بسم الله الرحمن الرحیم

هوش مصنوعی

۱

مدرس

امیر جلیلی ایرانی
amirjalili.ir
amirjaliliirani@gmail.com

تحصیلات

لیسانس کامپیوتر – نرم افزار
فوق لیسانس کامپیوتر – هوش مصنوعی
از دانشگاه علوم و تحقیقات تهران

۲

هوش مصنوعی
Artificial Intelligence (A. I.)

- هوش مصنوعی:

الهام از موجودات زنده به ویژه انسان برای ایجاد قابلیت های استدلال، استنتاج، یادگیری و رفتار هوشمند.

- گرایش هوش مصنوعی:

عبارت است از هوشمند سازی کامپیوترها و سیستم های مبتنی بر کامپیوتر.

مرجع درسی

هوش مصنوعی(رهیافتی نوین)
Prentice hall

تالیف

استوارت راسل – پیتر نورویگ
مرجع بین المللی

ترجمه

١. رامین رهنمون
ناقوس

٢. راحتی و بهداد و تیموری
دانشگاه امام رضا

مرجع کنکوری

هوش مصنوعی

۱- انتشارات پوران پژوهش

تالیف: مهدیه شادی

۲- انتشارات گسترش علوم پایه

تالیف: رضا رافع – حمید رضا مقسمی

سر فصلها

- هوش مصنوعی چیست؟
- عامل و انواع آن.
- حل مساله از طریق جستجو.
- روشهای جستجوی آگاهانه.
- عامل مبتنی بر دانش و نمایش منطق.
- منطق مرتبه اول.
- برنامه ریزی.
- عدم قطعیت.
- معرفی کاربردها.

پیش نیاز ها

- طراحی الگوریتم ها
- ساختمان داده ها
- ساختمانهای گسسته

۷

دلیل اهمیت درس

- **کاربردی و باعث تقویت برنامه نویسی هوشمند.**
- **پایه و پیش نیاز بسیاری از دروس تخصصی هوش.**
- **مطرح در کنکور کارشناسی ارشد گرایش های مختلف.**

۸

سیستم ارزشیابی

☐ کوئیز	۱
☐ تمرینات	۱
☐ میان ترم	۴
☐ پایان ترم	۱۴
☐ حضور مرتب و منظم	مازاد
☐ پروژه	اختیاری

فصل اول

هوش مصنوعی چیست

تعریف هوش مصنوعی

الهام از موجودات زنده به ویژه انسان برای ایجاد قابلیت های استدلال، استنتاج، یادگیری و رفتار هوشمند.

عاقلانه فکر کردن	مانند انسان فکر کردن
عاقلانه عمل کردن	مانند انسان عمل کردن

هوش مصنوعی

تعاریف AI به چهار قسمت تقسیم شده‌اند:

- پردازش فکري و استدلالي(reasoning)
- پردازش رفتاري
- هوشمندي ايده‌آل (منطقي بودن)
- کارايی انساني

هوش مصنوعی

۱۳

تست تورینگ

- در سال ۱۹۵۰آلن تورینگ آزمونی به شرح زیر برای تشخیص هوشمند بودن ماشین بیان کرد:

آزمونی از کامپیوتر به عمل آید که در آن آزمون گیرنده نتواند دریابد که در آن طرف انسان قرار دارد یا ماشین.

- برای این منظور ماشین باید قابلیت های خاصی داشته باشد.

۱٤

فصل دوم

عاملهای هوشمند

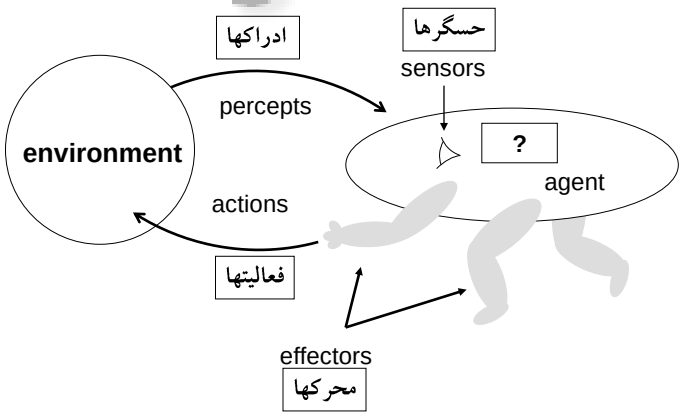
عامل یا agent

- هر چيزي كه قادر به درک محيط پيرامون از طريق حسگرها (sensor) و اثرگذاري بر روي محيط از طريق اثركننده‌ها (effector) باشد، عامل نام دارد.
- عامل نرم‌افزاري، رشته‌هاي بيتي را به عنوان درک محيط و عمل كدگذاري مي‌كند.
- عامل ابزاري براي تحليل سيستم‌ها است نه شخصيتي مطلق كه جهان را به دو بخش عامل و غيرعامل‌ها تقسيم كند.

عامل یا agent

- رفتار عامل بر دو پايه تجربه و دانش دروني بنا نهاده ميشود.
- عامل داناي كل معني خروجي واقعي اعمال خود را دانسته و بر پايه آن عمل مي‌كند اما دانش كل در واقعيت ناممكن است.
- اگر معين كنيم كه هر عامل هوشمند همواره بايد كاري را انجام دهد كه در عمل مناسب است هيچگاه نميتوان عاملي طراحي كرد كه اين مشخصات را مرتفع سازد.

نمایی از یک عامل



ساختار عامل

برنامه + معماري = عامل

- وظيفه هوش مصنوعي طراحي برنامه عامل است.
- برنامه عامل، تابع عامل را محقق ميسازد.
- تابع عامل، اقدامات عامل را در قبال ادراکات مشخص ميکند.
- رفتار عامل وابسته به دنباله ادراکي رويت شده تا حال است.

۱۹

عقلانیت

رفتار عقلانی یا خوب به موارد زیر بستگی دارد:

- معيار کارايي که ملاکهای موفقيت را تعريف ميکند.
- دانش قبلي عامل نسبت به محيط.
- فعاليتهايي که عامل ميتواند انجام دهد.
- دنباله ادراک عامل در اين زمان.

۲۰

عامل عقلانی

عاملی که کار درست را انجام میدهد.

عاملی که توسط ادراکات و دانش درونی خود اقدامی را انجام دهد که انتظار بیشترین کارایی را از آن داریم.

- عقلانیت با همه چیز دانی متفاوت است.
- یادگیری امکان طراحی عامل عقلانی را برای عملکرد در محیط های گوناگون مرتفع میسازد.

۲۱

خودمختاری

رفتار عامل مي‌تواند متكي بر دو پايه تجربه خود و دانش دروني بنا نهاده شود.

عاملی که هیچ دانش اولیه ای نداشته باشد و بر اساس تجربیات خود تصمیم گیری کند عامل خودمختار نام دارد.

عامل هوشمند واقعاً خود مختار، بايد قادر به عمل موفقيت‌آميز در دامنه وسيعي از محيطها باشد و البته بايد زمان كافي براي تطبيق نيز به آن داده شود.

۲۲

ویژگیهای محیط کار عامل

- کاملاً رویت پذیر در مقابل نیمه رویت پذیر.
- قطعی در مقابل اتفاقی.
- مرحله ای در مقابل ترتیبی.
- ایستا در مقابل پویا.
- گسسته در مقابل پیوسته.
- تک عاملی در مقابل چند عاملی.

برنامه های عامل

- **عاملهای واکنشی ساده**
- **عاملهای واکنشی مبتنی بر مدل**
- **عاملهای مبتنی برهدف**
- **عاملهای مبتنی بر سودمندی**
- نحوه تبدیل این عاملها به عاملهای یادگیرنده

عاملهای واکنشی ساده

- ساده ترین نوع عامل است.
- هوشمندی بسیار محدودی دارد.
- اقدامات را بر اساس ادراکات فعلی انجام میدهد و تاریخچه ادراکها را نادیده میگیرد.

۲۵

عاملهای واکنشی مبتنی بر مدل

- نگهداری سوابق بخشی از دنیا که عامل در حال حاضر قادر به رویت آن نیست بهترین راه برخورد عامل با نیمه رویت پذیری دنیا است.
- عامل، حالت داخلی یا تاریخچه ادراکات وابسته را نگهداری میکند.
- برای به روز در آوردن این حالت داخلی نیاز به:

1. اطلاعاتی در مورد چگونگی تغییر دنیا مستقل از عامل.
2. اطلاعاتی در مورد چگونگی تاثیر اقدامات عامل بر دنیا.

۲۶

عاملهای مبتنی بر هدف

- اطلاع از وضعيت كنوني محيط همواره براي تصميم‌گيري عامل نمي‌تواند كافي باشد.
- عامل علاوه بر توصيف وضعيت جاري، به نوعي نيازمند اطلاعات هدف (goal) مي‌باشد كه توضيح موقعيت مطلوب است.
- این عامل به رغم کمتر موثر بودن، انعطاف پذیرتر است.

۲۷

عاملهای مبتنی بر سودمندی

- اهداف به تنهايي براي توليد يک رفتار با كيفيت بالا كافي نيستند.
- اگر یک وضعیت در دنیا بر دیگری ارجحیت داشته باشد، سودمندی بیشتری برای عامل دارد.
- تابع سودمندی، حالت یا دنباله ای از حالتها را به یک عدد حقیقی نگاشت میکند که درجه رضایت را توصیف مِیکند.

۲۸

عاملهای یادگیرنده

- نظر تورینگ: ساخت ماشینهای هوشمند و سپس آموزش آنها.
- عامل های قبلی فاقد یادگیری هستند و دانش آنها در زمان طراحی تهیه میشود. آنها خودمختار نیستند و از تجربیات خود جهت تصمیم گیری و کسب دانش جدید استفاده نمیکنند.
- یادگیری به عامل اجازه عمل در محیطی ناشناخته میدهد تا ماهرتر گردد.
- عامل های یادگیرنده با استفاده از تجربیات خود به دانششان اضافه کرده و در صورت لزوم آنرا اصلاح می نمایند.

۲۹

عاملهای یادگیرنده

- عامل یادگیرنده شامل چهار بخش:

1. عنصر یادگیری.
2. عنصر کارایی.
3. مولد مساله.
4. منتقد.

۳۰

فصل سوم

حل مساله از طریق جستجو

عامل حل مسئله

- یک نوع عامل هدفگرا، عامل حل مسئله نامیده می‌شود.
- این عاملها با یافتن ترتیب عملیات، تصمیم به انجام آنها برای رسیدن به حالت‌هاي مطلوب دارند.

سه مرحله اساسي براى حل مسائل

- تدوين: ۱- هدف
  ۲- مساله
- جستجو
- اجرا

عامل بصورت تدوين، جستجو، اجرا طراحى ميشود.

۳۳

تدوين هدف

- اولين گام در حل مساله تدوين هدف بر اساس حالت فعلى و مقياس كارايى عامل ميباشد.
- اهداف با محدود كردن مقاصدى كه عامل براى رسيدن به آنها تلاش ميكند به سازماندهى رفتار عامل كمك ميكند.
- وظيفه عامل فهميدن اينست كه كدام دنباله از اقدامات آنرا به يک حالت هدف ميرساند.

۳٤

تدوین مساله

- فرایند تصمیم گیری در مورد اقدامات و حالات بر اساس یک هدف خاص است.
- چه فعاليتها و وضعيت هايي براى رسيدن به هدف موجود است.

۳۵

مرحله جستجو

- فرایند بررسی توالی های مختلف ممکن از اقدامات که به حالاتی با مقادیر معلوم منجر میشود و انتخاب بهترین آنها توسط عامل، جستجو نام دارد.
- رفتن از یک حالت ممکن به حالت ممکن دیگر برای رسیدن به حل مساله.(میتواند درخت یا گراف باشد)
- الگوريتم جستجو مسئلهاي را به عنوان ورودي دريافت نموده و راهحلي را به صورت دنباله عمليات بر ميگرداند.

۳۶

مرحله اجرا

- بمحض اینکه جستجو راه حلی پیدا کرد اقدامات پیشنهاد شده را اجرا میکند.
- پس از تدوین یک هدف و یک مساله برای حل، عامل یک روال جستجو را فراخوانی میکند. راه حل توالی اقدامات بعدی را برای عامل مشخص میکند و عامل اقدام به انجام و اجرای راه حل میکند.
- پس از اجرای راه حل عامل هدف جدیدی تدوین میکند.

۳۷

مسائل و راه حلهای خوش تعریف

- مسئله: در واقع مجموعه‌اي از اطلاعات است كه عامل از آنها براي تصميم‌گيري در مورد اينكه چه كاري انجام دهد، استفاده مي‌كند.
- یک مساله بصورت رسمی با چهار مولفه تعریف میشود:

1. حالت ابتدایی
2. اقدامات یا عملگرها یا حرکت ها
3. آزمون هدف
4. تابع هزینه مسیر

۳۸

حالت ابتدایی

- حالت نشان دهنده یک وضعیت از مساله است.
  وضعیت منحصر به فرد از محیط مساله.

- حالت شروع حالتی است که عامل از آن شروع بکار میکند.
  عامل خودش از بودن در آن آگاه است.

- حالت ابتدایی ممکن است بینهایت یا بیش از یکی باشد.

اقدامات یا عملگرها یا حرکت ها

- اقدامات، فعالیتها و عملیات ممکن که در دسترس عامل هستند.
- معمولاً از تابع پسین یا جانشین استفاده میشود.
- هر چیزی که باعث تبدیل حالات به یکدیگر شود.
- عملگر روی حالتی اعمال شده و آنرا به حالت دیگری تبدیل میکند.
- اگر عملگرها بینهایت باشند جستجو امکانپذیر نیست. پس حتماً باید عملگرها محدود باشند.

  تابع جانشین + حالت اولیه ← فضای حالت

فضای حالت

- مجموعه تمامی حالتهایی که از حالت اولیه میتوان به آنها رسید.
  مجموعه تمامی حالات ممکن(امکانپذیر و امکان ناپذیر) مساله.
- فضای حالت گرافی را تشکیل میدهد که گره های آن حالتها و کمانها اقدامات میباشند.
- یک مسیر در فضای حالت یک توالی از حالتهایی است که توسط دنباله اقدامات بهم متصل اند.
- فضای حالت با فضای جستجو متفاوت است.

٤١

آزمون هدف

- تعیین میکند که آیا یک حالت خاص، حالت هدف است یا خیر.
- هدف صریح: در مثال رومانی، رسیدن به بخارست.
  در این شکل حالت نهایی را داریم.
- هدف انتزاعی: در شطرنج، رسیدن به حالت کیش و مات.
  در این شکل به جای حالت نهایی الگویی برای شناسایی حالت نهایی داریم.

٤٢

تابع هزینه مسیر

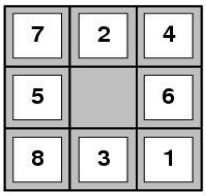
- برای هر مسیر یک هزینه عددی تخصیص میدهد.
- عامل حل مساله، تابع هزینه ای را انتخاب میکند که مقیاس کارایی خودش را منعکس کند.
- راه حل یک مسئله مسیری از حالت ابتدایی تا حالت هدف است.
- یک راه حل بهینه کمترین هزینه مسیر را در بین تمامی راه حلها دارد.

مسائل نمونه

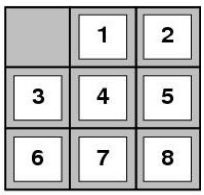
- رویکرد حل مساله در مجموعه وسیعی از محیط های کار استفاده میشود.
- مسائل اسباب بازی:
  هدف، نمایش یا تمرین روشهای مختلف حل مساله است.
  میتوان از آنها برای مقایسه کارایی الگوریتم ها استفاده کرد.
- مسائل دنیای واقعی:
  مسائلی که عموماً راه حلهای آن برای مردم واقعاً اهمیت دارد.

مساله اسباب بازی: پازل

- حالتها:
- حالت اولیه:
- تابع جانشین:
- آزمون هدف:
- هزینه مسیر:



Start State



Goal State

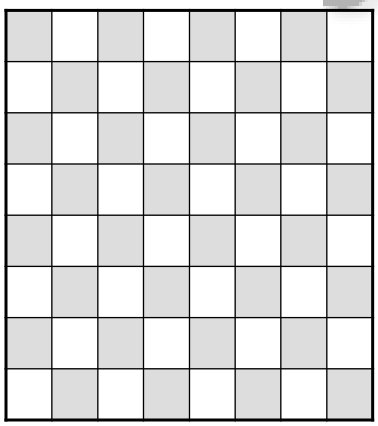
٤٥

مساله اسباب بازی: پازل

- پازل ۸ تایی 9!/2=181440 حالت دارد.
- پازل ۱۵ تایی حدود 1.3 تریلیون حالت دارد.
  حل آن با بهترین الگوریتم ها در حدود چند میلی ثانیه.
- پازل ۲۴ تایی حدود ۱۰۲۵تریلیون حالت دارد.
  حل آن با بهترین الگوریتم ها هنوز هم مشکل است.

