای ، برابر 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 : $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$

یک فرمول :

$P(A \vee \neg A) = P(A)+P(\neg A)-P(A \wedge \neg A) \Rightarrow P(True) = P(A)+P(\neg A)-P(False) \Rightarrow 1 = P(A) + P(\neg A) \Rightarrow \mathbf{P(A) = 1 - P(\neg A)}$

ظاهر گزاره ها

متغیّرهای تصادفی گزاره ای یا بولین : مثلا اگر Cavity برای آیا من کرم خوردگی دندان دارم ؟ ، باشد ؛ آن گاه ، Cavity = true یک گزاره می باشد .

متغیّرهای تصادفی گسسته (محدود یا نامحدود)

مثال : هوا (وضعیت جوی) [۱] ، دارای یکی از حالت های آفتابی ، بارانی ، ابری و برفی [۲] می باشد . هوا = بارانی ، یک گزاره می باشد .

متغیّرهای تصادفی پیوسته [۳] ، محدوده دار [۴] یا فاقد محدوده [۵] می باشند . مثال : دما = ۲۱.۶ ، یا Temp<22.0 .

احتمال شرطی [۶]

قبلا دیدیم که اگر هوا ابری باشد ، احتمال بارندگی بالا می رود ؛ که این یک احتمال شرطی نام دارد و به صورت P(Rain|Cloudy) نوشته می شود . برای توزیع های شرطی توجه نمایید که 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

---

[۱] Weather
[۲] snow
[۳] Continuous random variables
[۴] bounded
[۵] unbounded
[۶] conditional probability

$P(a|b) = \frac{P(a \wedge b)}{P(b)}$ ، با شرط $P(b) \neq 0$

با استفاده از فرمول بالا ، قانون ضرب [۱] را به صورت زیر خواهیم داشت :

$$P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a)$$

مثال :

P(Weather,Cavity)=P(Weather|Cavity)P(Cavity)

مثال : اگر $P(Rain \wedge Cloudy) = 0.15$ و $P(Cloudy) = 0.3$ باشد ؛ $P(Rain \mid Cloudy)$ چه قدر است ؟

$$P(Rain|Cloudy) = \frac{P(Rain \wedge Cloudy)}{P(Cloudy)} = \frac{0.15}{0.3} = 0.5$$

قانون زنجیره ای [۲] - برای کاربرد موفّقیّت آمیز قانون ضرب به وجود آمده است :

$$P(X_1,...,X_n) = P(X_1,...,X_{n-1})P(X_n \mid X_1,...,X_{n-1})$$
$$= P(X_1,...,X_{n-2})P(X_{n_1} \mid X_1,...,X_{n-2})P(X_n \mid X_1,...,X_{n-1})$$
$$= ... = \prod_{i=1}^{n} P(X_i \mid X_1,...,X_{i-1})$$

توزیع پیوسته [۳]

---

[۱] Product rule
[۲] Chain rule
[۳] joint distribution

برای k متغیّر تصادفی $X_1, \ldots, X_k$، توزیع پیوسته ی این متغیّرها ، جدولی است که در آن هر عـدد ، احتمال یک ترکیب از مقدارهای $X_1, \ldots, X_k$ را ارایه می نماید .

مثال :

	Toothache	¬Toothache
Cavity	0.04	0.06
¬Cavity	0.01	0.89

P(¬Cavity∧Toothache) P(Cavity∧¬Toothache)

با توجه به جدول بالا داریم :

P(Toothache) = P((Toothache ∧Cavity) v (Toothache∧¬Cavity))

= P(Toothache ∧Cavity) + P(Toothache∧¬Cavity) = 0.04 + 0.01 = 0.05
P(Toothache v Cavity) = P((Toothache ∧Cavity) v (Toothache∧¬Cavity)
v (¬Toothache ∧Cavity)) = 0.04 + 0.01 + 0.06 = 0.11

می توانیم همچنین احتمالات شرطی را محاسبه نماییم :

P(Cavity|Toothache) = P(Cavity∧Toothache) / P(Toothache)
P(Cavity∧Toothache) = ?

با توجه به جدول ، برابر است با ۰ . ۰۴

P(Toothache) = ?

حاصل این احتمال را هم که قبلا محاسبه کردیم ، برابر بود با : ۰ . ۰۵ ؛ درنتیجه داریم :

P(Cavity|Toothache) = 0.04/0.05 = 0.8

نرمال سازی

در فرمول احتمال شرطی ، مقسوم علیه [۱] می تواند به صورت یک ثابت نرمال سازی α دیده شود :

$$P(Cavity|Toothache) = P(Cavity \wedge Toothache) / P(Toothache) = \alpha\, [P(Cavity \wedge Toothache)] = \alpha \times 0.04 = \frac{1}{0.05} \times 0.04 = 20 \times \frac{4}{100} = 0.8$$

تعمیم

$$P(A \wedge B \wedge C) = P(A|B,C)\, P(B|C)\, P(C)$$

استدلال با شمارش

تصور نمایید X ، همه ی متغیّرها باشد . معمولاً ، ما توزیع پیوسته ی **متغیّرهای پرس و جوی** [۲] **Y** را می خواهیم که توسط مقادیر e برای **متغیّرهای ثابت** [۳] **E** ارایه شده اند . تصور نمایید متغیّرهـای مخفـی H=X-Y-E باشند . در این صورت ، مجموع تمام پیوند ها با جمع کردن متغیّرهای 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 هـم ، خروجـی آن هـا ، مجموعـه ای از متغیّرهای تصادفی می باشد .

مسایل آشکار :

۱) پیچیدگی زمانی در بدترین حالت در جایی که d بـزرگ تـرین مـی باشـد برابر است با $O(d^n)$

---

[۱] denominator

[۲] query variables

[۳] evidence variables

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 ؟؟؟

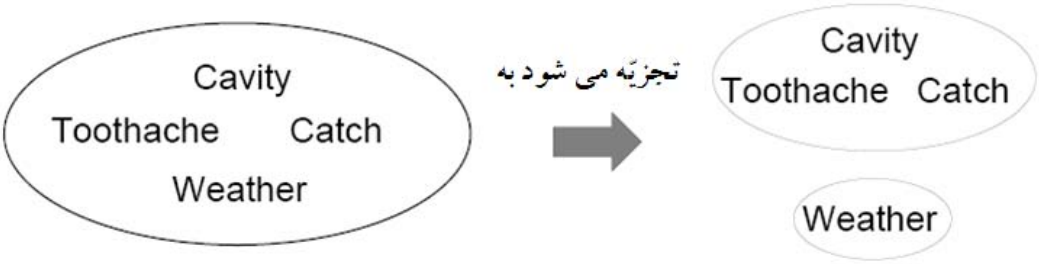
استقلال [۱]

در برخی از موارد ، ما می توانیم چیزها را با توجه به این که یک متغیّر دارای هیچ اثری بر روی دیگری نمی باشد ساده نماییم . برای مثال ، اگر ما یک متغیّر 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 :

$$P(Rain|Cloudy,Monday)=P(Rain|Cloudy,Tuesday)\ldots=P(Rain|Cloudy)$$

ما گفتیم که DayOfWeek و Rain مستقل می باشند . بنابراین 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 .

تعریف – A و B در صورتی مستقل هستند که ، $P(A|B)=P(A)$ یا $P(B|A)=P(B)$ یا $P(A,B)=P(A)P(B)$. با توجه به این تعریف برای مثال دندان درد داریم :



[۱] Independence

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 : $۲^n \rightarrow n$

استقلال مطلق ، خیلی مفید است ولی نادر یا کمیاب [۱] می باشد . مثلا ، دندانپزشکی ، زمینه ی بزرگی با صد ها متغیّر می باشد که هیچکدام مستقل نیستند . در این مورد چه کاری انجام بدهیم ؟

استقلال شرطی [۲]

اگر A و B مستقل شرطی باشند و C هم داده شده باشد ، داریم :

$$P(A,B|C) = P(A|C)P(B|C)$$

$$P(A|C,B) = P(A|C)$$

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 .

مثال اتومبیل – سه پارامتر گاز (Gas) ، باتری (Battery) و استارت (Start) را برای اتومبیل در نظر بگیرید ؛ داریم :

$$P(Battery|Gas) = P(Battery)$$

چون که گاز و باتری ، مستقل از هم هستند . ولی

$$P(Battery|Gas,Starts) \neq P(Battery|Starts)$$

---

[۱] rare

[۲] conditional independence

چون گاز و باتری با وجود استارت ، مستقل از هم نمی باشند .

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 :

$P(a \wedge b) = P(a \mid b)P(b) = P(b \mid a)P(a)$ ، در نتیجه قانون بیز را به صورت زیر خواهیم داشت :

$$P(a \mid b) = \frac{P(b \mid a)P(a)}{P(b)}$$

یا به شکل نرمال داریم :

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} = \alpha P(X \mid Y)P(Y)$$

مثال برای قانون بیز – این مفروضات را در نظر بگیرید : منژیت [1] باعث خشکی گردن [1] در ۵۰٪ از بیماران می شود ؛ P(stiffNeck|Meningities)=0.5 ؛ احتمال منژیت قدیمی برابر است با ۱/۵۰۰۰۰ ؛ P(meningitis)=0.00002 ؛ احتمال خشکی گردن قدیمی برابر است با ۱/۲۰ و P(stiffNeck)=0.05 . حال یک بیمار با خشکی گردن مراجعه می نماید . احتمال این که وی دارای منژیت باشد چقدر است ؟

$$P(meningitis \mid stiffNeck) = \frac{P(stiffNeck \mid meningitis)P(meningitis)}{P(stiffNeck)} = \frac{0.5 \times 0.00002}{0.05} = 0.0002$$

---

[1] meningitis

[1] stiff neck

مثال : جدول زیر را در نظر بگیرید ،

	Toothache	¬Toothache
Cavity	0.04	0.06
¬Cavity	0.01	0.89

اگر داشته باشیم : P(Cavity)=0.1 ، P(Toothache)=0.05 و P(Cavity|Toothache)=0.8 ؛ در این صورت P(Toothache|Cavity) را به دست آورید .

$$P(Toothache|Cavity) = (0.8 \times 0.05) / 0.1 = 0.4$$

تعمیم

$$P(A \wedge B \wedge C) = P(A \wedge B|C)\ P(C) = P(A|B,C)\ P(B|C)\ P(C)$$

$$P(A \wedge B \wedge C) = P(A \wedge B|C)\ P(C) = P(B|A,C)\ P(A|C)\ P(C)$$

$$P(B|A,C) = \frac{P(A|B,C)\ P(B|C)}{P(A|C)}$$

مثال : مدل ساده ی بیز [1]

[1] Naïve Bayes Model

در یک مدل قدیمی برای تشخیص بیماری ؛ علایمی که به صورت مستقل شرطی هستند برای بیماری داده شده اند ، بنابراین ، اگر $X_1,\ldots,X_n$ علایمی باشند که مریض گفته ؛ مثل ، سردرد ، تب بالا و غیره و H ، فرضیه هایی برای سلامتی بیمار باشد ، در این صورت ، داریم :

$$P(X_1,\ldots,X_n,H) = P(H)P(X_1|H)\ldots P(X_n|H)$$

این مدل ساده ی بیزی ، اجازه ی ارایه های فشرده را می دهد و فرضیه های مستقل نیرومندی دارد .

[۱]

فصل چهاردهم

شبکه های بیزی [۲]

[۱] تصویر مربوط به توماس بیز (Thomas Bayes) می باشد .

[۲] Bayesian networks

ریوس مطالب

- ساختار
- معنا
- توزیعات پارامتربندی شده [۱]

استنتاج احتمالی

در گذشته ، ما در مورد سیستم های برمبنای قانونی که می توانند استنتاج های منطقی را انجام دهند صحبت کردیم . مثلا اگر کبوتر خانگی [۲] ، گرسنه شود به بازار می رود :

---

[۱] parametrized distributions

[۲] Homer

Hungry(Homer) ⇒ GoesTo(Homer,Quickie-mart)

ما تمایل داریم این نوع از عملکرد را به جهان هایی با نامعلومی توسعه دهیم . مثلا :

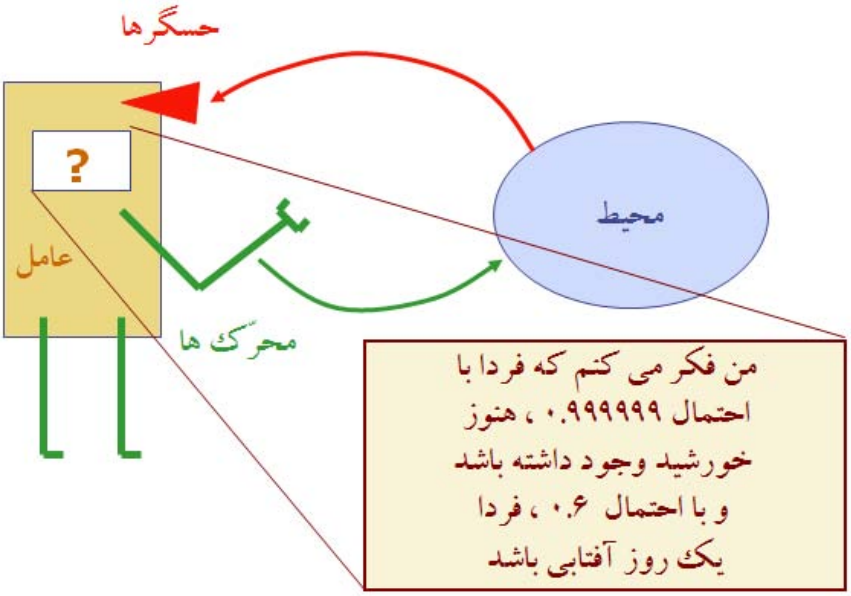
P(Hungry(Homer))=0.98

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 ، در این مورد ، مشکل در کار کردن با توزیع پیوسته ی احتمال می باشد ، چون دارای جدولی بزرگ به صورت نمایی می باشد . ما به ساختمان داده ای که در آن تعدادی متغیّر بر هم تاثیر نمی گذارند 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 .

عامل احتمالی



مسأله

در زمان مشخص t ، پایگاه دانش (KB) یک عامل ، مجموعه ای از حدس ها است ؛ در زمان t ، حسگرهای عامل ، یک دید را به وجود می آورند که احتمال یکی از حدس ها را افزایش می دهد ؛ ***حال سوال این است که در این موقع عامل باید چگونه احتمال موارد دیگر را به روز نماید ؟***

هدف شبکه های بیزی

توصیف یک مجموعه از حدس ها را با به وجود آوردن ارتباط های صریح میان حدس ها و پیدا کردن استقلال شرطی میان حدس ها آسان می کند و یک روش مناسب تر از استفاده از جدول های توزیع پیوسته را برای به روز کردن قدرت حدس ها در زمانی که شواهد جدید مشاهده می شوند ، به وجود می آورد .

نام های دیگر شبکه های بیزی

نام های دیگر شبکه های بیزی عبارتند از : شبکه های حدسی [۱] ، شبکه های احتمالی [۲] و شبکه های بیان کننده ی علت [۳] .

شبکه های بیزی

یک روش ساده و گرافیکی برای ادعاهای مستقل شرطی در یک نمایش فشرده برای توزیع پیوسته ی کامل است . به عبارت دیگر ، یک شبکه ی بیزی ، یک گراف مستقیم است که در آن هر گره با اطّلاعات احتمالی ، تفسیر می شود . یک شبکه دارای موارد زیر است :

---

[۱] belief networks

[۲] probabilistic networks

[۳] causal networks

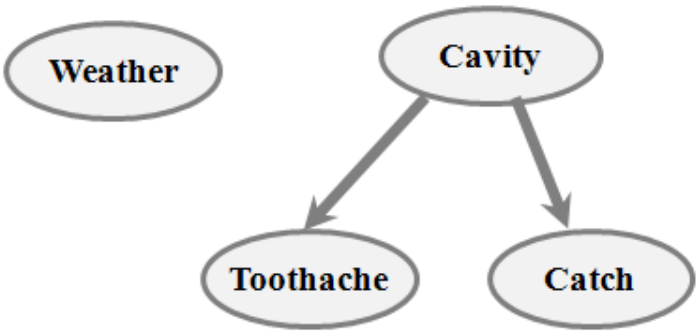
- یک مجموعه از متغیّرهای تصادفی که با گره های درون شبکه برابر است .
- یک مجموعه از یال ها (لبه ها یا پیکان ها) که گره ها را متصل می کنند . یال ها ، تاثیرات را نمایش می دهند . اگر پیکانی از X به طرف Y وجود داشته باشد ، آن گاه X ، پدر Y می باشد .

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 . در این شبکه ، هیچ حلقه ای وجود ندارد ؛ به عبارت دیگر ، این شبکه به صورت یک گراف مستقیم بدون دور می باشد .

پس ، یک شبکه ی بیزی ، یک نمایش ساده و گرافیکی برای موارد مستقل شرطی می باشد .

ساختار شبکه ی بیزی : در این ساختار هر متغیّر دارای یک گره است و شکل آن به صورت یک گراف غیرحلقه ای مستقیم می باشد . در ضمن ، 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 [1] ، توزیع را برمبنای X_i ، برای هر ترکیب از مقادیر والد ارایه می دهد .

مثال – توپولوژی شبکه ، موارد استقلال شرطی را بیان می کند :



[1] conditional probability table (CPT)

هوا مستقل از دیگر متغیّرها می باشد . دندان درد و دچار شدن به آن به صورت شرطی مستقل از پوسیدگی دندان می باشند .

مثال دزدی [۱] – من سرکار هستم و همسایه ام ، جان با من تماس می گیرد و می گوید آژیر خانه ی من فعال است ، اما همسایه ی دیگرم ، مری [۲] این کار را انجام نمی دهد . بعضی از وقت ها آژیر با کم ترین لرزه ای فعال می شود . آیا در خانه ی من واقعا دزد است ؟

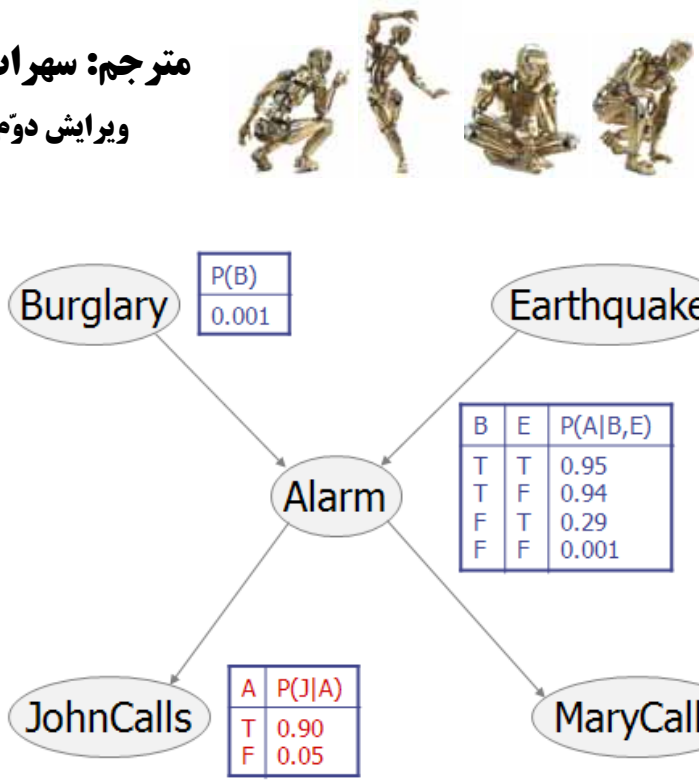
متغیّرها در این مثال عبارتند از : دزد ، لرزه [۳] ، آژیر [۴] ، تماس های جان [۵] ، تماس های مری [۶] .

توپولوژی شبکه آن چه که اتفاق افتاده است را برگشت می دهد (منعکس می کند) :

- 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 .

---

[۱] burglar
[۲] Mary
[۳] earthquake
[۴] alarm
[۵] JohnCalls
[۶] MaryCalls



تراکم [1]

شبکه های بیزی ، ارایه ی شبکه های بزرگ را امکان پذیر می سازند ؛ در این شبکه ها ، اطّلاعات اضافی (تکراری) [2] برداشته می شود .

یک CPT برای X_i بولین با والدهای بولین k دارای ۲^k سطر برای ترکیب مقدارهای 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 متغیّر دارای بیش از k والد نباشد ، شبکه ی کامل به تعداد $O(n.2^k)$ سطر 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 .

معانی عمومی

---

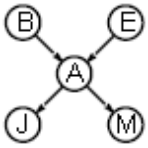
[1] compactness

[2] redundant

معانی کلی ، توزیع پیوسته ی کامل را به صورت ضرب توزیعات شرطی محلی ، تعریف می نماید :

$$P(x_1,\ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid parents\ (X_i))$$

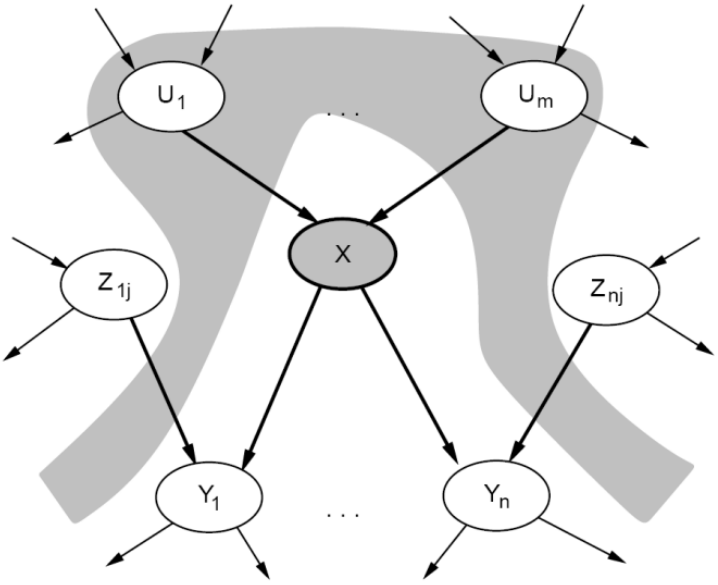
مثال :



$$P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) = P(j \mid a)P(m \mid a)P(a \mid \neg b, \neg e)P(\neg b)P(\neg e)$$
$$= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \approx 0.00063$$

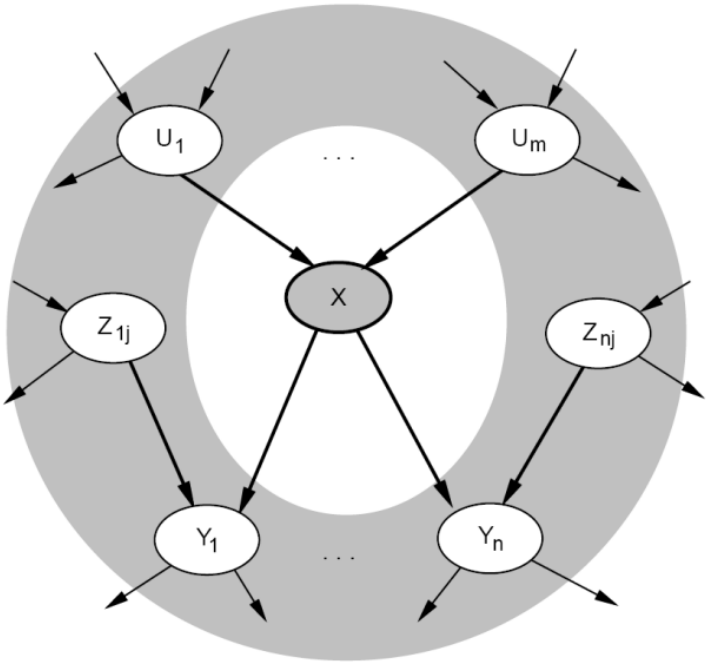
معانی محلّی

هر گره ، به صورت شرطی مستقل از فرزندانی است که توسط پدرشان به وجود آمده اند .



قضیّه : معانی محلی ⇔ معانی عمومی

پوسته ی مارکف [۱] – هر گره به صورت شرطی ، مستقل از همه ی گـره هـای دیگـری اسـت کـه توسط پوسته ی مارکوف ارایه می شوند ؛ پوسته ی مـارکوف تشـکیل شـده از : والـدین + فرزنـدان + والـدین فرزندان



ساختار شبکه

هر گره دارای یک جدول احتما ل شرطی است که این جدول ، احتمال هر مقـدار آن گـره و مقـادیر والدهای آن را ارایه می کند .

بیان نامعلومی (عدم قطعیّت) به طورخلاصه

---

[۱] Markov blanket

توجه نمایید که ما نیازی به داشتن گره هایی برای همه ی مواردی که مری ممکن است تماس نگیرد نداریم . یک روش احتمالی به ما اجازه می دهد که این اطّلاعات را به صورت $\neg M$ خلاصه نماییم . این روش ، به یک عامل ساده اجازه می دهد که به جهان های بزرگی که دارای تعداد زیادی از نتایج نامعلوم ممکن هستند رسیدگی کند .

ارایه ی ضمنی توزیع پیوسته ی کامل

به یاد بیاورید که توزیع پیوسته ی کامل به ما اجازه می داد که احتمال هر متغیّر و همه ی آن هایی که ارایه شده اند را محاسبه نماییم . رویدادهای مستقل می توانند به جدول های مستقل تقسیم شوند و این رویدادها ، CPT هایی هستند که در شبکه ی بیزی دیده می شوند . بنابراین ما می توانیم از این اطّلاعات برای انجام محاسبات استفاده نماییم . داریم :

$$P(x_1, x_2, \ldots, x_n) = \prod P(x_i \mid parents(x_i))$$

مثلا ؛

$$P(A \wedge \neg E \wedge \neg B \wedge J \wedge M) = P(J \mid A)P(M \mid A)P(A \mid \neg B \wedge \neg E)P(\neg B)P(\neg E) =$$
$$0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.00062$$

ساخت شبکه های بیزی

اغلب ، چند روش برای ساخت یک شبکه ی بیزی وجود دارد . در این مورد ، مهندسی دانش به کشف ارتباطات *مستقل شرطی* نیازمند است . والدهای یک گره باید آن هایی باشند که به طور مستقیم بر مقدار آن برتری دارند ؛ مثلا ، JohnCalls توسط لرزه دارای برتری می شود ، اما نه به طور مستقیم و تماس جان و ماری دارای برتری بر یکدیگر نمی باشند . به صورت صریح ، ما فرض می کنیم :

P(MaryCalls|JohnCalls,Alarm,Earthquake,Burglary)=P(MaryCalls|Alarm)

در زیر ، روشی برای ساخت شبکه های بیزی آمده است :

۱) یک مجموعه ی منظم از متغیّرهای $X_1,...,X_n$ را انتخاب نمایید .

۲) برای i=1 تا n کارهای زیر را انجام بده

X_i را به شبکه اضافه کن . والدهایی از $X_1,...,X_{i-1}$ را انتخاب کن که

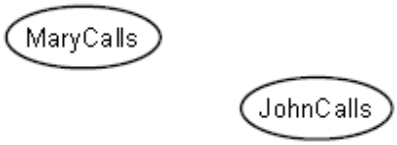
$$P(X_i \mid Parents(X_i)) = P(X_i \mid X_1,..., X_{i-1})$$

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 [۱])

$$= \prod_{i=1}^{n} P(X_i \mid Parents(X_i))$$

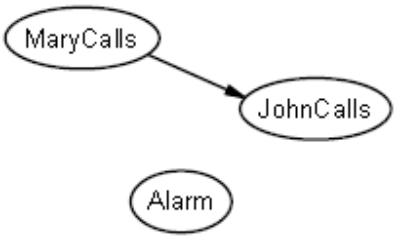
مثال – تصور کنید که ما مجموعه ی M,J,A,B,E را به ترتیب ، انتخاب می نماییم :



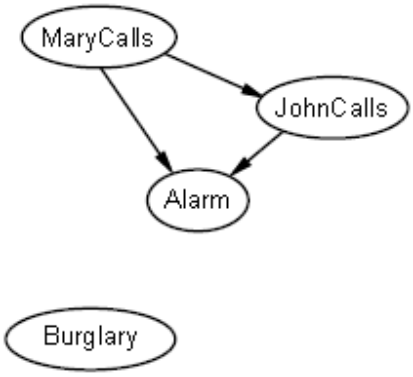
P(J|M)=P(J)? **نه**

---

[۱] chain rule

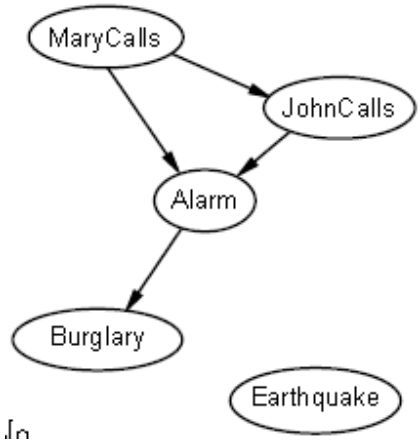


P(A|J,M)=P(A|J)? P(A|J,M)=P(A) ? نه



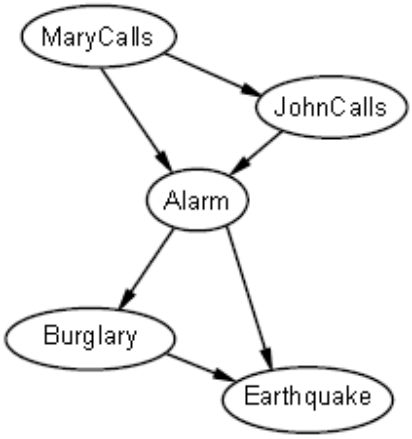
P(B|A,J,M)=P(B|A) ?بله

P(B|A,J,M)=P(B) ?نه



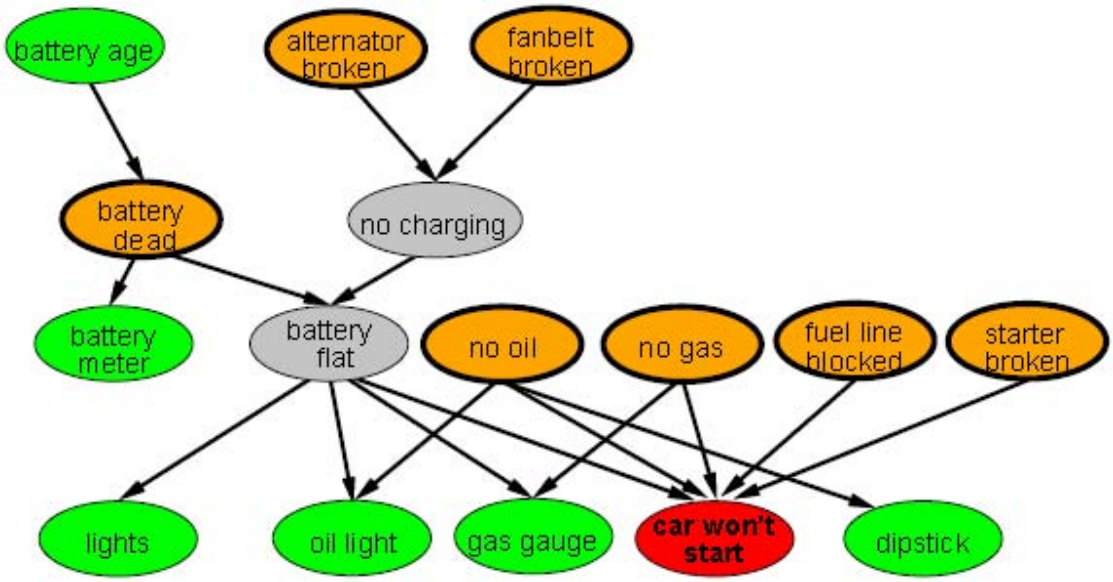
$P(E|B,A,J,M)=P(E|A)$? نه

$P(E|B,A,J,M)=P(E|A,B)$? بله

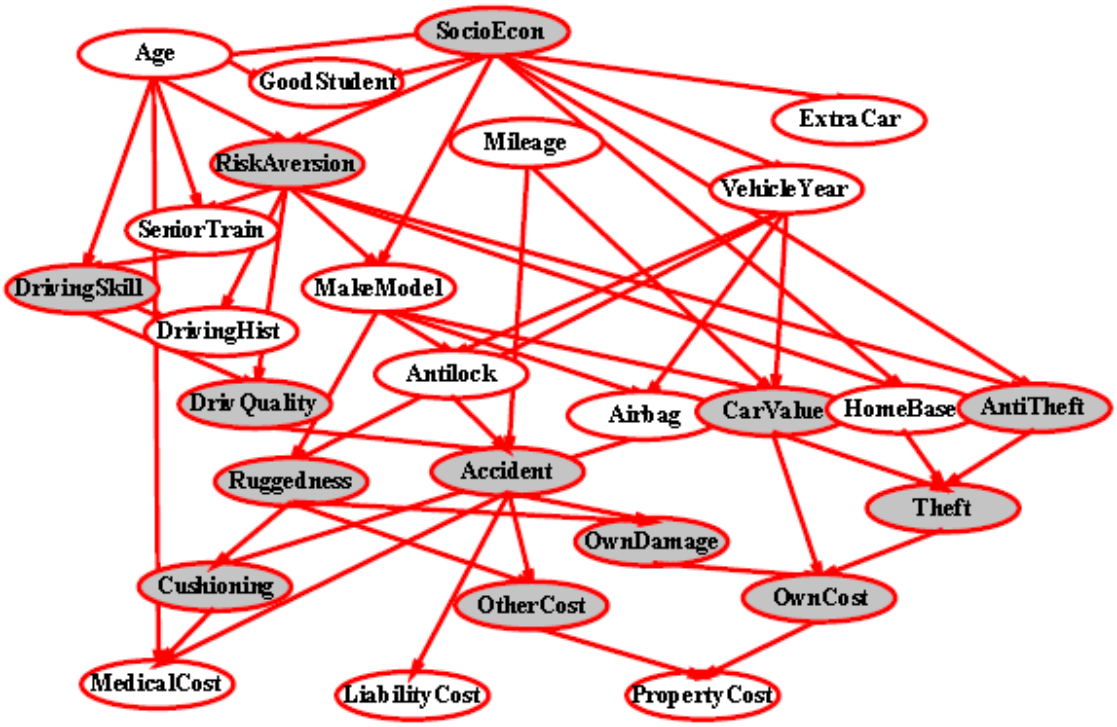


مثال : عیب اتومبیل

دلیل اوّلیه : ماشین روشن نمی شود . متغیّرهای قابل آزمایش در شکل زیر با رنگ سبز نشان داده شده اند ؛ متغیّرهایی که " شکسته اند و بنابراین آن ها را تعمیر نماییم " با رنگ نارنجی در شکل زیر نمایش داده شده اند ؛ برای متغیّر های مخفی (خاکستری) ، قطعات وابسته را چک نمایید و به این طریق پارامترها را کاهش دهید .