٤٦

مساله اسباب بازی: مساله n وزیر

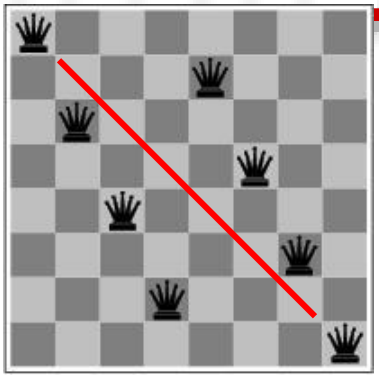


تدوین اول: افزایشی

- حالتها:
- حالت اولیه:
- تابع جانشین:
- آزمون هدف:
- هزینه مسیر:؟

۴۷

مساله اسباب بازی: مساله n وزیر



تدوین دوم: حالت کامل

- حالتها:
- حالت اولیه:
- تابع جانشین:
- آزمون هدف:
- هزینه مسیر:؟

۴۸

مساله اسباب بازی: مساله n وزیر

- تدوین اول:

در این روش باید 64*....*57=3*10^14 دنباله ممکن بررسی میشود.(برای n=100، 10^400 حالت)

- تدوین دوم:

این روش فضای حالت را از 3*10^14 به 2057 کاهش میدهد.(برای n=100، 10^52 حالت)

مساله دنیای واقعی

- مسیریابی:

الگوريتم‌هاي مسير يابي كاربردهاي زيادي دارند، مانند مسيريابي در شبكه‌هاي كامپيوتري، سيستم‌هاي خودكار مسافرتي و سيستم‌هاي برنامه‌نويسي مسافرتي هوايي.

- مساله فروشنده دوره گرد:

مسئله فروشنده دوره گرد مسئله مشهوري است كه در آن هر شهر حداقل يكبار بايد ملاقات شود. هدف يافتن كوتاهترين مسير است.

مساله و جستجو

برای مساله و جستجو چهار حالت امکانپذیر است:

1. مساله گراف ، جستجو گراف.
2. مساله گراف ، جستجو درخت.
   اگر گراف مساله دور داشته باشد و جستجوی درختی(در عمق نه در پهنا) انجام گیرد ممکن است جستجو در دور بینهایت بیافتد.
3. مساله درخت ، جستجو درخت.
4. مساله درخت ، جستجو گراف.

فقط زمان و فضا را بخاطر تست Ex از دست میدهیم.

۵۱

انواع جستجو

درختی
گرافی

بدنبال یک جواب
بدنبال همه جوابها
بدنبال بهترین جواب

در عمق
در پهنا

رو به جلو
روبه عقب
دو طرفه

بدون هزینه
با هزینه

با فرض وجود جواب
با فرض امکان عدم وجود جواب

۵۲

الگوریتم کلی جستجوی فضای حالت

1. Ex=[]; Un=[s];
2. n=head (Un);
3. Un= Un – [n];
4. if (n=G) then exit(“success”);
5. if (n∉Ex) then
   5.1. Ex = Ex + [n];
   5.2. expand n using operators and produce its children X1,…,Xk.
   5.3. add X1,…,Xk to the end of Un { Un = Un + [X1,…,Xk] }
6. go to 2.

Head تابعی است که یک عنصر برمیگرداند.

ویژگیهای الگوریتم کلی جستجوی فضای حالت

- الگوریتم جستجوی گرافی است.(برای مسایل گرافی)
- در این الگوریتم جستجو فرض بر وجود جواب است.
- در این الگوریتم جستجو بدنبال یک جواب است.
- این الگوریتم جستجوی در پهنا است.
- الگوریتم جستجوی رو به جلو است.
- الگوریتم جستجوی بدون هزینه است.

الگوریتم جستجوی درختی فضای حالت

1. Un=[s];
2. n=head (Un);
3. Un= Un – [n];
4. if (n=G) then exit(“success”);
5. expand n using operators and produce its children X1,…,Xk.
   5.1. add X1,…,Xk to the end of Un
   { Un = Un + [X1,…,Xk] }
6. go to 2.

۵۵

الگوریتم جستجو با فرض امکان عدم وجود جواب

1. Ex=[]; Un=[s];
2. if (Un=[]) then exit(“failure”);
   else n=head (Un);
3. Un= Un – [n];
4. if (n=G) then exit(“success”);
5. if (n∉Ex) then
   5.1. Ex = Ex + [n];
   5.2. expand n using operators and produce its children X1,…,Xk.
   5.3. add X1,…,Xk to the end of Un
   { Un = Un + [X1,…,Xk] }
6. go to 2.

۵۶

الگوریتم جستجوی فضای حالت برای همه جوابها

1. Ex=[]; Un=[s]; R=[];
2. if (Un=[]) then return(R)
   else n=head (Un);
3. Un= Un – [n];
4. if (n=G) then R =R + [n];
5. if (n∉Ex) then
   5.1. Ex = Ex + [n];
   5.2. expand n using operators and produce its children X1,…,Xk.
   5.3. add X1,…,Xk to the end of Un
   { Un = Un + [X1,…,Xk] }
6. go to 2.

۵۷

الگوریتم جستجوی در عمق فضای حالت

1. Ex=[]; Un=[s];
2. n=head (Un);
3. Un= Un – [n];
4. if (n=G) then exit(“success”);
5. if (n∉Ex) then
   5.1. Ex = Ex + [n];
   5.2. expand n using operators and produce its children X1,…,Xk.
   5.3. add X1,…,Xk to the front of Un
   { Un = [X1,…,Xk] + Un }
6. go to 2.

۵۸

الگوریتم جستجوی رو به عقب در فضای حالت

1. Ex=[]; Un=[G];
2. n=head (Un);
3. Un= Un – [n];
4. if (n=S) then exit("success");
5. if (n∉Ex) then
   5.1. Ex = Ex + [n];
   5.2. expand n using inverse of operators and produce its children X1,…,Xk.
   5.3. add X1,…,Xk to the end of Un
   { Un = Un + [X1,…,Xk] }
6. go to 2.

الگوریتم جستجوی دوطرفه

1. Ex1=[]; Un1=[s]; Ex2=[]; Un2=[G];
2. select direction using selector.
3. switch selector
   case forward :
   الگوریتم رو به جلو با Un1 و Ex1
   case backward :
   الگوریتم رو به عقب با Un2 و Ex2
4. if (Ex1 ∩ Ex2=[]) then goto 2.

- عامل مهم در انتخاب مسیر جستجو یا selector ، b میباشد.
  1. یک مرحله رو به جلو و یک مرحله رو به عقب.
  2. طرفی که (b) Un کوچکتری دارد.

الگوریتم جستجوی با هزینه در فضای حالت

ساختار Un تبدیل به زوج مرتب گره و هزینه میشود

g(n) هزینه رسیدن از s به n

1. Ex=[]; Un=[s];
2. Select n from Un with minimum g(n);
3. Un= Un – [n];
4. if (n=G) then exit("success");
5. if (n∉Ex) then
   5.1. Ex = Ex + [n];
   5.2. expand n using operators and produce its children X1,…,Xk.
   5.3. add X1,…,Xk to the end of Un { Un = Un + [X1,…,Xk] }
   5.4. g(Xi)=g(n) + C(n,Xi);
6. go to 2.

C(x,y) هزینه x به y
(x پدر y است)
باید از قبل موجود باشد

الگوریتم جستجو دنبال یک جواب از مجموعه معلوم

1. Ex=[]; Un=[s]; G=[goals];
2. n=head (Un);
3. Un= Un – [n];
4. if (n € G) then exit("success");
5. if (n ∉ Ex) then
   5.1. Ex = Ex + [n];
   5.2. expand n using operators and produce its children X1,…,Xk.
   5.3. add X1,…,Xk to the end of Un { Un = Un + [X1,…,Xk] }
6. go to 2.

الگوریتم جستجو دنبال جوابی از مجموعه نامعلوم

1. Ex=[]; Un=[s]; G=[conditions];
2. n=head (Un);
3. Un= Un – [n];
4. if (evaluate(n,G)) then exit(“success”);
5. if (n ∉ Ex) then
   5.1. Ex = Ex + [n];
   5.2. expand n using operators and produce its children X1,…,Xk.
   5.3. add X1,…,Xk to the end of Un { Un = Un + [X1,…,Xk] }
6. go to 2.

Evaluate(n,G) تابع ارزیابی هدف بودن n

اندازه گیری کارایی حل مساله

- کامل بودن: آیا الگوریتم تضمین میکند که در صورت وجود راه حل، آن را بیابد؟
- بهینگي: آیا این راه حل، بهینه ترین جواب را ارائه میکند.
- پیچیدگي زمانی: چقدر طول میکشد تا راه حل را پیدا کند؟
  تعداد گره های تولید شده در حین جستجو
- پیچیدگی فضا: برای جستجو چقدر حافظه نیاز دارد؟
  حداکثر تعداد گره های ذخیره شده در حافظه

کارایی در هوش مصنوعی

- مقیاس معمولاً اندازه گراف فضای حالت میباشد.
- گراف با استفاده از حالت اولیه و تابع پسین بطور صریح نشان داده میشود و اغلب نامتناهی است.
- پیچیدگی با چهار کمیت زیر نشان داده میشود:

1. b (فاکتور انشعاب):حداکثر تعداد پسینهای یک گره.
2. d(عمق): عمق کم عمقترین جواب.
3. m(طولانی ترین مسیر) : عمق درخت.
4. l(محدودیت): میزان محدودیت عمق.

راهبردهای جستجوی ناآگاهانه

- ناآگاهی این است که الگوریتم هیچ اطلاعات اضافی غیر از اطلاعاتی که در مسئله آمده درباره حالت دراختیار ندارد.
- این راهبردها فقط میتواند جانشینها یا پسینهایي را تولید و حالت هدف را از غیر هدف تشخیص دهند.
- راهبردهایي که تشخیص میدهد یک حالت غیر هدف نسبت به گره غیر هدف دیگر، امید بخش تر است، جست و جوی آگاهانه یا جست و جوی اکتشافي(هیورستیک)نامیده میشود.

راهبردهای جستجوی ناآگاهانه

- جستجوي اول سطح
- جستجوي با هزينه يكنواخت
- جستجوي اول عمق
- جستجوي عمق محدود
- جستجوي عميق‌شونده تكراري
- جستجوي دوطرفه

٦٧

راهبردهای جستجوی ناآگاهانه

- تمام راهبردهای جستجو بر اساس ترتیب گسترش گره ها، از یکدیگر متمایز میشوند.
- بطور کلی مسائل جستجو با پیچیدگی نمایی،جز برای نمونه های کوچک، توسط روشهای ناآگاهانه قابل حل نیستند .

٦٨

جستجوي اول سطح(BFS)

- در اين استراتژي ابتدا گره ريشه و سپس تمام فرزندان گره ريشه گسترش داده مي‌شوند و ... .
- به عبارت كلي‌تر، تمام گره‌هاي عمق d، قبل از گره‌هاي عمق d+1 گسترش داده مي‌شوند.
- به بیان ساده تر از بالا به پایین و از چپ به راست گره ها را گسترس میدهیم.

٦٩

جستجوي اول سطح

- این روش کم عمق ترین جواب را پیدا میکند .
- کم عمق ترین جواب لزوماً جواب بهینه نیست.

٧٠

جستجوي اول سطح : مثال

جستجوي اول سطح : كارايی

- کامل است.

(به شرط محدود بودن b)

- لزوماً بهینه نیست.

(اگر هزینه مسیر تابعی غیر کاهشی از عمق درخت باشد بهینه است)

- پیچیدگي زمانی $b+b^2+\ldots+b^d+(b^{d+1}-b)=O(b^{d+1})$

(تعداد گره های تولید شده در حین جستجو)

- پیچیدگی فضا $O(b^{d+1})$

(هر گره تولید شده باید در حافظه بماند)

جستجوي اول سطح : كارايى

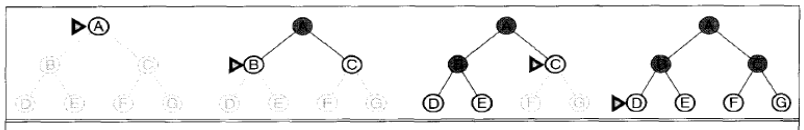


Figure 3.10 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabytes
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Figure 3.11 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 10,000 nodes/second; 1000 bytes/node.

٧٣

جستجوي هزينه يكنواخت(UCS)

- این روش بجای گسترش کم عمق ترین گره، گره n که کمترین هزینه مسیر دارد را گسترش میدهد.
- این روش جستجو به تعداد مراحل یک مسیر اهمیت نمیدهد بلکه فقط هزینه کل آنها را در نظر میگیرد.
- این روش یک جستجوی با هزینه است.

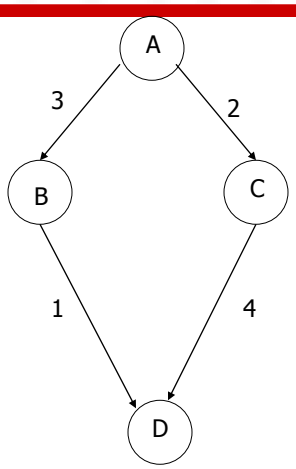
٧٤

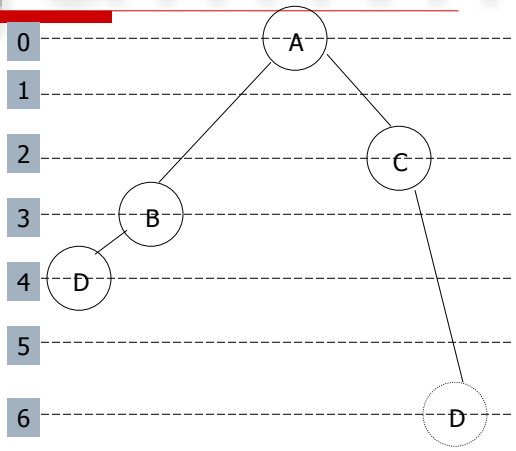
جستجوي هزينه يكنواخت و اول سطح

- اگر تمامی هزینه های مراحل یکسان باشد این جستجو تبدیل به جستجوی اول سطح میشود.(جستجوی سطحی حالت خاصی از UCS)
- در UCS از صف اولویت دار استفاده میشود و در اول سطح از صف معمولی.
- اگر بجای عمق درخت از سطوح هزینه استفاده و جستجوی سطحی انجام دهیم به UCS میرسیم.

۷۵

مثال: سطوح هزینه





۷۶

جستجوي هزينه يكنواخت: پارامترها

g(n)= هزینه کم هزینه ترین مسیر از s به n

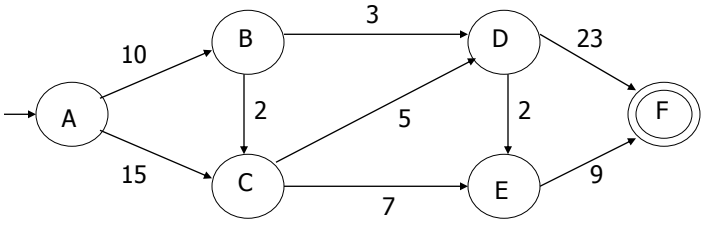
C(x,y)= هزینه رسیدن از x به y (به شرط x پدر y)

g(s)=0

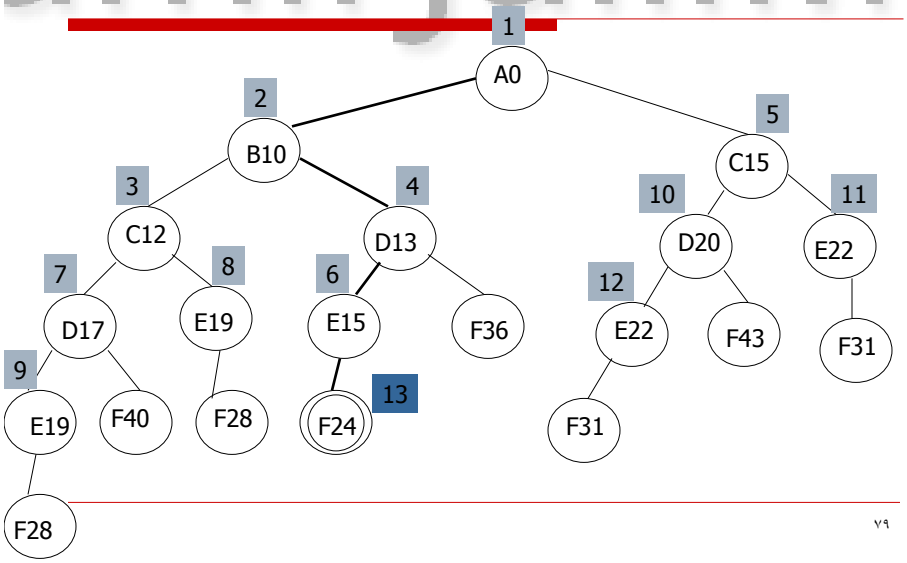
g(y)=g(x)+ C(x,y)

*این الگوریتم حریصانه نیست و بجای C به g توجه دارد.

جستجوي هزينه يكنواخت: مثال



جستجوي هزينه يكنواخت: مثال



جستجوي هزينه يكنواخت: كارايي

- كامل است.

(به شرطيكه هزينه هر مرحله بزرگتر يا مساوی يک مقدار ثابت و مثبت ε باشد، با هزينه صفر ممكن است در حلقه بينهايت گير كند)

- بهينه است.

(با همان شرط بالا يعنی C>0)

- پيچيدگي زماني $O(b^{[C^*/\varepsilon]})$
- پيچيدگي فضا $O(b^{[C^*/\varepsilon]})$

C^* هزينه راه حل بهينه

ε حداقل هزينه هر اقدام

- اگر تمامی مراحل هزينه يكسان داشته باشند

$$b^d = O(b^{[C^*/\varepsilon]})$$

اجتناب از حالات تکراري

- در برخی مسائل، حالات تکراري غیرقابل اجتناب هستند.
- وجود حالتهای تکراری در یک مسئله قابل حل، میتواند آن را به مسئله غیر قابل حل تبدیل کند.
- اگر الگوریتمی بتواند تمام حالتهایی را که از آن عبور کرده بخاطر بسپارد میتوان گفت گراف فضای حالت را مستقیماً کاوش کرده است.
- چنین روشی، جستجوی گرافی است.