مثال : بیمه ی اتومبیل



ساخت یک شبکه

اوّل ، ریشه ها را به وجود بیاورید ؛ دوّم ، متغیّرهای مستقیم را اضافه نمایید ؛ سوّم ، فرزندان مستقیم آن ها را اضافه نمایید ؛ به این ترتیب ، شما یک *مدل سببی* [1] را به وجود آورده اید . تخمین ها با این روش به مراتب آسان تر به دست می آیند . برای ساخت می توانیم از اثر به سبب هم تلاش نماییم ، امّا در این حالت ، شبکه به مراتب پیچیده تر خواهد بود .

کاربردهای شبکه های بیزی

عبارتند از : تشخیص ، که به طور گسترده ای در محصولات میکروسافت استفاده می شود ؛ تشخیص طبی ؛ فیلتر نمودن نامه های الکترونیکی که ناشناس می باشند [1] ؛ کاربرد در سیستم های خبره (کنترل وسایل ،مانیتورینگ) و کنترل روبوتیک .

---

[1] causal model

[1] spam filtering

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل پانزدهم

استنتاج در شبکه های بیزی

E را متغیّرهای مدرکی که دیده شده اند قرار دهید ؛ به عنوان مثال {JohnCalls,MaryCalls} .
X را متغیّر پرس و جو قرار دهید ؛ به عنوان مثال ، Burglary و فرض کنید می خواهیم P(X|E) را به دست آوریم .

J	M	P(B|...)
T	T	?

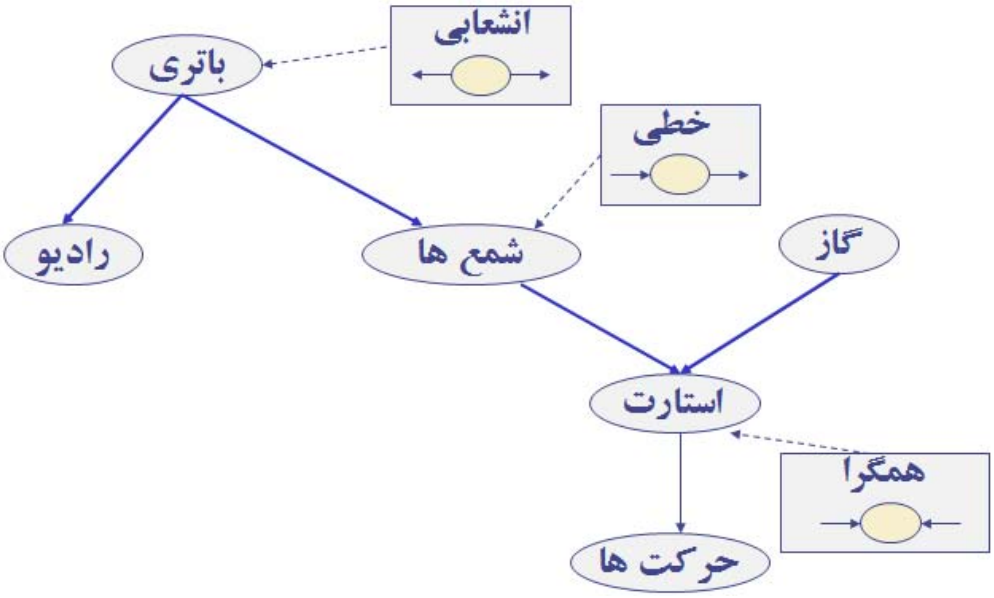
کاربردهای شبکه های بیزی

کاربرد اصلی یک شبکه ی بیزی : ارایه ی مشاهدات جدید و محاسبه ی توانایی های جدید برخی (یا همه ی) تصورّات [1] .

[1] beliefs

کاربرد دیگر : ارایه ی توانایی یک تصور و این که کدام مشاهده را باید برای به وجود آوردن بیش ترین تغییر در توانایی تصور به کار ببریم .

انواع گره های موجود در یک مسیر



با داشتن یک مجموعه ی E از گره های مدرک ، دو گره که توسط یک مسیر غیر مستقیم به هم وصل شده اند در صورتی *مستقل* هستند که دارای یکی از سه شرط زیر باشند :

۱- یک گره ی موجود در مسیر به صورت خطی [۱] و در E باشد .
۲- یک گره ی موجود در مسیر به صورت انشعابی [۲] و در E باشد .
۳- یک گره ی موجود در مسیر به صورت همگرا باشد و نه این گره و نه هیچ یک از فرزندانش در E باشند .

---

[۱] linear
[۲] diverging

گاز و رادیو با توجه به مدرک شمع ها [1] مستقل هستند . گاز و رادیو با توجه به مدرک باتری مستقل هستند .

استنتاج در شبکه های بیزی

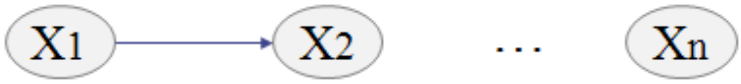
در ساده ترین حالت داریم :

$$P(B) = P(a)P(B|a) + P(\sim a)P(B|\sim a)$$

$$P(B) = \sum_{A} P(A)P(B \mid A)$$

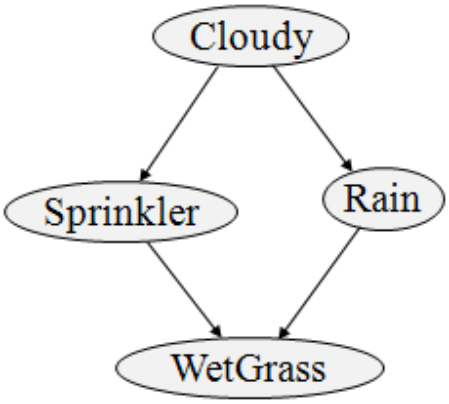
حال سوال این جاست که اگر شکل زیر را داشته باشیم ، آن گاه P(C) برابر چیست ؟

اگر یک زنجیره به صورت زیر داشته باشیم پیچیدگی زمانی برای محاسبه ی P(Xn) چه قدر است ؟



مثال :

[1] Spark plugs



$$P(W) = \sum_{R,S,C} P(w \mid r,s) P(r \mid c) P(s \mid c) P(c)$$

$$= \sum_{R,S} P(w \mid r,s) \sum_{C} P(r \mid c) P(s \mid c) P(c)$$

اگر $\sum_{C} P(r \mid c) P(s \mid c) P(c)$ را $f_c(R,S)$ در نظر بگیریم داریم :

$$P(W) = \sum_{R,S} P(w \mid r,s)\, f_C(r,s)$$

الگوریتم در حال محاسبه ی احتمال ها به صورت تکی نمی باشد ، بلکه در حال محاسبه ی تمام موارد است . دو ایده برای جلوگیری از انفجار نمایی محاسبه ها وجود دارد :

۱- با توجه به ساختار شبکه ی بیزی ، برخی از زیر عبارت ها در محل های اتصال فقط به تعداد کمی از متغیّرها وابسته اند .

۲- با یکبار محاسبه ی آن ها و نگهداری نتایج ، ما می توانیم از به وجود آمدن آن ها به صورت نمایی در بسیاری از موارد جلوگیری نماییم .

روش حذف متغیّر

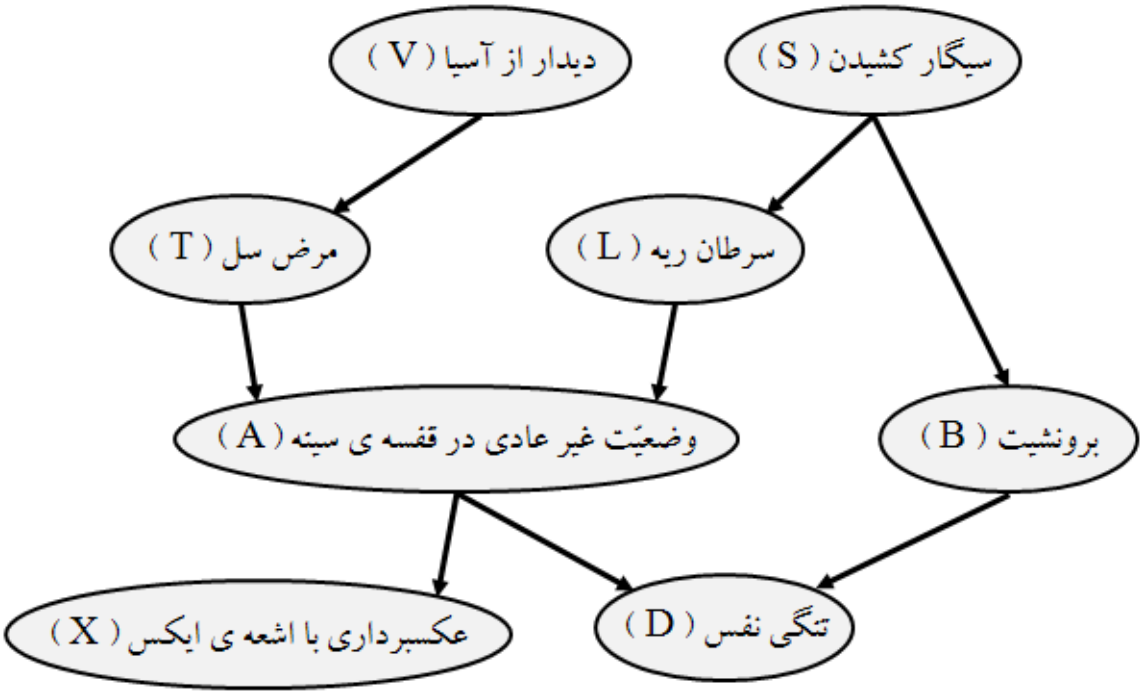
ایده ی اصلی این است که پرس و جو را به صورت زیر بنویسیم :

$$P(X_n, \boldsymbol{e}) = \sum_{x_k} \cdots \sum_{x_3} \sum_{x_2} \prod_i P(x_i \mid pa_i)$$

و به صورت تکراری کار های زیر را انجام دهیم :

۱- تمام عبارت های نامربوط را به خارج از داخلی ترین جمع انتقال دهیم .
۲- داخلی ترین جمع را انجام دهیم و عبارت جدید را به دست آوریم .
۳- عبارت جدید را در ضرب وارد نماییم .

یک مثال پیچیده تر



ما می خواهیم احتمال تنگی نفس (P(D)) را محاسبه نماییم . باید v , s , x , t , l , a , b را حذف نماییم . عامل های اوّلیه عبارتند از :

$$P(v)P(s)P(t|v)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

برای حذف v داریم :

$$f_v(t) = \sum_v P(v)P(t|v)$$

در نتیجه داریم :

$$f_v(t)P(s)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

توجّه کنید که $f_v(t) = P(t)$. در حالت کلّی ، نتیجه ی حذف لزوماً یک عبارت احتمالی نمی باشد .

برای حذف s داریم :

$$f_s(b,l) = \sum_s P(s)P(b|s)P(l|s)$$

در نتیجه :

$$f_v(t)\, f_s(b,l)P(a|t,l)P(x|a)P(d|a,b)$$

در جمع بندی بر روی نتایج s که دارای یک عامل و دو آرگومان $f_s(b,l)$ است ، در حالت کلّی نتیجه ی حذف ممکن است یک تابع با چند متغیّر باشد .

برای حذف x داریم :

$$f_x(a) = \sum_x P(x \mid a)$$

در نتیجه داریم :

$$f_v(t)\, f_s(b,l)\, f_x(a) P(a \mid t,l) P(d \mid a,b)$$

توجه کنید که $f_x(a)$ برای همه ی مقدارهای a برابر با یک می باشد .

برای حذف t داریم :

$$f_t(a,l) = \sum_t f_v(t) P(a \mid t,l)$$

در نتیجه داریم :

$$f_s(b,l)\, f_x(a)\, f_t(a,l) P(d \mid a,b)$$

برای حذف l داریم :

$$f_l(a,b) = \sum_l f_s(b,l)\, f_t(a,l)$$

در نتیجه داریم :

$$f_l(a,b)\, f_x(a) P(d \mid a,b)$$

برای حذف a و b داریم :

$$f_a(b,d) = \sum_a f_l(a,b)\, f_x(a)\, p(d \mid a,b) \qquad f_b(d) = \sum_b f_a(b,d)$$

ما حالا فهمیدیم که حذف متغیّر به صورت یک رشته از عملیّات ***بازنویسی*** می باشد ، محاسبه ی واقعی در مرحله ی حذف انجام می شود و محاسبه ، وابسته به ترتیب حذف می باشد .

رسیدگی به مدارک

تصوّر کنید که مدارک $V = t, S = f, D = t$ را داریم و می خواهیم $P(L, V = t, S = f, D = t)$ را محاسبه نماییم ؛ با نوشتن عامل ها شروع می کنیم :

$$P(v)P(s)P(t\,|\,v)P(l\,|\,s)P(b\,|\,s)P(a\,|\,t,l)P(x\,|\,a)P(d\,|\,a,b)$$

از آنجایی که ما می دانیم که $V = t$ می باشد ، نیازی به حذف V نداریم و به جای این کار می توانیم عامل های $P(V)$ و $P(T|V)$ را به صورت زیر جایگزین نماییم :

$$f_{P(V)} = P(V = t) \quad f_{p(T|V)}(T) = P(T\,|\,V = t)$$

که این ها اجزای مناسب عامل های اصلی را که مدارک بیان می کنند انتخاب می کنند . توجّه کنید که $f_{p(V)}$ یک ثابت است و بنابراین در حذف دیگر متغیّرها ظاهر نمی شود .

عامل های اوّلیه پس از تنظیم مدرک ها به صورت زیر خواهند بود :

$$f_{P(v)}\, f_{P(s)}\, f_{P(t|v)}(t)\, f_{P(l|s)}(l)\, f_{P(b|s)}(b) P(a\,|\,t,l) P(x\,|\,a)\, f_{P(d|a,b)}(a,b)$$

پس از حذف x داریم :

$$f_{P(v)}\, f_{P(s)}\, f_{P(t|v)}(t)\, f_{P(l|s)}(l)\, f_{P(b|s)}(b) P(a\,|\,t,l)\, f_x(a)\, f_{P(d|a,b)}(a,b)$$

با حذف t داریم :

$$f_{P(v)}\, f_{P(s)}\, f_{P(l|s)}(l)\, f_{P(b|s)}(b)\, f_t(a,l)\, f_x(a)\, f_{P(d|a,b)}(a,b)$$

پس از حذف a داریم :

$$f_{P(v)} f_{P(s)} f_{P(l|s)}(l) f_{P(b|s)}(b) f_a(b,l)$$

با حذف b داریم :

$$f_{P(v)} f_{P(s)} f_{P(l|s)}(l) f_b(l)$$

الگوریتم حذف متغیّر

تصوّر کنید X_1 , X_2 , ... , X_m رشته ای از متغیّرهای غیر پرس و جو باشند :

$$\sum_{X_1}\sum_{X_2}\cdots\sum_{X_m}\prod_j P(X_j \mid Parents(X_j))$$

برای I مساوی با m تا ۱ کارهای زیر را انجام بده :

- در مجموع یابی برای X_i فقط عامل هایی که X_i را بیان می کنند ، نگهدارید .
- عامل ها را ضرب کنید ، یک عامل را که شامل یک شماره ی X_i برای هر مقدار متغیّرهای مذکور X_i هستند را به دست آورید .
- جمع را انجام دهید و یک عامل f را که شامل یک عدد برای هر مقدار متغیّرهای ذکر شده که شامل X_i نمی باشند را به دست آورید .
- عامل ضرب شده را در جمع جایگزین نمایید .

پیچیدگی حذف متغیّر

فرض کنید در یک مرحله ی حذف ما عبارت زیر را محاسبه می کنیم :

$$f_x(y_1, \dots, y_k) = \sum_x f'(x, y_1, \dots, y_k)$$

$$f'_x(x, y_1, \dots, y_k) = \prod_{i=1}^{m} f_i\,(x, y_{1,1}, \dots, y_{1,l_i})$$

که به $m \cdot |\mathrm{Val}(X)| \cdot \prod_i |\mathrm{Val}(Y_i)|$ ضرب نیاز دارد . برای هر مقدار $x, y_1, \dots, y_k$ ، ما به m ضرب نیاز داریم .

روش های استنتاج

به دو دسته ی استنتاج دقیق [۱] و استنتاج تقریبی [۲] تقسیم می شوند . استنتاج در زنجیره های ساده ، حذف متغیّر و الگوریتم های طبقه بندی درختی [۳] ، جزو روش های استنتاج دقیق هستند و شبیه سازی احتمالی یا تصادفی [۴] یا روش های نمونه [۵] برداری و روش های زنجیره ی مارکوف مونت کارلو [۶] جزو روش های استنتاج تقریبی می باشند .

شبیه سازی تصادفی مستقیم

فرض کنید که شما دارای مقادیری برای برخی از زیرمجموعه های متغیّرها (G) هستید و می خواهید مقدارهایی را برای متغیّرهای ناشناخته (U) به دست آورید . تعداد زیادی از نمونه ها را از شبکه ی

---

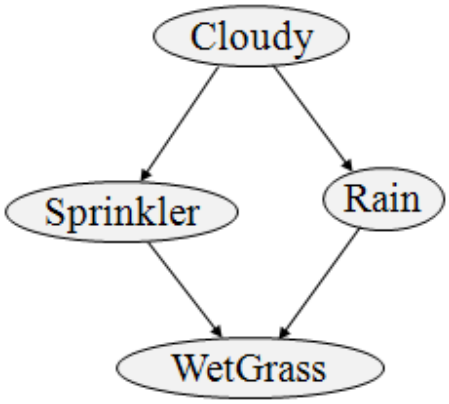
[۱] Exact inference
[۲] Approximate inference
[۳] Clustering tree algorithms
[۴] Stochastic simulation
[۵] sampling methods
[۶] Markov chain Monte Carlo methods

بیزی به صورت تصادفی به وجود بیاورید ؛ نمونه ها را برای همه ی متغیّرها به وجود آورید ؛ از متغیّرهای ریشه شروع کنید و این کار را به صورت مستقیم (رو به جلو) [۱] ادامه دهید .

ردّ نمونه برداری : نمونه هایی که با مقدارهای G سازگارند را نگه دارید .

مقدارها را برای U به کار ببرید تا احتمال های تخمین زده شده را به دست آورید . دقّت نتیجه ها به تعداد نمونه ها بستگی دارد .

مثال : با داشتن شکل زیر ، P(WetGrass | Cloudy) چیست ؟



$$P(WetGrass|Cloudy) = P(WetGrass \wedge Cloudy) / P(Cloudy)$$

۱. کارهای زیر را N بار انجام دهید :

۱.۱. فرض کنید به تصادف هوا ابری [۲] است .

۱.۲. برای هر فرض ابری بودن هوا ، آبپاش [۱] و باران [۲] را فرض کنید و سپس خیس بودن چمن [۳] را فرض نمایید .

---

[۱] forward

[۲] Cloudy

۲. نسبت تعداد مواردی که خیس بودن چمن و ابری بودن هوا درست است را در صورتی ابری بودن هوا درست باشد محاسبه نمایید .

وزن احتمال [۴]

ایده ی این روش این است که نمونه هایی که لازم است در مرحله ی اوّل رد شوند را تولید ننمایید ؛ فقط از متغیّرهای ناشناخته ی Z نمونه بگیرید و هر نمونه را با داشتن مدرک E با توجّه به احتمالی که رخ می دهد مورد ارزیابی قرار دهید .

الگوریتم زنجیره ی مارکوف مونت کارلو

در زنجیره ی مارکوف هر مورد تولید شده در نمونه ، وابسته به نمونه ی قبلی می باشد و در روش مونت کارلو از روش نمونه برداری آماری استفاده می شود .

یک پیمایش تصادفی را در میان فضای متغیّر انجام دهید و آمارها را در موقع جلو رفتن نگه دارید ؛ برای این کار با یک نمونه ی تصادفی که با متغیّرهای مدرک سازگار است شروع کنید و در هر مرحله ، برای برخی از متغیّرهای غیر مدرک که با دیگر انتساب های جاری سازگارند ، به صورت تصادفی از مقدارش نمونه برداری نمایید . با داشتن نمونه های کافی ، زنجیره ی مارکوف مونت کارلو (MCMC) یک تخمین دقیق از توزیع درست مقدارها را ارایه می کند .

---

[۱] Sprinkler

[۲] Rain

[۳] WetGrass

[۴] Likelihood weighting

فصل شانزدهم

شناخت سخن یا سخن شناسی[۱] (به طور خلاصه)

[۱] Speech recognition

ریوس مطالب

- **سخن به صورت استدلال احتمالی** [۱]
- **صداهای سخن** [۲]

سخن شناسی

عملیّات لازم برای توانمندسازی یک کامپیوتر برای شناسایی و واکنش دادن به صداهای به وجود آمده در سخن انسان می باشد [۳]. به عنوان تعریفی دیگر ، سخن شناسی یا تشخیص صدا [۴] ، توانایی سیستم های کامپیوتری برای دریافت سخن به صورت ورودی و پردازش بر روی آن یا بیان آن به صورت نوشته می

---

[۱] speech as probabilistic inference

[۲] speech sounds

[۳] فرهنگ آکسفورد / Babylon

[۴] voice recognition

باشد . کاربردهای عملی سخن شناسی ، شامل سیستم های پایگاه داده – پرس و جو [۱] و سیستم های بازیابی اطّلاعات [۲] می باشد . سخن شناسی دارای کاربرد در روبوتیک و مخصوصاً توسعه ی روبوت هایی که می توانند "بشنوند" می باشد . [۳]

سخن به صورت استدلال احتمالی

سیگنال های سخن ، پارازیت دار (اغتشاش دار) [۴] ، متغیّر و مبهم [۵] می باشند . شبیه ترین ترتیب کلمات و سیگنال سخن ارایه شده چیست ؟ ، برای این کار از قانون بیز استفاده نمایید :

$$P(Words|signal)=\alpha P(signal|Words)P(Words)$$

توجّه کنید که ، سخن ، به مدل صوتی [۶] و مدل زبانی [۷] ، تجزیه می شود . **کلمات** [۸] ، ترتیب وضعیّت های پنهان می باشند و **سیگنال** [۹] ، ترتیب مشاهده می باشد . در مورد **اصوات** [۱۰] ، باید بدانیم که ، تمام سخنان بشر ترکیبی از ۴۰ الی ۵۰ صوت می باشد . برای یک سطح میانی وضعیّت های مخفی شده ی

---

[۱] database-query systems
[۲] information retrieval systems
[۳] دایره المعارف بریتانیکا
[۴] noisy
[۵] ambiguous
[۶] acoustic model
[۷] language model
[۸] words
[۹] signal
[۱۰] phones

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

میان کلمات و سیگنال ها داریم ، مدل صوتی [۱] = مدل تلفظ [۲] + مدل صوت [۳]. آرپابت [۴] تعیین 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
⋮	⋮	⋮	⋮	⋮	⋮

برای مثال ، برای کلمه ی " ceiling " داریم : [s iy l en] / [s iy l ix ng] / [s iy l ih ng]

---

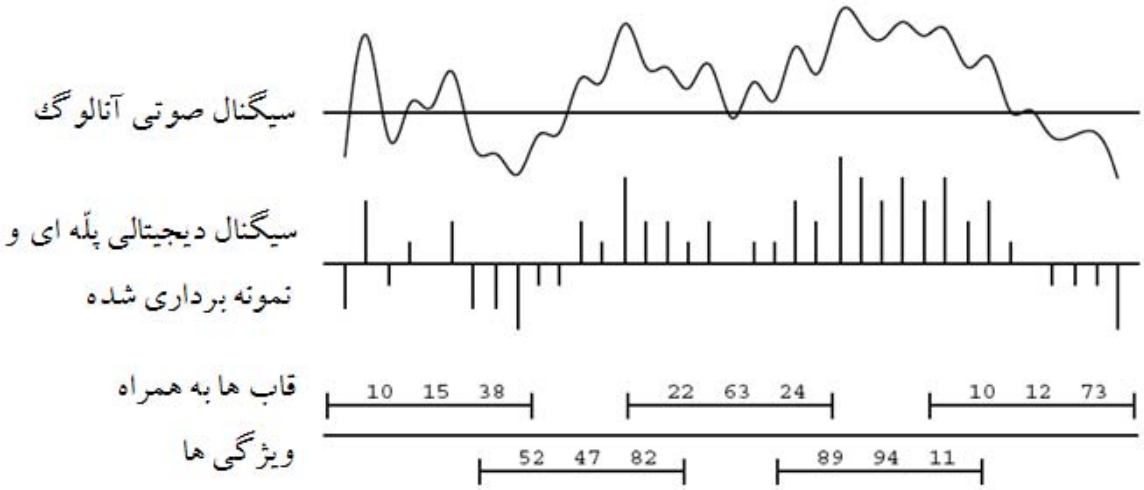
[۱] acoustic model

[۲] pronunciation model

[۳] phone model

[۴] ARPAbet

صداهای سخن – 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 .



پستی و بلندی های قاب با فرمت های معمولی می باشند و قله [۱] ها دارای طیف [۲] قوی می باشند .

اصوات سه حالته [۳] : هر صوت دارای سه وجه می باشد (آغاز [۴] ، وسط [۵] ، پایان [۶]) ، به عنوان مثال ، حرف [t] دارای ابتدای آرام [۷] ، وسط قوی [۱] ، انتهای خشن [۲] می باشد .

---

[۱] peak
[۲] spectrum
[۳] Three-state phones
[۴] Onset
[۵] Mid
[۶] End
[۷] silent Onset

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل هفدهم

تصمیم گیری های عاقلانه [۳]

(تیوری تصمیم گیری [۱])

---

[۱] explosive Mid

[۲] hissing End

[۳] Rational decisions

ریوس مطالب

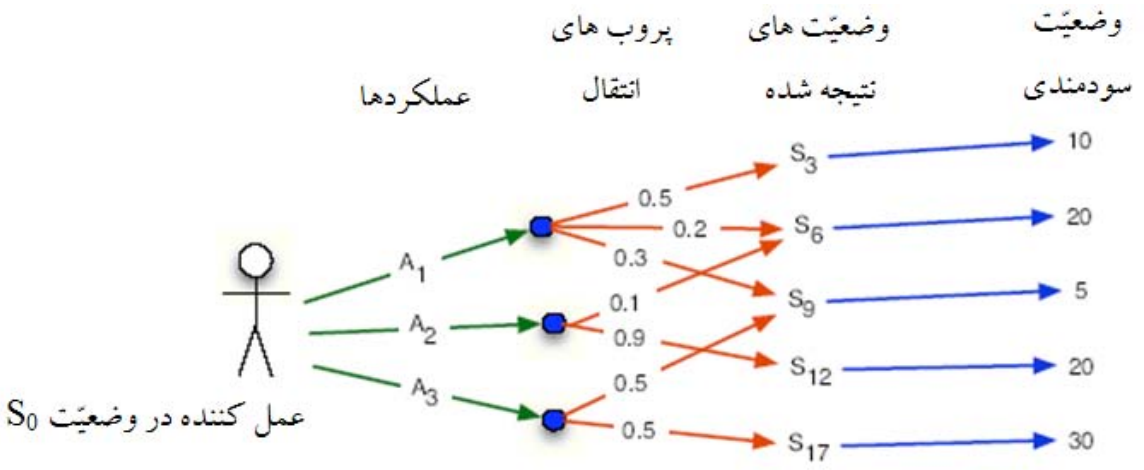
- تقدم های عاقلانه [۲]
- تسهیلات [۳]
- پول [٤]
- تسهیلات چند خصوصیتی [٥]
- شبکه های تصمیم گیری [٦]
- ارزش اطّلاعات [۱]

---

[۱] Decision theory
[۲] Rational preferences
[۳] utilities
[۴] Money
[۵] Multiattribute utilities
[۶] Decision networks

تصمیم گیری عاقلانه

تصمیم گیری ای عاقلانه است که منجر به انجام کار درست می شود ؛ کار درست را انجام دهید ؛ یعنی ، عملی که سود مورد انتظار را بیشینه می نماید را انتخاب نمایید . اصل بیش ترین سودمندی مورد انتظار [۲] ، می گوید که یک عامل هوشمند باید کاری که سودمندی مورد انتظار [۳] عامل را بیشینه می کند انتخاب نماید .



$$EU(A_1) = (0.5)(10) + (0.2)(20) + (0.3)(5) = 10.5$$

$$EU(A_2) = (0.1)(20) + (0.9)(20) = 20 \text{ , } EU(A_3) = (0.5)(5) + (0.5)(30) = 17.5$$

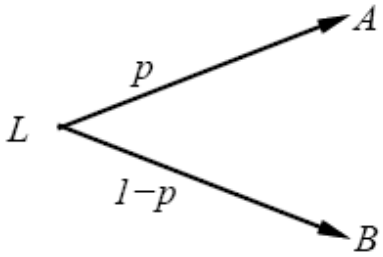
---

[۱] Value of information

[۲] Maximum Expected Utility (MEU)

[۳] Expected Utility (EU)

تقدّم ها – تیوری سودمندی [۱] به تقدّم ها توجه می کند . در این مورد ، حالت ها ، جایزه ها (A ، B و غیره) می باشند . نتیجه های ممکن ، مثل قرعه کشی هایی با جایزه های نامعلوم می باشند .



قرعه کشی L برابر است با : [p,A; (1-p),B] .

یادداشت : عبارت $A \succ B$ ؛ یعنی ، A بر B مقدّم است ؛ عبارت $A \sim B$ ؛ یعنی ، بی تفاوتی میان A و B و عبارت $A \succsim B$ ؛ یعنی ، B بر A مقدّم *نمی باشد* .

اولویّت های عاقلانه

اولویّت های عامل هوشمند ، تابع محدودیّت ها می باشد که محدودیّت ها عبارتند از :

توانایی نظم دهی [۲] $(A \succ B) \vee (B \succ A) \vee (A \sim B)$

انتقال پذیری [۳] $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

پیوستگی [۱] $A \succ B \succ C \Rightarrow \exists p[p, A; 1-p, C] \sim B$

---

[۱] utility theory

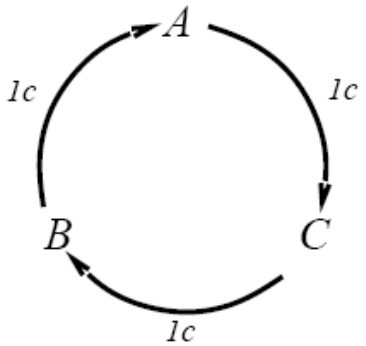
[۲] Orderability

[۳] Transitivity

تعویض پذیری [۲] $A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$

یکنواختی [۳] $A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1-p, B] \succsim [q, A; 1-q, B])$

تخلّف از محدودیّت ها ، منجر به نمودها یا آشکارسازی نامعقولانه می شود ، برای مثال : یک عامل با اولویّت های لازم می تواند منجر به از دست دادن همه ی پول شود :



در صورتی که $B \succ C$ باشد ، آن گاه یک عامل که دارای C می باشد باید یک سنت [۴] برای دریافت B پرداخت نماید . در صورتی که $A \succ B$ باشد ، آن گاه یک عامل که دارای B می باشد باید یک سنت برای دریافت A بپردازد . در صورتی که $C \succ A$ باشد ، آن گاه یک عامل که دارای A می باشد باید یک سنت برای دریافت C بپردازد .

بیشینه کردن سودمندی (تسهیلات) مورد انتظار

---

[۱] Continuity

[۲] Substitutability

[۳] Monotonicity

[۴] cent که معادل یک صدم دلار آمریکایی می باشد

اصل های تیوری سودمندی در مورد اولویّت ها صحبت می کنند . تابع سودمندی از اصل های سودمندی پیروی می کند .

قضیّه (رمزی [۱] ، ۱۹۳۱ ؛ ون نیومن [۲] و مورگنسترن [۳] ، ۱۹٤٤) : اگر اولویّت های یک عامل از اصل های سودمندی ، پیروی کند ، در این صورت یک تابع با مقدار حقیقی U وجود دارد که :

اصل بیش ترین سودمندی مورد انتظار [٤] : سودمندی یک قرعه کشی ، مجموع سودمندی های هر نتیجه و مطابق با احتمال زیر می باشد :

$$U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i U(S_i)$$

که با قانون تصمیم گیری با بیش ترین سودمندی مورد انتظار ، هم تراز است .

قانون بیش ترین سودمندی مورد انتظار (MEU) : عملکردی را که سود مورد نیاز را بیشینه می کند را انتخاب نمایید .

نکته : یک عامل می تواند کاملاً عاقلانه باشد (سازگار با قانون MEU باشد) بدون حتّی ارایه یا به کارگیری سود و احتمالات . مثلاً ، یک جدول انتخابی [۵] برای تیکتاکتوی کامل .

---

[۱] Ramsey
[۲] von Neumann
[۳] Morgenstern
[۴] MEU principle
[۵] lookup table

سودمندی پول

تیوری سودمندی دارای ریشه هایی در اقتصاد است . آیا پول می تواند به صورت یک معیار سودمندی مورد استفاده قرار گیرد ؟ .

به عنوان مثال ، فرض کنید که شما در یک مسابقه ی تلویزیونی مبلغ یک میلیون دلار ، برنده می شوید . برای ادامه ی بازی شما باید یک سکّه را به هوا پرتاب نمایید ؛ اگر شیر آمد ، مبلغ سه میلیون دلار می گیرید ، ولی اگر خط آمد تمام پولی را که قبلاً برده اید از دست می دهید . در این موقع ، شما چه کاری را انجام خواهید داد ؟ . شما می توانید در این مورد ، بازی را به صورت زیر بیان نمایید :

$L : [0.5, Heads(triple); 0.5, Tails(loose)]$

$EU(L) = 0.5 \times 3,000,000 + 0.5 \times 0 = 1,500,000$

استاندارد نزدیک به صرفه های بشر

اگر در یک بازی قمار روسی [۱] شانس شما برای کشته شدن ، یک میلیونیم باشد ، آیا در این بازی شرکت می کنید . قمار روسی ، یک بازی خطرناک شانسی است 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 . [۲]

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) معمولاً U(L)<U(EMV(L)) می باشد ، توجه کنید که افراد مخالف ریسک هستند .

---

[۱] Russian roulette

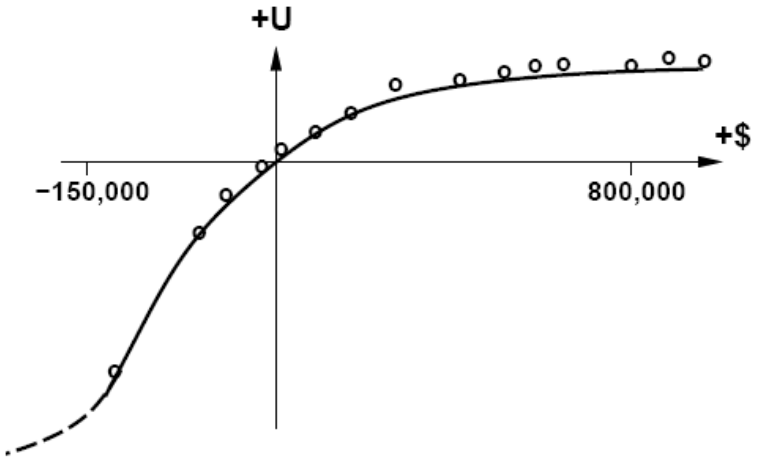
[۲] فرهنگ آکسفورد

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

منحنی سودمند [۱] **:** در یک مسابقه ، با چه مقدار احتمال p ، در بی تفاوتی میان یک پاداش x و [p,$M;(1-p),$0] دارای شانسی برای بردن M بزرگ هستم ؟

داده های تجربی [۲] معمولی و قیاسی با ریسک متعادل به صورت زیر رفتار می کنند :



اهمّیّت پول

ارزش (سودمندی) پول ، وابسته به مقدار پولی است که شما دارید . 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 که شیب آن برای همه ی مقادیر S یکسان نمی باشد .

برنولی [۱] **(۱۷۳۸) :** U(Money) تقریبا برابر است با log(Money)

---

[۱] utility curve

[۲] empirical

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

بیزاری از خطر (ریسک) و بیمه [۲]

یک شانس بخت آزمایی L=[p,$M;(1-p),$0] داده شده است ، چه مقدار پول شما می گیرید 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** [۳] بخت آزمایی نام دارد . در این مورد ، اجرت بیمه [۴] برابر است با 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 .

یک مثال واقعی بخت آزمایی

در یک مسابقه ی بخت آزمایی اگر داشته باشیم : L=[p=.02,$10;q=.000005,$1,000,000;(1-p-q)=.9799995,$0] و پول بلیط بخت آزمایی برابر یک دلار می باشد . آن گاه EMV(L) چه قدر است ؟ چه هنگام معقولانه است که یک بلیط بخت آزمایی را بخریم ؟

---

[۱] Bernoulli

[۲] insurance

[۳] certainty equalent

[۴] insurance premium

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

عقلانیّت و سود بخت آزمایی

فرض کنید S_k ، وضعیّت دارایی های جاری افراد باشد . 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) است یا نه ، 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 .

مزیّت چند خصوصیّتی بودن [۱]

چگونه می توانیم توابعی سودمند با تعدادی متغیّر X_1 تا X_n را به کار بگیریم ؟ برای مثال ، U(Deaths,Noise,Cost) چیست ؟ . چگونه می توانیم توابع سودمندی که دارای تقدّم در رفتار هستند را ترکیب کنیم ؟

ایده ی ۱ : وضعیّت هایی را تحت تصمیم گیری هایی که می توانند بدون شناسنامه ی کامل $U(x_1,...,x_n)$ ساخته شوند را تعیین نماییم .

---

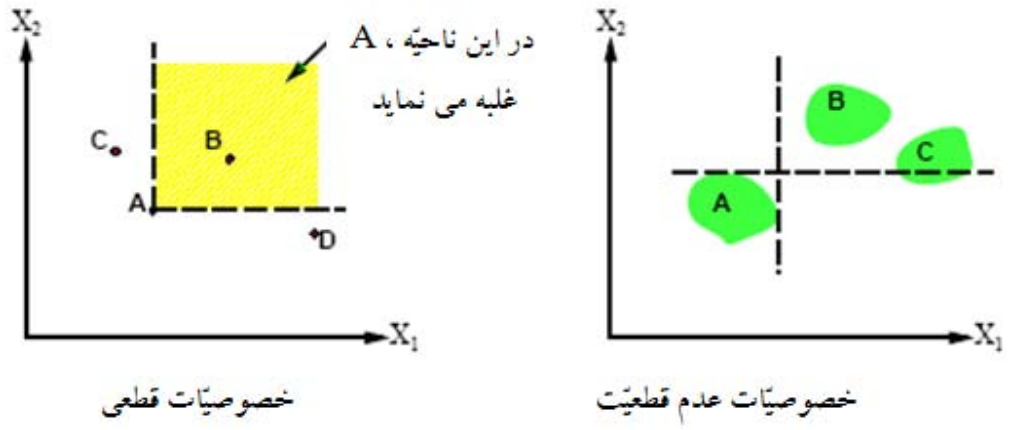
[۱] Multiattribute utility

ایده ی ۲ : انواع مختلف *مستقل* در اولویّت ها را شناسایی نماییم و فرم های استاندارد نتیجه را برای $U(x_1,\ldots,x_n)$ به دست آوریم .

تسلط (نفوذ) سخت [۱]

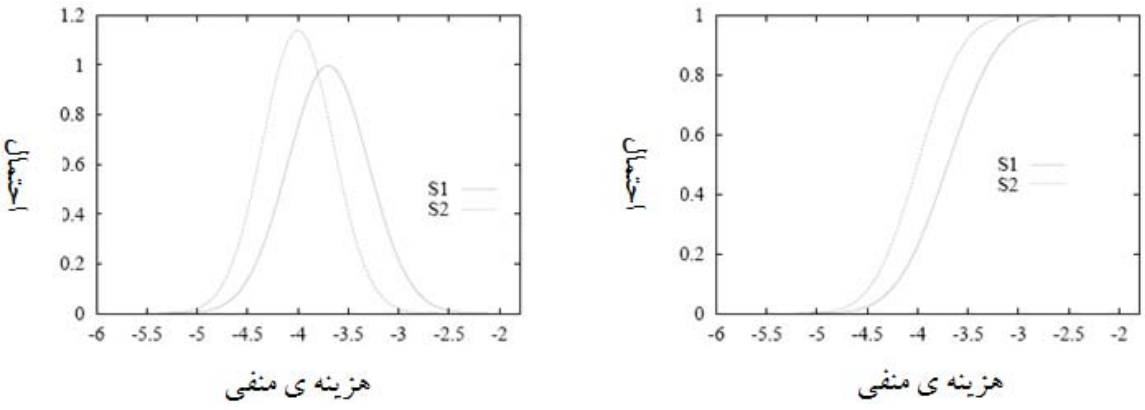
معمولاً خصوصیّات را به گونه ای تعریف می کنیم که U یکنواخت باشد .

تعریف تسلّط (نفوذ سخت) : انتخاب B در صورتی نفوذ شدید بر انتخاب A دارد که $\forall i X_i(B) \geq X_i(A)$ و بنابراین U(B)≥U(A) می باشد . نفوذ سخت ، در عمل به ندرت اتفاق می افتد .



نفوذ اتّفاقی

[۱] strict dominance



توزیع p_1 در صورتی بر توزیع p_2 اثر اتّفاقی 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 به دست آمده از p_1 ، بر A_2 ، با توزیع به دست آمده از p_2 اثر اتفاقی می گذارد اگر :

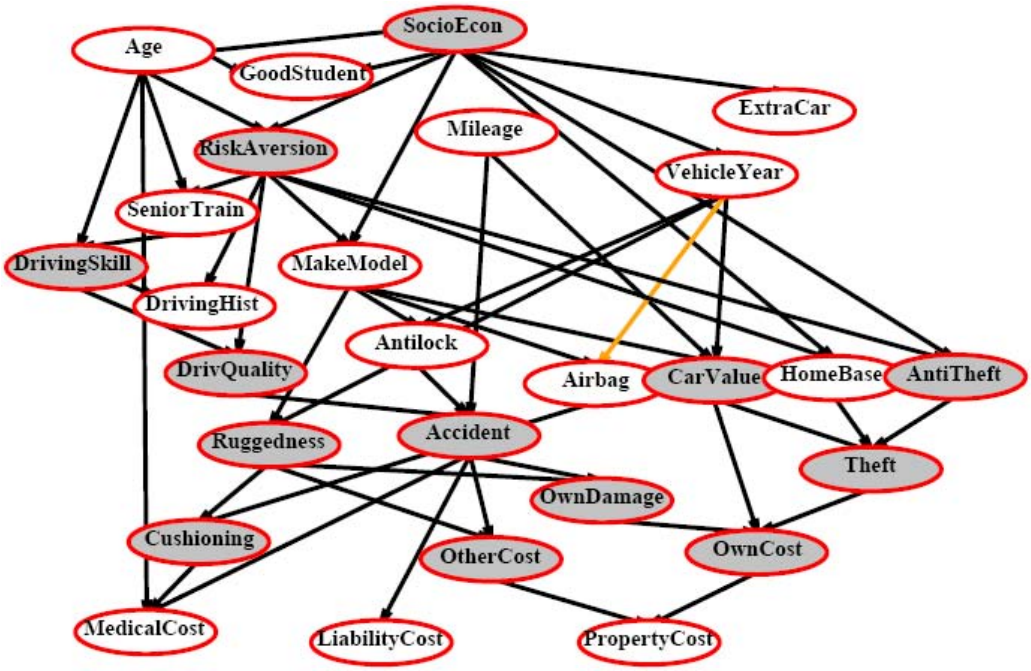
$$\int_{-\infty}^{\infty} p_1(x)U(x)\,dx \;\ge\; \int_{-\infty}^{\infty} p_2(x)U(x)\,dx$$

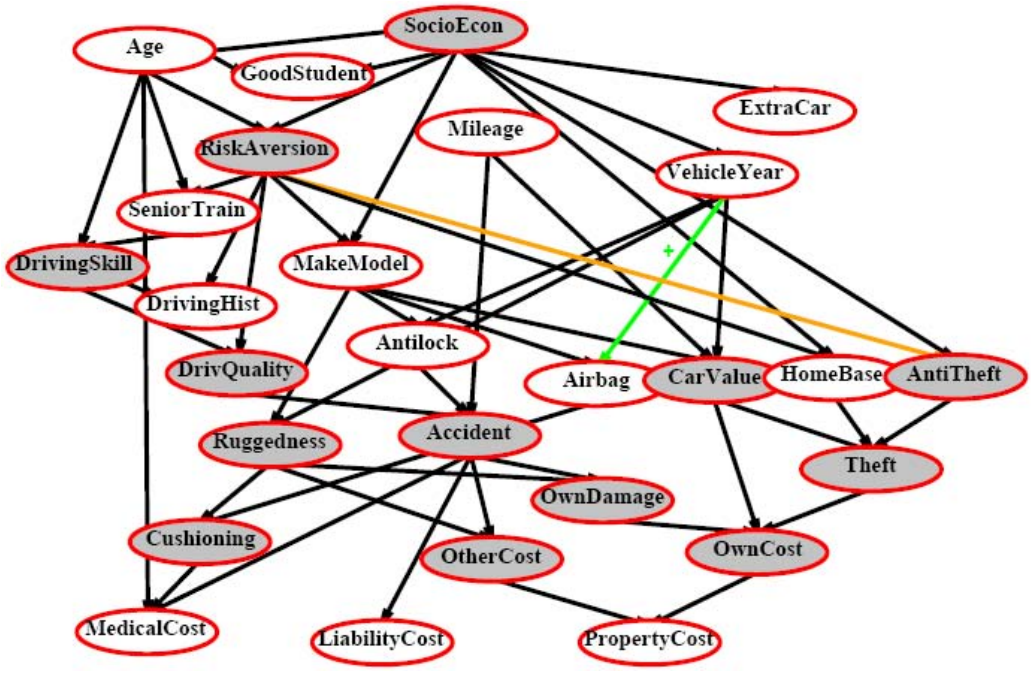
کاربرد چند خصوصیّتی : در صورت وجود اثر تصادفی در همه ی خصوصیّات ، چند خصوصیّتی ، بهینه خواهد بود .

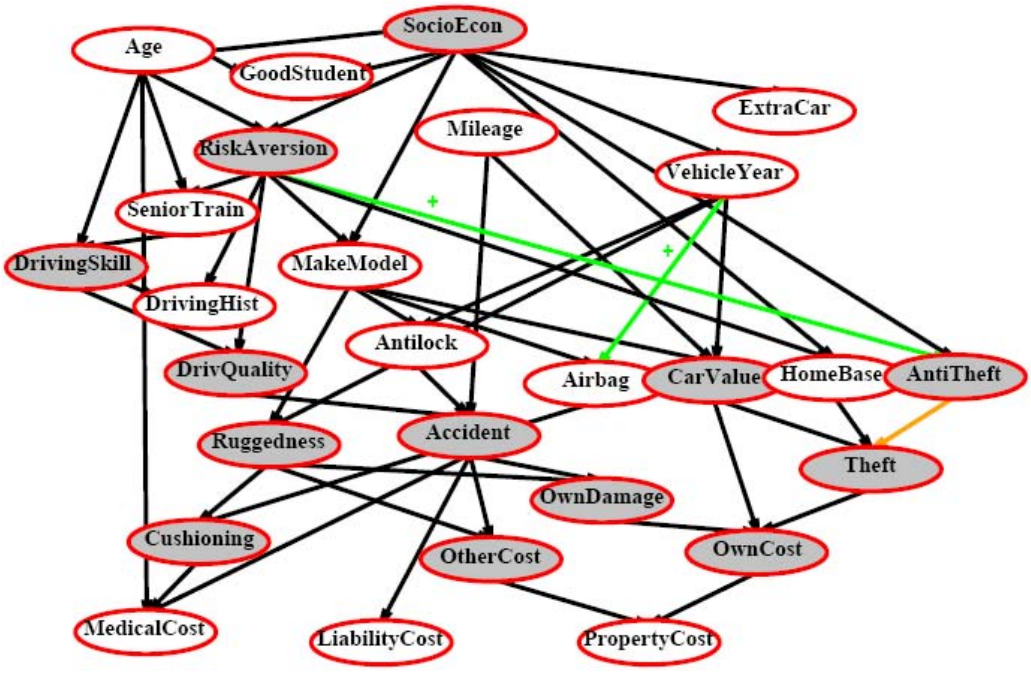
اثر اتّفاقی ، اغلب می تواند بدون توزیعات دقیق و با استفاده از چگونگی منطق ، تعیین شود . به عنوان مثال ، هزینه ی تولید با افزایش فاصله شهر S_1 از شهر S_2 افزایش می یابد ، در نتیجه ، S_1 دارای اثر تصادفی بر S_2 از نظر هزینه می باشد . به طور مثال ، آسیب دیدگی (صدمه) با افزایش سرعت در لحظه ی تصادف افزایش می یابد . می توان رفتار شبکه ها را با اطّلاعات اثر تصادفی تفسیر نمود : $X \xrightarrow{+} Y$ (X دارای اثرات مثبت بر

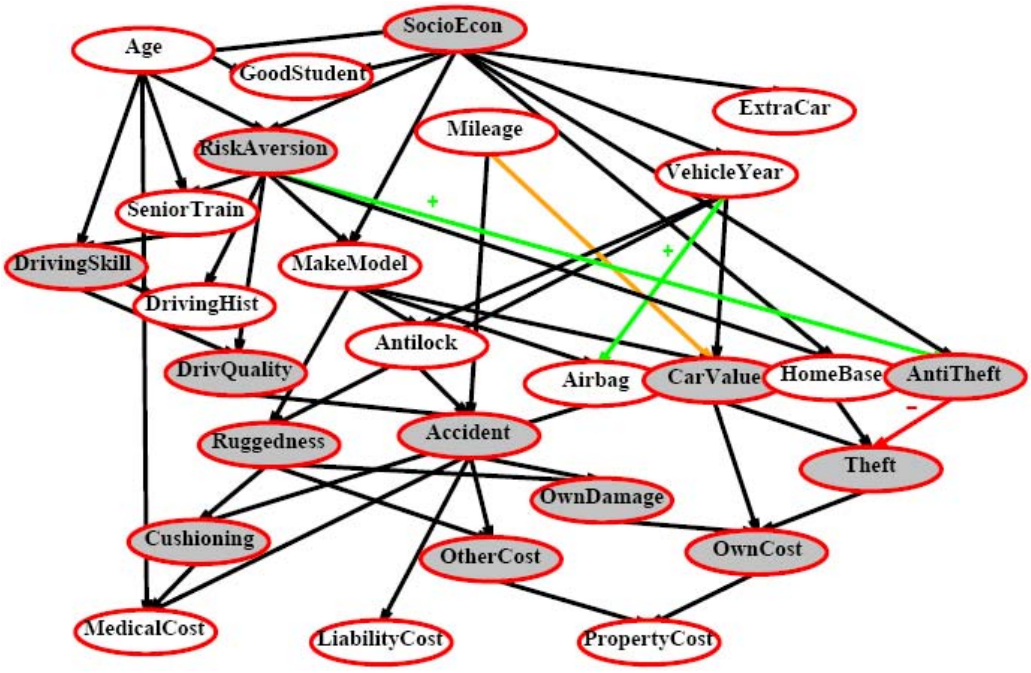
Y مـی باشـد) دارای ایـن معنـی اسـت کـه : بـرای هـر مقـدار z از سـایر والـدهای Z متعلـق بـه Y $\forall x_1, x_2 : x_1 \geq x_2 \Rightarrow P(Y \mid x_1, z)$ دارای اثر اتفاقی بر $P(Y \mid x_2, z)$ می باشد .

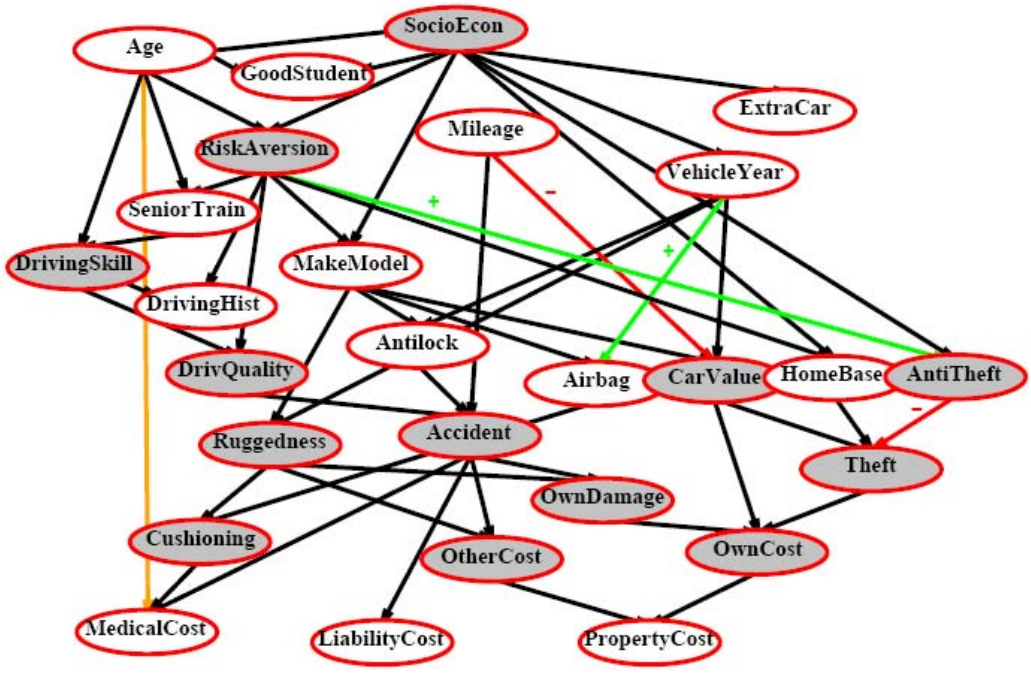
برچسب قوس های + یا –

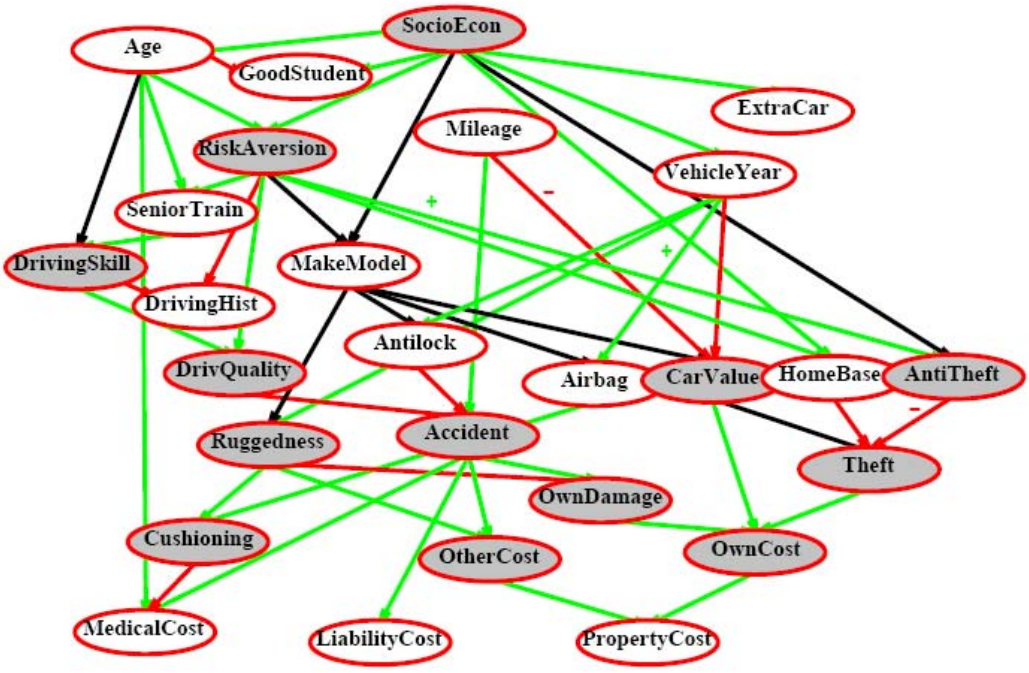



SocioEcon
Age
GoodStudent
ExtraCar
Mileage
RiskAversion
VehicleYear
SeniorTrain
DrivingSkill
MakeModel
DrivingHist
Antilock
DrivQuality
Airbag
CarValue
HomeBase
AntiTheft
Ruggedness
Accident
Theft
OwnDamage
Cushioning
OtherCost
OwnCost
MedicalCost
LiabilityCost
PropertyCost


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


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



اولویّت ساختار : قطعی

X_1 و X_2 ، از نظر اولویّت ، مستقل از $\mathbf{X_3}$ هستند درصورتی که : اولویّت میان $\langle x_1, x_2, x_3 \rangle$ و $\langle x_1', x_2', x_3 \rangle$ وابسته به $\mathbf{X_3}$ نباشد . مثلاً در <Noise,Cost,Safety> :

<۰.۰۶ و ۴.۶ بیلیون دلار و ۲۰۰۰۰> و <۰.۰۶ و ۴.۲ بیلیون دلار و ۷۰۰۰۰> .

قضیّه مستقل از لحاظ اولویّت دو طرفه (لیون تیف [1] ،۱۹٤۷) : اگر هر جفت از خصوصیّات از لحاظ اولویّت مستقل از متمّم خود باشد ، بنابراین هر زیر مجموعه از خصوصیّات از لحاظ اولویّت مستقل از متمّم خود می باشد .

---

[1] Leontief

قضیّه (دبرو [۱] ، ۱۹۶۰) : اگر قضیّه ی از لحاظ اولویّت مستقل بودن دو طرفه برقرار باشد ، در نتیجه تابعی به صورت جمع وجود دارد که :

$$V(S) = \sum_i V_i(X_i(S))$$

بنابراین به دست آوردن توابع تک خصوصیتی [۲] ، n تایی ؛ اغلب با تقریب خوبی انجام می شود .