۸۱

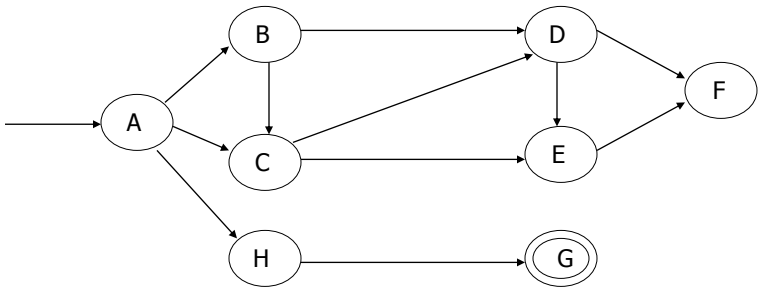
جستجوی گرافی

- در جستجوی گرافی زمان بسط گره تکراری صرفه جویی میشود اما در مقابل زمانی را برای بررسی گره های بسط داده شده (Ex) تلف میکند.
- اگر مساله حالات تکراری زیادی داشته باشد، جستجوی گرافی بهتر خواهد بود.
- بهینگی در جستجوی گرافی مساله بغرنجی است. زیرا هنگام رویت گره تکراری دو مسیر برای یک حالت پیدا شده وباید تصمیم بگیریم که کدام گره یا مسیر را رها کنیم.

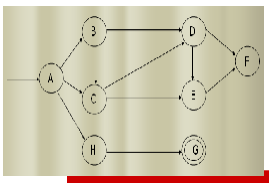
*UCS و BFS با هزینه ثابت، راهبرد بهینه جستجوی گرافی هستند.

۸۲

مثال: جستجوی سطحی (درختی و گرافی)



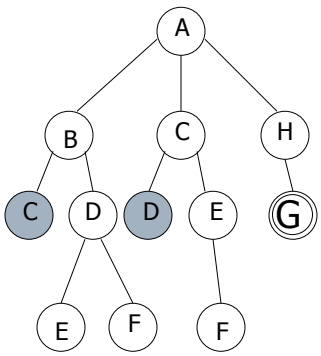
۸۳



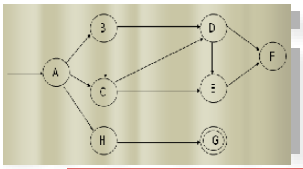
جستجوی سطحی گرافی

Un=[A]
Un=[B,C,H]
Un=[C,H,C,D]
Un=[H,C,D,D,E]
Un=[C,D,D,E,G]
Un=[D,D,E,G]
Un=[D,E,G,E,F]
Un=[E,G,E,F]
Un=[G,E,F,F]
G

Ex=[]
Ex =[A]
Ex =[A,B]
Ex =[A,B,C]
Ex =[A,B,C,H]
Ex =[A,B,C,H]
Ex =[A,B,C,H,D]
Ex =[A,B,C,H,D]
Ex =[A,B,C,H,D,E]

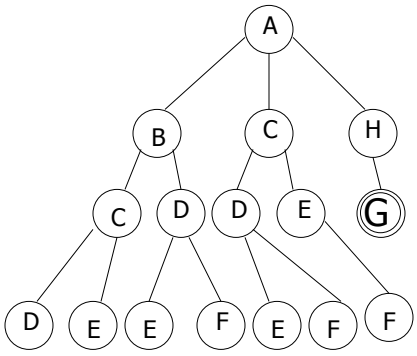


۸۴



جستجوی سطحی درختی

Un=[A]
Un=[B,C,H]
Un=[C,H,C,D]
Un=[H,C,D,D,E]
Un=[C,D,D,E,G]
Un=[D,D,E,G,D,E]
Un=[D,E,G,D,E,E,F]
Un=[E,G,D,E,E,F,E,F]
Un=[G,D,E,E,F,E,F,F]
G



۸۵

جستجوي اول عمق (DFS)

- در این روش، جستجو به عمیق ترین سطح درخت جستجو هدایت میشود.
- جستجو به عمیق ترین گره بعدی بر میگردد که هنوز فرزند پیمایش شده ندارد.
- جستجوی عمقی چپ ترین جواب را پیدا میکند.

۸۶

جستجوي اول عمق: مثال۱

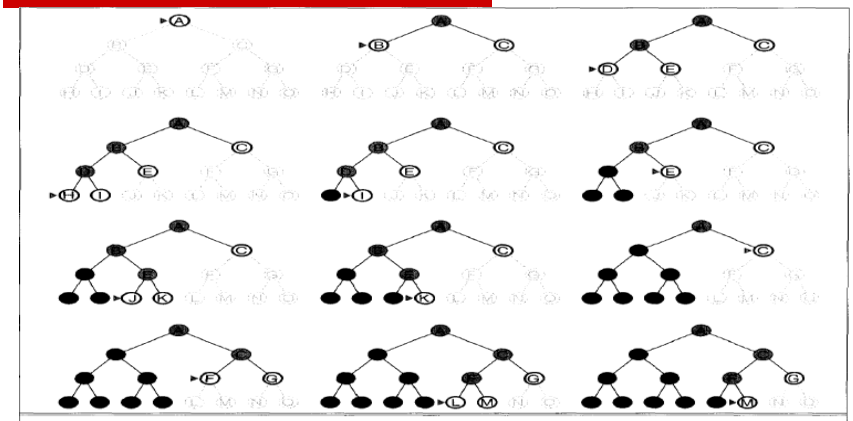


Figure 3.12 Depth-first search on a binary tree. Nodes that have been expanded and have no descendants in the fringe can be removed from memory; these are shown in black. Nodes at depth 3 are assumed to have no successors and M is the only goal node.

۸۷

جستجوي اول عمق: مثال۲

A

۸۸

جستجوي اول عمق: كارايی

- کامل نیست.

(اگر زیر درخت چپ دارای عمقی نامحدود و بدون راه حل باشد در حلقه بینهایت گیر میکند)

- بهینه نیست.

(چپ ترین جواب را پیدا میکند)

- پیچیدگي زمانی $O(b^m)$ (m>>d)

(در بدترین حالت تمامی گره ها جستجو میشود)

- پیچیدگی فضا O(bm)

(مسیری از ریشه تا برگ همراه با گره های هم نیا)

* برای d=12 اول سطح 10PB و اول عمق 118KB نیاز دارد.

۸۹

جستجوي اول عمق و جستجوی عقبگرد

- نوع خاصی از جستجوی اول عمق که حافظه کمتری نیاز دارد.
- برای اینکار هر گره باید برادر کوچکتر و پدر خود را بشناسد (در اختیار داشته باشد).
- در این روش هر گره گسترش داده نشده کنترل را به فرزند اول خود می سپارد و همین کار تکرار میشود تا گرهی فاقد فرزند باشد یا کنترل از فرزند برگشته باشد آنگاه کنترل را به برادر کوچکتر خود میدهد اگر برادر کوچکتری نداشته باشد کنترل را به پدر می سپارد.

۹۰

جستجوی عقبگرد: مزایا

- دیگر نیازی به Un نیست.
- بجای همه فرزندان درهر لحظه فقط یک فرزند تولید میشود.
- حافظه O(m) نیاز دارد بجای O(bm).

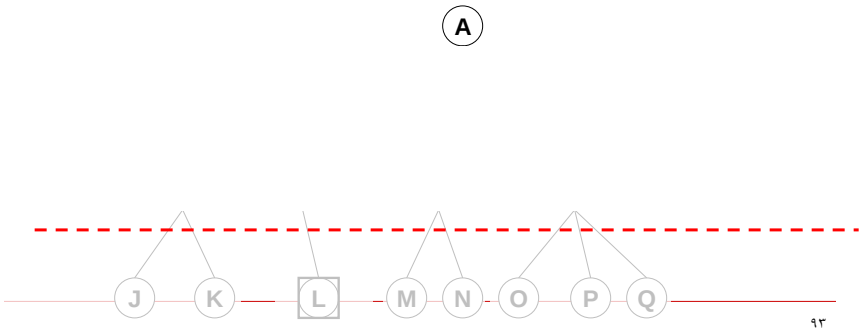
*فقط در عمق معنا پیدا میکند.

جستجوی عمق محدود (DLS)

- مشکل درختهای نامحدود میتواند به وسیله جستجوی عمقي با عمق محدود l بهبود یابد.
- با گره های عمق l مانند برگ برخورد میشود.
- جستجوی اول عمق حالت خاصی از جستجوی عمق محدود است که در آن l=∞ .

* بهترین مقدار l=d میباشد!!!

جستجوی عمق محدود:مثال



۹۳

جستجوی عمق محدود: کارایی

- کامل نیست. (اگر l<d)
- بهینه نیست. (اگر l>d)
- پیچیدگی زمانی: $O(b^l)$
- پیچیدگی فضا: $O(bl)$

۹٤

جستجوی عمیق شونده تکراری (IDS)

- این روش اغلب با جستجوی اول عمق استفاده میشود و بهترین محدودیت عمق را پیدا میکند.
- اینکار با افزایش تدریجی عمق تا پیدا شدن یک هدف صورت میگیرد.
- برای حل مشکل روش قبلی یعنی تعیین l بهینه پیشنهاد شده است.

۹۵

جستجوی عمیق شونده تکراری : مزایا

- ترکیبي از مزاياي جستجوي سطحي و عمقي را دارد:
1. در هر تکرار جستجوی عمقی انجام میدهد.
2. در هر تکرار مانند جستجوی اول سطح همه گره های آخرین سطح جاری پیمایش میشود.
- کم عمق ترین جواب را پیدا میکند.
- اگر مساله جواب نداشته باشد در حلقه بینهایت گیر میکند.
- اگر l=d آنگاه DLS بهتر از IDS خواهد بود.

۹۶

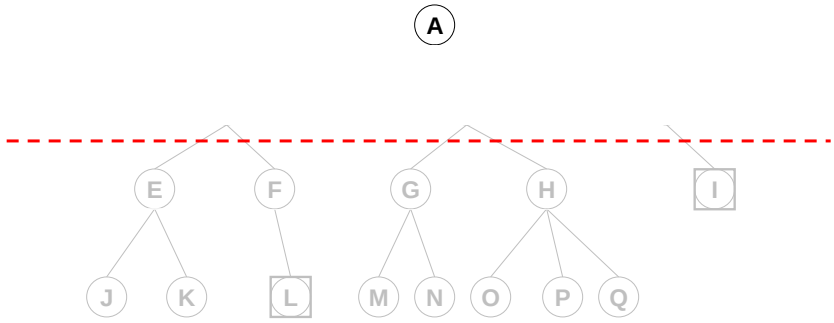
جستجوی عمیق شونده تکراری : الگوریتم

```
for l=0 to ∞
   if (DLS(l)="success" )  then
      exit "success"
```

- **در حالت کلي، زماني که فضاي جستجوي بزرگي وجود دارد و عمق راه حل نيز مجهول است IDS روش جستجوي مطلوبي است.**

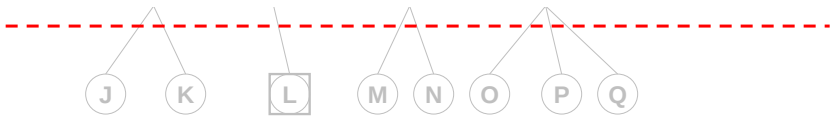
۹۷

جستجوی عمیق شونده تکراری:مثال۱



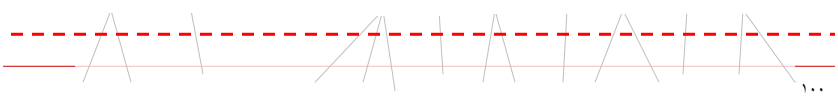
۹۸

جستجوی عمیق شونده تکراری: مثال ۱



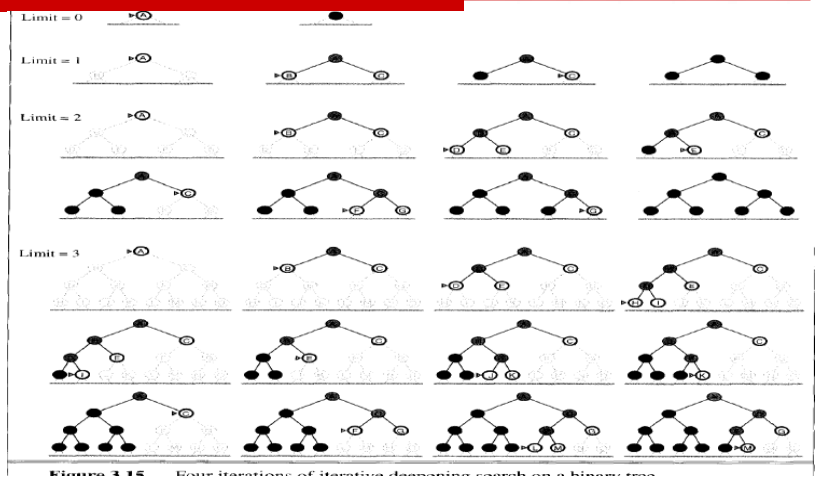
۹۹

جستجوی عمیق شونده تکراری: مثال ۲



۱۰۰

جستجوی عمیق شونده تکراری: مثال ۲



۱۰۱

مقایسه IDS با BFS

- مثال برای d=10,b=5:

$$IDS: (d)b + (d-1)b^2 + ... + (1)b^d = 50 + 400 + 3000 + 20000 + 100000 = 123450$$

$$BFS: b + b^2 + ... + b^d + (b^{d+1} - b) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100$$

- **IDS به رغم تولید تکراری برخی حالتها (در عمق کم با گره های کم) از BFS سریعتر است چون BFS در عمق d+1 نیز گره هایی تولید میکند.**

۱۰۲

جستجوی عمیق شونده تکراری : کارایی

- کامل است.

(به شرط محدود بودن b)

- لزوماً بهینه نیست.

(اگر هزینه مسیر تابعی غیر کاهشی از عمق درخت باشد بهینه است)

- پیچیدگي زمانی: $(d)b+(d-1)b^2+...+(1)b^d=O(b^d)$

$O(b^d)$

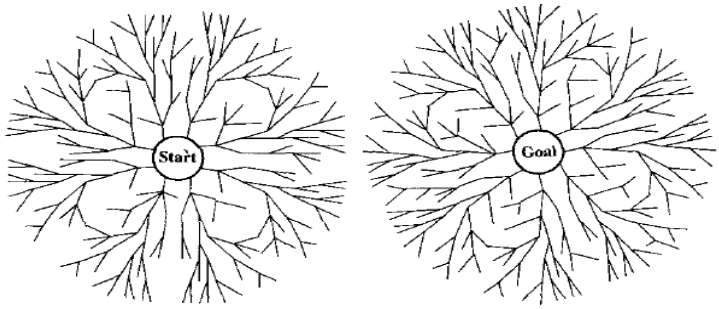
- پیچیدگی فضا:

$O(bd)$

۱۰۳

جستجوی دو طرفه

- انجام دو جست و جوی همزمان، یکي از حالت اولیه به جلو و دیگری از هدف به عقب تا زمانی که ایندو جستجو به هم برسند.
- انگیزه : $b^d >> b^{d/2}+b^{d/2}$



۱۰٤

جستجوی دوطرفه : مزایا و معایب

- نیاز به فضای زیاد و تعیین نوع جستجو در هر سمت نقطه ضعف این روش است.
- کاهش پیچیدگی زمانی مزیت این روش است.
- اگر هر دو طرف اول عمق باشند به احتمال زیاد به جواب نمیرسد.
  - اما آیا این روش همیشه امکانپذیر است؟
  - اهداف انتزاعی
  - معکوس عملگرها

هیچ راه حل کلی برای انجام اینکار بشکل موثر موجود نیست.

۱۰۵

جستجوی دوطرفه : کارایی

- کامل است.

(به شرط اول سطح بودن هر دو جستجو و محدودیت b)

- بهینه است.

(به شرط اول سطح بودن هر دو جستجو و هزینه یکنواخت)

- پیچیدگی زمانی: $O(b^{d/2})$
- پیچیدگی فضا: $O(b^{d/2})$

(حداقل یکی از درخت ها جهت بررسی عضویت باید در حافظه ذخیره شود)

* نیاز به فضای زیاد نقطه ضعف این روش است.

۱۰۶

مقایسه راهبردهای جستجوی ناآگاهانه

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^{d+1}	b^{d}	b^{m}	b^{l}	b^{d}	$b^{d/2}$
Space	b^{d+1}	b^{d}	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete	Yes	Yes	No	No	Yes	Yes

۱۰۷

جستجو با اطلاعات ناقص

- در بخشهای قبلی فرض بر این بود که :
1. محیط کاملاً رویت پذیر است.
2. عامل اثر هر اقدام را میداند.

لذا عامل نتیجه انجام یک توالی از اقدامات را دقیقاً میدانست.

- اگر دانش در مورد حالات یا اقدامات ناقص باشد، این نقص به سه نوع مساله مختلف منجر میشود:

۱۰۸

انواع مساله با اطلاعات ناقص

- مسائل فاقد حسگر: زمانی که عامل فاقد حسگر باشد.
- مسائل اقتضايي: زمانی که محیط نیمه رویت پذیر باشد یا اگر اقدامات غير قطعي باشد.
- مسائل تخاصمی: اگرعدم قطعیت در اثر اقدامات عامل دیگری بوجود آید.
- مسائل اکتشافی: وقتی حالتها و اقدامات محیط ناشناخته باشند، عامل بايد سعي كند آنها را كشف كند. مسئله های اکتشافی را میتوان شکل نهایی مسئله های اقتضايي دانست.

فصل چهارم

جستجوی آگاهانه

و

اکتشاف

راهبردهای جستجوی آگاهانه

- راهبردهای جستجوی ناآگاهانه میتوانستند راه حلها را با تولید نظام مند حالتهای جدید و آزمودن آنها در مقابل هدف بیابند، ولی ناکارا هستند.
- این بخش نسخه های آگاهانه الگوریتم های فصل قبل را توصیف میکند.
- یک راهبرد جستجوی آگاهانه با استفاده از دانش خاص مساله(فراتر از تعریف مساله)، میتواند راه حلها را بطور مؤثرتری نسبت به راهبردهای جستجوی ناآگاهانه بیابد.

جستجوی اول بهترینBFS

- نمونه ای از الگوریتم های عمومی (درختی یا گرافی) است که در آنها یک گره براساس یک تابع ارزیاب $f(n)$ جهت گسترش انتخاب میشود.
- گرهی که کمترین ارزیابی را دارد انتخاب میشود زیرا مقیاس ارزیابی فاصله تا هدف میباشد.
- تابع ارزیابی کاملاً دقیق نیست و ممکن است گاهی اوقات ما را به بیراهه نیز بکشاند.
- تابع ارزیابی برای انتخاب بهترین گره ظاهری (نزدیک ترین به هدف)، از یک تابع تخمین استفاده میکند.

توابع هیوریستیک

- برای اعمال آگاهی و هوشمندی به الگوریتم های جستجو از هیوریستیک استفاده میکنیم.
- توابع هیوریستیک معمول ترین شکل رساندن اطلاعات اضافی مساله به الگوریتم هستند.
- در واقع تخمینی است برای زود رسیدن به جواب.
- این توابع حدس ما را در مورد جواب مساله به الگوریتم اعمال میکنند.
- چون حدس وارد الگوریتم شده قطعیت را از دست میدهیم.

۱۱۳

تعاریف

- تابع هزینه مسیر g(n) :هزینه مسیر ازگره S تا گره n.
- تابع اکتشافی h(n) :هزینه تخمینی ارزانترین مسیر از گره n به G.
if(n=G) then h(n)=0
- تابع بهترین مسیر h*(n) :ارزان ترین مسیر واقعی از گره n تا گره G.
- تابع ارزیابي f(n) :هزینه تخمینی ارزان ترین مسیر از طریق n.
f(n)= g(n) + h(n)
- f*(n) : هزینه ارزان ترین مسیر واقعی یا قطعی از طریقn.
f*(n)= g(n) + h*(n)

۱۱٤

انواع توابع ارزیابی

- **f(n)= g(n)**

چون عموماً g(n) قطعی است این روش همان UCS خواهد بود.

- **f(n)= h(n)**

یک الگوریتم حریصانه است که فقط مسیری را بسط میدهد که ظاهراً به هدف نزدیکتر است.

- **f(n)= g(n) + h(n)**

یک تابع ارزیاب مناسب که اثر بایاس g و h را خنثی میکند.

در الگوریتم A* کاربرد دارد.

ویژگی تابع ارزیاب f(n)

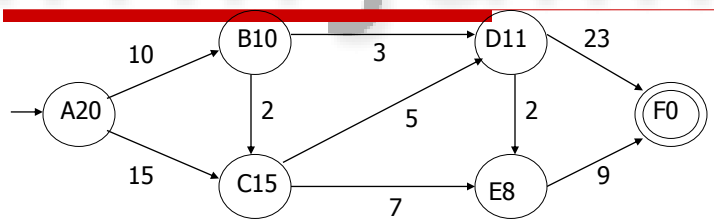
1. یک تخمین نزدیک به واقعیت به ما بدهد.
2. حتی الامکان خطای کمتری داشته باشد.
3. محاسبه آن ساده باشد.

- دقت h در تخمین مساله شدیداً دخالت دارد.
- اگر h(n)=∞ باشد یعنی که این گره اصلاً به جواب نمیرسد.
- اگر f<f* گوئیم دچار understimate شده و منجر به جواب صحیح (بهینه) خواهد شد.
- اگر f>f* گوئیم دچار overstimate شده و ممکن است منجر به جواب صحیح (بهینه) نشود.

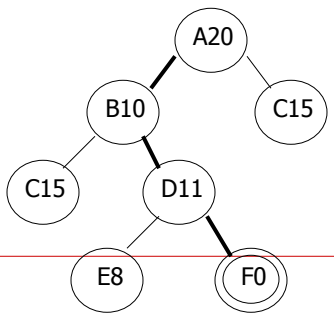
جستجوی اول بهترین حریصانه

- این روش سعی میکند تا نزدیکترین گره به هدف را گسترش دهد.
- پس در این روش $f(n)= h(n)$ خواهد بود.
- چون عموماً h تخمینی است یک اشتباه کوچک ممکن است ما را از رسیدن به جواب بهینه، باز دارد.

جستجوي اول بهترين حريصانه: مثال ۱



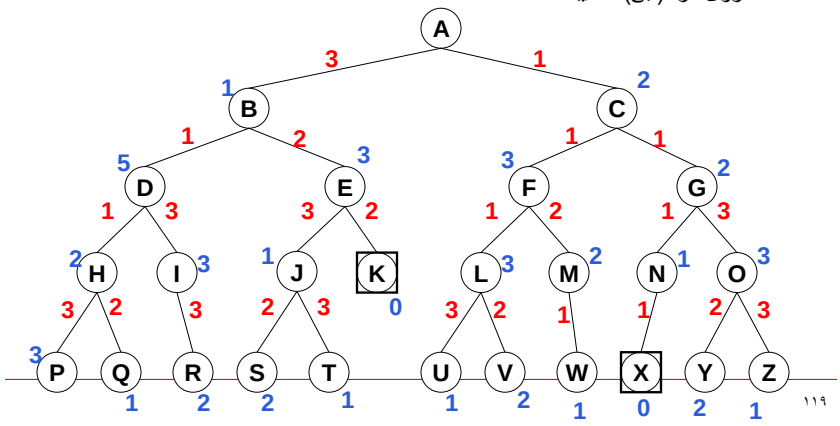
اعداد داخل گره، h میباشد.



جستجوي اول بهترين حريصانه: مثال۲

اعداد روی یال (قرمز) g میباشد.

اعداد روی گره (آبی) h میباشد.



۱۱۹

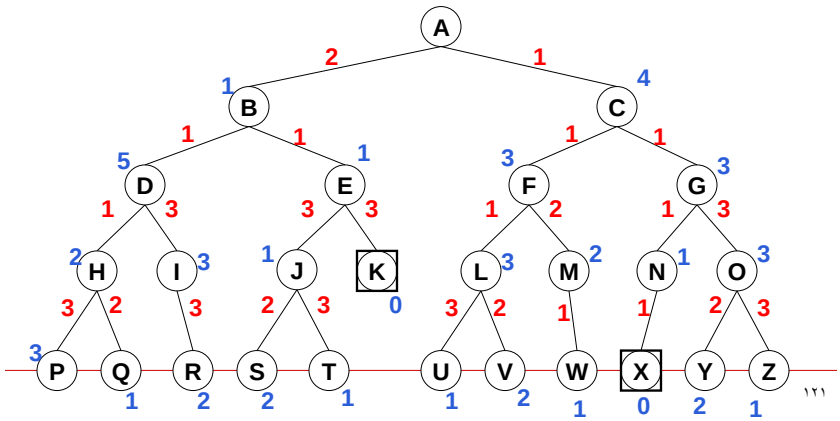
amirjalili.ir

جستجوي اول بهترين حريصانه: مثال۲

A

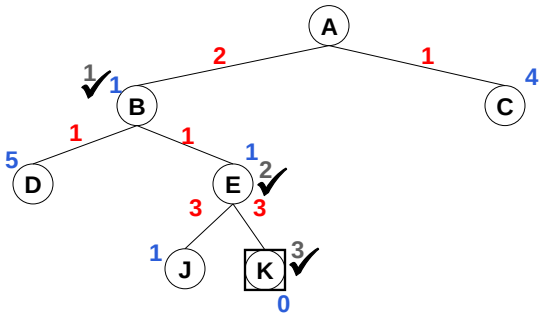
۱۲۰

جستجوي اول بهترين حريصانه: مثال۳



۱۲۱

جستجوي اول بهترين حريصانه: مثال۳



۱۲۲

جستجوی اول بهترین حریصانه: کارایی

- کامل بودن: خیر
  - اما اگر h <= h* آنگاه جستجو کامل میشود
- بهینگی: خیر
  - اما اگر h = h* آنگاه جستجو بهینه میشود
- پیچیدگی زمانی: $O(b^m)$
  - اما اگر h = h* آنگاه $O(bd)$
- پیچیدگی فضا: $O(b^m)$
  - اما اگر h = h* آنگاه $O(bd)$

۱۲۳

جستجوی کمینه کردن کل هزینه تخمین راه حل: A*

- رایجترین شیوه از جستجوی اول بهترین میباشد.
- در این روش تابع ارزیابی عبارت است از:

f(n)= g(n) + h(n)

- چون عموماً g هزینه قطعی (از گره S به گره n) و h هزینه تخمینی (از n گره به گره G) میباشد پس:

هزینه تخمینی ارزانترین راه حل از طریق n = f(n)

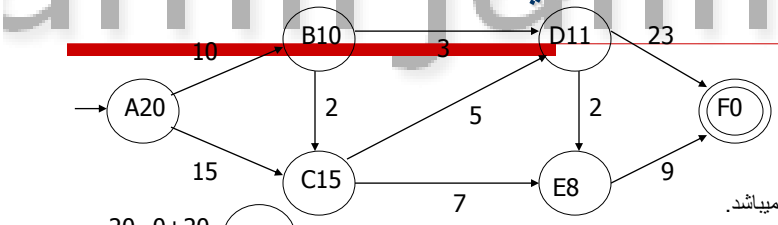
- این راهبرد کاملاً عاقلانه است.

۱۲٤

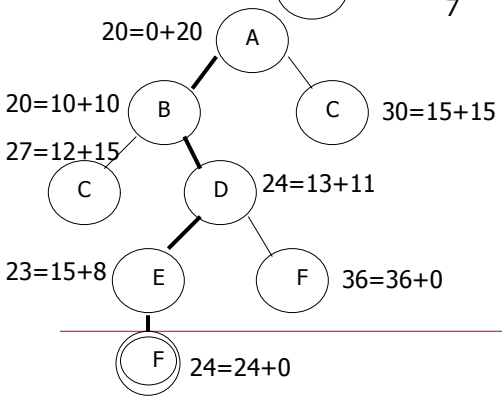
جستجوي A*

- یک h قابل قبول است اگر overstimate نباشد.
- هر h قابل قبول ذاتاً بهینه است.
- یک h سازگار است اگر: $h(n)<=C(n,x)+h(x)$ که در آن x فرزند n است.
- اگر h سازگار باشد آنگاه f در طول هر مسیری غیر نزولی است.
- A* تمام گره هایی که $f(n)< C^*$ هستند را توسعه میدهد.
- A* ممکن است برخی از گره هایی که $f(n)= C^*$ هستند را نیز قبل از انتخاب گره هدف توسعه دهد.

جستجوي A* : مثال۱



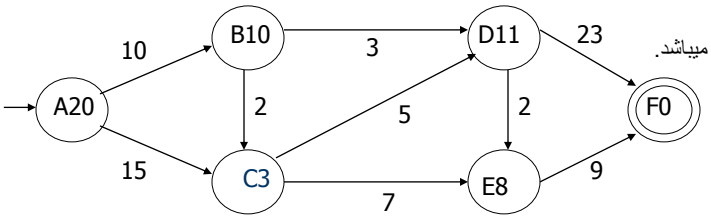
اعداد داخل گره، h میباشد.



f=g+h

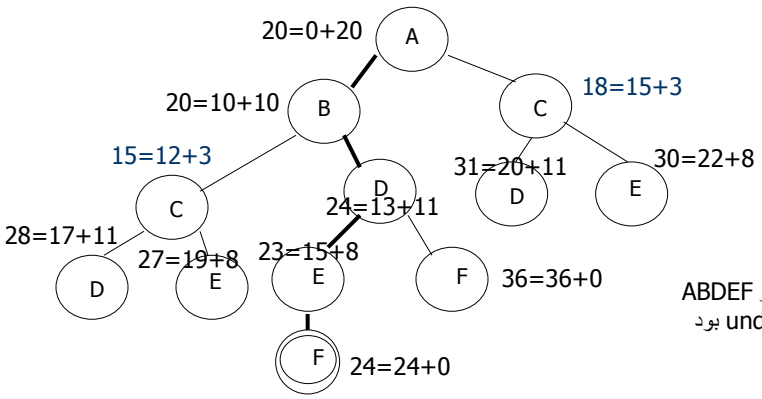
جواب مسیر ABDEF

جستجوي A* : مثال ۱ با understimate



اعداد داخل گره، h میباشد.

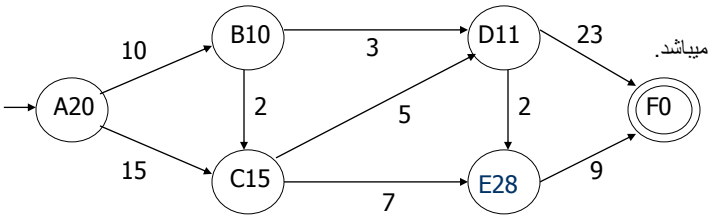
جستجوي A* : مثال ۱ با understimate



f=g+h

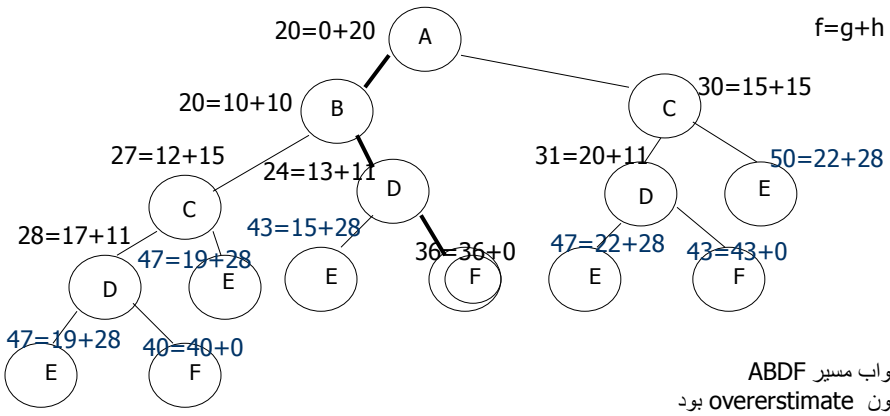
همچنان جواب مسیر ABDEF
چون understimate بود

جستجوي A* : مثال ۱ با overstimate



اعداد داخل گره، h میباشد.