اولویّت ساختار : اتّفاقی

– لازم است که اولویّت های بخت آزمایی را بدانیم : X دارای استقلال سودمندی [۳] ، Y می باشد ، در صورتی که اولویّت های شانس در X وابسته به y نباشند .

مستقل از لحاظ سودمندی بودن دو طرفه : هر زیر مجموعه مستقل از لحاظ سودمندی از مکمّل خود می باشد ، در نتیجه ، تابع سودمندی به صورت ضرب وجود دارد که :

$$U = k_1U_1 + k_2U_2 + k_3U_3 + k_1k_2U_1U_2 + k_2k_3U_2U_3 + k_3k_1U_3U_1 + k_1k_2k_3U_1U_2U_3$$

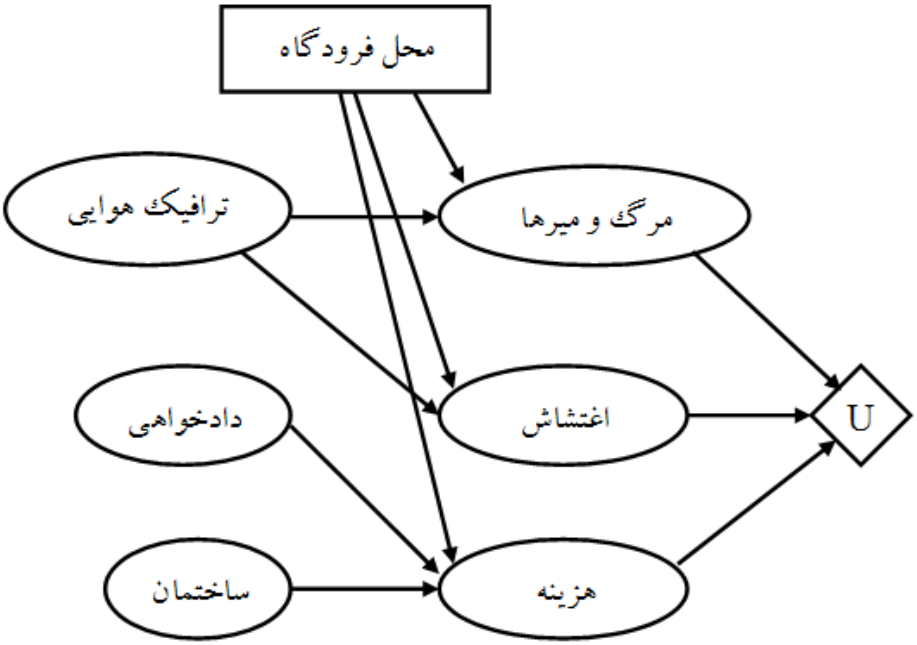
روال زیر برنامه ها و بسته های نرم افزاری برای تولید تست های اولویّت ، خانواده هایی با استاندارد های متفاوت توابع سودمند را معرّفی می نماید .

شبکه های تصمیم گیری

گره های عملکرد [۱] و گره های سودمند [۲] را به شبکه های رفتاری برای فعّال ساختن تصمیم گیری منطقی اضافه نمایید .

---

[۱] Debreu
[۲] single-attribute
[۳] utility-independent



الگوریتم :

برای هر مقدار گره ی عملکرد ،

مقدار مورد انتظار سودمندی عملکرد و ثبات (پایداری) گره ی ارایه شده را با استفاده از استنتاج محاسبه نمایید

عملکرد MEU را برگردان

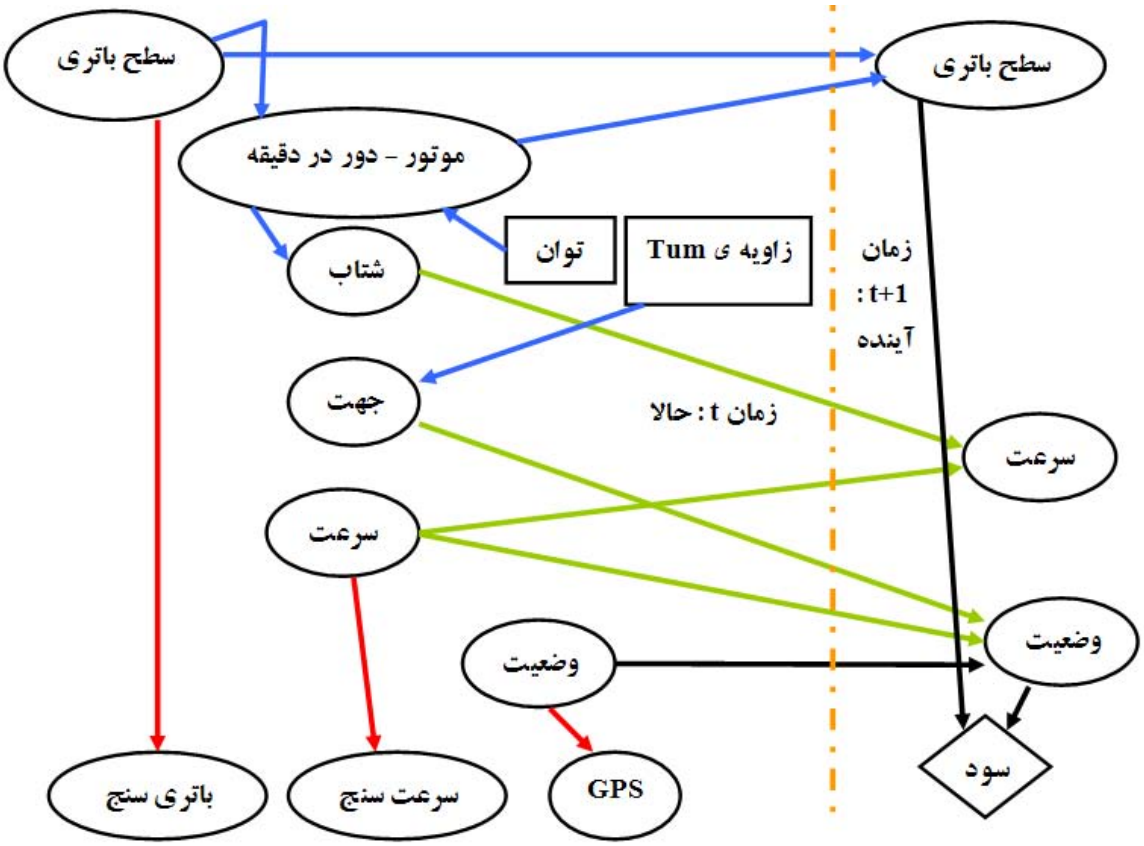
پایان الگوریتم

شبکه ی تصمیم گیری روبوت

---

[1] action nodes

[2] utility nodes



ارزیابی شبکه ی تصمیم گیری

۱. متغیّرهای مدرک را برای وضعیّت جاری سیستم قرار دهید .

۲. برای هر بردار انتساب های مقدار برای متغیّرهای *تصمیم گیری* (به عنوان مثال ، فرمان یا توان) :

۱) گره های تصمیم گیری را به مقادیر آن ها انتساب دهید .

۲) از استنتاج بیزی (دقیق یا تقریبی) برای محاسبه ی احتمال های قبلی همه ی والدهای گره سود استفاده نمایید .

۳) 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 متغیّرهای مدرک نسبت دهید :

باتری سنج ، سرعت سنج و GPS [۱] (همه در زمان t)

۲. برای هر ترکیب ممکن از توان و زاویه ی چرخش [۲] :

۱) از استنتاج بیزی برای محاسبه ی توزیع احتمال برای سطح باتری و وضعیت در زمان t+1 استفاده نمایید .

۲) سپس آن توزیع ها را برای محاسبه ی سود به کار ببرید .

۳. بهترین توان را به دست آورید .

نکته : از سود مقادیر *آینده ی طرح* [۱] سطح باتری و وضعیّت به صورت پایه ای برای انتخاب عملکردهای جاری ، استفاده نمایید .

---

[۱] مخفّف Global Positioning System است و سیستمی ناوبری [مکان نمایی] است که با شبکه ای گسترده از ماهواره ها کار می کند .

[۲] turn-angle

ارزش اطّلاعات

– **ایده :** مقدار به دست آمده برای هر جزء ممکن مدرک 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 ، دقیقاً 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 A مناسب است ؟

راه حل : ارزش مورد انتظار اطّلاعات را محاسبه نمایید ، که برابر است با ، مقدار مورد انتظار بهترین عملکرد اطّلاعات ارایه شده منهای مقدار مورد انتظار بهترین عملکرد بدون اطّلاعات . عملکردها خرید A (Buy(A)) و خرید B (Buy(B)) می باشند . بدون اطّلاعات ، ارزش مورد انتظار هر عملکرد برابر است با :

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 ! با داشتن اطّلاعات ، مقدار مورد انتظار برای بهترین عملکرد ، برابر است با :

[۰.۵ * ارزش داده شده ی " خرید A " و با فرض این که " روغن درون A می باشد " + ۰.۵ * ارزش " خرید B " با فرض این که " روغن درون A نمی باشد "] که برابر است با :

$$=(0.5*k/2)+(0.5*k/2)-0=k/2$$

بنابراین ارزش اطّلاعات مورد انتظار برابر است با :

---

[1] PROJECTED FUTURE

انتخاب اندازه گیری [۱]

اطّلاعات معمولاً مستقل نمی باشند . به دست آوردن اندازه گیری ها می تواند پرهزینه باشد : آزمایش های طبی ، تحلیل های شیمیایی اعماق دریا و ... . احتیاج به اولویّت بندی اندازه ها یا آزمایش ها داریم . برای این کار ، آن هایی که منجر به وضعیّت های با بالاترین سود مورد انتظار می شوند را انتخاب نمایید .

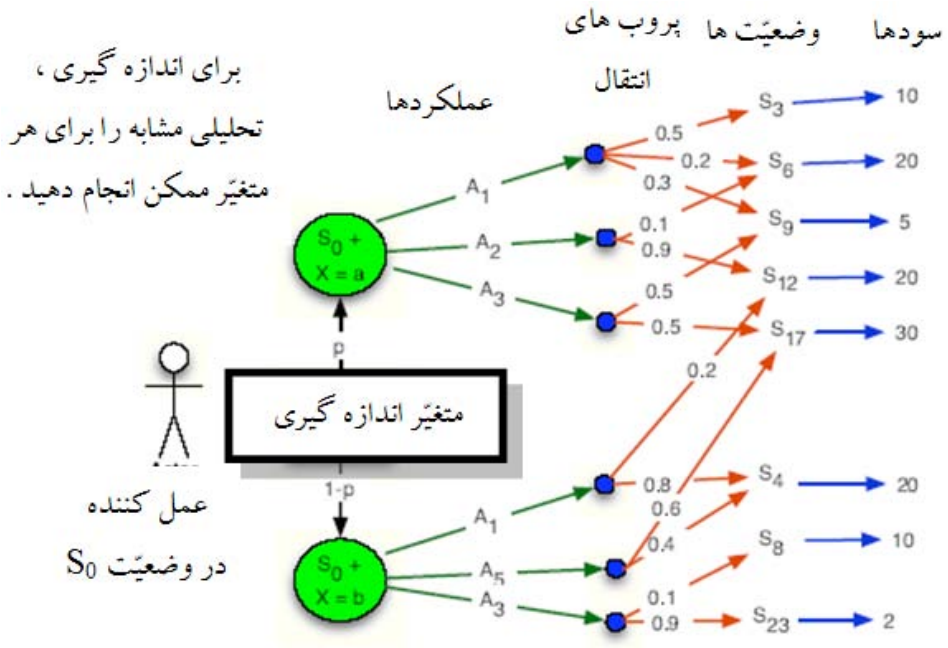
اندازه گیری ها ← اطّلاعات وضعیّت ها ← انتخاب عملکردها ← وضعیّت های کلّی جدید ← ارزشیابی سود [۲]

از آنجایی که تعدادی از این اتّصالات به صورت اتّفاقی می باشند ، ما به توزیع احتمالی که هم نتایج اندازه گیری و هم عملکردهای منظّم را دارا هستند رسیدگی می نماییم .

---

[۱] Measurement Selection

[۲] Utility Assessment



فرمول کل – اگر برای مدرک جاری E ، بهترین عملکرد جاری ، α باشد . نتیجه ها ی عملکرد ممکن S_i و توان بالقوه ی (پتانسیل) مدرک جدید E_j برابر است با :

$$EU(\alpha \mid E) = \max_a \sum_i U(S_i) P(S_i \mid E, a)$$

تصور نمایید که ما می دانیم $E_j = e_{jk}$ ، بنابراین ما $\alpha_{e_{jk}}$ را انتخاب می کنیم .

$$EU(\alpha_{e_{jk}} \mid E, E_j = e_{jk}) = \max_a \sum_i U(S_i) P(S_i \mid E, a, E_j = e_{jk})$$

E_j ، یک متغیّر تصادفی است که مقدار فعلی آن ناشناخته می باشد ، در نتیجه ، باید مقادیر آن را حول تمام مقادیر ممکن آن محاسبه نماییم :

$$VPI_E(E_j) = (\sum_k P(E_j = e_{jk} \mid E) EU(\alpha_{e_{jk}} \mid E, E_j = e_{jk})) - EU(\alpha \mid E)$$

VPI مخفف Value of Perfect Information می باشد ، به معنی ارزش اطّلاعات کامل می باشد .

خصوصیّات ارزش اطّلاعات کامل

- **نا منفی** [۱] می باشد : $\forall j, E : VPI_E(E_j) \geq 0$

- **جمع ناپذیری** [۲] ، 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 [۳] :

$$VPI_E(E_j, E_k) = VPI_E(E_j) + VPI_{E,E_j}(E_k) = VPI_E(E_k) + VPI_{E,E_k}(E_j)$$

توجّه : در زمانی که بیش تر از یک بخش مدرک می تواند جمع آوری شود ، بیشینه کردن ارزش اطّلاعات کامل برای هر قسمت ، برای انتخاب یکی از آن ها که همیشه هم بهینه نمی باشد منجر به این می شود که جمع کردن مدرک ها تبدیل به یک مسأله ی تصمیم گیری *ترتیبی* شود .

کیفیّت (چگونگی) رفتارها

اگر انتخاب ، قابل مشاهده می باشد ، اطّلاعات دارای ارزش کمی می باشند . اگر انتخاب ، غیرقابل مشاهده می باشد ، اطّلاعات ، به دلیل بالا بودن گوناگونی سودمندی (U) ، دارای ارزش زیادی می باشند .

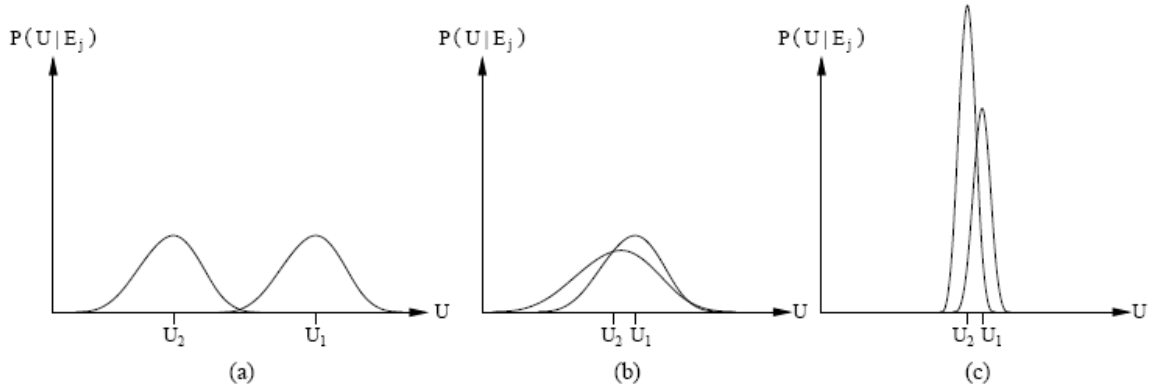
---

[۱] nonnegative

[۲] nonadditive

[۳] order-independent

اگر انتخاب غیرقابل مشاهده می باشد ، اطّلاعات ، به دلیل پایین بودن گوناگونی سودمندی (U) ، دارای ارزش کمی می باشند .



مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل هیجدهم

شبکه های عصبی[1]

[1] neural networks

ریوس مطالب

- مغز [۱]
- شبکه های عصبی
- پرسپترون [۲] ها
- پرسپترون های چند سطحی
- کاربردهای شبکه های عصبی

شبکه های عصبی

---

[۱] brains

[۲] Perceptron

هوش مصنوعی

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

بسیاری از آنچه که تا کنون ما مطالعه کرده ایم می تواند به صورت *هوش مصنوعی نمادین یا نشانه ای* [۱] طبقه بندی شود ؛ در هوش مصنوعی نمادین ، تأکید بر سمبل ها و ارتباطات میان آن ها می باشد . به عنوان مثال ، جستجو ، منطق ، درخت های تصمیم گیری و برنامه ریزی جزء هوش مصنوعی نمادین می باشند . سمبل ها برای رفتار هوشمند ، کلیدی می باشند . شبکه های عصبی بر رفتار *نمادین* تمرکز می نماید . توجّه کنید که رفتار هوشمند از ارتباط اجزای ساده به وجود می آید .

زیست شناسی و علم کامپیوتر

در نورون های [۲] زیست شناسی ، سیگنال های دریافت شده توسط دندریت ها [۳] دریافت می شوند و از طریق آکسون [۴] به دیگر نورون ها می رسند . [در انتقال سیگنال ها] ، علامت دهی [۵] و شلیک [۶] ، خیلی پیچیده می باشند . فکر و رفتار در تعامل هزاران نورون به وجود می آیند . محقّقان هوش مصنوعی اغلب جذب توسعه ی الگوریتم های مؤثّر می باشند ؛ مانند الگوریتم های ژنتیکی ، ما برروی ایده هایی که کاملاً طبیعی هستند کار می کنیم و آن هایی که مفید هستند را به کار می بندیم .

---

[۱] symbolic AI

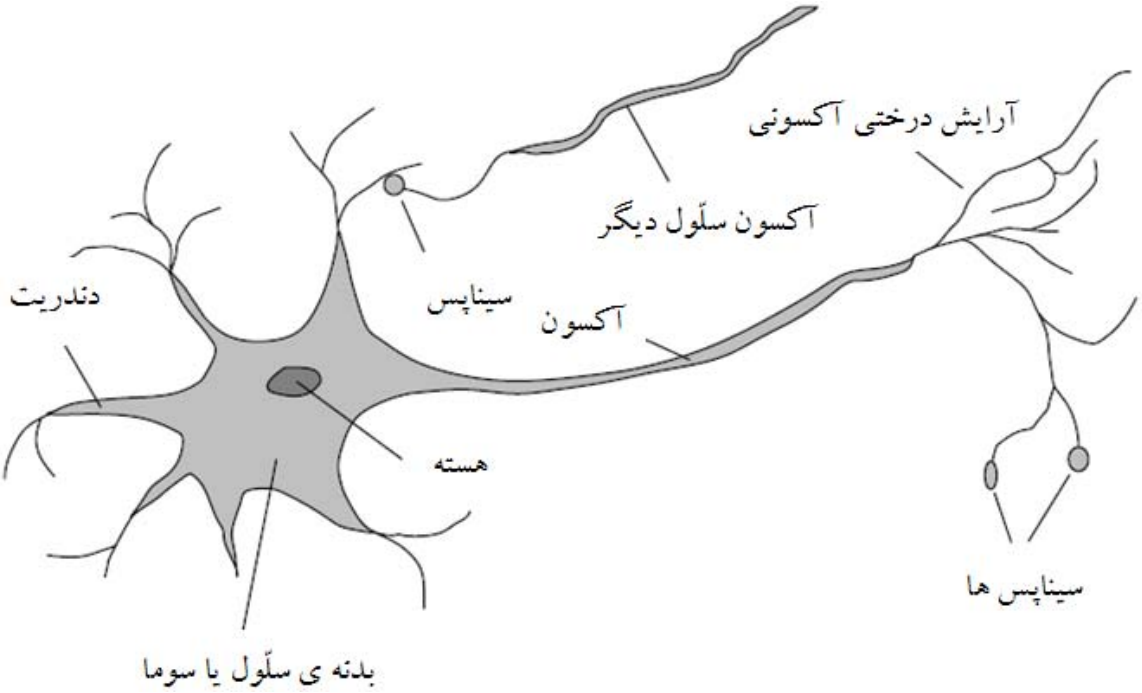
[۲] neurons

[۳] dendrites

[۴] axon

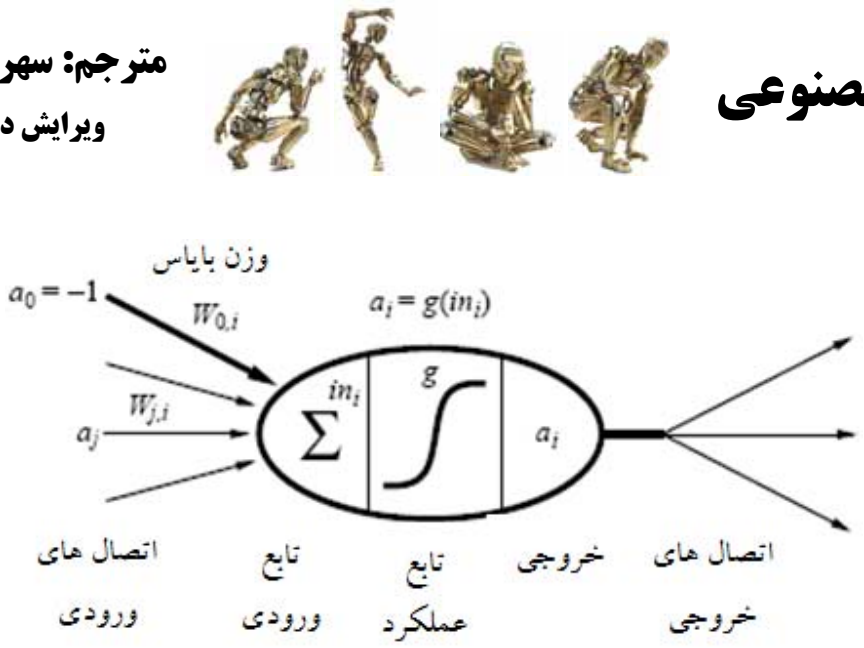
[۵] signaling

[۶] firing



شبکه های عصبی محاسباتی

شبکه های عصبی از گره ها تشکیل شده اند . این گره ها توسط اتّصال هایی جدا از آکسون ها به هم متّصل می باشند . 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 مدیریّت می نماید .



توابع عملکرد

در حالت کلّی ، از هر تابع غیرخطی می توان استفاده نمود ؛ عمومی ترین توابع ، دو تابع به صورت زیر می باشند :

۱- **تابع مرحله ای (تابع آستانه)** – خروجی ها ، اگر ورودی مثبت باشد ، یک می باشد و در غیر این صورت صفر می باشد .

۲- **تابع S شکل (سیگموید)** [۱] **یا نقلی** [۲] **:** $1/(1+e^{-x})$ ؛ که به صورت مشتق پذیر و پیوسته [۳] می باشد و دارای تغییر سریع در نزدیکی آستانه می باشد و به صورت تدریجی به سمت بی نهایت می رود .

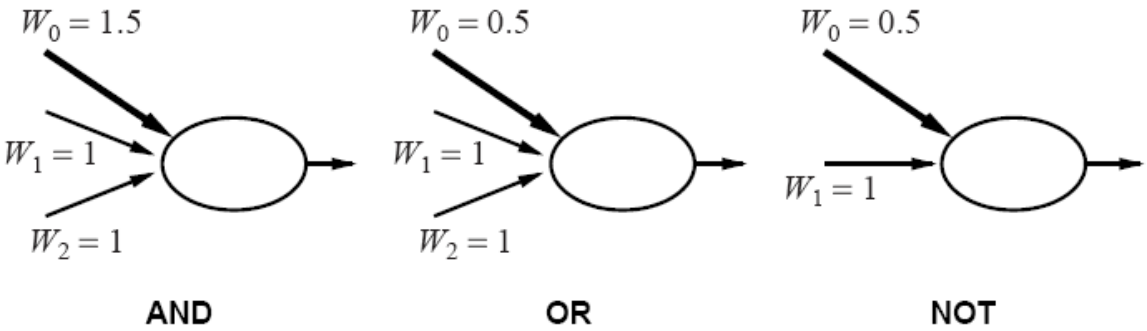
مثال ها :

---

[۱] sigmoid

[۲] logistic

[۳] continuously differentiable



شبکه های عصبی می توانند به سادگی برای انجام برخی از عملیّات استاندارد منطقی با استفاده از تابع عملکرد آستانه ای ساخته شوند ؛ که در این مورد ، باید شما بسته به تابعی که می خواهید ، آستانه را تغییر دهید .

انواع گره ها

ما می توانیم سه نوع گره را تشخیص دهیم :

۱- گره های ورودی
۲- گره های خروجی
۳- گره های پنهان

ما همچنین می توانیم انواع شبکه های زیر را داشته باشیم :

۱- **شبکه های با تغذیه ی مستقیم** [۱] **:** علامت (سیگنال) ها در یک جهت حرکت می کنند و بدون دور یا سیکل می باشند . شبکه های تغذیه ی مستقیم ، توابع را تکمیل می کنند و وضعیت داخلی ندارند .
۲- **شبکه های بازگشت کننده** [۱] **:** در انتشار علامت (سیگنال) ، دور یا سیکل وجود دارد .

---

[۱] feed-forward networks

ما در ابتدا روی شبکه های تغذیه ی مستقیم تمرکز می نماییم .

شبکه های تغذیّه 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 غیرخطّی* [۲] قرار می گیرند ؛ خروجی یک شبکه ی عصبی تابعی از ورودی هایش می باشد . تابع فعّال سازی غیرخطّی ، 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 یک یا بیش تر خروجی تبدیل می نمایند و دامنه ی خروجی به کلاس هایی مجزّا تقسیم می شوند و برای کارهای آموزشی در جایی که نمی دانیم " در جستجوی چه هستیم " ، خیلی مفید می باشد ، مثل : صورت شناسی [۳] ، دست خط شناسی [۴] و رانندگی یک اتومبیل .

مثال تغذیه ی مستقیم

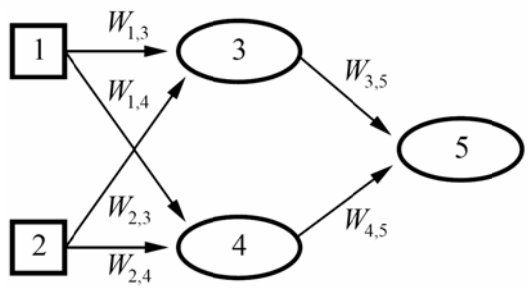
---

[۱] recurrent networks

[۲] nonlinear function approximators

[۳] face recognition

[۴] handwriting recognition



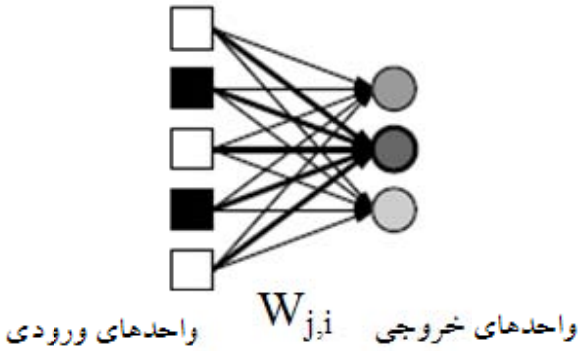
شبکه ی تغذیه ی مستقیم ، یک خانواده ی پارامتربندی شده از توابع غیرخطی است ، داریم :

$$a_5 = g(W_{3,5}.a_3 + W_{4,5}.a_4)$$
$$= g(W_{3,5}.g(W_{1,3}.a_1 + W_{2,3}.a_2) + W_{4,5}.g(W_{1,4}.a_1 + W_{2,4}.a_2))$$

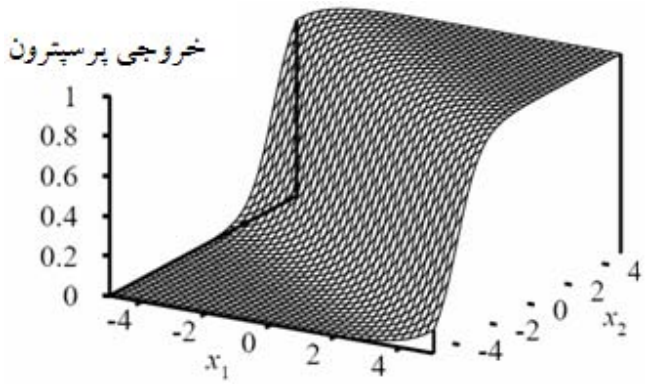
پرسپترون ها

پرسپترون ، شبکه ای از نورون هاست که در آن یک ارتباط دارای وزن میان خروجی برخی از نـرون ها و ورودی دیگر نرون ها وجود دارد ؛ به عنوان تعریفی دیگر ، ابزاری است که می تواند الگوهای یـادگیری (آموزشی) را تشخیص دهد . [۱]

پرسپترون های تک لایه ای



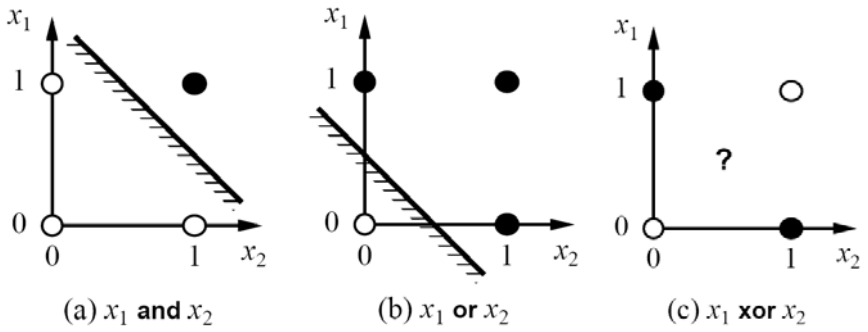
[۱] Babylon / English - English



پرسپترون های تک لایه ای ، ساده ترین شکل شبکه ی عصبی می باشند ؛ برای فهم ، ساده می باشند ولی برای محاسبه ، محدود می باشند و در آن ها هر واحد ورودی به صورت مستقیم به یک یا بیش تر واحد خروجی متّصل شده است . 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 .