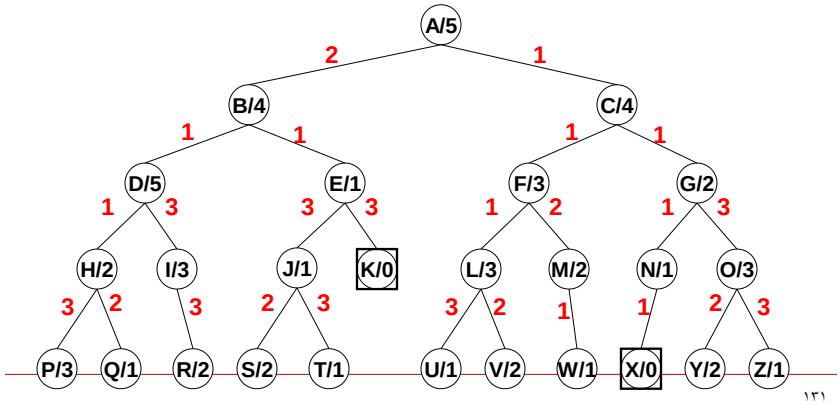
جستجوي A* : مثال ۱ با overstimate



f=g+h

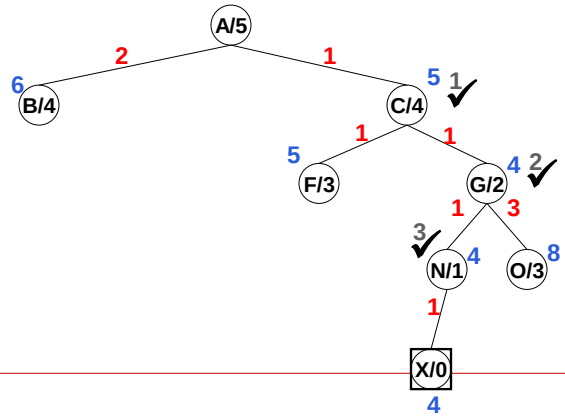
جواب مسیر ABDF
چون overerstimate بود

جستجوي A* : مثال۲



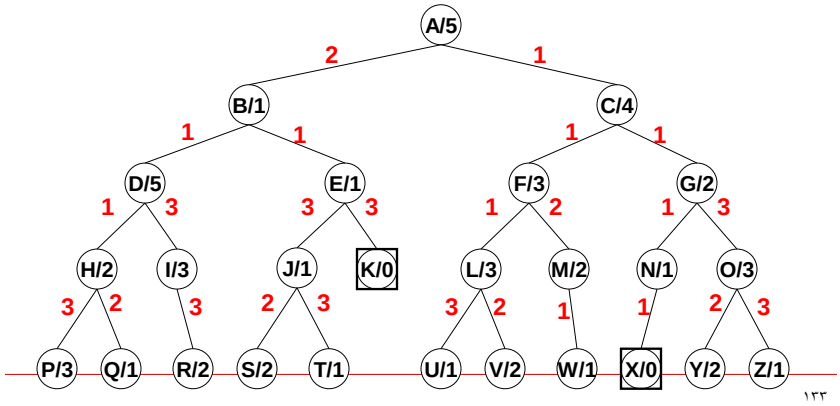
۱۳۱

جستجوي A* : مثال۲



۱۳۲

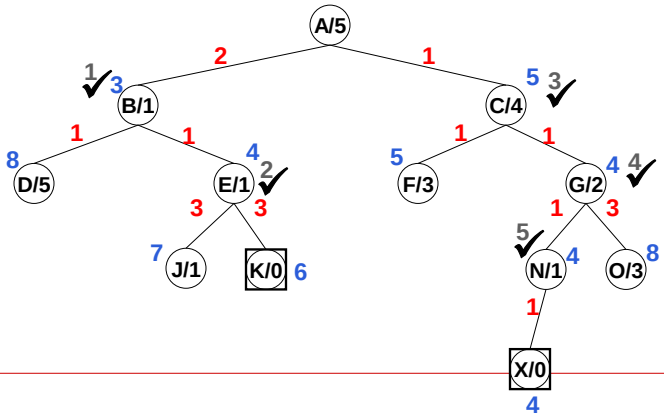
جستجوي A* : مثال٣



١٣٣

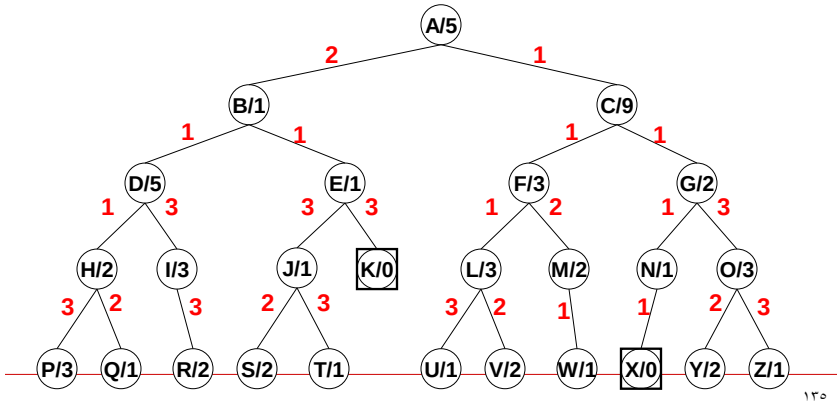
amirjalili.ir

جستجوي A* : مثال٣



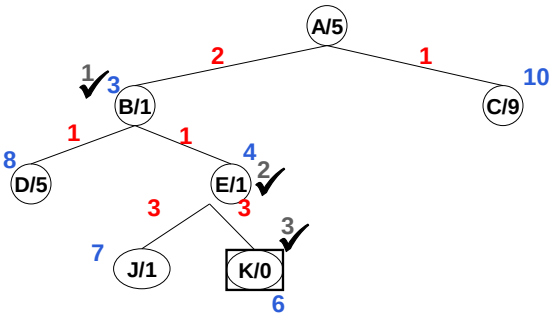
١٣٤

جستجوي A* : مثال۴



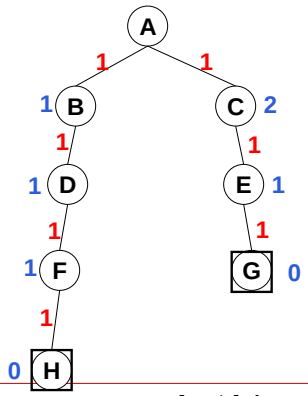
۱۳۵

جستجوي A* : مثال۴

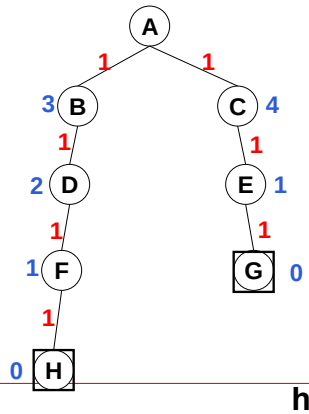


۱۳۶

جستجوي A* : مثال۵



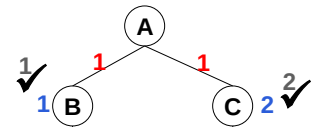
$h \leq h^*$



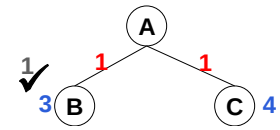
$h \nleq h^*$

amirjalili.ir

جستجوي A* : مثال۵

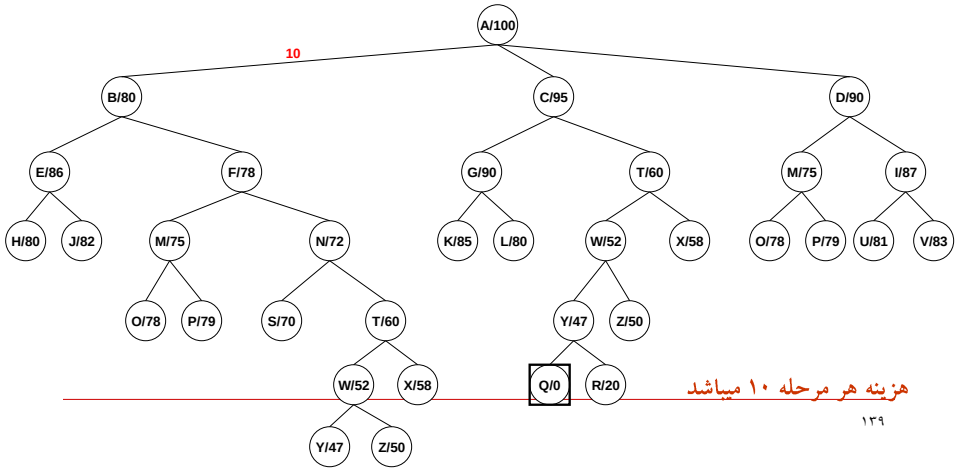


$h \leq h^*$



$h \nleq h^*$

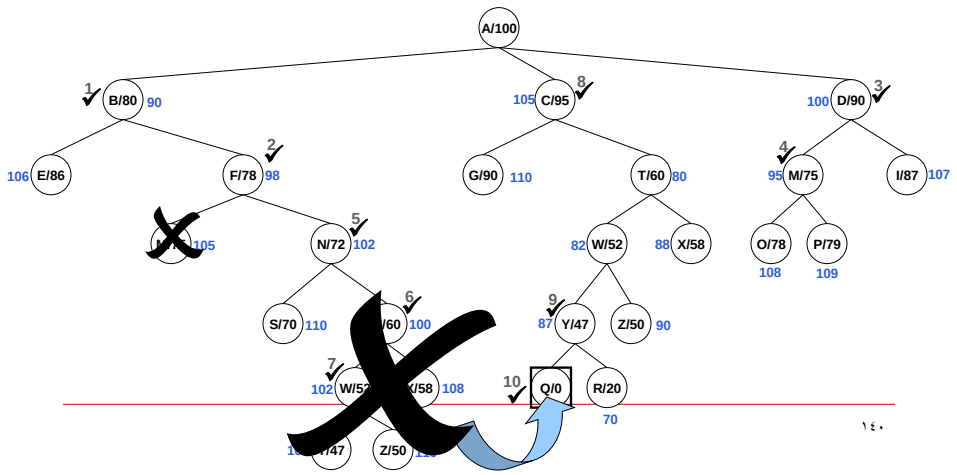
جستجوي A*(اجتناب از گره های تکراری): مثال۶



هزینه هر مرحله ۱۰ میباشد

۱۳۹

جستجوي A*(اجتناب از گره های تکراری): مثال۶



۱۴۰

جستجوی A* : کارایی

- کامل بودن: بله
  به شرط h قابل قبول و سازگار
- بهینگی: بله
  به شرط h قابل قبول و سازگار
- پیچیدگی زمانی: $O(b^m)$
  - اما اگر h = h* آنگاه $O(bd)$
- پیچیدگی فضا: $O(b^m)$
  - اما اگر h = h* آنگاه $O(bd)$

جستجوی A* : بهبود

دو روش برای بهبود A* وجود دارد:

- جستجوی A* با ضریب آلفا
- جستجوی A* با حافظه محدود: - روش SMA*
  - روش IDA*

جستجوی A* با آلفا

- میتوان برای f از رابطه زیر استفاده کرد:

$$f(n) = \alpha.g(n) + (1-\alpha).h(n)$$
$$0 \le \alpha \le 1$$

- آلفا را از نزدیک به صفر شروع کرده و در نزدیک یک به هدف میرسیم.
- با اینکار ابتدا با اتکا به h(تخمینی) یک سرعت اولیه میگیریم و بتدریج با اتکا به g(دقیق) دقت را افزایش میدهیم.
- مشکل این الگوریتم تعیین آلفا میباشد.

جستجوی A* با حافظه محدود

- الگوریتم A* بهینه موثر است یعنی هیچ الگوریتم بهینه دیگری تضمین نمیکند که تعداد گره هایی که توسعه میدهد از A* کمتر باشد.
- با اینکه A* کامل، بهینه و بهینه موثر است با این حال نیازهای جستجوی ما را بطور کامل رفع نمیکند.
- A* بیشتر از آنکه وقت کم بیاورد، حافظه کم می آورد. زیرا تمامی گره های تولید شده را جهت تضمین بهینگی در حافظه نگهداری میکند و به این دلیل برای مسائل بزرگ عملی نیست.

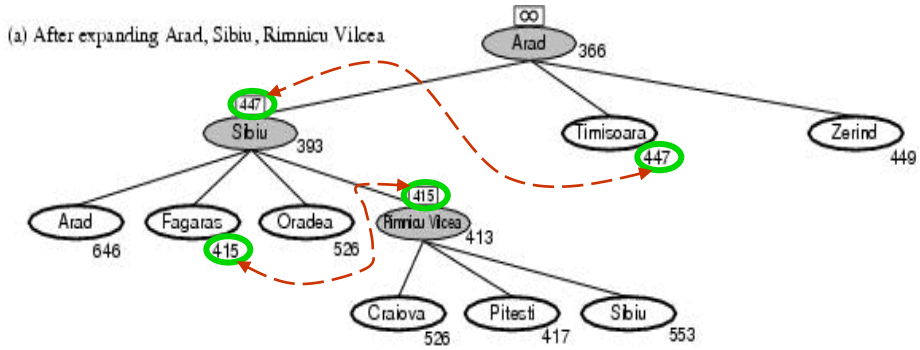
جستجوی A* عمیق کننده تکراری IDA*

- ساده ترین راه برای کاهش حافظه مورد نیاز A* استفاده از ایده عمیق کننده تکراری در زمینه جستجوی اکتشافي است.
- در جستجوی IDA*، برش مورد استفاده عمق نیست بلکه هزینه f=g+h است.
- IDA* اغلب برای مسئله هایی با هزینه گام واحد مناسب است و از سربار ناشي از نگهداری صف مرتبي از گره ها اجتناب میکند.

جستجوی اول بهترین بازگشتیRBFS

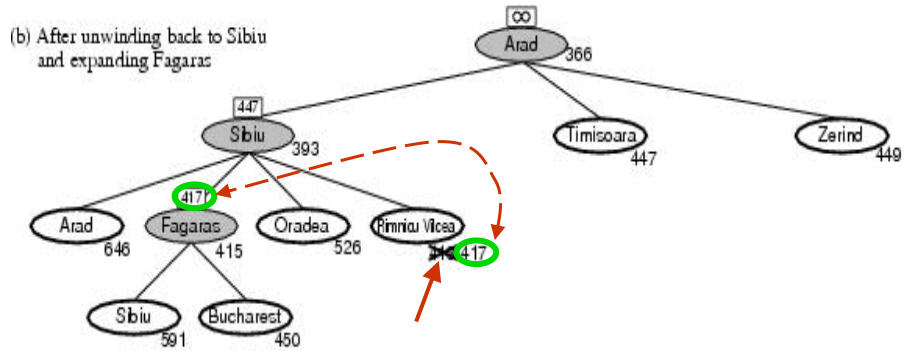
- این روش یک الگوریتم بازگشتی ساده است که سعی در تقلیدعملیات جستجوی اول بهترین را در فضای خطی دارد.
- ساختار آن شبیه جستجوی عمقي بازگشتي است، اما به جای اینکه دائما به طرف پایین مسیر حرکت کند مقدار f مربوط به بهترین مسیر از هر جد گره فعلی را نگهداری میکند، اگر گره فعلی از این حد تجاوز کند، بازگشتی به عقب برمیگردد تا مسیر دیگري را انتخاب کند.
- در حین بازگشت مقدار f هر گره را نیز با بهترین مقدار فرزندان آن گره تغییر میدهد.

جستجوی اول بهترین بازگشتی:مثال



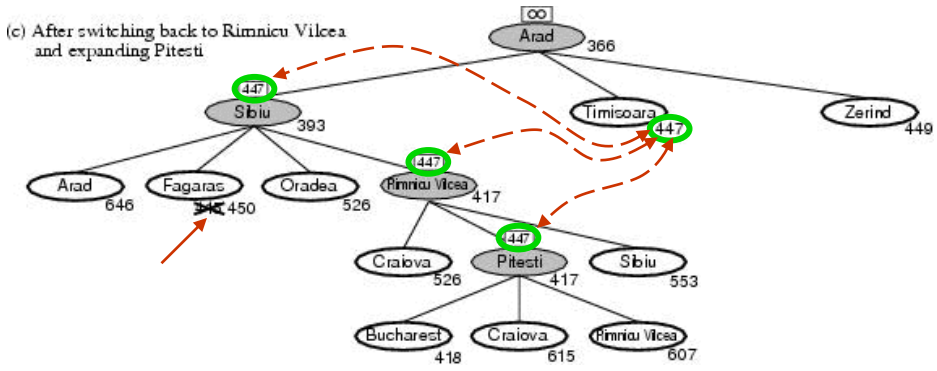
١٤٧

جستجوی اول بهترین بازگشتی:مثال



١٤٨

جستجوی اول بهترین بازگشتی:مثال



مقایسه RBFS و IDA*

- RBFS تا حدی از IDA* کارآمدتر است، اما هنوز هم گره های زیادی تولید میکند.
- IDA* و RBFS در معرض افزایش نمایی پیچیدگي قرار دارند که در جستجوی گرافها مرسوم است. زیرا نمیتوانند حالتهای تکراری را در غیر از مسیر فعلي بررسي کنند. لذا، ممکن است یک حالت را چندین بار بررسي کنند.
- IDA* و RBFS از فضای اندکي استفاده میکنند که به آنها آسیب میرساند.
- IDA* در هر تکرار فقط یک عدد نگهداری میکند(مقدار فعلي f).
- RBFS اطلاعات بیشتری در حافظه نگهداری میکند.

جستجوی RBFS : کارایی

- کامل بودن: بله
- بهینگی: بله

به شرط h قابل قبول و سازگار

- پیچیدگي زماني:

به دقت تابع هیوریستیک و تعداد تغییر مسیر بستگی دارد.

- پیچیدگی فضا: $O(bd)$

جستجوی A* مقید به حافظه ساده SMA*

- بهترین گره را تا زمانی بسط میدهد که حافظه پر شود.
- در این نقطه بدون از بین بردن گره های قبلي نمیتواند گره جدیدی اضافه کند.
- SMA* همیشه بدترین گره برگ را حذف میکند و مانند RBFS مقدار گره فراموش شده را به پدرش برمیگرداند.
- جد زیر درخت فراموش شده، کیفیت بهترین مسیر را در آن زیر درخت میداند بدین ترتیب زیر درخت فراموش شده فقط وقتی بسط داده میشود که از تمامی مسیرهای دیگر بهتر باشد.
- این الگوریتم بر خلاف RBFS، قادر است تا از تمام حافظه موجود براي اجراي جستجو استفاده کند.
- استفاده از حافظه بیشتر کارایي جستجو را وسعت مي‌بخشد.

جستجوی SMA*: کارایی

- کامل بودن: بله
  - به شرط آنکه حافظه براي ذخيره کم عمق‌ترين مسير راه حل کافي باشد.
- بهینگی: بله
- پيچيدگي زماني:؟
- پیچیدگی فضا:؟

- SMA* بهترین الگوریتم همه منظوره برای یافتن راه حلهای بهینه میباشد بخصوص در یک گراف با هزینه های غیریکنواخت.

یادگیری برای جستجوی بهتر

- عامل با استفاده از فضای حالت فرا سطح میتواند یاد بگیرد که بهتر جستجو کند.
- هر حالت در فضای حالت فرا سطح، حالت(محاسباتی) داخلی برنامه ای را که در یک فضای حالت سطح شیء در حال جستجو است، ثبت میکند.
- هر اقدامی در فضای حالت فرا سطح، یک گام محاسباتی است که حالت داخلی را تغییر میدهد.
- الگوریتم یادگیری فرا سطح میتواند چیزهايي را از تجربیات یاد بگیرد تا زیردرختهای غیر قابل قبول را کاوش نکند.
- هدف یادگیری، کمینه کردن کل هزینه حل مسئله است.

توابع هیوریستیک:مثال پازل

- هزینه متوسط راه حل یک نمونه تصادفی حدود ۲۲ گام است با فاکتور انشعاب تقریباً ۳.(حرکتها ۲یا۳یا۴)
- جست و جوی جامع تا عمق ۲۲ : $3^{22} \approx 3.1 \times 10^{10}$
- با نگهداری سابقه حالات تکراری، از یک حالت فقط 9!/2=181440 حالت متمایز قابل دسترسی وجود دارد.؟؟؟

(کل حالات 9!=362880)

- پس به یک هیوریستیک خوب جهت کاهش مراحل جستجو نیاز داریم.
- یک تابع هیوریستیک خوب هرگز تعداد گامها تا هدف را بیش از اندازه واقعی برآورد نمیکند.

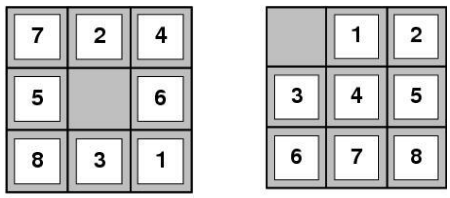
۱۵۵

تابع هیوریستیک اول برای پازل

- h1 = تعداد خانه‌هايي كه در مكان‌هاي نادرست هستند.

h1 یک هیوریستیک خوب است، زیرا واضح است که هر خانه‌اي كه خارج از مكان درست باشد حداقل يكبار بايد جابجا شود.

h1=8



۱۵۶

تابع هيوريستيک دوم برای پازل

- h2 = مجموع فواصل خانه‌ها از مکان‌هاي هدف صحيحشان است. فاصله‌اي که ما حساب مي‌کنيم، مجموع فواصل عمودي و افقي است (نه حرکت قطری) که فاصله بلوک شهر يا منتهن ناميده مي‌شود.

$$h_2 = 3+1+2+2+2+3+3+2 = 18$$

- h2 قابل قبول است زيرا هر جابجايي که ميتواند انجام گيرد به اندازه يک مرحله به هدف نزديک ميشود.
- هيچکدام بيش از هزينه واقعی(۲۶گام) برآورد ندارند.

تاثير دقت هيوريستيک بر کارايي

- يک روش براي تشخيص کيفيت هيوريستيک فاکتور انشعاب مؤثر b* است.
  - اگر تعداد کل گره هايي که برای يک مسئله خاص توسط A* توليد ميشود برابر با N و عمق راه حل برابر با d باشد، آن گاه b* فاکتور انشعابی است که درخت يکنواختی به عمق d بايد داشته باشد تا حاوی N+1 گره باشد. پس:

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

- فاکتور انشعاب مؤثر برای نمونه های مختلف يک مساله ميتواند متفاوت باشد ولی معمولاً برای مسئله های سخت ثابت است.
- مقدار b* در اکتشافي با طراحي خوب، نزديک به ۱ است تا حل مسائل بزرگ را ممکن سازد.

تاثير دقت هيوريستيک بر کارايي:مثال

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Figure 4.8 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle, for various solution lengths.

۱۵۹

تاثير دقت هيوريستيک بر کارايي

- اگر براي هر گره n داشته باشيم: h2(n) >= h1(n)
  - h2 بر h1 برتری دارد.
  - برتر بودن مستقيما به کارايي ترجمه ميشود.
  - تعداد گره هايي که A* با بکارگيری h2 بسط ميدهد، هرگز بيش از بکارگيری h1 نيست.
- همیشه بهتر است از تابع هیوریستیک با مقادیر بالاتر استفاده کنیم. به شرطی که:
  1. بيش از اندازه تخمين نزند.
  2. زمان محاسبات برای اکتشاف، خيلي بزرگ نباشد.

۱۶۰

ابداع توابع هیوریستیک قابل قبول

- مساله ای که نسبت به مساله اصلی تعداد کمتری محدودیت برای اقدامات داشته باشد یک مساله تعدیل شده است.
- هزینه یک راه حل بهینه برای یک مساله تعدیل شده یک هیوریستیک قابل قبول برای مساله اصلی است.
- زیرا راه حل بهینه در مساله اصلی، راه حلی در مساله تعدیل شده است. در نتیجه باید از نظر هزینه با هزینه راه حل بهینه در مساله تعدیل شده یکی باشد.
- چون هیوریستیک بدست آمده یک هزینه دقیق برای مساله تعدیل شده میباشد طبق نامساوی مثلث، سازگار است.

ابداع توابع هیوریستیک قابل قبول

- اگر تعریف یک مساله به زبان رسمی باشد، ساخت خودکار مساله تعدیل شده امکانپذیر است.
- برنامه ABSOLVER از تعاریف مساله با روش مساله تعدیل شده، بطور خودکار قادر به تولید هیوریستیک میباشد.
- مشکل تولید توابع هیوریستیک جدید اینست که معمولاً نمیتوان مشخصاً بهترین را پیدا کرد.
- هیوریستیک قابل قبول را میتوان از هزینه راه حل یک زیر مساله از مساله اصلی بدست آورد.

ابداع توابع هیوریستیک قابل قبول

- یک زیرمساله برای پازل ۸ تایی:

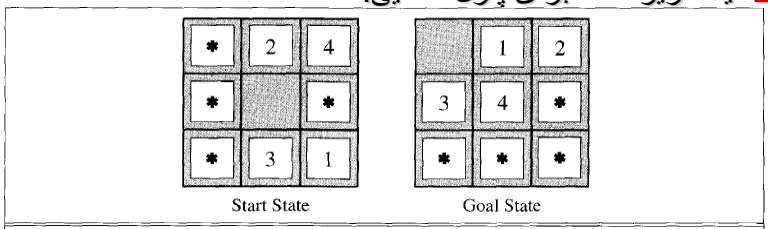


Figure 4.9 A subproblem of the 8-puzzle instance given in Figure 4.7. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles.

- هزینه راه حل بهینه برای این زیر مساله، یک حد پایین برای هزینه مساله کامل است.

۱۶۳

فراگیری هیوریستیکها از تجربه

- علاوه بر راه حل مساله تعدیل شده، یک روش دیگر فراگیری از طریق تجربه میباشد.

- منظور از تجربه در اینجا مثالهای زیادی از پازل ۸تایی است.
  با این مثالها میتوان یک الگوریتم فراگیری استقرایی یافت.

- اگر روشهای فراگیری استقرایی را بجای فقط چند توصیف خام حالت، با ویژگیهایی از حالت که با ارزیابی آن مرتبط هستند تغذیه کنیم از همیشه بهتر عمل کرده ایم.

۱۶٤

الگوریتم های جستجوی محلی و مسائل بهینه سازی

- الگوریتمهای جستجو، فضای جستجو را بطور سیستماتیک بررسی میکنند و:
  - تا رسیدن به هدف یک یا چند مسیر نگهداری میشوند.
  - مسیر رسیدن به هدف، راه حل مسئله را تشکیل میدهد.
- در بسیاری از مسائل مسیر به هدف ارتباطی ندارد.
- اگر مسیر هدف اهمیت نداشته باشد، الگوریتم های متعددی مطرح میگردد که اصلاً به مسیر اهمیت نمیدهند.

۱۶۵

الگوریتم های جستجوی محلی و مسائل بهینه سازی

- الگوریتم های جستجوی محلی با استفاده از حالت فعلی عمل میکنند و عموماً فقط به همسایگان آن حالت تغییر مکان میدهند.
- معمولاً مسیرهای دنبال شده توسط جستجو، نگهداری نمیشوند.
- این الگوریتم ها اگرچه نظام مند نیستند ولی:

1. بسیار کم از حافظه استفاده میکنند.
2. اغلب در فضاهای بزرگ یا نامتناهی پیوسته(که الگوریتم های نظام مند مناسب نیستند)، به راه حل معقولی میرسند.

۱۶۶

الگوریتم های جستجوی محلی و مسائل بهینه سازی

- الگوریتم های جستجوی محلی علاوه بر یافتن اهداف در مسائل خالص بهینه سازی نیز سودمند میباشند.
- در مسائل خالص بهینه سازی هدف، یافتن بهترین حالت بر اساس تابع هدف میباشد.

- یک الگوریتم جستجوی محلی کامل، همیشه هدف را در صورت وجود پیدا میکند.
- یک الگوریتم جستجوی محلی بهینه، همیشه یک کمینه یا بیشینه سراسری را پیدا میکند.

۱۶۷

الگوریتم های جستجوی محلی و مسائل بهینه سازی

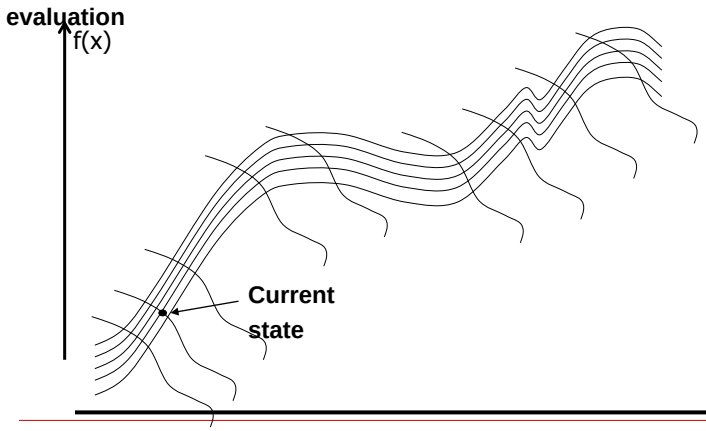
- برای درک جستجوی محلی، توجه به دورنمای فضای حالت بسیار مفید خواهد بود. یک دور نما شامل محل و ارتفاع میباشد.

- اگر ارتفاع متناظر هزینه باشد، آنگاه هدف یافتن یک کمینه یا بیشینه سراسری میباشد.

- الگوریتمهای جستجوی محلی این دور نما را کشف میکنند.

۱۶۸

الگوریتم های جستجوی محلی و مسائل بهینه سازی



۱۶۹

جستجوی تصادفی

- ساده ترین و ابتدایی ترین الگوریتم جستجوی محلی است.
- عدد تصادفی x را تولید و فرض میکنیم که f(x) بهینه است!!!!
- K بار عدد تصادفی x را تولید و برای هر کدام f(x) را پیدا میکنیم. سپس بهترین مقدار بین آنها را بهینه در نظر میگیریم.
- عملاً در این حالت نیز احتمال رسیدن به جواب بهینه کم است.

۱۷۰

الگوریتم های جستجوی محلی و مسائل بهینه سازی

- این الگوريتم‌ها به دو گره اصلي تقسيم مي‌شوند:

- الگوريتم‌هاي تپه‌نوردي Hill-climbing
  - تپه نوردی استاندارد
  - تپه نوردی با k شروع مجدد تصادفی
- سخت سازی شبیه سازی شده Simulated Annealing
  - پرتو محلی
  - الگوریتم ژنتیک

۱۷۱

جستجوی تپه نوردی

- در این روش در یک حلقه، مدام در جهت افزایش مقدار حرکت میکنیم و الگوریتم هنگامی پایان مییابد که به قله ای برسیم که هیچیک از همسایه ها مقدار بیشتری نداشته باشد.

- این الگوریتم درخت جستجو را نگهداری نمیکند و هر گره فعلی تنها نیاز به ثبت حالت و مقدار تابع هدفش دارد.

- تپه نوردی فراتر از همسایه های مجاور حالت فعلی را نگاه نمیکند.

۱۷۲

جستجوی تپه نوردی

- حلقه اي كه مدام در جهت افزايش مقدار (بطرف بالای تپه) حركت ميكند.
- رسيدن به بلندترين قله در همسايگی حالت فعلی، شرط خاتمه است.
- در صورت وجود بيش از يک بهترين پسين، معمولاً از اين مجموعه به تصادف يکی را انتخاب ميكند.

جستجوی تپه نوردی

- جستجوی تپه نوردی، جستجوی محلی حریصانه نیز نام دارد. چون یک حالت همسایه خوب را بدون فکر قبلي كه از آنجا به كجا خواهد رفت، انتخاب ميكند.
- الگوریتم های حریصانه اغلب بسیار خوب عمل میکنند.
- تپه نوردی اغلب پیشرفت سریعی به سمت راه حل دارد چون بهبود یک حالت بد خیلی ساده است.

جستجوی تپه نوردی

- در تپه نوردی یک x تصادفی تولید میکنیم و یک مقدار $\pm\Delta x$ به آن اعمال میکنیم هر کدام بیشتر باشد آنرا دنبال میکنیم تا زمانیکه:

$$f(x\pm\Delta x)<f(x)$$

- مشکل این الگوریتم تعیین ΔX میباشد.
- تعداد پسین ها برای فضای n بعدی:

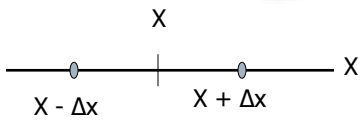
$$3^n-1$$

- این همان فاکتور انشعاب یا b میباشد.

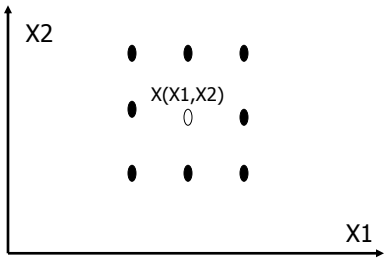
- چون فاکتور انشعاب نمایی است برای ابعاد بالا مناسب نیست.

جستجوی تپه نوردی:مثال

- فضای یک بعدی:



- فضای دو بعدی:

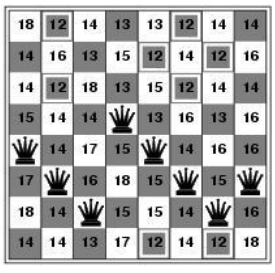


جستجوی تپه نوردی: مثال ۸ وزیر

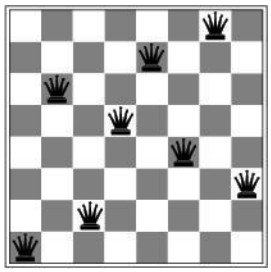
- الگوریتم های جستجوی محلی اغلب از فرمول بندی حالت کامل استفاده میکنند.
- در این روش هر حالت شامل ۸ وزیر، یکی در هر ستون میباشد.
- تابع پسین تمام حالاتی که با جابجا کردن یک وزیر به خانه دیگر در همان ستون تولید میشود.هر حالت 8*7=56 حالت پسین دارد.
- تابع هیوریستیک h، تعداد جفت وزیرهایی که نسبت به هم، مستقیم یا غیر مستقیم گارد دارند.
- کمینه سراسری این تابع صفر است.(در راه حل کامل اتفاق می افتد)

۱۷۷

جستجوی تپه نوردی: مثال ۸ وزیر



حالتی با h=17
مقادیر تمام پسینها در شکل مشخص است
بهترین پسین دارای h=12میباشد



یک کمینه محلی با h=1
با ۵ گام فاصله از شکل قبلی
پسینها دارای h بالاتری میباشد

۱۷۸

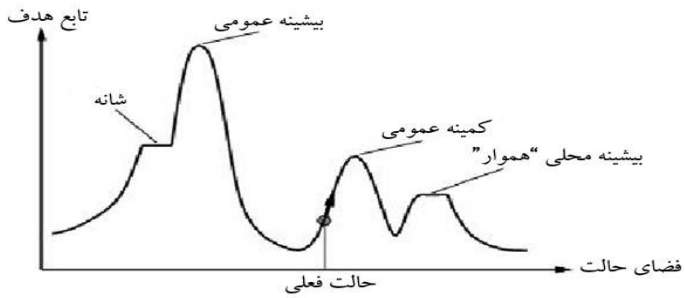
جستجوی تپه نوردی: معایب

1. بیشینه محلی: قله ای که از تمام حالتهای همسایه بلندتر و از بیشینه سراسری کوتاهتر است.
2. دماغه: یک رشته بیشینه محلی هستند که گذر از آنها برای الگوریتمهای حریصانه بسیار مشکل است.

1. فلات: ناحیهای که در آن تابع ارزیاب ثابت است. میتواند یک بیشینه محلی مسطح نیز باشد که هیچ مسیر رو به بالایی ندارد یا یک شانه باشد که بتوان از آن بالاتر هم رفت. ممکن است.

یک دور نمای فضای حالت یک بعدی

الگوریتم تپه نوردی ممکن است به نقطه ای برسد که قادر به خروج از هیچیک از سه حالت قبلی نباشد.



جستجوی تپه نوردی: انواع

- تپه نوردی اتفاقی: بصورت تصادفی از میان حرکات رو به بالا یکی را انتخاب میکند.
- تپه نوردی اولین گزینه: از تپه نوردی اتفاقی استفاده میکند و بصورت تصادفی پسین تولید میکند تا زمانیکه پسین تولید شده از حالت فعلی بهتر باشد.
- همه این روشها ناکامل هستند و معمولاً در بیشینه محلی گیر میکنند.
- تپه نوردی با K شروع مجدد تصادفی مستقل

تپه نوردی با K شروع مجدد تصادفی

- این روش یک مجموعه جستجوی تپه نوردی را از حالتهای شروع تصادفی اجرا میکند و هنگام پیدا شدن هدف متوقف میشود.
- به احتمال زیاد این روش کامل است. چون بالاخره یک حالت هدف را به عنوان حالت شروع تولید خواهد کرد.

- از بین k تپه نوردی بهترین جواب را انتخاب میکنیم.
- با افزایش k احتمال رسیدن به جواب افزایش پیدا میکند.

جستجوی تپه نوردی: ارزیابی

- موفقيت hill-climbing خيلي به ظاهر دور نمای فضاي حالت بستگي دارد: اگر فلات و ماكزيمم‌هاي محلي كمي وجود داشته باشد، تپه‌نوردي با شروع تصادفي خيلي سريع يک راه حل خوب را پيدا خواهد كرد.
- بسیاری از مسائل واقعی دارای دورنمایی شبیه جوجه تیغی هستند.
- مسائل سخت معمولاً دارای تعداد نمایی بیشینه محلی هستند.