توجّه نمایید که یک پرسپترون با تابع مرحله ای g ، می تواند AND ، OR ، NOT و ... را ارایه نماید ولی XOR را نمی تواند بیان نماید و در فضای ورودی ، یک مجزّا کننده ی خطی [1] را ارایه می نماید :

$$W.x>0 \text{ یا } \sum_j W_j x_j > 0$$



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

[1] linear separator

توان نمایش پرسپترون ها

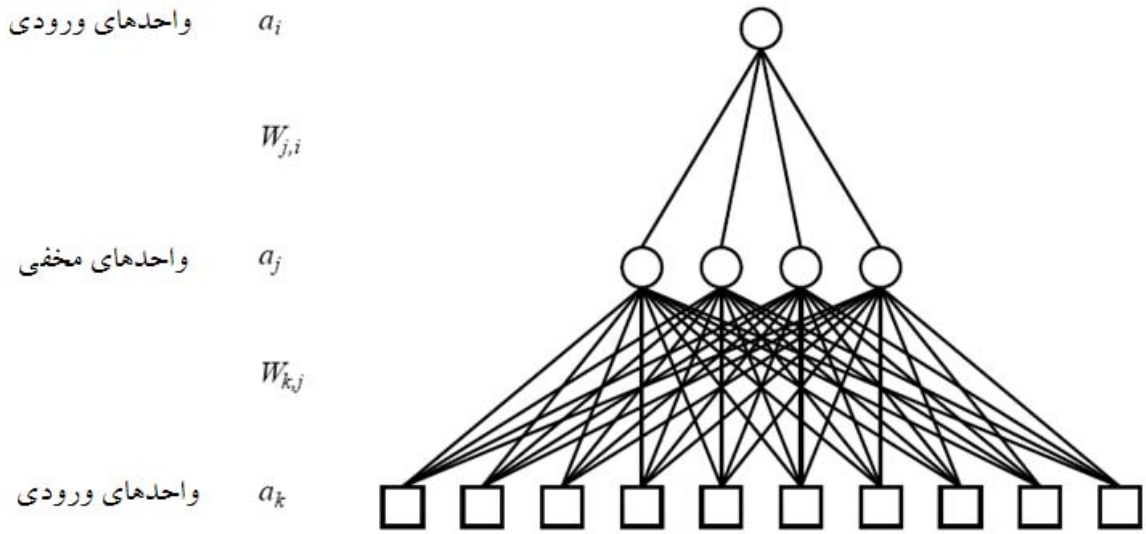
خروجی شبکه ، $\sum_{j=0}^{n} W_j i_j > 0$ می باشد . پرسپترون ها قادر به ارایه ی هر تابع خطی جداشدنی هستند ، ولی متأسّفانه ، تعدادی از توابع مثل XOR به دلیل این که به صورت خطّی جدا شدنی نمی باشند قابل نمایش توسّط پرسپترون ها نمی باشند . در روزهای ابتدایی هوش مصنوعی ، پرسپترون ها ساده بودند ، به علّت این واقعیّت که وزن آن ها می توانست به سادگی پیدا شود .

آموزش پرسپترون

اگر سیگنال خروجی ، خیلی بزرگ باشد ، اوزان باید کاهش یابند . برای این کار ، وزن هایی را که در خروجی شرکت دارند را کاهش دهید ، توجّه کنید که وزن های با ورودی صفر ، مؤثّر نمی باشند . اگر خروجی خیلی کوچک باشد ، اوزان باید افزایش یابند . برای این کار ، وزن هایی را که در خروجی شرکت دارند را افزایش دهید ، در این مورد هم توجّه کنید که وزن های با ورودی صفر ، مؤثّر نمی باشند . آموزش ، در واقع بهینه سازی وزن ها می باشد و به طور متناوب ما در حال انجام یک جستجوی تپّه نوردی در میان فضای وزن هستیم .

پرسپترون های چند لایه ای

در پرسپترون های چند لایه ای ، لایه ها معمولاً به صورت کامل متّصل می شوند و به طور معمول ، تعدادی از واحدهای مخفی به صورت دستی انتخاب می شوند . پرسپترون ها دارای این مزیّت هستند که یک الگوریتم آموزشی ساده دارند ولی عیب آن ها در این است که دارای محدودیّت های محاسباتی هستند . حال سؤال این است که اگر ما یک لایه ی مخفی دیگر اضافه نماییم چه اتفاقی می افتد ؟ پاسخ این است که *توان محاسباتی افزایش می یابد* ؛ با یک لایه ی پنهان ، می تواند هر تابع پیوسته را نمایش دهد و با دو لایه ی پنهان ، می تواند هر تابعی را نمایش دهد . در این مورد ، مشکل این است که چگونه وزن های صحیح را برای گره های پنهان ، پیدا نماییم ؟



انتشار معکوس [۱]

انتشار معکوس ، شاخه ای از الگوریتم آموزشی پرسپترون برای رسیدگی کردن به چندلایه ی گره هاست . گره ها از تابع فعال سازی سیگموید استفاده می نمایند ، داریم :

$$g(input_i) = \frac{1}{1 + e^{-input_i}}$$

$$g'(input_i) = g(input_i)(1 - input_i))$$

یادداشت :

- $h_w(x)$ - بردار خروجی های شبکه
- y – خروجی مطلوب برای یک مثال آموزشی

---

[۱] backpropagation

- 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 با ورودی پنهان :

ایده : هر گره ی پنهان دارای یک کسر از خطا به صورت Δ_i می باشد . Δ_i را با توجّه به شدّت اتّصال 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 با ورودی پنهان به روز نمایید : $W_{k,j} = W_{k,j} + \alpha * input_k * \Delta_j$

الگوریتم انتشار معکوس

مادامی که انجام نشده است کارهای زیر را انجام بده :

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

$$W_{j,i} = W_{j,i} + \alpha * \Delta_i * input_j$$

متوقّف کردن شرط ها

سؤال : چه موقع ، آموزش را متوقّف نماییم ؟

جواب : وقتی که تعداد تکرارها ثابت شده باشد ، مجموع خطا ، زیر یک مجموعه ی آستانه باشد و همگرایی [۱] وجود داشته باشد ؛ یعنی ، هیچ تغییری در وزن ها به وجود نیاید .

صورت شناسی با استفاده از شبکه های عصبی

یکی از مزایای شبکه های عصبی این است که آن ها به یک توضیح صریح از ارتباط های میانِ بَعدی ها نیاز ندارند . آن ها واقعاً 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 .

نکاتی در مورد انتشار معکوس

---

[۱] convergence

همیشه برای چند لایه ی پنهان کار می کنند . انتشار معکوس فقط برای گرایش دادن (همگرایی) به یک کمینه ی محلی ضمانت شده است و ممکن است مجموعه ی وزن های کامل را پیدا نکنیم . وزن های اولیه ی پایین می توانند در کارکردن شبکه به صورت به مراتب خطّی تر به ما کمک نمایند ، همچنین می توانند از شروع مجدّد تصادفی استفاده نمایند .

شبکه های با عملکرد شعاعی [۱]

یک مسأله 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 شعاعی ، یک راه حل را برای این مورد بیان می نمایند .

بینش : هر گره در شبکه یک جزء از فضای ورودی را نمایش می دهد و دنبال طبقه بندی نمونه هایی که در اطراف آن رخ می دهند می باشد .

نگرش وانیلی [۲] : برای هر نقطه ی آموزشی $< x_i , f(x_i) >$ ، 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 ϕ که تابع اساسی است برابر است با $\phi = \exp(-\frac{x^2}{2\sigma^2})$.

هر گره دارای یک " ناحیّه ی تأثیر " [۳] است که می تواند نمونه های نزدیک را طبقه بندی نماید . آموزش با طبقه بندی غلط ، فقط بر روی گره هایی که نزدیک نمونه هایی که به صورت نادرست طبقه بندی شده اند تأثیر خواهد گذاشت . همچنین ، این شبکه ، تک لایه ای می باشد ؛ وزن ها می توانند با یک معادله ی

---

[۱] Radial Basis Function networks

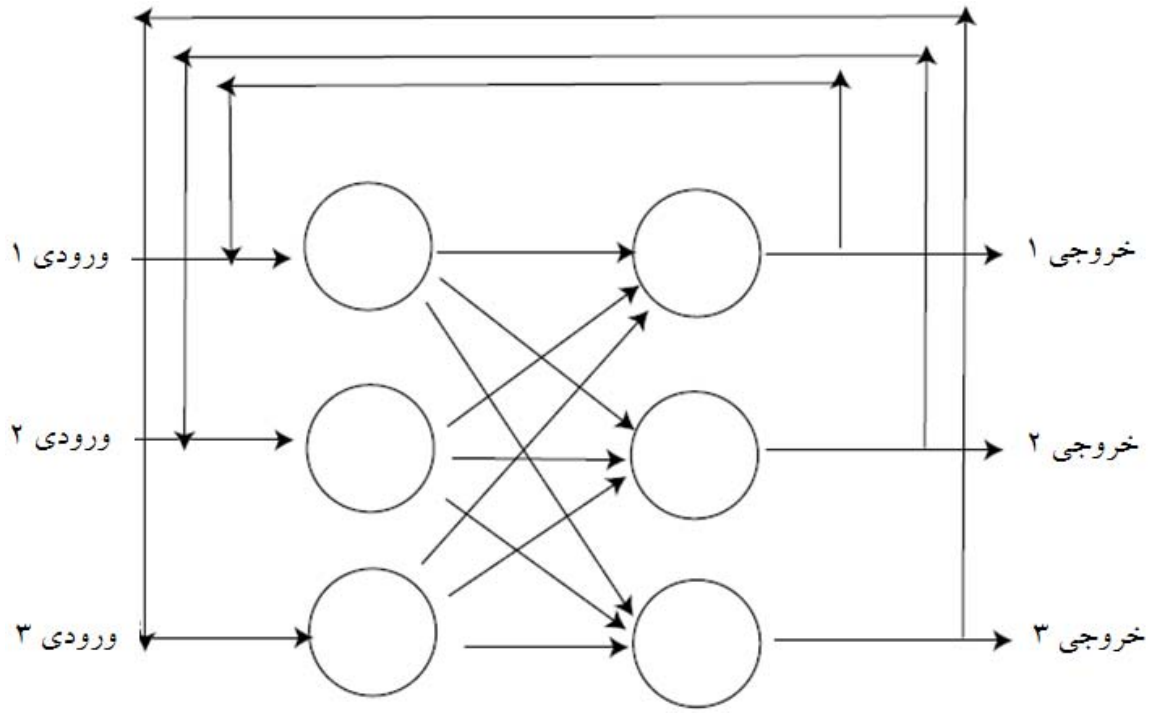
[۲] vanilla approach

[۳] zone of influence

ماتریسی به دست آیند ، داریم : $\Phi W = t$، در نتیجه ، $W = \Phi_t^{-1}$. توجّه کنید که معکوس کردن یک ماتریس ، به مراتب سریع تر از ساختن آن با انتشار معکوس می باشد .

شبکه های عصبی برگشت کننده [۱]

تا کنون ما فقط در مورد شبکه های با تغذیّه 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 پشت ورودی ها متّصل شده اند . در زیر یک شبکه عصبی برگشت کننده ی تک لایه ای را می بینید :



[۱] recurrent NNs(Neural Networks)

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

شبکه های هوپفیلد [۱]

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 متّصل شده است و معمولاً ، از توابع آستانه ای استفاده می شود ؛ شبکه بلافاصله یک خروجی را تولید نمی نماید و مردّد می باشد ؛ تحت برخی از وضعیّت های به آسانی قابل دسترس ، سرانجام به حالت موازنه می رسد ؛ وزن ها با استفاده از روش گرم و سرد کردن شبیه سازی شده به دست می آیند . شبکه های هوپفیلد می توانند برای ساختن یک حافظه ی شرکت *پذیر* [۲] به کار روند ، در این شبکه ها ، یک قسمت از یک الگو برای شبکه ارایه می شود و شبکه تمام الگو را به یاد می آورد . برای حرف شناسی [۳] هم به کار می رود ، همچنین برای مسایل بهینه سازی هم به کار می رود و اغلب برای مدل فعّالیّت مغز استفاده می شود .

---

[۱] Hopfield networks

[۲] associative memory

[۳] letter recognition

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل نوزدهم

الگوریتم های ژنتیکی [۱]

[۱] Genetic Algorithms

تاریخچه

در دهه ی هفتاد میلادی توسط جان هُلَند [۱] بیان شد ؛ در دهه ی هشتاد میلادی عمومیّت یافت و براساس تئوری تکاملی داروین (باقی ماندن شایسته ترین) بود . الگوریتم های ژنتیکی می توانند برای حلّ انواعی از مسایل که برای حل ، آسان نمی باشند با استفاده از روش های دیگر مورد استفاده قرار گیرند .

تکامل در دنیای واقعی

هر سلّول یک موجود زنده دارای *کروموزوم* [۲] هایی می باشد . هر کروموزوم شامل یک مجموعه از *ژن* [۳] ها می باشد . هر ژن برخی از خواص موجود زنده (مانند رنگ چشم) را تعیین می کند . یک مجموعه از ژن ها در برخی از موارد *ژنوتیپ* (نوع معرّف و نماینده ی یک جنس) [۴] نامیده می شود . یک مجموعه از

---

[۱] John Holland
[۲] chromosome
[۳] gene
[۴] genotype

خصوصیّات (نظیر رنگ چشم) گاهی *فنوتیپ* [۱] نامیده می شود . انتشار ، شامل بازترکیب [۲] ژن ها از والـدین و سپس مقدار کوچکی *جهش* [۳] (خطاها) در کپی کردن می باشد . شایستگی [۴] یک موجود زنده مقـداری اسـت که می تواند قبل از مردنش انتشار دهد . تکامل ، براساس "بقای مناسب ترین [۵] " می باشد .

شروع با یک تصوّر

فرض کنید که شما مشکلی دارید و نمی دانید که چگونه آن را حل نمایید . برای حل این مشکل چـه کار می توانید بکنید ؟ آیا می توانید به طریقی از یک کامپیوتر برای پیدا کردن یک راه حل اسـتفاده نماییـد ؟ این کار باید خوب باشد ! می توانید آن را انجام دهید ؟

یک راه حل گنگ

یک الگوریتم "تولید و تست گنگ" :

تکرار کن

یک راه حل ممکن تصادفی را تولید کن

راه حل را امتحان کن و خوب بودن آن را بسنج

تا موقعی که راه حل به اندازه ی کافی خوب باشد

---

[۱] phenotype
[۲] recombination
[۳] mutation
[۴] fitness
[۵] survival of the fittest

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 . *امّا 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 ، 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

تکرار کن

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 مسأله می باشد ! الگوریتم های ژنتیکی ، اغلب راه حل ها را به صورت رشته های بیتی [۱] باطول ثابت ، کد می کنند (به عنوان مثال ، ۱۰۱۱۱۰ ، ۱۱۱۱۱۱ ، ۰۰۰۱۰۱) . هر بیت برخی از خصوصیّات راه حل های ارایه شده برای مسأله 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

تصوّر کنید که شما باید برای روغن جایی را در طول یک جاده ی بیابانی یک کیلومتری ، حفر نمایید .

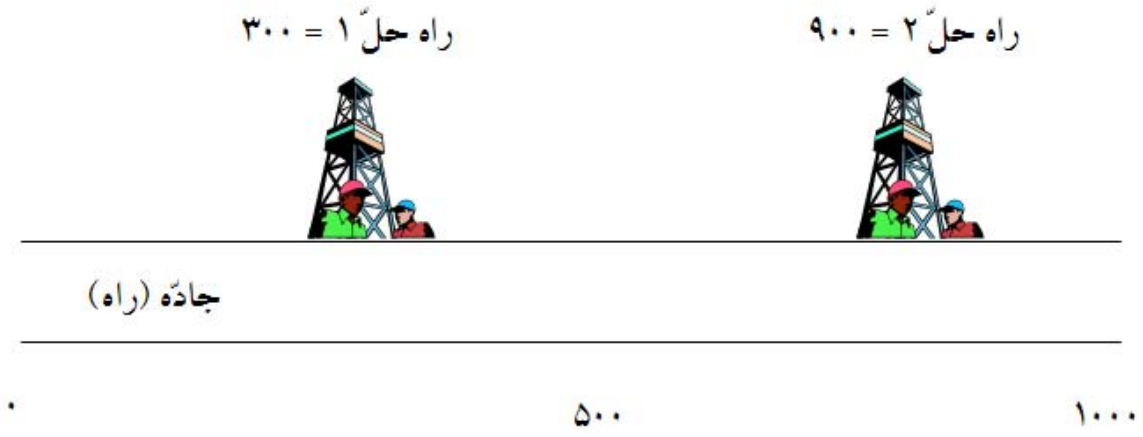
مسأله : بهترین مکان در جادّه را که بیش ترین مقدار روغن را در روز تولید می کند ، انتخاب نمایید .

ما می توانیم هر راه حل را به صورت یک وضعیّت در جادّه نمایش دهیم . تصوّر نمایید که تمام اعداد بین [۱۰۰۰، ۰] باشند .

کجا را برای روغن ، حفر نماییم ؟

---

[۱] bitstrings

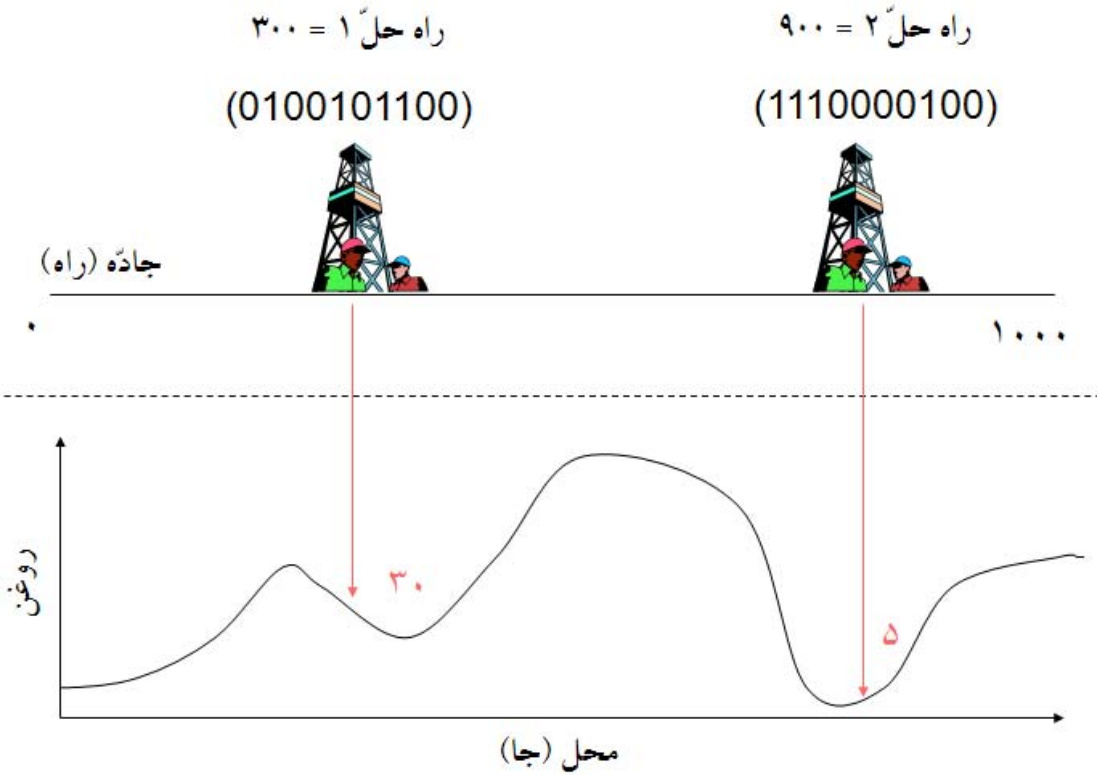


مجموعه ی همه ی راه حل های ممکن [۰،۱۰۰۰] ، فضای جستجو یا فضای حالت نام دارد . در این مورد ، این فقط یک عدد است اما می تواند تعداد زیادی عدد یا سمبل باشد . اغلب ، الگوریتم های ژنتیکی اعداد را به صورت دودویی (باینری) کد می کنند . در مورد این مثال ، ده بیت را که برای نمایش ۰ ... ۱۰۰۰ کافی است را انتخاب می نماییم .

تبدیل به رشته ی باینری

	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 تکی گاهی اوقات ژن ها نامیده می شوند .



خلاصه

ما تا کنون دیدیم که چگونه راه حل ها را به صورت یک عدد بیان نماییم . یک عدد را به صورت یک رشته ی بیتی کد کردیم . یک رتبه یا درجه را برای هر عدد داده شده به منظور تعیین میزان خوب بودن آن راه به آن عدد نسبت دادیم که اغلب *تابع شایستگی* [1] نامیده می شود . مثال روغن در واقع بهینه سازی با استفاده از یک تابع f(x) است که ما باید پارامتر x را تطبیق دهیم .

---

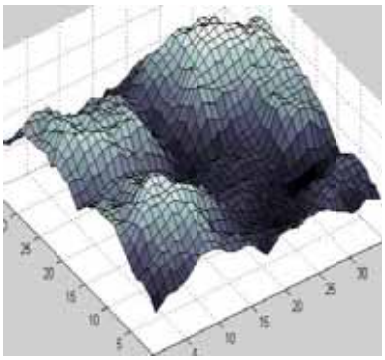
[1] fitness function

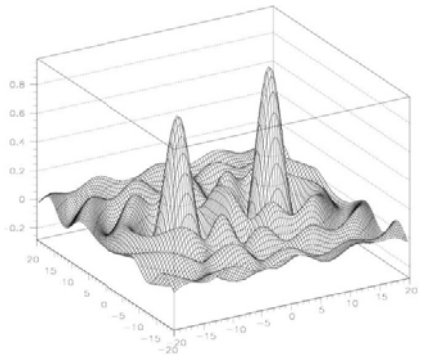
مترجم: سهراب جلوه گر

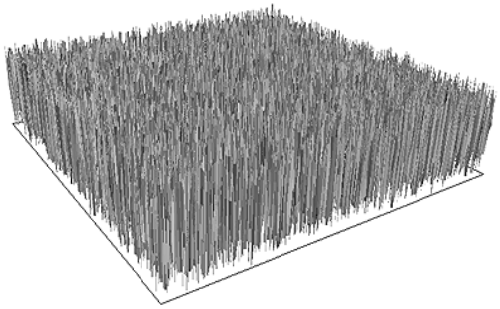
ویرایش دوّم ، بهار ۱۳۸۸

فضای جستجو

برای یک تابع ساده ی $f(x)$ ، فضای جستجو یک بعدی است . امّا با کد کردن چند مقدار به کروموزوم ، ابعاد زیادی می تواند به وجود آید به عنوان مثال برای حالت دوبعدی تابع f به صورت $f(x,y)$ خواهد بود . فضای جستجو می تواند به صورت یک سطح باشد . هر ژنوتایپ ممکن ، یک نقطه در فضا است . یک الگوریتم ژنتیکی تلاش می کند که نقطه ها را به جاهای بهتر (با شایستگی بالاتر) در فضا منتقل نماید . در زیر سه نمونه از منظره های شایستگی را مشاهده می نماییم .







بدیهی است که نوع فضای جستجو تعیین می کند که چگونه یک الگوریتم ژنتیکی کار کند . یک فضای کاملاً تصادفی برای یک الگوریتم ژنتیکی ، بد خواهد بود . همچنین الگوریتم های ژنتیکی ، اگر فضاهای جستجو شامل تعداد زیادی از موارد تصادفی باشند ، ممکن است در ماکزیمم محلّی بیفتند .

اضافه کردن جنسیّت – Crossover

اگرچه که الگوریتمی که ما گفتیم برای فضاهای جستجوی ساده کار می کند امّا این الگوریتم هنوز خیلی ساده است . این الگوریتم به جهش های تصادفی برای یافتن یک راه حلّ خوب ، وابسته می باشد . با اضافه نمودن جنسیّت به این الگوریتم می توانیم نتایج بهتری را به دست آوریم . این کار با انتخاب دو والد در موقع تکثیر [۱] و ترکیب ژن های آن ها برای تولید فرزند انجام می شود . دو رشته ی بیتی (کروموزوم) والد با امتیاز بالا انتخاب می شوند و با نرخ Crossover با هم ترکیب می شوند . دو فرزند (رشته ی بیتی) به وجود می آیند و هر فرزند ممکن است به صورت تصادفی تغییر داده شود (جهش) .

انتخاب والدها

روش های زیادی برای انتخاب کروموزوم های با امتیاز بهتر وجود دارد . امتیاز ، اغلب *شایستگی* [۲] نامیده می شود . از روش " چرخش رولت " [۳] می توان به صورت زیر استفاده نمود :

- شایستگی همه ی کروموزوم ها را اضافه نمایید .
- یک عدد تصادفی R را در محدوده ی آن به وجود آورید .
- اوّلین کروموزوم موجود در جمعیّت را با این شرط که زمانی که همه ی شایستگی های قبلی اضافه می شوند ، لااقل مقدار R را می دهد انتخاب نمایید .

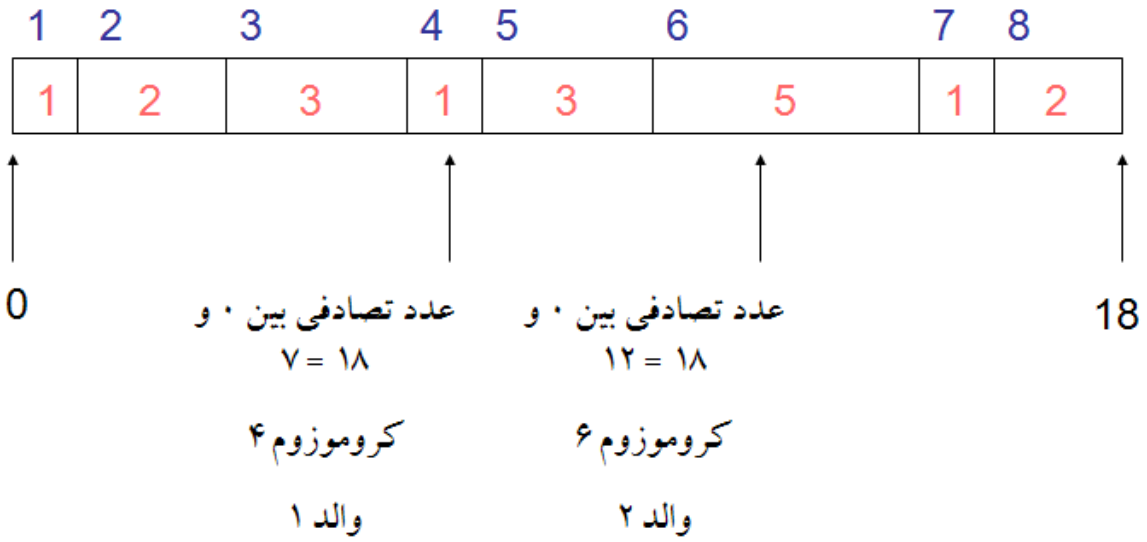
---

[۱] reproduction
[۲] fitness
[۳] Roulette Wheel

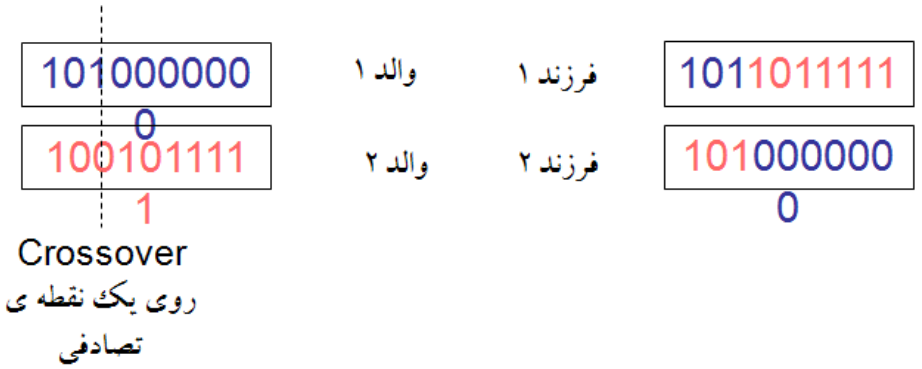
جمعیّت نمونه

شایستگی	کروموزوم	شماره
1	1010011010	1
2	1111100001	2
3	1011001100	3
1	1010000000	4
3	0000010000	5
5	1001011111	6
1	0101010101	7
2	1011100111	8

انتخاب چرخش رولت



Crossover – بازترکیب (جفت گیری) [1]



با احتمال بالا (نرخ Crossover) ، Crossover را به والدها اعمال نمایید . (مقدارهای عادّی بین ۰.۸ تا ۰.۹۵ می باشند)

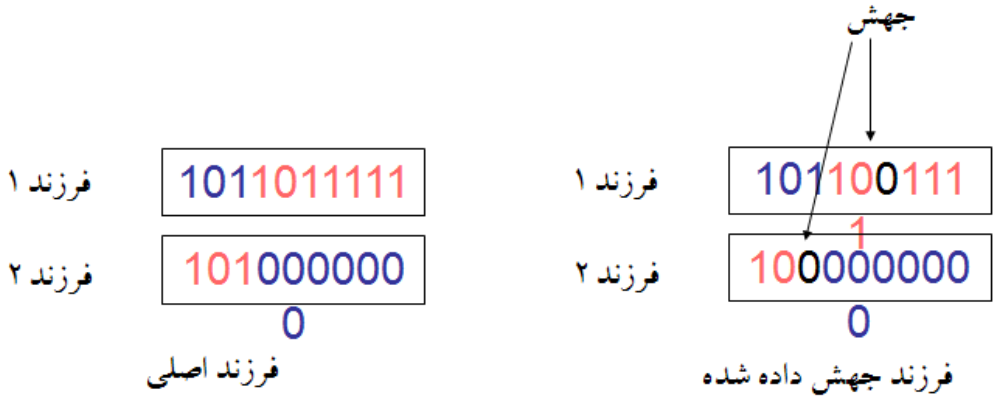
جهش

[1] Recombination : ترکیب جدیدی از فنوتیپها در نتاج حاصل از والدین با فنوتیپی متفاوت .

یک یا بیش تر ژن به صورت تصادفی به یک مقدار تصادفی ، جهش داده می شوند ، مثلاً :

۱۱۱۱۱۱۱۱۱۱ → ۱۱۰۱۰۱۱۱۱۱

در مورد مثال قبلی داریم :



برگشت به الگوریتم ژنتیکی

یک جمعیّت را با استفاده از کروموزوم های تصادفی ، تولید کن

برای هر نسل کارهای زیر را انجام بده

شایستگی هر کروموزوم را محاسبه کن

کارهای زیر را تکرار کن

انتخاب رولت را برای انتخاب جفت های والدها به کار ببر

فرزند را با استفاده از Crossover و جهش تولید نما

تا زمانی که جمعیّت جدید تولید شود

تا زمانی که بهترین راه حل به اندازه ی کافی خوب شود

تنوّع های زیاد الگوریتم های ژنتیکی

به خاطر موارد زیر ، الگوریتم های ژنتیکی دارای تنوّع زیادی هستند :

- انواع مختلف انتخاب که رولت نیستند ؛ مثل ، مسابقه [۱] ، نخبه سالاری [۲] و غیره .
- تفاوت در جفت گیری ؛ مثل ، Crossover چند نقطه ای .
- انواع مختلف کد کردن رشته ی بیتی ؛ مثل ، مقدارهای صحیح و مجموعه ی منظّم از سمبل ها
- انواع مختلف جهش

پارامترهای زیاد برای تنظیم

هر پیاده سازی الگوریتم ژنتیکی به تصمیم گیری در مورد تعدادی از پارامترها نیاز دارد ؛ مثل ، اندازه ی جمعیّت (N) ، نرخ جهش (m) ، نرخ Crossover (c) . که اغلب ، این ها باید براساس نتیجه های به دست آمده تنظیم شوند . مقدارهای معمولی شاید $N = 50$ ، $m = 0.05$ و $c = 0.9$ باشند .