جستجوی سخت سازی شبیه سازی شده

- نامهای مختلفی از جمله شبیه سازی حرارت دارد.
- چون الگوریتم تپه نوردی هرگز به سمت پایین (حالاتی با مقادیر کم یا هزینه بالا) حرکت نمیکرد ناکامل است.
- زیرا ممکن است در یک بیشینه محلی گیر کند.
- در مقابل جستجوی تصادفی کامل ولی ناکارا است.
- لذا ترکیب تپه نوردی و جستجوی تصادفی کار معقول است.
- شبیه سازی حرارت چنین الگوریتمی است.
- حرارت با درجه بالا و به تدریج سرد کردن.

جستجوی سخت سازی شبیه سازی شده

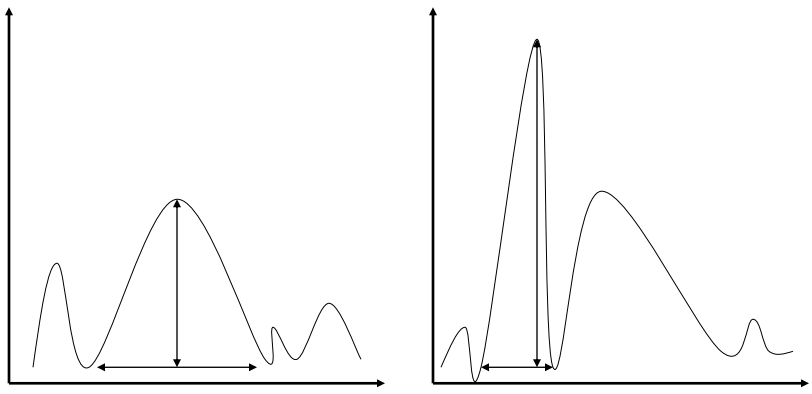
- در این الگوریتم علاوه بر ΔX یک Δt نیز داریم.

$$\Delta t >> \Delta x$$

- Δt ما را از یک بیشینه محلی بیرون می آورد.
- در هر مرحله Δt را کاهش میدهیم.
- زمانیکه قله جدید با قله قبلی یکی شد کار تمام است.
- به شرط برابر بودن نسبت ارتفاع به پهنای قله های مختلف جواب بهینه را پیدا میکند.

۱۸۵

جستجوی شبیه سازی حرارت : مثال



۱۸۶

جستجوی شبیه سازی حرارت : مثال

- پایین آمدن توپ از شیب در سطحی ناهموار(کمینه کردن هزینه)
  - توپ در فرود از تپه به کمینه محلی میرود
  - با تکان دادن سطح، توپ از کمینه محلي خارج میشود
  - با تکان شدید شروع(دمای زیاد)
  - بتدریج کاهش شدت تکان (دمای پایین تر)

شدت تکانها باید مناسب باشد تا فقط از کمینه محلی خارج گردد.

اگر زمانبندی دما به اندازه کافی آهسته کاهش یابد، الگوریتم یک بهینه عمومی را مي یابد

جستجوی پرتو محلی

- نگهداری یک گره در حافظه واکنشی افراطی نسبت به مساله محدودیت حافظه است.
- جستجوی پرتو محلی اطلاعات k حالت را نگهداری میکند.
- K حالت تصادفی تولید و تمام پسینهای آنها تولید میشود اگر هر کدام از آنها هدف بود پایان میپذیرد در غیر اینصورت بهترن k پسین از لیست انتخاب و الگوریتم ادامه می یابد.
- اگر فاکتور انشعاب b باشد در هر مرحله bk پسین داریم.

جستجوی پرتو محلی

- تفاوت پرتو محلی با k شروع مجدد تصادفی:
  - در جستجوی پرتو محلی، اطلاعات مفید بین k فرایند موازی مبادله میشود.
  - در جستجوی شروع مجدد تصادفی، هر فرایند مستقل از بقیه اجرا میشود.

بهترینها اینجاست!!!!

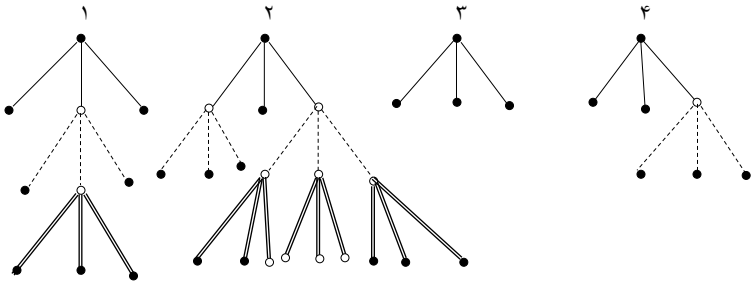
- پس کاملاً با یکدیگر تفاوت دارند.

- این الگوریتم به سرعت جستجوهای بی ثمر را رها و منابع را به جایی منتقل میکند که بیشترین پیشرفت صورت گیرد.

۱۸۹

جستجوی پرتو محلی:مثال

- با فرض k=4 و b=3 :



۱۹۰

جستجوی پرتو محلی:بهبود

- جست و جوی پرتو اتفاقی یا غیرقطعی:
  - به جای انتخاب بهترین k پسین، k جانشین تصادفی را انتخاب میکند. بطوریکه احتمال انتخاب یک پسین خاص، یک تابعی صعودی از مقدارش باشد.

- جست و جوی پرتو بهبود یافته:
  - در هر مرحله به bk فرزند تولید شده، N نقطه تصادفی جدید تولید و اضافه کنیم سپس از بین bk+N گره، k گره بهینه را انتخاب کنیم و مرحله بعدی را تکرار کنیم. با اینکار یک تنوع به الگوریتم میدهیم.

الگوریتم های ژنتیک(GA)

- شکلی از جستجوی پرتو غیر قطعی است که:
  حالتهای پسین از طریق ترکیب دو حالت والد تولید میشود.
- از نظریه تکامل گرفته شده است.
- الگوریتم ژنتیک در سه زمینه مطرح است:

1. طبیعت
2. بهینه سازی
3. جستجوی فضای حالت

الگوریتم های ژنتیک

- در GA نیز شبیه جستجوی پرتو با یک مجموعه تصادفی که جمعیت یا population نامیده میشود شروع میکنیم.
- میزان جمعیت بر اساس اندازه جمعیت یا popsize مشخص میگردد.
- براساس تعداد نسل یا maxgen الگوریتم اقدام به تکرار تولید نسل میکند.

۱۹۳

الگوریتم های ژنتیک: عملگرهای اصلی

- افراد جمعیت راتوسط تابع برازش یا Fitness function ارزیابی میکنیم.

1. تابع انتخاب یا selection بر اساس ارزیابی Ff اقدام به انتخاب از بین نسلها به اندازه popsize میکند.
2. تابع برش(تقاطع) یا crossover نقطه پیوند را جهت تولید فرزندان(offspring) از والدین(parents) نشان میدهد.
3. تابع جهش یا mutation نیز نشان دهنده میزان جهش در بیتها(ژنها) میباشد.

۱۹٤

الگوریتم های ژنتیک

- هر حالت یا فرد از جمعیت (کروموزوم) بصورت یک رشته از الفبای محدود (ژن) نشان داده میشود.(معمولاً 0,1)
- مثال ۸ وزیر:

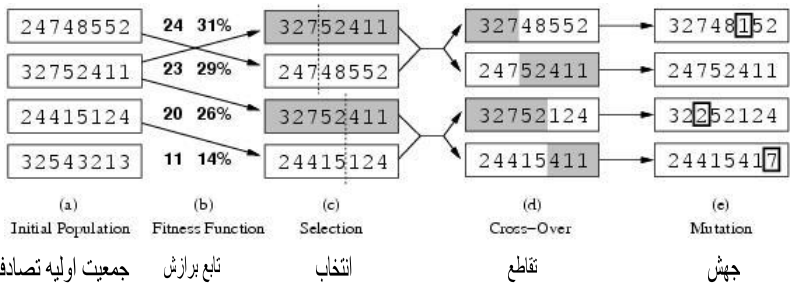
1. در هر حالت باید موقعیت ۸ وزیر را هر یک در یک ستون (با ۸ خانه) نشان داد. پس تعداد بیت(ژن) مورد نیاز:

$$8*\log_2^8 = 24$$

2. هر حالت با ۸ رقم که هر یک بین ۱ تا ۸ هستند نشان داده شود. پس این روش ۸ ژن خواهد داشت.

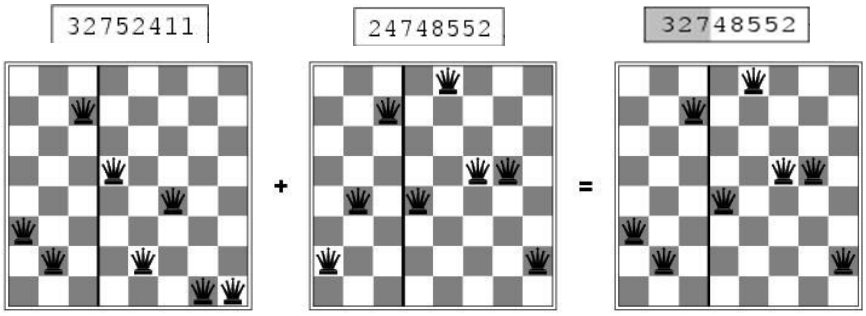
الگوریتم های ژنتیک:مثال ۸ وزیر

در این شکل در a جمعیت اولیه (۴ رشته ۸ بیتی) دیده میشود.
تولید نسل بعدی حالتها در b تا e دیده میشود.



الگوریتم های ژنتیک:مثال ۸ وزیر

حالتهای ۸ وزیر متناظر با دو والد اول و فرزند اول



۱۹۷

الگوریتم های ژنتیک

- یک تابع برازش برای حالتهای بهتر باید عدد بزرگتری برگرداند.
- در این مثال تعداد وزیرهایی که به هم حمله نمیکنند. برای یک راه حل برابر ۲۸میباشد.؟؟؟
- در این روش خاص از GA، احتمال انتخاب مستقیماً متناسب است با Ff.
- یک مورد دو بار انتخاب شده است و موردی دیگر اصلاً.
- برای هر جفت یک نقطه پیوند تصادفی انتخاب میشود.

۱۹۸

الگوریتم های ژنتیک

- اغلب در اوایل فرایند تقاطع جمعیت کاملاً متفاوت هستند. بنابراین در اوایل پیوند گامهای بزرگی در فضای حالت بر میدارد (مانند SA) و بعدها که اکثر نمونه ها مشابه شدند گامهای کوچکتری بر میدارد.
- GA شبیه جستجوی پرتو اتفاقی با ترکیب اکتشاف اتفاقی و تبادل اطلاعات میان رشته های موازی بالاتر میرود.
- مزیت اصلی GA عمل پیوند میباشد.

الگوریتم های ژنتیک:شبه کد

1. Random initial population (popsize)
2. for i=2 to maxgen do
   - crossover (Pc) نرخ برش
   - mutation (Pm) نرخ جهش
   - selection (Ff) تابع برازش

الگوریتم های ژنتیک

- به حاصل یک crossover, mutation, selection نسل جدید میگویند.
- در مسائل بهینه سازی بهترین نفر از جمعیت آخر به عنوان جواب برگردانده میشود.
- Crossover میتواند چند نقطه ای نیز باشد. یعنی برش و اتصال از بیش از یک نقطه صورت گیرد.
- جهش میتواند تصادفی نباشد. یعنی کپی از فرد دیگر باشد یا با چرخ رولت یا با تورنمنت انتخاب گردد.

۲۰۱

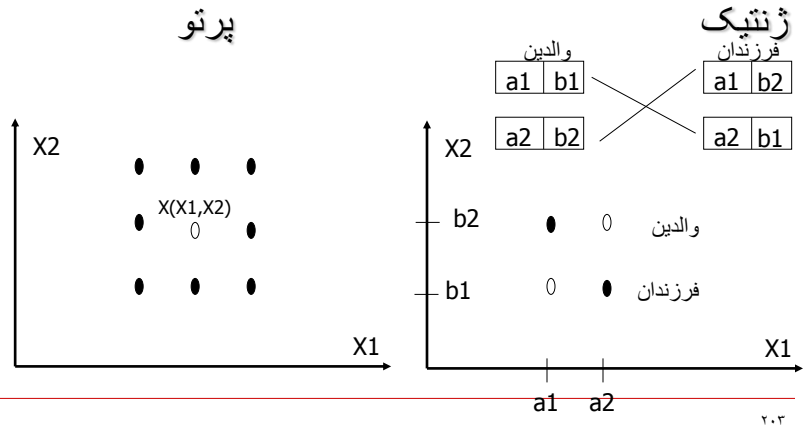
الگوریتم های ژنتیک و جستجوی فضای حالت

- GA حالت خاصی از جستجوی پرتو اتفاقی است که bk فرزند توسط crossover تولید میشود و N فرزند تصادفی نیز توسط mutation ایجاد میگردد.
- GA فضای حالت را به شکل گسسته بررسی میکند.
- برای بررسی پیوسته فضای حالت نیاز به PSO داریم.

۲۰۲

الگوریتم ژنتیک :مثال

- فضای دو بعدی گسسته:



۲۰۳

بهینه سازی گروهی ذرات(PSO)

- مانند الگوریتم GA تعدادی جمعیت دارد ولی بر خلاف آن اقدام به تولید جمعیت جدید نمیکند (تا ضعیف ها از بین بروند) بلکه اقدام به حرکت دادن جمعیت ها میکند.

- جمعیت یا (swarm) به سمت بهترین فرد یا (partial) حرکت میکنند.

۲۰٤

بهینه سازی گروهی ذرات(PSO)

- ممکن است افراد به دو سمت نیز حرکت کنند:
  - Global best:بهترین نقطه که همه به آن رسیده اند.
  - personal best: بهترین نقطه که خود به آن رسیده اند.

- ممکن است وقتی تعداد افراد زیاد است به گروههای مختلف تقسیم کنیم . در اینجا سه حرکت خواهیم داشت:
  - Global best:بهترین نقطه که همه به آن رسیده اند.
  - Personal best:بهترین نقطه که خود به آن رسیده اند.
  - Local best:بهترین نقطه که افراد گروه به آن رسیده اند.

۲۰۵

مقایسه GA با PSO

- هر دو بهینه سازی یک هدفه هستند.

- GA فضای حالت را گسسته بررسی میکند ولی PSO پیوسته.

- در PSO نقاط بوجود آمده از بین نمیروند بلکه به سمت بهترین نقطه متمایل میشود.

۲۰۶

بهینه سازی چند هدفه(MOO)

- زمانی مطرح است که بیش از یک (معمولاً دو) هدف برای بهینه سازی مطرح باشد.

- هر دو میتواند maximization باشد.
  مانند کارخانه: بیشترین استفاده از خط تولید و بیشترین سود.
- هر دو میتواند minimization باشد.
  مانند سوئیچ شبکه: کمترین ترافیک و کمترین فاصله.
- میتواند ترکیبی باشد.
  مانند خرید: بیشترین کیفیت و کمترین قیمت.

۲۰۷

فصل پنجم

مسائل

ارضای محدودیت

۲۰۸

مسائل ارضای محدودیت(CSP)

- در دو فصل قبلی به روشهای جستجو در فضای حالات پرداخته شد.
- این روشها حالات را یک جعبه سیاه میبینند که ساختار داخلی آنها قابل مشاهده و تشخیص نمیباشد.
- حالات تنها بوسیله روالهای مختص مساله یعنی تابع پسین، تابع هیوریستیک و آزمون هدف قابل دستیابی هستند.

۲۰۹

مسائل ارضای محدودیت(CSP)

- در این فصل به مسائل ارضای محدودیت پرداخته میشود که حالات و آزمون هدف آنها ساختیافته، استاندارد و دارای بازنمایی بسیار ساده ای هستند.
- الگوریتمهای جستجو طوری تعریف میشوند که بتوانند نهایت استفاده را از ساختار مساله و حالات آن ببرند.
- بجای استفاده از روشهای خاص، از روشهای کلی استفاده میکنند که قابلیت حل مسائل بزرگتری را دارند.
- با بازنمایی استاندارد آزمون هدف، ساختار مساله جهت ایجاد روشهایی برای تجزیه مساله و درک روابط میان ساختار آنها روشن میشود.

۲۱۰

مسائل ارضای محدودیت(CSP)

- این مسائل شامل متغیرهایی دارای محدودیت هستند.
- بسیاری از مسائل دنیای واقعی میتوانند بعنوان CSP در نظر گرفته شوند.
- ساختار یک CSP توسط گراف محدودیت نشان داده میشود.
- جستجوی عقبگرد که نوعی جستجوی اول عمق است معمولاً برای حل CSP استفاده میشود.
- جستجوی محلی با استفاده از هیوریستیک حداقل تناقضات در حل مسائل ارضای محدودیت بسیار مفید است.

فصل ششم

جستجوی تخاصمی

بازیها

- بازیها حالتی از محیطهای چند عاملي هستند که:
  - هر عامل نیاز به در نظر گرفتن فعالیت سایر عاملها و چگونگی تأثیر آنها دارد.
  - رفتارهای غیر قابل پیش بینی عاملهای دیگر میتواند باعث بروز مقتضیات بسیاری در فرایند حل مساله شود.
- اثر هر عامل بر عاملهای دیگر معنا دار است.
- بین محیطهای چند عاملی رقابتي و همکار تمایز وجود دارد:
  - محیطهای رقابتی، که در آنها اهداف عاملها با یکدیگر برخورد دارند، منجر به مسائل تخاصمی میشود که به عنوان بازی شناخته میشوند.