چگونه Crossover کار می کند ؟

تعداد زیادی نظریّه در این مورد وجود دارد البتّه برخی مخالفت ها هم با این نظریّه ها وجود دارد . آقای هُلَند نظریّه ی " الگو " را معرّفی کرد ؛ این ایده می گوید که Crossover ، بیت های خوب والدین مختلف را نگهداری می کند و آن ها را برای تولید راه حل های بهتر ترکیب می نماید . بنابراین ، یک الگوی خوب کد کننده باید بیت های خوب را در طول Crossover و جهش نگهداری نماید .

---

[۱] Tournament

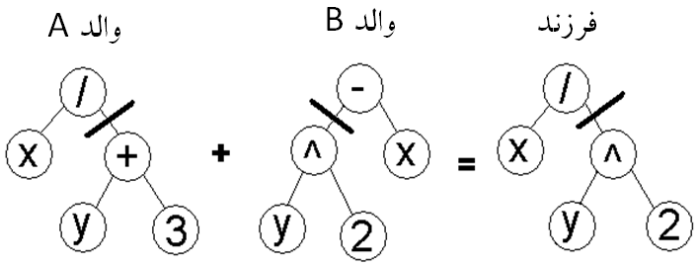
[۲] Elitism

کاربردهای الگوریتم های ژنتیکی

الگوریتم های ژنتیکی در بسیاری از موارد و زمینه های تحقیقاتی ، نظیر مسایل عددی ، مسأله 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 .

برنامه نویسی ژنتیکی

زمانی که کروموزوم یک برنامه یا تابع را کد می کند ، این کار برنامه نویسی ژنتیکی نامیده می شود . برای انجام این کار معمولاً از ارایه ی درختی استفاده می شود . Crossover ، باعث جابه جا شدن زیر درخت ها میان والدین می شود .



این کار برای برنامه های کوچک ، مفید است ولی برای برنامه های بزرگ با توابع پیچیده ، ناکارآمد می باشد .

توابع شایستگی ناآشکار (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 .

مسأله

در مسأله 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 مسأله ای مشکل می باشد ، زیرا تعداد مسیرهای ممکن برابر !N می باشد ، که N ، برابر با تعداد شهرها می باشد . 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 Crossover را برای مسأله ی مسافرت شخص دوره گرد پیدا می نماید . فرض کنید تعداد شهرها (N) برابر ۱۰ می باشد . بعد از همه ی عملیّات ، کروموزوم های تولید شده باید همیشه مسافرت های ممکن مجاز را ارایه نمایند (هر شهر فقط باید یک بار ملاقات شود) . برای این مسأله ، یک راه حل وجود ندارد و تعداد زیادی الگوهای مختلف ، قبلاً استفاده شده اند .

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل بیستم

سیستم های خبره [۱]

[۱] Expert systems

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

سیستم های خبره ، برنامه هایی طرّاحی شده برای مدل سازی و استدلال در مورد دانش انسان می باشند و معمولاً حول یک دامنه مثل ، تشخیص بیماری ، پرواز فضایی ، لجستیک (اقدامات مربوط به تهیه و توزیع) و عیب یابی نرم افزار تمرکز می کنند . در این سیستم ها ، برنامه نویسان ، *واقعیّات* و *قوانین* را اضافه می کنند و سیستم ، پردازه ی استنتاج را به صورت خودکار انجام می دهد و می توانند یا از زنجیره ی مستقیم (سیستم های تولید) یا از زنجیره ی معکوس (ثابت کننده های قضیّه) 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 اطّلاعات کلّی تر را به نمایش نمی گذارد و در یک محدوده ی نسبی و با تخصّص نسبی کار می کند . امّا در آن زمینه عملکردش در سطح خبره می باشد . این سیستم ها ممکن است برای طبقه بندی ، تشخیص عیب ، پیداکردن خرابی ، تفسیر اطّلاعات ، طرّاحی ، 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 تکثیر خبره ی نادر و پرهزینه ؛ برای فرمول بندی دانش خبره و برای جمع آوری منابع دانش متمایز یک سیستم خبره را می سازیم . 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 . البتّه ، سیستم های خبره معایب زیادی هم دارند ، که باید استفاده از آن ها را به دامنه های مشخّص ، محدود نماییم . سیستم های خبره دارای بینش ، دلسوزی ، فهم انگیزه ی بشری ، توانایی حدس ، توانایی یادگیری (معمولاً) ، دانش قضاوت صحیح (عقل سلیم) کمی می باشند . در این مورد ، کاربر می تواند دلسوزی و قضاوت صحیح را اضافه نماید .

خصوصیّات خوب یک سیستم خبره

در صورتی که سیستم خبره ، به صورت 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 این موارد می باشد : شیوه ی سؤالاتی که سیستم می پرسد

[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 معمولاً شامل کربن ، هیدروژن ، نیتروژن و اکسیژن می باشند . تمام موجودات زنده از این ترکیبات آلی تشکیل شده اند .

باید طبیعی باشد . سؤالات 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 سؤال را می پرسد و هر نتیجه ای که به آن می رسد را توجیه نماید و باید به کاربر در مورد سیستم ، اطمینان دهد .

در استدلال ، باید یک سیستم خبره ، قادر به انجام این موارد باشد : باید استنتاج هایش باورکردنی و نه الزاماً صحیح باشند ؛ به عنوان مثال ، یک سیستم خبره که یک وضعیت بهداشتی را از یک مجموعه از علایم ، استنتاج می کند ؛ بعید است که وضعیّت ، رشته ای منطقی از علایم (نشانه ها) باشد . باید قادر به کار با دامنه ی دانش ناقص و مواردی که اطّلاعات ناقص هستند ، باشد . اغلب ، دانش و / یا مورد اطّلاعات ، ناکامل می باشند و دانش و / یا داده ها ممکن است قابل اعتماد نباشند (مثلاً ، 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) [۱] طرّاحی شده باشند . باید بتوانند به سادگی اطّلاعات جدید را یکی نمایند (یا از طریق مهندسی دانش بیش تر یا با استفاده از آموزش ماشینی) .

سیستم های خبره ی قانون گرا [۲]

روش های زیادی می توانند برای ساخت سیستم های خبره استفاده شوند . امّا ، اکثر سیستم های خبره که تا کنون ساخته شده اند سیستم های قانون گرا می باشند ؛ یعنی ، پایگاه دانش ، شامل واقعیّت ها و قانون ها می باشد .

پایگاه دانش

---

[۱] maintainability

[۲] rule-based

پایگاه دانش ، شامل واقعیّت ها و قوانین می باشد . تصوّر کنید که ما در حال ساخت یک سیستم خبره ی قانون گرا برای تشخیص پزشکی بودیم . قانون ها ارتباط های میان علایم و عبارت های پزشکی را کد خواهند کرد . واقعیّت 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 p_1 AND p_2 … AND p_n then q

$(p_1 \wedge \ldots \wedge p_n) \Rightarrow q$

$q \vee \neg p_1 \vee \ldots \vee \neg p_n$

برای سیستم های قانون گرا ، استفاده از منطق گزاره ای ، زمانی که گزاره ها دارای هیچ آرگومانی نمی باشند ، کاملاً معمولی است . بنابراین متغیّرها و سمبل های تابعی در واقعیّت ها و قانون ها وجود ندارند . ما در مثال های زیر خودمان را با این وضعیّت ، 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

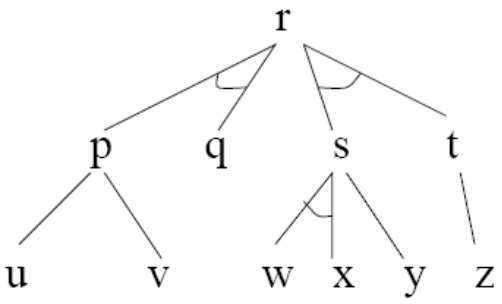
if v **then** p

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*[1]

if *animalHasHair* **then** *animalIsMammal*

if *animalIsMammal* AND *animalChews*[2]*Cud*[3] **then** *animalIsUngulate*[4]

if *animalIsUngulate* AND *animalHasLongNeck*[5] **then** *animalIsGiraffe*[6]

if *animalIsUngulate* AND *animalIsStriped*[7] **then** *animalIsZebra*[8]

ما از این مثال ، بعداً استفاده خواهیم کرد .

---

[۱] پستاندار
[۲] می جَوَد
[۳] نُشخوار می کند
[۴] سُم دار
[۵] گردن
[۶] زرّافه
[۷] راه راه یا خط دار
[۸] گورِ خر

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 [۱] انجام می دهد . از یک نقطه نظر منطقی ، اساساً 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 مؤثّر ، ما به دنبال یک مسیر که ریشه و برگ ها را به هم متّصل می کند و این موارد 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 واقعیّت را که باید درست باشد را نمایش می دهد .

در سیستم های خبره ی محاوره ای ، ما فرض نمی کنیم که همه ی واقعیّات شناخته شده اند و قبلاً در پایگاه دانش وجود داشته اند . بنابراین ، در زمانی که ما در تلاش برای این هستیم که ببینیم که آیا یک گره ی برگ (واقعیّت) درست است یا نه ، ما پایگاه دانش را برّرسی خواهیم نمود ، اما اگر در پایگاه دانش نباشد ، در این صورت ما از کاربر پرسش می کنیم که آیا واقعیّت 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 – 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) . امّا ، 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 عبارت

---

[۱] Rule-Based Reasoning (RBR)

های 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 [۱] – برخی از سیستم ها منحصراً از یکی از این روش ها یا هر دوی این روش ها استفاده می کنند . امّا 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 توجّه 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 اطّلاعات دیگری خواسته شود . و این کار (پروسه) باز هم تکرار شود .

توضیح های استدلال

ما قبلاً اهمّیّت داشتن توضیح روان را در یک سیستم خبره بیان کردیم . سیستم های قانون گرا معمولاً دو امکان را برای توضیحات ارایه می کنند . ممکن است یا بپرسند چرا و یا بپرسند چگونه . سیستم ، سؤالات را با نمایش برخی از قوانین مربوط ، جواب می دهد . برای مثال ، تصوّر 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 سؤال بپرسد که : چرا شما این سؤال 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*

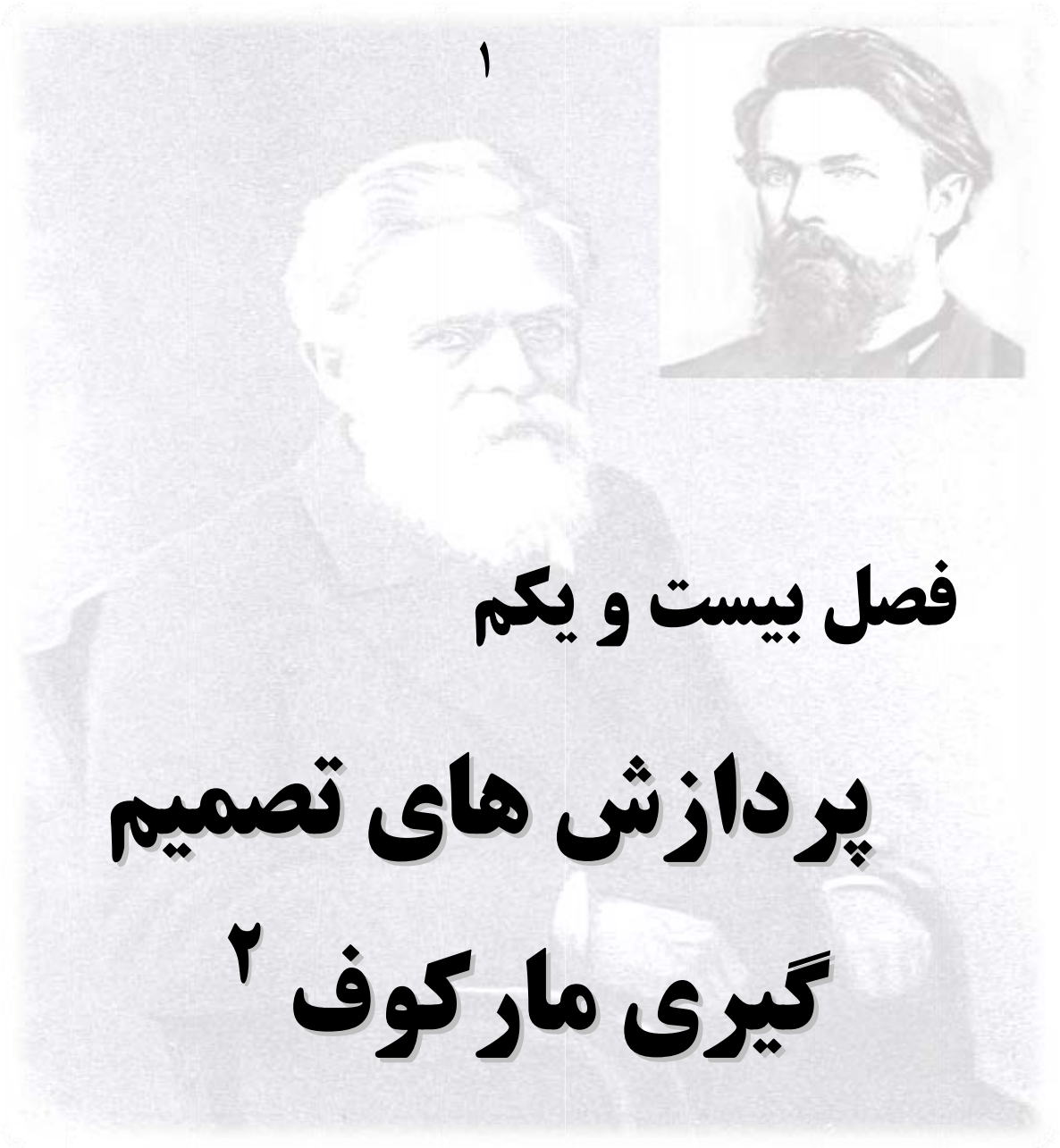
---

[۱] Interleaved Backwards- and Forwards-Chaining

کند . قبلاً 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 زرّافه است کمک می کند . در صورتی که حیوان سم دار باشد و دارای گردن بلند باشد ، زرّافه می باشد . به بیان دیگر ، تصوّر 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 ؛ یعنی این که بپرسد : چگونه به این نتیجه رسیدی ؟ برای این کار ، سیستم خبره با نمایش قسمت های موفّق 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 ، تصوّر 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 مدّت 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 برّرسی ، کامل کردن و پیکربندی تقاضاهای تجهیزات کامپیوتری مشتری ، می باشند . بعد از برخی از آزمایش ها با ساخت چند

سیستم خبره ی قانون گرا ، محقّقان هوش مصنوعی دریافتند که فقط پایگاه دانش ، وابسته به دامنه می باشد . موتور استنتاج و رابط کاربر (نسبتاً) مستقل از دامنه بودند . به طور کلّی ، 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 .



۱

فصل بیست و یکم

پردازش های تصمیم گیری مارکوف ۲

[۱] تصویر بالا متعلّق به آندرِی اِی . مارکوف ((۱۹۲۲ – ۱۸۵۶) Andrei A. Markov) ، دانشمند و ریّاضیدان معروف روسی است .

[۲] Markov Decision Processes(MDP)

پردازش های تصمیم گیری مارکوفی

یک مدل گسسته برای تصمیم گیری تحت نـامعلومی مـی باشـد . چهـار جـزء ایـن مـدل عبارتنـد از : *وضعّیت ها* ؛ که جهان به وضعیّت هایی تقسیم شده است . *عملکردها* ؛ هر وضعیّت دارای یک تعـداد محـدود از عملکردها می باشد . *تابع انتقال* ؛ ارتباطی احتمالی میان وضعیّت ها و عملکردهای قابـل دسترسـی بـرای هـر وضعیّت و *تابع پاداش* ؛ پاداش مورد انتظار انجام یک کار ، تحت وضعیّت s می باشد .[۱]

به وجود آوردن تصمیم گیری های ترتیبی

قبلاً ما در مورد این موارد صحبت کردیم : به وجود آوردن تصـمیم گیـری هـای بـه یکبـاره در یـک محیط قطعی ؛ به وجود آوردن تصمیم گیری های ترتیبی در یک محیط قطعـی ، مثـل : جسـتجو – اسـتنتاج – برنامه ریزی ؛ به وجود آوردن تصمیم گیری های به یکباره در یک محیط اتّفـاقی ، مثـل : شـبکه هـای بـاور و

---

[۱] http://www.sci.brooklyn.cuny.edu/~parsons/courses/790-spring-2004/notes/incremental.ppt

احتمال - پیش بینی سودمندی . حال این سؤال مطرح است که : تصمیم گیری های ترتیبی در یک محیط اتّفاقی ، چگونه می باشند ؟ .

سودمندیِ مورد انتظار

به یاد بیاورید که سودمندی مورد انتظار یک عملکرد ، سودمندی هر نتیجه ی ممکن است و توسّط احتمالی که نتیجه اتّفاق بیفتد ارزیابی (وزن) می شود ؛ به طور صریح تر ، از وضعیّت s ، یک عامل ممکن است عملکردهای a_1 ، a_2 ، ... ، a_n را بگیرد . هر عملکرد a_i می تواند منجر به وضعیّت های $s_{i1}, s_{i2}, \ldots, s_{im}$ با احتمال $p_{i1}, p_{i2}, \ldots, p_{im}$ شود ؛ در نتیجه ، سودمندی مورد انتظار برای عملکردها برابر است با : $EU(a_i) = \sum p_{ij} s_{ij}$. ما مجموعه ای از احتمال ها و وضعیّت های وابسته را *مدل انتقال وضعیّت* [1] می نامیم . عامل باید وضعیّت 'a ای که سودمندی مورد انتظار را بیشینه می کند ، انتخاب نماید .

محیط های مارکوفی

ما می توانیم این ایده را برای محیط های ترتیبی بسط دهیم . در این مورد ، مسأله این است که چگونه احتمال وضعیّت ها را تشخیص دهیم ؟ ؛ احتمال رسیدن به وضعیّت s ارایه شده توسط عملکرد a ممکن است وابسته به عملکردهای قبلی ای که انجام شده اند باشد . فرض مارکوف می گوید که احتمال تغییر وضعیّت ها فقط وابسته به یک تعداد محدودی از والدها می باشد . ساده ترین ، پردازه ی مرتبه ی اوّل مارکوف در این صورت است که احتمال تغییر وضعیّت ها فقط وابسته به وضعیّت قبلی باشد . ما نیز روی پردازش مرتبه ی اوّل مارکوف تمرکز می نماییم .

توزیع های ثابت [2]

---

[1] state transition model

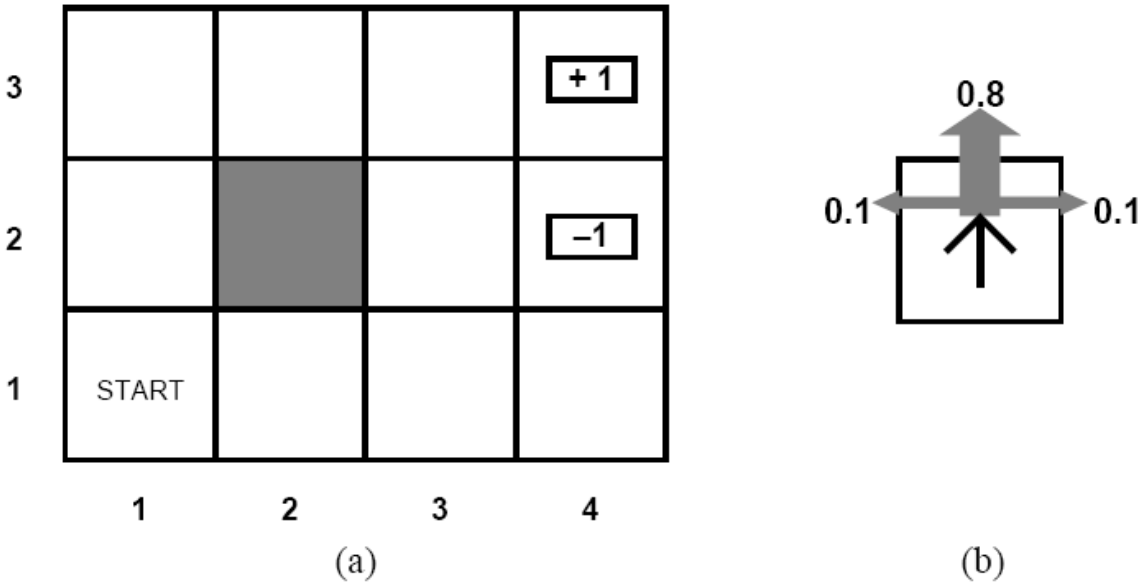
[2] Stationary distributions

ما فرض می کنیم که توزیعمان ثابت باشد ، *که این به این معنی است که احتمال رسیدن به یک وضعیّت* s' *ارائه شده توسط عملکرد* a *از وضعیّت* s *با تاریخچه ی* H *تغییر نمی یابد* ؛ تاریخچه های مختلف ، ممکن است احتمال های مختلف را به وجود بیاورند . داشتن تاریخچه های یکسان ، تغییر وضعیّت های یکسان را سبب خواهد شد . ما همچنین فرض خواهیم کرد که سودمندی وضعیّت ، تغییری را در روش مسأله به وجود نمی آورد .

حل مسایل ترتیبی

در این مورد ، سودمندی ، وابسته به یک رشته از وضعیّت های $s_1, s_2, \ldots, s_n$ می باشد و برای هر وضعیّت ، یک پاداش $R(s_i)$ را در نظر می گیریم . 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 یک پردازه ی تصمیم گیری مارکوفی داریم : یک وضعیت اوّلیه ی s_0 ؛ یک مجموعه از وضعیّت ها و عملکردهای گسسته ؛ یک مدل انتقال : $T(s,a,s')$ که احتمال رسیدن به وضعیّت s' از s وقتی که عمل a انجام می شود را نشان می دهد و یک تابع پاداش $R(s)$.

مثال : مسأله ی شبکه



در این مثال ، عامل در جهت خواسته شده با احتمال ۰.۸ حرکت می کند و در یک جهت راست یا چپ با احتمال ۰.۲ حرکت می کند ؛ در هر وضعیّت ، یک عامل باید چه کاری انجام دهد تا پاداش را بیشینه نماید ؟

راه حل های پردازه ی تصمیم گیری مارکوفی – از آنجایی که محیط ، تصادفی یا اتّفاقی می باشد ، یک راه حل به صورت رشته ای از عملکرد نمی باشد . در عوض ، ما باید معیّن نماییم که یک عامل باید چه کاری در هر وضعیّت ممکن انجام دهد . ما این خصوصیّت 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 وضعیّت s ، مشخّص 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]

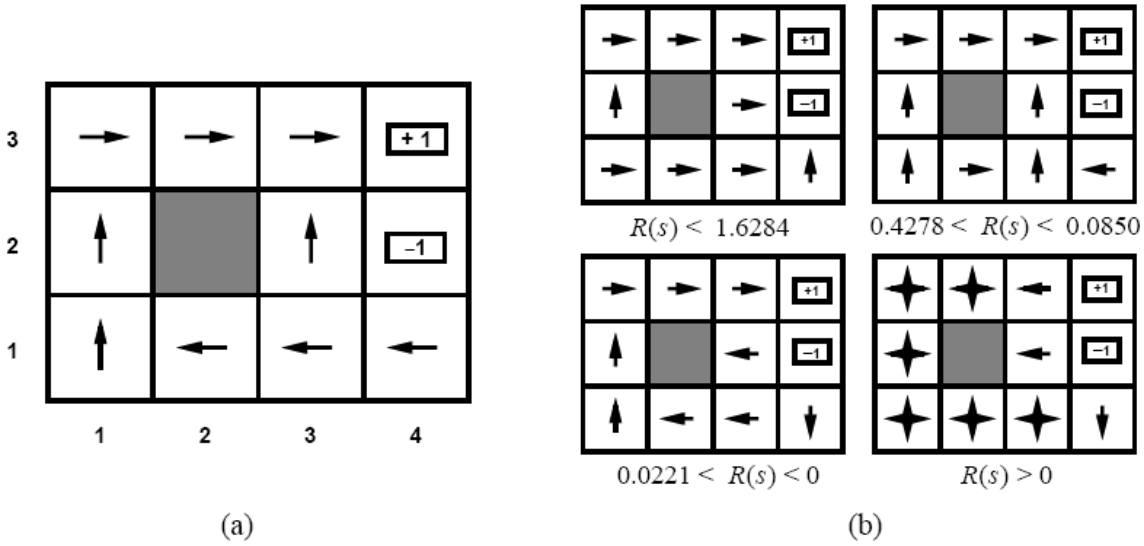
---

[1] policy

[2] optimal policy

نام دارد . اوّلین باری که یک روش پیدا می شود ، عامل فقط می تواند بهترین عملکرد را برای هر وضعیّت پیدا نماید .

مثال : مسأله ی شبکه



توجّه کنید که در شکل سمت چپ ، R(s) برابر است با ۰.۰۴ − و در شکل ، تمام پاداش های غیر صفر ، باید منفی باشند . از آنجایی که هزینه های تغییر در وضعیّت های غیرپایانی تغییر می یابد ، بنابراین ، روش بهینه را انجام می دهد . وقتی که هزینه خیلی بالا باشد ، عامل تلاش می کند که بلافاصله خارج شود ؛ در زمینه ی میانی ، عامل تلاش می نماید که از خروج بد جلوگیری نماید و در پاداش مثبت ، عامل تلاش نمی کند که خارج شود .

مطالب بیش تری در مورد توابع پاداش

در حل یک پردازه ی تصمیم گیری مارکوفی ،یک عامل باید به مقدار عملکردهای آینده توجّه نماید . انواع مختلفی از مسایل که باید به آن ها توجّه نماید ، وجود دارد :

- وسعت – آیا جهان برای همیشه ادامه می یابد ؟

o وسعت محدود : بعد از N عملیّات ، جهان متوقّف می شود و هیچ پاداشی دریافت نمی شود .

o وسعت نامحدود : جهان به صورت نامحدود ادامه می یابد یا این که ما نمی دانیم چه هنگامی متوقّف 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* [۱] در پاداش های آینده این مورد را مدل سازی می نماییم .

اگر γ ، فاکتور تخفیف باشد داریم :

$$U(s_0, s_1, s_2, s_3, ...) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + ..., \gamma \in [0,1]$$

در صورتی که γ بزرگ باشد ، ما وضعیّت 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 ۱- (γ / ۱) . تخفیف به ما اجازه می دهد که مقدار محسوسی ، مسایل با وسعت نامحدود را داشته باشیم . در غیر این صورت ، همه ی سودمندی های مورد انتظار باید نزدیک به مقدار نامحدود باشند . سودمندی های 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 :

$$\pi^* = \arg\max_{\pi} EU\left(\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right)$$

---

[۱] discounting

تکرار مقدار

چگونه یک روش بهینه را پیدا نماییم ؟ ما با محاسبه ی سودمندی مورد انتظار هر وضعیّت و سپس انتخاب عملکردهایی که سودمندی مورد انتظار را بیشینه می کنند شروع می نماییم . در یک مسأله ی ترتیبی ، سودمندی یک وضعیّت ، سودمندی مورد انتظار همه ی رشته های وضعیّتی 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 وضعیّت s می باشد .

سودمندی وضعیّت ها

3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

توجّه کنید که سودمندی ها دارای بیش ترین مقدار برای وضعیّت های نزدیک به خروجی ۱+ هستند . سودمندی یک وضعیّت ، پاداش فوری برای آن وضعیّت به اضافه ی سود تخفیف داده شده ی مورد انتظار وضعیّت بعدی می باشد و فرض براین 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 وضعیّت نهایی به وضعیّت های اولیّه 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 وضعیّت ها وابسته به سودمندی سایر وضعیّت ها می باشد ، چگونه یک راه حل نزدیک را پیدا نماییم ؟ ما می توانیم از یک نگرش تکراری به صورت زیر استفاده نماییم :

- هر وضعیّت را به صورت تصادفی مقداردهی اولیّه نمایید .
- برای یک وضعیّت ، طرف سمت چپ را با توجّه به مقدار های همسایگانش محاسبه نمایید .
- این کار را برای به روز رسانی طرف سمت راست دیگر وضعیّت ها ادامه دهید .
- قانون به روز رسانی به صورت زیر است :

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U_i(s')$$

الگوریتم تکرار مقدار

انجام دهید [۱]

برای s های موجود در وضعیّت ها

$$U(s) = R(s) + gamma * \max T(s,a,s') U(s')$$

تا هنگامی که [۲]

تمام سودمندی ها با مقدار کم تر از دلتا تغییر نمایند (که دلتا برابر است با : $\delta = error * (1-\gamma)/\gamma$) .

مثال :

1 0.1	2 -0.1	3 0.05	**+1**
4 -0.02		5 0.15	**-1**
6 0.0	7 0.1	8 -0.1	9 0.15

---

[۱] do

[۲] until

با نسبت دادن سودمندی های تصادفی به هر وضعیّت ، شروع می کنیم ؛ فرض می کنیم γ برابر با ۸. ۰ باشد ؛ هزینه ی زمان برابر با ۰۴ . ۰ - باشد ($R(s) = -0.04$) و خطا برابر با ۰۱ . ۰ باشد ($\delta = 0.0025$)

بعد از یک بار تکرار ، مقادیر تخمین زده شده در اینجا آورده شده است :

1 0.03	2 -0.02	3 0.62	**+1**
4 0.02		5 0.05	**-1**
6 0.02	7 0.02	8 0.08	9 0.06

بعد از دو بار تکرار ، مقادیر تخمین زده شده در اینجا آورده شده است :

1 -0.02	2 0.35	3 0.65	+1
4 -0.02		5 0.28	-1
6 -0.02	7 0.01	8 0.02	9 0.01

بعد از سه بار تکرار ، مقادیر تخمین زده شده در اینجا آورده شده است :

1 0.19	2 0.43	3 0.69	+1
4 -0.06		5 0.32	-1
6 -0.04	7 -0.03	8 0.14	9 -0.03

بعد از چهار بار تکرار ، مقادیر تخمین زده شده در اینجا آورده شده است :

1 0.25	2 0.47	3 0.68	+1
4 0.07		5 0.34	-1
6 -0.07	7 0.04	8 0.16	9 -0.03

بعد از پنج بار تکرار ، مقادیر تخمین زده شده در اینجا آورده شده است :

1 0.27	2 0.47	3 0.68	+1
4 0.13		5 0.34	-1
6 0.0	7 0.07	8 0.18	9 -0.02

بعد از شش بار تکرار ، مقادیر تخمین زده شده در اینجا آورده شده است :

1 0.29	2 0.47	3 0.68	+1
4 0.15		5 0.34	-1
6 0.04	7 0.08	8 0.18	9 -0.01

و در این نقطه ، ما به یک نزدیکی (همگرایی) رسیده ایم .

تشریح مطلب

نقاط قوّت روش تکرار مقدار : برای نزدیک شدن (همگرایی) به راه حل صحیح ، ضمانت شده است و الگوریتم تکرار ، ساده می باشد .

نقاط ضعف روش تکرار مقدار : نزدیک شدن (همگرایی) می تواند به صورت کند باشد ، ما در واقع به تمام این اطّلاعات نیاز نداریم و فقط به این نیاز داریم که بدانیم در هر وضعیّت چه کاری را انجام دهیم .

تکرار روش [1] (بهبود روش تکرار مقدار)

[1] Policy iteration

به طور مستقیم برای روش های بهینه جستجو می کنیم ؛ به بیان دیگر ، روش ها را با تکرار برای هر وضعیّت ، به روز می نماییم . این کار دارای دو مرحله می باشد : ارایه ی یک روش ، محاسبه ی سودمندی ها برای هر وضعیّت . به دست آوردن یک روش جدید براساس این سودمندی های جدید .

الگوریتم

سودمندی همه ی وضعیّت ها را برابر با صفر قرار بده

رشته ی روش های تصادفی شاخص گذاری شده توسط وضعیّت را در Pi قرار بده

انجام دهید [۱]

سودمندی هر وضعیّت را برای Pi ارزیابی نمایید و آن را در U قرار دهید .

برای s موجود در وضعیّت ها

عملکردی که سودمندی مورد انتظار را برای آن وضعیّت ، بیشینه می نماید پیدا نمایید و در a قرار دهید

a را در Pi(s) بریزید

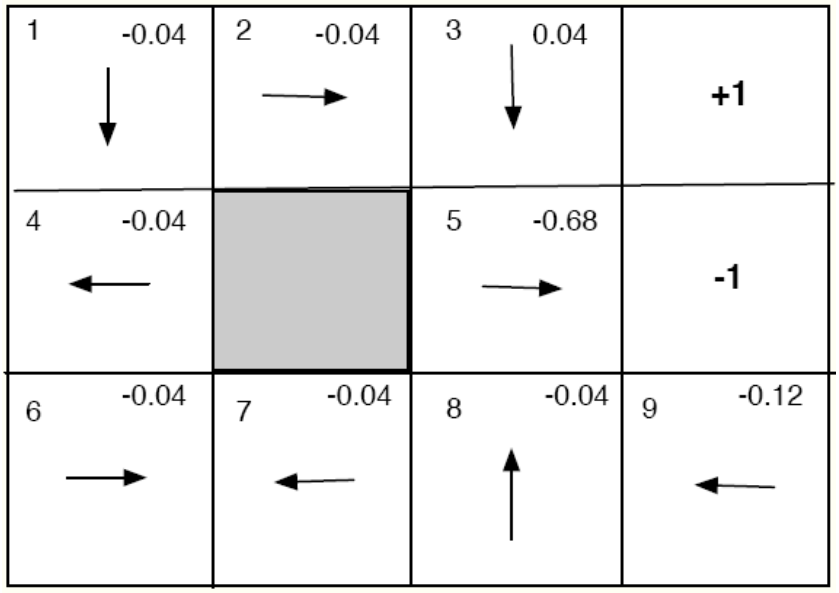
مادامی [۲] که برخی از عملکردها تغییر کردند

مثال :

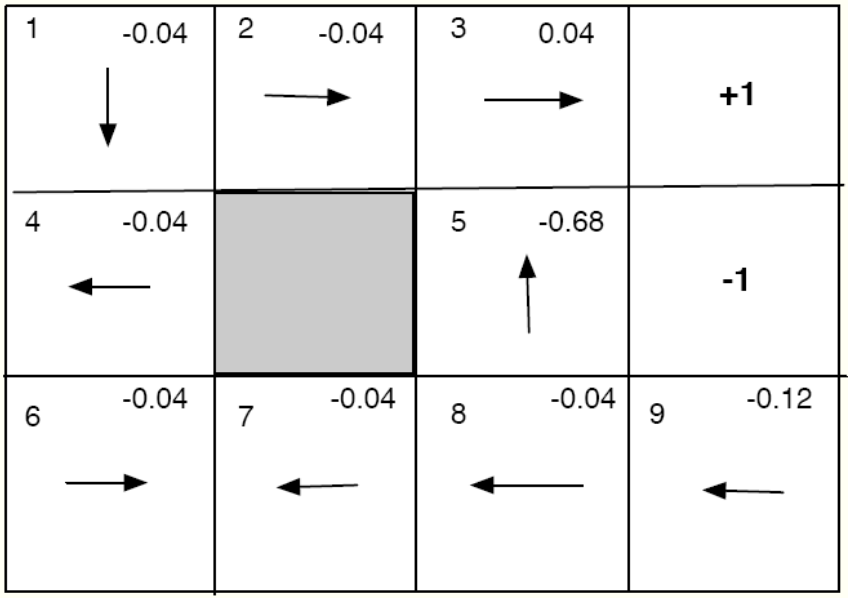
---

[۱] do

[۲] while



روش های تصادفی را بـه وجـود آوریـد ، سـودمندی وضـعیّت هـا را براسـاس ایـن روش هـا ، مـورد ارزیابی قرار دهید .



روش های بهینه ای که این سودمندی ها را ارایه می نمایند ، انتخاب نمایید .

1 -0.07 ↓	2 -0.02 →	3 0.55 →	+1
4 -0.07 ←		5 -0.14 ↑	-1
6 -0.07 →	7 -0.07 ←	8 -0.12 ←	9 -0.12 ←

براساس روش های جدید به دست آمده ، سودمندی های جدید را برای هر وضعیّت ، تخمین بزنید .

1 -0.04 →	2 -0.02 →	3 0.55 →	+1
4 -0.04 ←		5 -0.14 ↑	-1
6 -0.04 →	7 -0.04 ←	8 -0.12 ←	9 -0.12 ↓

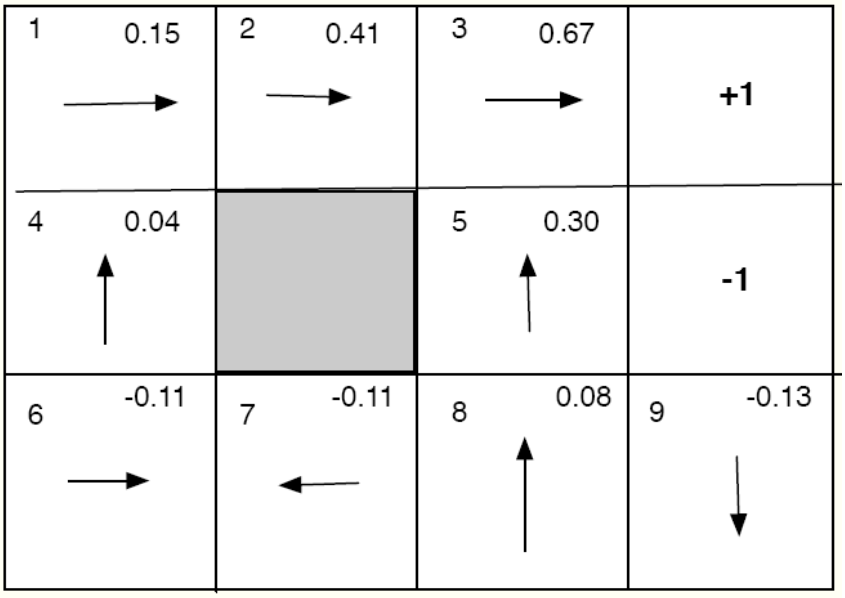
براساس این سودمندی های جدید ، روش های بهینه را انتخاب نمایید .

1 -0.06 →	2 0.31 →	3 0.63 →	+1
4 -0.09 ←		5 0.22 ↑	-1
6 -0.09 →	7 -0.09 ←	8 -0.10 ←	9 -0.12 ↓

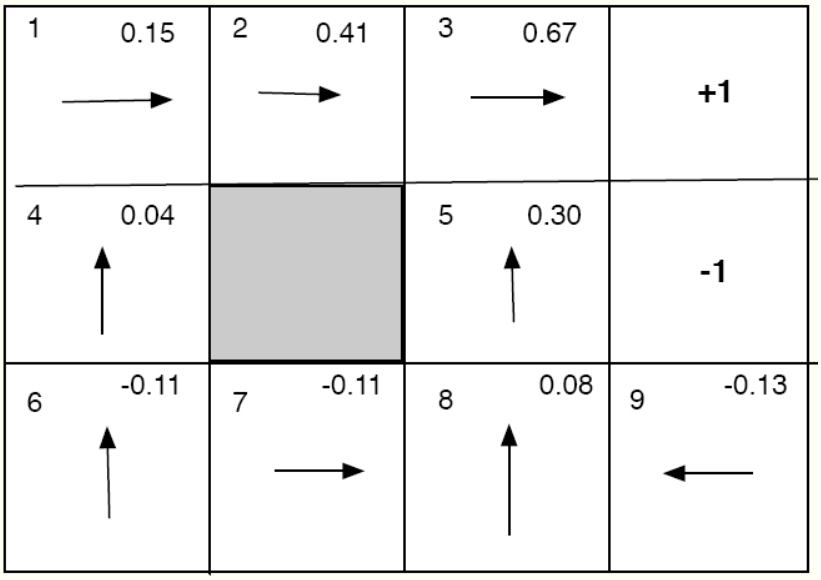
از این روش های جدید برای تخمین مجدّد سودمندی ها استفاده نمایید .

1 -0.06 →	2 0.31 →	3 0.63 →	+1
4 -0.04 ↓		5 0.22 ↑	-1
6 -0.04 →	7 -0.04 ←	8 -0.08 ↑	9 -0.12 ↓

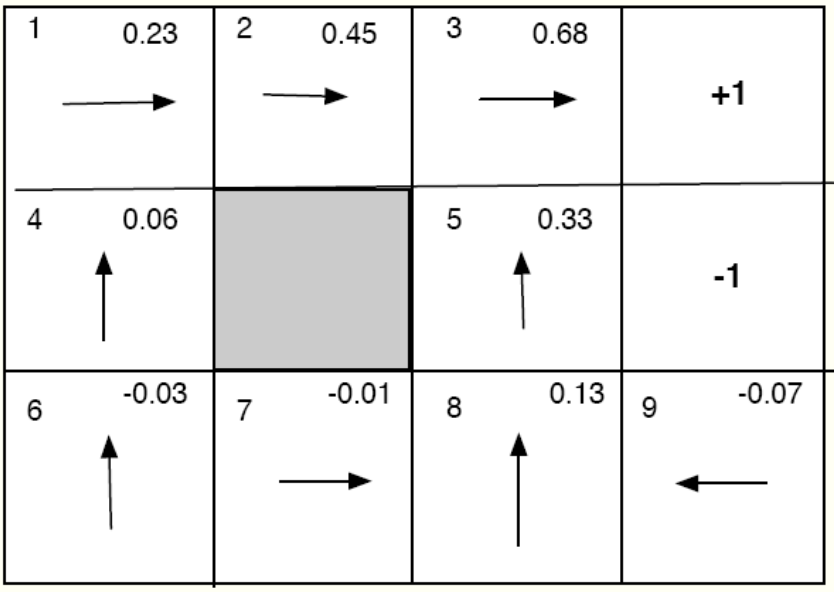
و از این تخمین های سودمندی جدید برای به دست آوردن روش های بهینه استفاده نمایید .



مجدّداً ، از روش های به دست آمده برای تخمین مجدّد سودمندی ها استفاده نمایید .



و سپس از سودمندی ها برای به روز کردن روش های بهینه استفاده نمایید .



و دوباره سودمندی های تخمین ها را براساس روش جدید ، به روز نمایید . اگر ما روش را براساس تخمین های جدید به روز کردیم و دیدیم که تغییری به وجود نمی آید ، در این صورت ، کار به پایان رسیده است . حُسن تکرار روش در این است که نزدیک شدن یا همگرایی سریع تر می باشد .

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل بیست و دوّم

سیستم های طبقه بندی کننده[۱]

[۱] Classifier Systems

سیستم های طبقه بندی کننده

الگوریتم های ژنتیکی برای مسایل بهینه سازی ، خیلی خوب کار می کنند و داریم $f(x_1, x_2, x_3, ..., x_n) = R$. 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 بادقّت تر باشیم . *طبقه بندی* ، پردازه ی انتساب یک ورودی به یکی از کلاس های چندگانه می باشد .

سیستم های تولید [۱]

یک سیستم تولید ، از یک مجموعه از تولیدات (قوانین) ، حافظه ای که در آن واقعیّات قرار می گیرند و یک الگوریتم که با استفاده از روش زنجیره ی مستقیم اجرا می شود و واقعیّات جدید را با استفاده از قبلی ها به وجود می آورد تشکیل شده است . یک قانون ، زمانی اجرا می شود که مجموعه ای از شرایط ، که در حافظه هستند ، برقرار باشند . [۲] سیستم های تولید ، یک روش رایج در هوش مصنوعی هستند . و در آن ها

---

[۱] Production system

[۲] Babylon / FOLDOC

از اگر – آن گاه یا قانون های شرط – عملکرد استفاده می شود . مثلاً در دنیای جاروبرقی داریم : در صورتی که خانه ی [۱و۱] جارو نشده باشد و در خانه ی مجاور یا همسایه باشد ، آن گاه به [۱،۱] برو . یک سیستم تولید ، از تعداد زیادی قانون تشکیل شده است . برخی از قوانین ممکن است باعث عمل کردن قانون های دیگر شوند . در سیستم های تجاری ، مسأله این است که وقتی برای یک مورد ، بیش از یک قانون داریم ، چه کار کنیم [و کدام را اجرا نماییم] ، ما بعداً به این مورد خواهیم پرداخت .

طبقه بندی کننده ها

الگوریتم های ژنتیکی ، به صورت بالقوّه ، به ما یک روش برای استنتاج و انتخاب قانون ها را ارایه می دهند و می گویند ، قانونی که دارای بیش ترین شایستگی می باشد را انتخاب نمایید . توسّط ۰،۱ و * قانون ها را به صورت یک رشته ی بیتی کد نمایید . بیت های وضعیّت ، همان بیت های عملکرد می باشند . مشکلاتی که در مورد وجود دارند عبارتند از اینکه : 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 ، اطّلاعات را که به صورت یک رشته ی بیتی کد شده اند دریافت می نمایند ؛ این اطّلاعات ، پیامی از محیط می باشد . این پیام طبقه بندی کننده ها (قانون ها) را با تطبیق شرط ها فعّال می نماید . 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 فعّال نمایند .

مثال – فرض کنید سیستم ما دارای طبقه بندی کننده های زیر می باشد :

1. 01## : 0000
2. 00#0 : 1100
3. 11## : 1000
4. ##00 : 0001

اگر پیام ۰۱۱۱ از محیط دریافت می شود . در ابتدا ، قانون یک اجرا می شود ، ۰۰۰۰ در لیست پیام قرار می گیرد . سپس ، قانون های دو و چهار اجرا می شود ، ۱۱۰۰ و ۰۰۰۱ در لیست پیام قرار می گیرند . بعد ، قانون سه اجرا می شود ، ۱۰۰۰ در لیست پیام قرار می گیرد ، که با قانون شماره ی ۴ مطابقت دارد که پیامش در لیست پیام ها قرار دارد . حالا چند پیام در لیست پیام قرار دارد – کدام به عمل کننده ها فرستاده می شود ؟

دسته سطل [۱] – الگوریتم دسته سطل به قانون ها برای پیشنهاد اجرا براساس عملکرد قبلی اجازه می دهد . وقتی که یک قانون برقرار می شود ، در یک " مزایده [۲] " شرکت می نماید . هر قانون براساس عملکرد قبلی ، دارای یک قوّت 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 فعّال کرده (اند) فرستاده می شود .

مثال : اگر به صورت اوّلیّه (t=0) داشته باشیم : 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

---

[۱] Bucket Brigade : زنجیره ای از افراد که با دست به دست کردن سطل هایی از آب سعی می کنند آتش را خاموش نمایند (Babylon / Merriam – Webster Collegiate ® Dictionary) .

[۲] auction

محیط S=0

در 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

در 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

در t = 3:

1. 01## : 0000 S=220 1000
2. 00#0 : 1100 S=218 0001
3. 11## : 1000 S=180
4. ##00 : 0001 S=162 B=16

محیط S=20

در t = 4 :

1. 01## : 0000 S=220 0001
2. 00#0 : 1100 S=208
3. 11## : 1000 S=196
4. ##00 : 0001 S=156

محیط S=20

و در t = 5 – درخواست به سیستم می آید و به آخرین طبقه بندی کننده ی فعّال نسبت داده می شود .

1) 01## : 0000 S=220
2) 00#0 : 1100 S=208
3) 11## : 1000 S=196
4) ##00 : 0001 S=206

محیط S=20

تولید قوانین جدید – دسته سطل ، 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 جمعیّت که با هم اشتراک ندارند استفاده می نماید ؛ همه ی جمعیّت در زمان t ، در زمان t+1 جایگزین می شود . به خوبی برای بهینه سازی عمل می کند ، ولی برای یادگیری ، زیاد مناسب نمی باشد . در عوض ، ما از نخبه سالاری [1] برای حفظ

[1] elitism

برخی از قانون ها استفاده می نماییم ، از روش انتخاب رولت [۱] برای حفظ قانون ها استفاده نمایید ، البتّه این روش ، کندتر استنتاج می کند .

برنامه نویسی ژنتیکی – فراتر از قوانین – روش های تکاملی برای تولید برنامه ها به کار گرفته شده اند . ایده ی این روش این است که یک برنامه می تواند به صورت یک S- عبارتی بیان شود ، مثلاً :

(+ 3 (* 4 x))

ما می توانیم این را به صورت یک درخت بکشیم . ما سپس تکامل را روی یک جمعیّت از برنامه ها انجام می دهیم . شایستگی ، کارایی یک برنامه برای یک عملکرد ارایه شده است . Crossover ، زیر درخت ها را با هم عوض می نماید . جهش ، عملگرها را عوض می نماید و برای تکامل استفاده می شوند ، مثال ها برنامه های فوتبال روبوت ، مدارهای کنترل آنالوگ ، فیلترها و مدارهای پیچیده می باشند . در این مورد چالش ها عبارتند از ، فضای پهناور (نامحدود) برنامه ها و کمبود ساختارهای سطح بالاتر برنامه .

---

[۱] roulette selection

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل بیست و سوّم

درخت های تصمیم گیری [۱]

[۱] Decision trees

در گذشته ، ما فرض کردیم که دانش اوّلیّه توسّط خبره ها به ما داده شده است و بر چگونگی استفاده از این دانش تمرکز کردیم . حال می خواهیم در این مورد صحبت کنیم که چگونه دانش را از راه مشاهده به دست آوریم و بر قانون های گزاره ای تمرکز می کنیم . مثلاً اگر هوا آفتابی و گرم باشد ، آن گاه تنیس بازی کنید و می نویسیم :

$$\text{sunny} \wedge \text{warm} \rightarrow \text{PlayTennis}$$

یا اگر هوا خنک باشد و بارانی باشد یا باد شدید بوزد ، آن گاه تنیس بازی نکنید و می نویسیم :

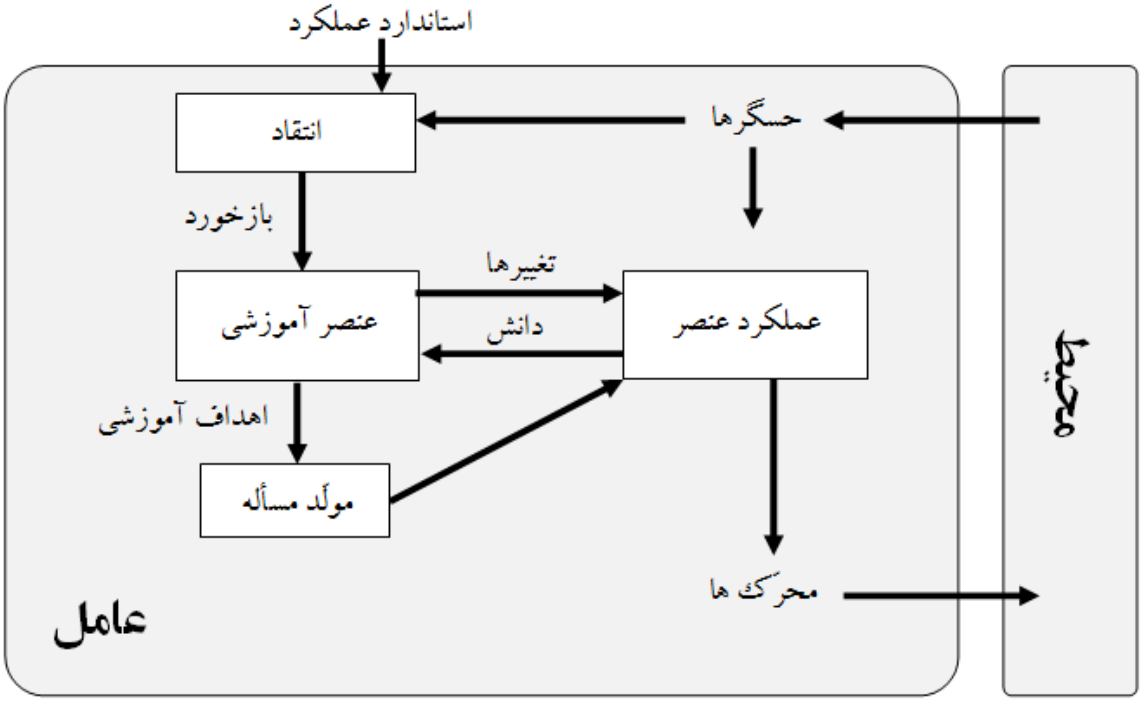
$$\text{cool} \wedge (\text{rain} \vee \text{strongWind}) \rightarrow \neg\ \text{PlayTennis}$$

یادگیری

برای یک عامل ، یادگیری به چه معنی می باشد ؟ عامل ، دانش جدید را دریافت می نماید ، دانش جدید را به کار می گیرد ، رفتارش را تغییر می دهد و در یک کار معیّن ، معیار کارایی خود را بهبود می بخشد .

عامل های آموزشی

به یاد بیاورید که ما قبلاً در مورد عامل های آموزشی صحبت کردیم :



یک عامل آموزشی دارای یک *عنصر کارایی* [۱] و یک *عنصر آموزشی* [۲] 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 و 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 واقعیّات ، بازنگری

---

[۱] performance element

[۲] learning element

یک مکاشفه و تغییر یک تابع جانشین [۱] باشد . یک عامل ، برای بازنگری کردن رفتارش ، به اطّلاعاتی که به عامل بگوید چگونه به خوبی کارش را انجام می دهد نیازمند می باشد ، این اطّلاعات ، بازخورد [۲] نام دارد .

انواع بازخورد

در اصل سه نوع عملکرد آموزشی وجود دارد که هر کدام بازخورد متفاوتی را دارد :

یادگیری نظارت شده [۳] (کنترل شده) که در این مورد ، یک منبع خارجی (که اغلب آموزش دهنده [۴] نام دارد) ، عامل را با *نمونه های برچسب زده شده* [۵] ارایه می نماید . عامل ، موارد / عملکردهای معینی را در طول طبقه بندی اشان می بیند .

یادگیری بدون نظارت [۶] (کنترل نشده) که در این مورد ، آموزش دهنده ای برای ارائه ی نمونه ها وجود ندارد و عامل ، معمولاً برای پیداکردن الگوهایی در داده ها تلاش می نماید .

یادگیری تقویتی [۷] *که* 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 و "بهترین" عملکرد را برای انجام نمی داند .

یادگیری نظارت شده (کنترل شده)

---

[۱] Successor function
[۲] feedback
[۳] Supervised learning
[۴] teacher
[۵] labled examples
[۶] unsupervised learning
[۷] reinforcement learning

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

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 کلّی 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 سؤالات برای طبقه بندی حیوانات چیست ؟ صورت چه کسی در این تصویر می باشد ؟ این پردازش یادگیری قانون های کلی از واقعیّات مشخص ، *استنتاج* [1] نام دارد .

تعریف یک مسأله ی یادگیری – ما می توانیم مسأله ی یادگیری 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* [2] 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* [3] می نامیم . هدف ، پیداکردن یک فرض سازگار که در نمونه های دیده نشده هم به خوبی کار کند . ما می توانیم یادگیری را به صورت جستجو در میان یک فضا از فرض ها تصوّر نماییم .

---

[1] induction

[2] hypothesis

[3] consistent hypothesis

بایاس استنتاجی – توجّه 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 [۱] ، معیّن ترین فرض ، کلّی ترین فرض و تابع خطی می باشند .

مشاهده ی داده ها – عامل ها ، ممکن است دارای ابزار مختلف مشاهده کننده ی نمونه های یک فرض باشند . یک الگوریتم یادگیری *دسته ای* [۲] ، یک مجموعه ی بزرگ داده ها که همه با هم ارایه می شوند و یک فرضیّه ی منفرد را انتخاب می کنند ، می باشد . یک الگوریتم یادگیری *افزایشی* ، نمونه های همه در یک زمان را دریافت می کند و دایماً فرض آن را بازنگری می نماید ؛ دسته معمولاً با دقّت تر می باشد ، اما افزایشی ممکن است با محیط عامل ، مناسب تر باشد . یک عامل یادگیری فعّال می تواند نمونه ها را انتخاب نماید . یک عامل یادگیری *غیرفعّال* [۳] ، دارای نمونه های ارایه شده با آن به وسیله ی یک منبع خارجی هستند می باشد ؛ یادگیری فعال توانمندتر می باشد ، اما ممکن است با محدودیّت های دامنه ، مناسب نباشد .

درخت های تصمیم گیری یادگیری – 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 .

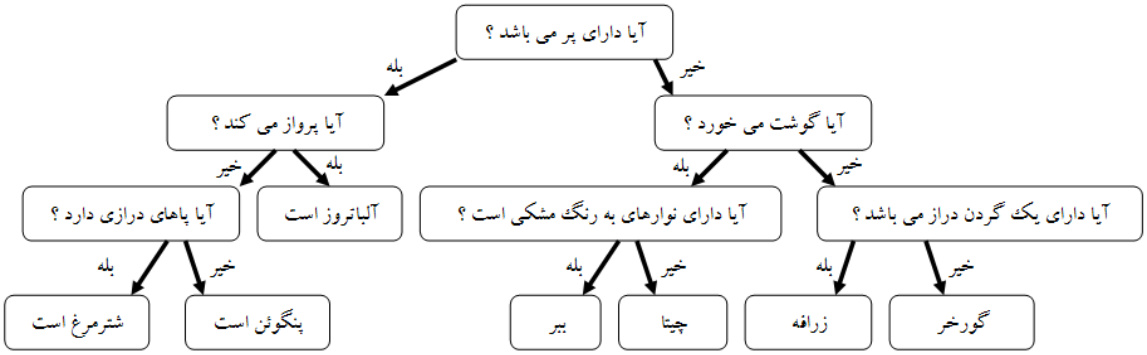
---

[۱] 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)

[۲] batch

[۳] passive

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸



اطّلاعات

در مورد دنیای جاروبرقی ، اتاق ها ممکن است تمیز یا کثیف باشند که برای بیان این مورد به یک بیت نیاز داریم . اگر یک اتاق دارای چهار وضعیّت باشد یا هشت وضعیّت چه طور ؟ اگر یک اتاق فقط دارای یک وضعیّت باشد چه طور ؟ به چند بیت برای بیان نیاز داریم ؟

تئوری اطّلاعات

به صورت رسمی تر ، بیایید بگوییم که n پاسخ ممکن $v_1, v_2, ..., v_n$ برای یک سؤال وجود دارد و هر جواب دارای احتمال رویداد $P(v_n)$ می باشد . *محتوای اطّلاعات* [1] جواب برای سؤال به این صورت می باشد :

$$I = \sum_{i=1}^{n} - P(v_i) \log_2 P(v_i)$$

برای یک سکّه ، این برابر است با :

$$-\frac{1}{2}\log_2 \frac{1}{2} + -\frac{1}{2}\log_2 \frac{1}{2} = 1$$

[1] information content

سؤالاتی با یک احتمال زیاد جواب ، دارای محتوای اطّلاعاتی کمی خواهند بود . (در صورتی که سکّه در یک بار ۹۹/۱۰۰ شیر بیاید ، I=0.08 خواهد بود .) ، محتوای اطّلاعاتی ، برخی اوقات بی نظمی (آنتروپی [۱]) نامیده می شود و اغلب در فشرده سازی و الگوریتم های انتقال داده ها استفاده می شود .

استفاده از تئوری اطّلاعات

برای درخت های تصمیم گیری ، ما می خواهیم بدانیم ارزشمندی هر مورد ممکن چگونه می باشد ، یا چه میزان از اطّلاعات را نتیجه می دهد . ما می خواهیم احتمال جواب های ممکن از یک مجموعه ی مورد آموزش را تخمین بزنیم . معمولاً ، یک مورد تکی برای به طور کامل مجزا نمودن نمونه های مثبت و منفی کافی نمی باشد . در عوض ، ما نیاز داریم در مورد این فکر کنیم که چه مقدار بهتری بعد از سؤال وجود خواهد داشت و این ، *سود اطّلاعات* [۲] نام دارد .

سود اطّلاعات

در صورتی که یک مجموعه ی مورد آزمایش دارای p نمونه ی مثبت و n نمونه ی منفی باشد ، بی نظمی یا آنتروپی 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 خصوصیّتی که برای مجزّا کردن نمونه های مثبت ومنفی ، نزدیک ترین می باشد را پیدا نماییم . ما با محاسبه ی باقی مانده شروع می کنیم – این اطّلاعاتی 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 ،

---

[۱] entropy

[۲] information gain

v زیر مجموعه ی جدید از داده ها را به نام $E_1, E_2, ..., E_v$ خواهد ساخت . باقی مانده ، مجموع اطّلاعات موجود در هر کدام از این زیرمجموعه ها می باشد .

$$\text{Re}\ mainder\ \ (A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} * I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

سود اطّلاعات سپس می تواند به صورت تفاوت میان اطّلاعات اصلی (قبل از آزمایش) و اطّلاعات جدید (بعد از آزمایش) بیان شود .

$$Gain(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{Re}\ mainder\ (A)$$

مکاشفه : ویژگی با بیش ترین سود اطّلاعات را انتخاب نمایید .

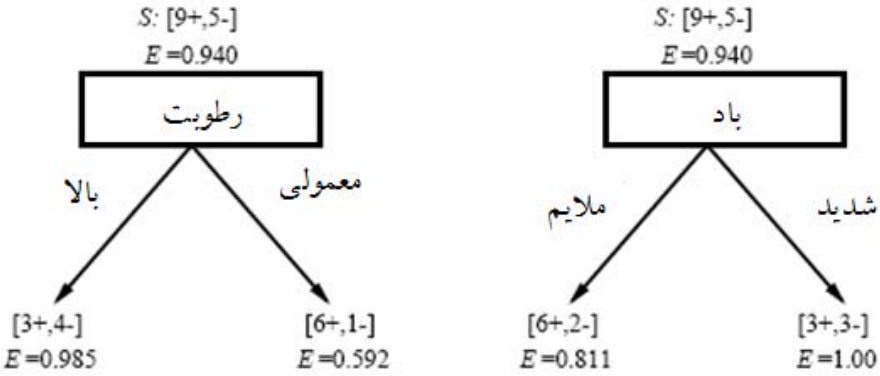
سؤال : این کدام نوع از جستجو می باشد ؟

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	ملایم	بله
D6	بارانی	خنک	معمولی	شدید	نه
D7	ابری	خنک	معمولی	شدید	بله
D8	آفتابی	متوسط	بالا	ملایم	نه

D9	آفتابی	خنک	معمولی	ملایم	بله
D10	بارانی	متوسط	معمولی	ملایم	بله
D11	آفتابی	متوسط	معمولی	شدید	بله
D12	ابری	متوسط	بالا	شدید	بله
D13	ابری	گرم	معمولی	ملایم	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 یا پارازیت [1]

[1] Noise

در صورتی که دو نمونه ی دارای ویژگی های یکسان ولی مقدارهای مختلف وجود داشته باشد ، یک درخت تصمیم گیری قادر به طبقه بندی آن ها به صورت مجزّا نخواهد بود و در این صورت می گوییم که این داده ها دارای اغتشاش می باشند .

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 . در این مورد برای پیداکردن مقداری که بالاترین سود اطّلاعات را نتیجه می دهد از جستجوی تپّه نوردی استفاده نمایید .

فصل بیست و چهارم

یادگیری Q ای [۱]

[۱] Q - learning

ما به محیط های تصادفی جذب شده ایم . ما می توانیم این مسأله را به صورت یک پروسه ی تصمیم گیری مارکوفی فرمول بندی نماییم ؛ مسأله دارای یک حالت اولیّه ی s_0، یک مجموعه ی گسسته از وضعیّت ها و عملکردها ، یک مدل انتقالی $T(s,a,s')$ که احتمال رسیدن وضعیّت s' از s در موقع انجام عمل a می باشد و یک تابع پاداش ($R(s)$) می باشد .

روش ها [۱]

یک روش ، راه حلّی برای یک پردازه ی تصمیم گیری مارکوفی می باشد و عملکرد بهینه را برای انجام در هر وضعیّت ، مشخّص می نماید . عملکرد بهینه ، عملکردی است که سودمندی مورد انتظار کاهش داده شده از این وضعیّت را بیشینه می نماید . بنابراین ، سودمندی یک وضعیّت ، پاداشی برای آن وضعیّت ، به اضافه ی پاداش کم شده ی مورد انتظار برای دنبال کردن روش بهینه از این وضعیّت می باشد . داریم :

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

[۱] policies

که عبارت فوق معادله ی بلمن [۱] نام دارد .

تکرار مقدار [۲]

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 : به سودمندی های غیرپایانی ، مقدارهای تصادفی نسبت دهید . برای هر وضعیّت ، عملکرد بهینه ی ارایه کننده ی این مقادیر را محاسبه نمایید . از این روش برای به روز رسانی سودمندی تخمین زده شده ی وضعیّت استفاده نمایید و این کار را تا زمان نزدیک شدن به وضعیّت مطلوب (همگرایی) ، ادامه دهید .

تکرار روش [۳]

در روش تکرار مقدار ؛ حتّی در زمانی که روش تغییر نمی کند ، همگرایی ، کند می باشد و ما واقعاً نمی توانیم نسبت به سودمندی یک وضعیّت ، بی تفاوت باشیم . تکرار روش ، روش ها را به صورت مستقیم به روز رسانی می نماید .

یادگیری یک روش

تا حالا ، ما فرض کردیم که دارای دانش زیادی هستیم . *در حالت به خصوص ، ما فرض کرده ایم که مدلی از جهان شناخته شده است ؛ این ، مدل انتقال وضعیّت می باشد* . اگر ما یک مدل نداشته باشیم چطور ؟ همه ی چیزی که ما می دانیم این است که مجموعه ای از وضعیّت ها و یک مجموعه از عملکردها وجود دارند . ما هنوز می خواهیم یک روش بهینه را یاد بگیریم .

---

[۱] Bellman equation

[۲] Value Iteration

[۳] Policy Iteration

یادگیری Q ای

آموزش یک روش به صورت مستقیم ، مشکل می باشد . مسأله این است که داده های ما به یک شکل نمی باشند : < وضعیّت ، عملکرد> . در عوض ، به شکل $s_1, s_2, s_3, ..., R$ می باشد . از آنجایی که ما تابع انتقال را نمی دانیم ، مشکل است که سودمندی یک وضعیّت را یاد بگیریم . در عوض ، ما یک تابع Q(s,a) را یاد خواهیم گرفت که " سودمندی " انجام عملکرد a را در وضعیّت s ، تخمین می زند . به طور صریح تر ، Q(s,a) مقدار a را در وضعیّت s را ارایه می کند و بعد از این به صورت بهینه عمل می کند .

$$Q(s,a) = R(s,a) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

روش بهینه این است که عملکرد با بالاترین مقدار Q را در هر وضعیّت ، به دست آوریم . اگر عامل بتواند Q(s,a) را یاد بگیرد ، می تواند عملکردهای بهینه را حتّی بدون دانستن تابع پاداش یا تابع انتقال داشته باشد .

یادگیری تابع Q

برای یادگیری Q ، ما باید بتوانیم مقدار یک عملکرد را در یک وضعیّت ، حتّی اگر پاداش ها در طول زمان پخش شده اند تخمین بزنیم . ما می توانیم این کار را به صورت تکراری انجام دهیم . توجّه 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 $\hat{Q}(s,a)$ ، مشخّص نماییم . ما یک جدول لیست کننده ی هر جفت وضعیّت – عملکرد و مقدار تخمین زده شده ی Q را نگهداری می نماییم . عامل ، وضعیّت s را درک می کند ، عملکرد a را انتخاب می کند و سپس درک می کند که پاداش برابر با r = R(s,a) می باشد ، که

آن را دریافت می کند و وضعیّت 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 . توجّه کنید که عامل نیاز به هیچ دانشی در مورد R یا تابع انتقال برای اجرای این کار ندارد . آموزش Q ای برای نزدیک شدن (همگرایی) ضمانت شده است به شرطی که پاداش ها محدود شده باشند ، عامل جفت وضعیّت – عملکرد های این شکلی را به روشی که معمولاً نامحدود می باشد انتخا ب می کند . 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 وضعیّت – عملکرد با نگرش نامحدود باشد .

اکتشاف

بنابراین چگونه آن را به وجود بیاوریم ؟ یادگیری Q ای دارای یک تفاوت مشخّص از دیگر الگوریتم هایی که تا کنون دیده ایم می باشد و آن این است که *عامل می تواند عملکردها و دیدی که آن ها به دست می آورند را انتخاب نماید و این یادگیری پویا* [1] *نام دارد* . عامل ، مایل به بیشینه کردن عملکرد می باشد ؛ این به این معنی است که کاری را که در حال حاضر به نظر می رسد بهترین باشد را انجام می دهد ، امّا اگر عامل هیچ گاه عملکردهای با " دید بد " نداشته باشد ، نمی تواند از خطاها بیرون بیاید . $\hat{Q}$ خیلی بادقّت نمی باشد ، بنابراین ، 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 [۱]

یک راه برای انجام این کار استفاده از اکتشاف بولتزمان است . ما یک عمل با احتمال زیر را انجام می دهیم :

$$P(a \mid s) = \frac{k^{\hat{Q}(s,a)}}{\sum_j k^{\hat{Q}(s,a_j)}}$$

که k پارامتر دما می باشد . این فرمول ، شبیه به فرمولی است که ما در روش شبیه سازی گرم و سرد کردن استفاده کردیم .

یادگیری مستحکم [۲]

یادگیری Q ای نمونه ای از آن چه که ما یادگیری مستحکم می نامیم می باشد . در یادگیری مستحکم ، 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 ، حتّی در زمانی که عمل غیر بهینه ای را انجام می دهد . تعمیم ها به پاداش تأخیر داده شده و پردازه های غیرقطعی تصمیم گیری مارکوفی رسیدگی می کنند .

خلاصه

یادگیری Q ای برخی اوقات به صورت یادگیری مستقل از مدل [۱] شناخته می شود . در این روش ، عامل فقط نیاز به انتخاب عملکردها و دریافت پاداش ها دارد . مشکلات این روش ، چگونه تعمیم بدهد ؟ ،

---

[۱] Boltzmann exploration

[۲] reinforcement learning

مقیاس پذیری [۲] و سرعت نزدیک شدن (همگرایی) می باشد . یادگیری Q ای برای یک الگوریتم مفید تجربی به سطح مطلوبی رسیده است .

---

[۱] model-free learning

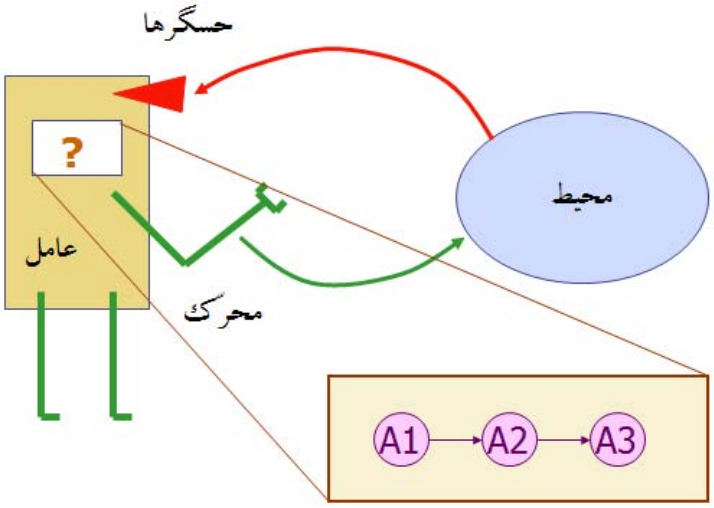
[۲] scalability

فصل بیست و پنجم

برنامه ریزی [۱]

[۱] planning

عامل برنامه ریزی



برنامه ریزی

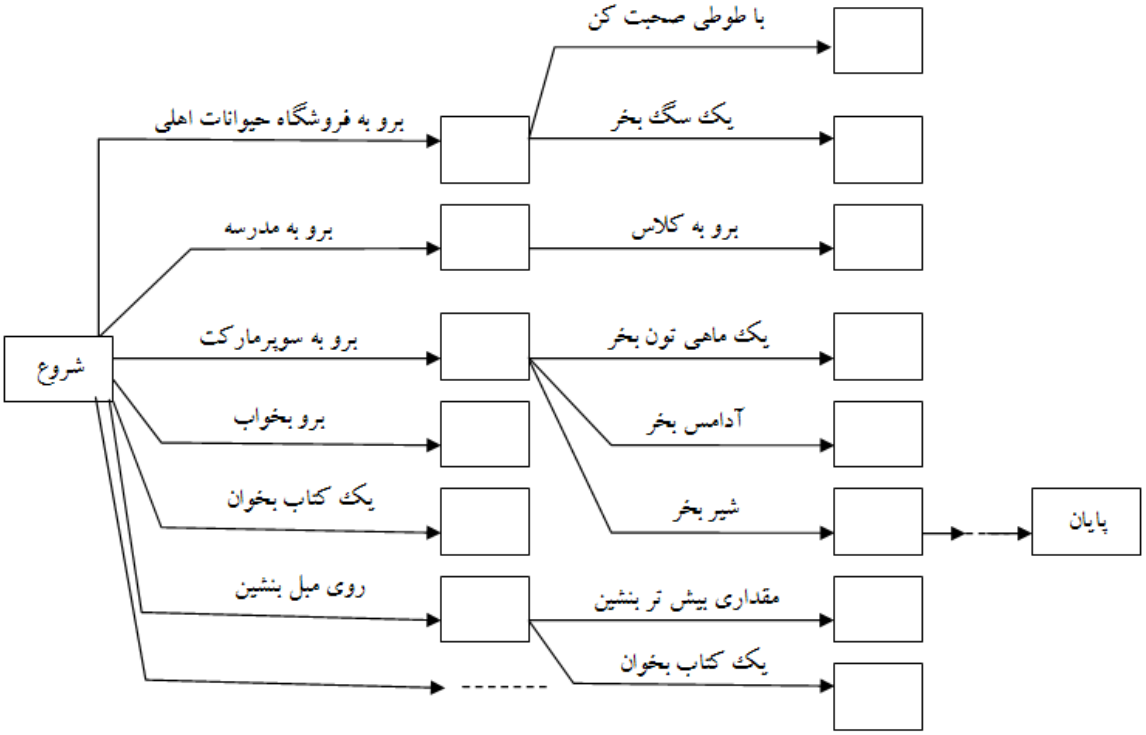
مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

برنامه ریزی ، تعبیه کردن یک رشته از عملکردها برای دسترسی به یک هدف می باشد . برنامه ریزی کلاسیک ، کاملاً قابل مشاهده ، قطعی ، محدود ، ثابت [۱] و گسسته است . زبان مشخصی برای ارایـه ی مسـایل برنامه ریزی وجود دارد .

جستجو و برنامه ریزی

الگوریتم های جستجوی استاندارد به نظر می رسد که به سختی شکست بخورند :



[۱] static

مشکلاتی که در روش جستجو وجود دارد عبارتند از : تعداد زیادی عملکردهای نامربوط وجود دارد ، پیدا کردن مکاشفه های خوب مشکل می باشد و از تجزیه ی مسأله نمی توان سودی برد . به صورت خودکار مکاشفه ها از ارایه مشتق می شوند ؛ مانند مسأله ی ارضای محدودیّت .

مقایسه ی برنامه ریزی با منطق

	جستجو	برنامه ریزی
وضعیّت ها	ساختمان داده های جاوا	عبارت های منطقی
عملکردها	کد جاوا	پیش شرط ها [1] / نتیجه ها
هدف	کد جاوا	عبارت منطقی
برنامه	رشته ای از S_0	محدودیّت های روی عملکردها

زبانی برای مسایل برنامه ریزی

حل کننده ی مسأله ی مؤسّسه ی تحقیق استنفورد [2] (اِستریپس) : جهان توسّط شرط های منطقی توصیف می شود ، وضعیّت به صورت اجتماع حرف های مثبت می باشد و می تواند به صورت

[1] preconditions

[2] STanford Research Institute Problem Solver (STRIPS)

گزاره ای ؛ به عنوان مثال ، $Happy \wedge Hungry$ برای نمایش وضعیّت عامل و یا منطق مرتبه ی اوّل و عبارت های مستقل از تابع باشد ؛ 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 ، وضعیّتی است که به صورت جزیی مشخّص شده است . اگر محدودیّت ها همه لفظ هایی از هدف باشند یک وضعیّت 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 توسّط موارد زیر مشخّص می شوند :

پیش شرط ها [1] **:** زمانی را مشخّص می کنند که عملکرد می تواند به کار گرفته شود .

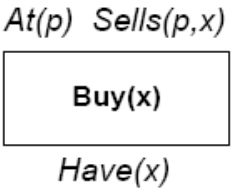
اثرات : وضعیّت توسّط عملکردها تغییر می یابد . در این مورد ، لیست اضافه ، گزاره هایی هستند که صحیح از آب در می آیند و لیست حذف ، گزاره هایی که غلط از آب در می آیند .

عملکردها شامل متغیّرها می باشند ، یک طرح عملکرد منفرد عملکردهای مختلف را نمایش می دهد (نمونه ای از متغیّرها) . توصیف عملکردها به صورت منظّم می باشد و زبان ، محدود شده است .

طرح عملکرد :

---

[1] preconditions



Buy(x)

پیش شرط ، $At(p) \wedge Sells(p,x)$ می باشد و اثر ، Have(x) است و باید به این نکته توجه کنیم که ، هیچ اطّلاعاتی در مورد چگونگی اجرای عملکرد وجود ندارد ! . چون که زبان محدود شده است پس الگوریتم کارا می باشد . عملکرد ، نام و لیست پارامتر را مشخّص می نماید . پیش شرط ، اجتماع لفظ های مثبت و اثر ، اجتماع لفظ ها (مثبت یا منفی) است . یک مجموعه ی کامل از عملگرهای استریپس می تواند به یک مجموعه از اصول جانشین وضعیّت [۱] ترجمه شود .

معانی [۲]

با داشتن یک وضعیّت (اجتماع لفظ ها) ، اگر انتساب یک متغیّر ، در ارتباط با لفظ های شامل شده ی وضعیّت باشد ، پیش شرط ارضا می شود ؛ به عنوان مثال ، وضعیّت $At(HW) \wedge Sell(HW, Drill)$ ، 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 از لیست حذف ، اضافه نمودن عناصر به لیست اضافه ، وضعیّت جدید :

$$At(HW) \wedge Sell(HW, Drill) \wedge Have(Drill)$$

مثال : خرید

---

[۱] successor – state axioms

[۲] semantics

عملکردها عبارتند از :

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 : $At(c,a) \wedge At(p,a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

اثر : $\neg At(c,a) \wedge In(c,p)$

Unload(c,p,a)

پیش شرط : $In(c,p) \wedge At(p,a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

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 عبارتند از Load ، Unload و نتیجه ی Fly . گزاره ها ، In(0,0) و At(0,0) می باشند . انواع گزاره ها هم Cargo(0) ، Plane(0) و Airport(0) هستند .

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

مسأله ی برنامه ریزی استریپس : یک رشته از عملکردهایی که به هدف منجر می شوند را پیدا نمایید . وضعیّت ها و هدف ها توسّط یک اجتماع از لفظ ها تعریف می شوند .

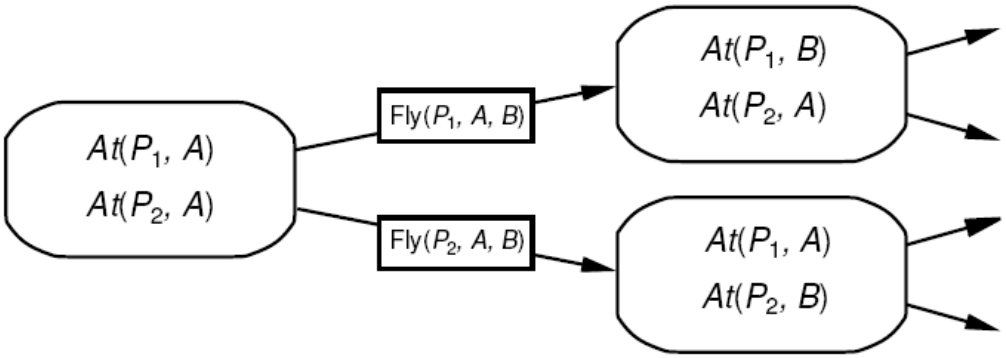
فضای حالت جستجو : جستجوی مستقیم از وضعیّت اولیّه تلاش می کند تا به هدف برسد . جستجوی معکوس از هدف تلاش و طرح ریزی می کند که به وضعیّت ابتدایی برسد .

فضای برنامه ی جستجو : برنامه ریزی با مرتبه ی جزیی [۱] ، فضای جزیی ساخته شده توسّط طرح ها را جستجو می کند .

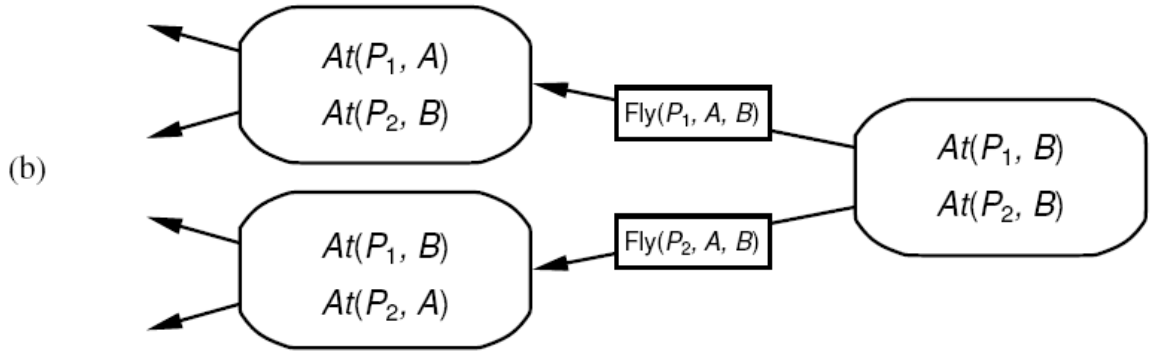
فضای حالت جستجو

مسأله ی برنامه ریزی ، مسأله ی جستجو را تعریف می کند . وضعیّت اوّلیّه ، وضعیّت شروع می باشد ، آزمون هدف ، بررسی می کند که آیا هدف برآورده (ارضا) می شود یا نه ، عملکردها ، عملگرها را تعریف می کنند ، هزینه مرحله ، معمولاً ۱ می باشد . در زیر ، شکل a ، جستجوی مستقیم و شکل b ، جستجوی معکوس می باشد :

(a)



[۱] Partial-Order Planning(POP)



جستجوی مستقیم – حلقه ی جستجوی اصلی : یک عملکرد را انتخا ب نمایید و پیش شرط را با وضعیّت یکی نمایید . در صورتی که پیش شرط برآورده شد ، عملکرد تولید کننده ی یک وضعیّت جدید را به کار ببرید . برّرسی کنید که آیا وضعیّت جدید ، هدف را برآورده می نماید یا نه . در صورتی که فضای حالت ، محدود باشد ، برای الگوریتم های جستجوی کامل ، کامل می باشد . این روش به دلیل عملکردهای نامربوط ، ناکارآمد می باشد و به مکاشفه های خوب ، نیازمند می باشد .

جستجوی معکوس – ایده ی این روش این است که عملکردهای مربوط را فقط با شروع از هدف انتخاب نمایید . عملکرد ، در زمانی که به یکی از متّحدانش دست می یابد (لیست اضافه) ، مربوط می باشد . در زمانی که هر لفظ مربوط را بی اثر [1] نمی کند (لیست حذف) ، سازگار می باشد .

یک عملکرد مربوط و سازگار را انتخاب نمایید و وضعیّت جدید را تولید نمایید : لیست اضافه را پاک نمایید و پیش شرط ها را اضافه نمایید .

این روش با یک وضعیّت برآورده شده توسّط وضعیّت ابتدایی خاتمه می یابد و روی وضعیّت های مربوط و سازگار منشعب می شود .

فضای حالت و فضای برنامه

[1] undo

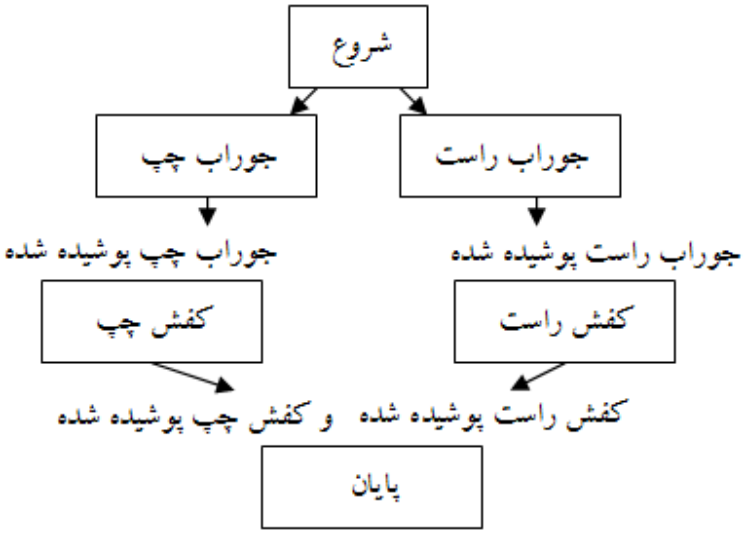
جستجوی مستقیم و معکوس ، رشته های خطی عملکردها را اکتشاف می نماید که این کار لازم نمی باشد ؛ به راه حل انتقال هوایی محموله توجّه نمایید :

$$Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$$
$$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)$$

تنها ترتیب لازم ، میان Load ، Fly و Unload می باشد که توسط اثرات سببی به هم متّصل شـده انـد ؛ بـه عنوان مثال ، $Load\ (C_1, P_1, SFO)$ به $In(C_1, P_1)$ استفاده شده توسط پیش شرط Unload دست می یابـد . 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 ≺)

مثال : طرح با مرتبه ی جزیی :



[1] partial-order plan

طرح های با مرتبه ی کلّی :

طرح های منظّم شده به صورت جزیی - مجموعه ای از مراحل با **مرحله ی شروع** [۱] دارای توصیف وضعیّت اوّلیّه به صورت اثرش می باشد ، **مرحله ی پایانی** [۲] دارای توصیف هدف به صورت پیش شرطش می باشد ، **اتصالات سببی** [۳] از نتیجه های یک مرحله به پیش شرط دیگری به وجود می آیند و **مرتب سازی زمانی** [۴] میان جفت های مراحل می باشد .

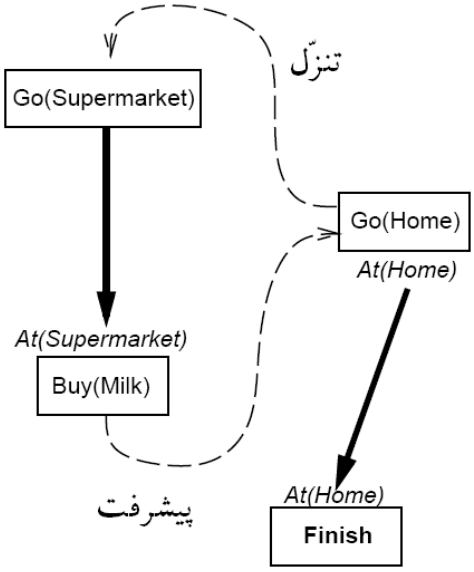
شرط باز [۵] ، پیش شرط یک گام که هنوز به صورت سببی متّصل نشده است می باشد . **طرح** در صورتی کامل است که هر پیش شرط در دسترس باشد ، **پیش شرط** در صورتی قابل دسترس می باشد که اثر یک

---

[۱] start step
[۲] finish step
[۳] causal links
[۴] temporal ordering
[۵] open condition

مرحله قبل تر باشد و هیچ مرحله ی میانی ، آن را بی اثر ننماید (تناقض نداشته باشند) . طرح ها نمی توانند دارای حلقه باشند .

ناسازگاری ها و پیشرفت [۱] / تنزل [۲] – یک عملکرد ناسازگار یک مرحله ی میانی بالقوّه است که شرط قابل دسترس توسط اتّصال سببی را خراب می نماید . به عنوان مثال ، Go(Home) ، At(Supermarket) را حذف می نماید :



تنزّل : قبل از Go(Supermarket) قرار دهید

پیشرفت : بعد از Buy(Milk) قرار دهید

---

[۱] promotion

[۲] demotion

فهرست برخی از منابع و مآخذ

بیش تر مطالب استفاده شده برای ترجمه ی این کتاب ، مطالب به زبان انگلیسی موجود در پایگاه های اینترنتی دانشگاه های کشور ایالات متحده ی آمریکا می باشد . در زیر ، فهرست برخی از پایگاه های اینترنتی ای که در ترجمه ی این کتاب از آن ها استفاده شده را مشاهده می نمایید :

۱- مطالب موجود در پایگاه اینترنتی دانشگاه سانفرانسیسکو ایالات متحده ی آمریکا ، به نشانی اینترنتی www.cs.usfca.edu

۲- مطالب موجود در پایگاه اینترنتی دانشکده ی برکلی دانشگاه کالیفرنیا ایالات متحده ی آمریکا ، با آدرس اینترنتی http://www.cs.berkeley.edu (مطالب استوارت راسل و پیتر نورویگ)

۳- مطالب موجود در پایگاه اینترنتی دانشکده ی علوم کامپیوتر دانشگاه دولتی تگزاس ایالات متحده ی آمریکا ، به آدرس اینترنتی www.cs.txstate.edu

۴- مطالب موجود در پایگاه اینترنتی http://www-formal.stanford.edu (مطالب جان مک کارتی ، استاد دانشگاه استنفورد)

۵- مطالب گردون اس. نواک جی. آر [۱] استاد دانشگاه تگزاس ایالات متحده ی آمریکا به نشانی اینترنتی www.cs.utexas.edu

۶- مطالب Milos Hauskrecht استاد دانشگاه شهر پیتسبورگ کشور ایالات متحده ی آمریکا به نشانی اینترنتی www.cs.pitt.edu

۷- مطالب موجود در پایگاه اینترنتی دانشگاه مریلند کشور ایالات متحده ی آمریکا با آدرس اینترنتی http://www.cs.umd.edu

۸- مطالب موجود در پایگاه اینترنتی مدرسه ی علوم کامپیوتر و مهندسی ، به نشانی اینترنتی www.cse.unsw.edu.au

۹- مطالب موجود در پایگاه اینترنتی جامعه ی هوش محاسباتی ، به نشانی اینترنتی

http://ebrains.la.asu.edu/~jennie/tutorial/index.htm

۱۰- مطالب موجود در پایگاه اینترنتی مهندسان سیاهپوست ، به آدرس اینترنتی www.nsbe.org

۱۱- مطالب موجود در پایگاه اینترنتی دانشگاه آزاد بوزن ایتالیا ، به نشانی اینترنتی http://www.inf.unibz.it

۱۲- مطالب موجود در پایگاه اینترنتی دانشگاه کوبلنز آلمان ، به نشانی اینترنتی http://www.uni-koblenz.de

---

۱- Gordon S. Novak Jr.

۱۳- مطالب موجود در پایگاه اینترنتی دانشکده ی علوم کامپیوتر دانشگاه کورک ایرلند ، با آدرس اینترنتی http://www.cs.ucc.ie

۱۴- مطالب موجود در پایگاه اینترنتی دانشکده ی علوم کامپیوتر و اطّلاعات نروژ ، به آدرس اینترنتی www.idi.ntnu.no

۱۵- مطالب موجود در پایگاه اینترنتی دانشگاه ملی تایوان ، به آدرس اینترنتی www.csie.ntu.edu.tw

۱۶- مطالب موجود در پایگاه اینترنتی دانشگاه لیورپول [۱] انگلستان به نشانی اینترنتی http://www.csc.liv.ac.uk

۱۷- مطالب موجود در پایگاه اینترنتی دانشگاه خصوصی سیبنسای [۲] کشور ترکیه

۱۸- مطالب موجود در پایگاه اینترنتی دانشگاه شمال غربی [۳] کشور ایالات متّحده ی آمریکا به نشانی اینترنتی http://www.eecs.northwestern.edu

۱۹- دانشکده ی علوم کامپیوتر دانشگاه بولوگنا کشور ایتالیا به نشانی اینترنتی http://www.cs.unibo.it

---

۱- Liverpool

۲- Sabancı University

۳- NORTHWESTERN UNIVERSITY