۲۱۳

بازیها

- برخلاف بیشتر مسائل سرگرمی، بازیها بسیار سخت حل میشوند و بدان دلیل جذابیت دارند.
- هرگونه ناکارآمدی در بازی جریمه سختی خواهد داشت.
- در این بحث، محیط معین و کاملاً رویت پذیر میباشد که در آن دو عامل بصورت نوبتی عمل میکنند.
- وجود تضاد در توابع امتیاز عاملها باعث تخاصمی شدن شرایط میشود.

۲۱٤

بازیها: مثال بازی شطرنج

- دارای میانگین فاکتور انشعاب ۳۵ میباشد.
- هر بازیکن در طول بازی حدود ۵۰ حرکت انجام میدهد.
- درخت جستجو حدود $35^{100} \approx 10^{154}$ گره خواهد داشت.
- گراف جستجو دارای 10^{40} گره مجزا میباشد.

بازی بعنوان یک مساله جستجو

- حالت اولیه: شامل موقعیت اولیه بازی و شخص شروع کننده میباشد.
- تابع پسین: لیستی از (حالت,حرکت) که معرف یک حرکت معتبر است.
- آزمون هدف: مشخص میکند که پایان بازی چه موقع است؟ پایان بازی که به آن حالتهای پایانی گویند.
- تابع سودمندی: برای هر حالت پایانه یک مقدار عددی را ارائه میکند. مثلاً برنده(+۱) ،بازنده(-۱)، مساوی(۰).

بازی بعنوان یک مساله جستجو

- عمدتاً مجموع امتیازات در بازیها صفر خواهد بود.
- حالت اولیه و حرکات مجاز برای هر بازیکن، درخت بازی را برای آن بازی ایجاد میکند.
- بطور کلي يک بازي با دو بازيکن را در نظر مي‌گيريم که آن را MAX-MIN مي‌ناميم.
- MAX شروع کننده بازی است و سپس طرفین به نوبت بازی میکنند تا بازی خاتمه یابد.

۲۱۷

بازیها: مثال دوز

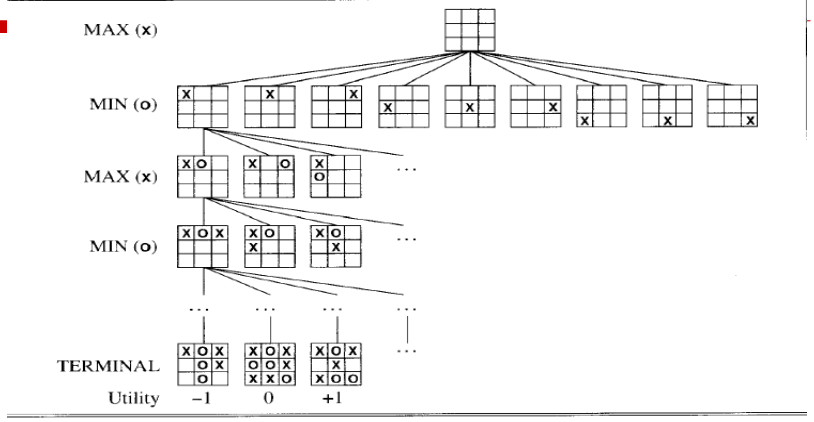


Figure 6.1 A (partial) search tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the search tree, giving alternating moves by MIN (O) and MAX, until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

۲۱۸

بازیها: مثال دوز

- MAX در اولین حرکت، ۹ انتخاب دارد.
- در هر حرکت به نوبت MAX یک X و MIN یک O در خانه مد نظر خود قرار میدهند تا به حالت نهایی برسیم.
- اعداد نوشته شده زیر هر گره پایانی، مبین سودمندی از دیدگاه MAX میباشد.
- مقادیر بزرگ برای MAX مناسب اند و برای MIN نامناسب. (مقادیر کوچک برای MIN مناسب اند) دلیل MAX-MIN

تصمیمات بهینه در بازیها

- در یک مساله جستجوی عادی، راه حل بهینه ترکیبی از حالاتی است که در نهایت منتهی به هدف گردد.
- در یک بازی، از آنجاییکه MIN نیز یک طرف بازی است MAX باید یک راهبرد اقتضایی تشکیل دهد تا علاوه بر حرکت اولیه در شروع بازی، تمامی حرکات ممکن بعدی تحت تاثیر MIN را مشخص کند(در نظر گیرد).
- یک راهبرد بهینه باید منجر به نتایجی گردد که هنگام بازی با یک حریف بدون اشتباه، حداقل با هر راهبرد دیگری برابری کند.

الگوريتم MINIMAX

- MAX بايد يک استراتژي بيابد که به يک حالت پاياني برنده (بدون توجه به عملکرد MIN) منجر شود. اين استراتژي شامل حرکات درست براي MAX به ازاء هر حرکت ممکن از MIN مي‌باشد.

- الگوريتم MINMAX به منظور تعيين استراتژي بهينه براي MAX طراحي شده است و از اينرو مي‌توان بهترين حرکت را تصميم‌گيري کرد.
- اين الگوريتم شامل ۴ مرحله است:

مراحل الگوريتم MINIMAX

1. توليد درخت کامل بازي، تمام راه تا مراحل پاياني.
2. درخواست تابع سودمندي براي هر حالت پاياني.
3. استفاده از سودمندي حالات پاياني به منظور تعيين سودمندي گره‌ها يک مرحله بالاتر و ... .
4. MAX حرکتي را انتخاب مي‌کند که به بالاترين مقدار منتهي مي‌شود.

الگوریتم MINIMAX

- مقدار بیشینه کمینه از رابطه زیر محاسبه میشود:

$$\text{MINIMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node.} \end{cases}$$

الگوریتم MINIMAX:مثال

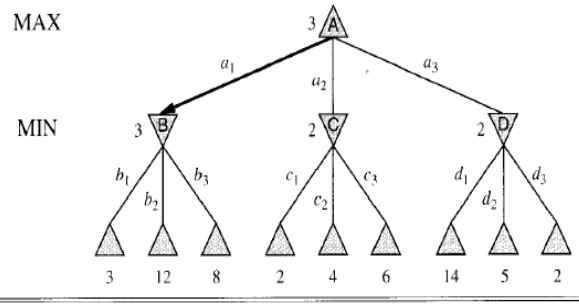


Figure 6.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is a_1, because it leads to the successor with the highest minimax value, and MIN's best reply is b_1, because it leads to the successor with the lowest minimax value.

الگوریتم MINIMAX:مثال

- یک بازی ویژه که با انجام یک حرکت از MAX و یک حرکت از MIN به اتمام میرسد.
- در اصطلاح بازی، این درخت دارای عمق حرکت یک است. که از دو نیم حرکت تشکیل شده و هر یک، یک لایه نام دارند.
- در این شکل گرههای MAX با Δ و گرههای MIN با ▽ نشان داده شده اند.

الگوریتم MINIMAX

- با این تعریف بهینه از بازی، MAX فرض میکند که MIN نیز بصورت بهینه بازی میکند.
- اگر MIN بهینه بازی نکند، چه اتفاقی می افتد؟
  - در این حالت نیز MAX بهتر عمل کرده است.
- ممکن است راهبردهای دیگری موجود باشند که در مقابل حریفان غیر بهینه از MINIMAX بهتر عمل کنند. ولی مسلماً در مقابل حریفان بهینه، بدتر عمل خواهند کرد.

الگوریتم MINIMAX:کارایی

- کامل بودن: بله (اگر درخت محدود باشد)
- بهینگي: بله
- پیچیدگي زمانی: $O(b^m)$
- پیچیدگی فضا: $O(bm)$

الگوریتم MINIMAX یک جستجو عمقي در درخت بازی است.

۲۲۷

تصمیمات بهینه در بازیهای چند نفره

- در این نوع بازیها حالات و انتخابهای بیشتری وجود دارد و به جای یک مقدار، یک بردار به هر گره تخصیص داده میشود.
- در حین بازی اتحادهای رسمی و غیر رسمی میان بازیکنان تشکیل یا شکسته میشود.
- اگر مجموع امتیازات بازی صفر نباشد ممکن است یک همکاری بین دو بازیکن تا انتها صورت گیرد.(برای رسیدن به هدف مشترک)

۲۲۸

بازیهای چند نفره:مثال

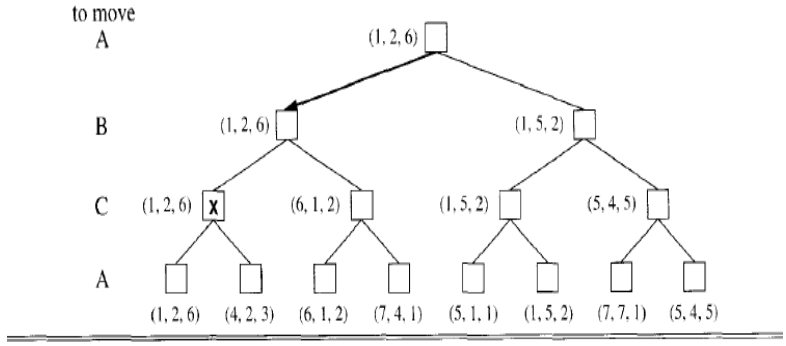


Figure 6.4 The first three ply of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

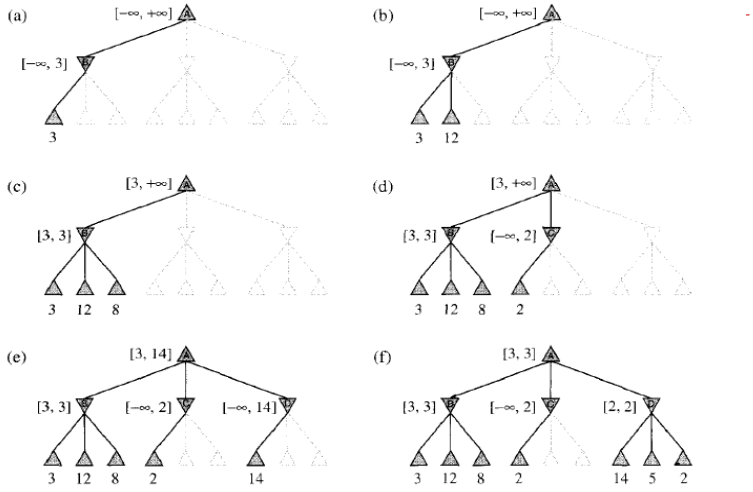
۲۲۹

هرس آلفا بتا

- مشکل الگوریتم MINIMAX نمایی بودن تعداد حالات مورد بررسی در حین بازی میباشد.
- میتوان بدون احتیاج به دیدن تمام گره های درخت جستجو، بیشینه-کمینه صحیح را محاسبه نمود.
- برای این منظور با ایده هرس کردن، شاخه هایی از درخت جستجوکه درتصمیم گیری نهایی تاثیری ندارندهرس میشوند.

۲۳۰

هرس آلفا بتا:مثال



۲۳۱

هرس آلفا بتا

- مقدار گره ریشه به کمک رابطه زیر بدست آمده است:

$$\begin{aligned}\text{MINIMAX-VALUE}(root) &= \max(\min(3,12,8),\min(2,x,y),\min(14,5,2))\\ &= \max(3,\min(2,x,y),2)\\ &= \max(3,z,2) \qquad \text{where } z \le 2\\ &= 3.\end{aligned}$$

- مقدار گره ریشه (انتخاب بیشینه-کمینه) مستقل از مقادیر برگهای هرس شده x و y میباشد.

۲۳۲

هرس آلفا بتا

- هرس آلفا بتا را میتوان بر روی درختهایی با هر عمق اعمال کرد و در برخی موارد میتوان بجای برگها، زیر شاخه های اصلی را نیز هرس نمود.

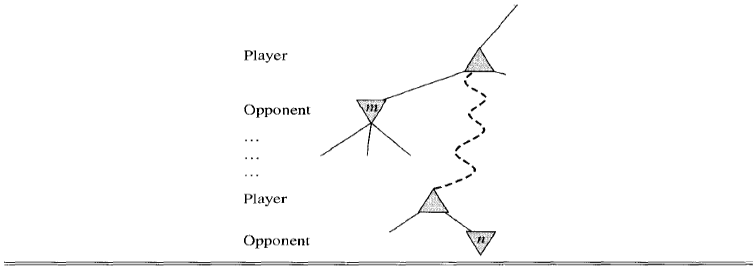


Figure 6.6 Alpha–beta pruning: the general case. If m is better than n for Player, we will never get to n in play.

۲۳۳

هرس آلفا بتا

- این روش از دو پارامتر آلفا و بتا استفاده میکند که مشخص کننده کران مقادیر نگهداری شده در طول مسیر هستند:

- آلفا: مقدار بهترین انتخاب ممکن (بالاترین) در طول مسیر برای MAX که در طول مسیر بدست آمده است.

- بتا: مقدار بهترین انتخاب ممکن (کمترین) در طول مسیر برای MIN که در طول مسیر بدست آمده است.

۲۳٤

هرس آلفا بتا

- چون جستجوی بیشینه-کمینه یک جستجوی اول عمق است، درهر زمان باید گره های در راستای یک مسیر از درخت بازی، مورد توجه قرار گیرند.

- در جستجوی آلفا-بتا، مقدار α و β با پیشروی در درخت بازی به روز درمي‌آید و به محض اینکه مقدار گره جاری به ترتیب بدتر از مقادیر فعلی α و β برای MIN و MAX تشخیص داده شد شاخه های باقیمانده آن گره هرس میشود.

هرس آلفا بتا:کارایی

- بسیار وابسته به ترتیب پسینها است.(مثال قبلی شاخه D)

- اگر بتوان ابتدا پسینهایی که احتمال دارد بهترین باشند را انتخاب کرد آنگاه این روش بجای $O(b^d)$ الگوریتم بیشینه-کمینه قادر خواهد بود با $O(b^{d/2})$بررسی، بهترین حرکت را انتخاب کند.

- فاکتور انشعاب موثر در این روش بجای b برابر با $\sqrt{b}$ خواهد بود.
- این روش در زمانی برابر تا دو برابر روش بیشینه-کمینه پیش بینی میکند.

تصمیمهای بلادرنگ ناقص

- الگوریتم minimax کل فضای جستجوی بازی را تولید میکند.
- الگوریتم آلفا-بتا با وجود هرس بخش وسیعی از درخت، کل مسیر حالتهای پایانی را حداقل برای بخشي از فضای حالت باید جستجو کند.
- این عمق جستجو معمولاً عملي نیست زیرا حرکات باید در زمانی معقول انجام شود.
- میتوان بجای استراتژی (فکر کامل،حرکت کامل) یعنی رسم کامل درخت بازی، سراغ استراتژی (کمی تفکر،یک حرکت) رفت.
- به اینکار تفکر حین بازی نیز میگویند.

۲۳۷

نظریه شانون

- برنامه ها باید زودتر از زمان معمول قطع شوند و حین جستجو با اعمال توابع ارزیاب هیوریستیک به حالات، بطور موثر گره های غیر انتهایی را به برگ تبدیل کرد.
- پس الگوریتم های بیشینه-کمینه یا آلفا بتا به شکل زیر تغییر کنند:

- تابع سودمندی با یک تابع ارزیاب هیوریستیک EVAL که تخمینی از میزان سودمندی موقعیت ایجاد میکند، جایگزین گردد.
- آزمون پایانی با یک آزمون قطع CUTOFF-TEST جایگزین گردد که تشخیص دهد چه زمانی باید EVAL اعمال گردد.

۲۳۸

توابع ارزیاب

- تابع ارزیابی، تخمینی از سودمندی مورد انتظار بازی ازیک موقعیت خاص میباشد. مانند توابع اکتشافی، تخمینی از فاصله تا هدف را بر میگرداندند.

1. تابع ارزيابي با تابع سودمندي در مورد حالت پاياني بايد به توافق برسند.
2. نبايد زياد طول بكشد.
3. تابع ارزيابي بايد به درستي شانس‌هاي واقعي براي برد را منعكس كند.

آزمون قطع

- مرحله دوم بهینه سازی جستجوی آلفا بتا به ترتیبی است که باید در زمان مناسب جستجو را قطع و تابع EVAL را فراخوانی نماید.

- صريح‌ترين رهيافت براي كنترل ميزان جستجو قراردادن محدوديتي براي داشتن يک عمق ثابت است، بنابراين تست قطع براي تمام گره‌ها در زير عمق d موفق مي‌شود.

- عمق طوري انتخاب مي‌شود كه ميزان زمان استفاده شده از آنچه كه قوانين بازي اجازه مي‌دهد تجاوز نكند.

بازیهای جایزه دار

- در این بازیها درخت از حالت MAX-MIN خارج میشود.
- در حقیقت چند مرحله MAX و چند مرحله MIN انجام میگیرد.
- کلیات بحث الگوریتم MINIMAX اینجا نیز صادق خواهد بود.

بازیهای دارای عامل شانس

- در برخی مسائل حوادث غیر قابل پیش بینی زیادی پیش می آید که باعث ایجاد شرایطی پیش بینی نشده میگردد.
- در بسیاری از بازیها این واقعیت با افزودن یک عنصر احتمالی مانند پرتاب تاس اعمال میشود.
- در این بازیها بازیکن هیچ آگاهی از حرکت حریف ندارد چون پرتاب تاس آنرا معین خواهد کرد.

بازیهای دارای عامل شانس

- در این بازیها نمیتوان درخت بازی استاندارد را ساخت.
- در این شرایط درخت بازی علاوه بر گره های MAX و MIN دارای گره های شانس نیز میباشد.
- گره های نشانگر عامل شانس را در این درخت ها با دایره نشان میدهیم.

بازیهای دارای عامل شانس

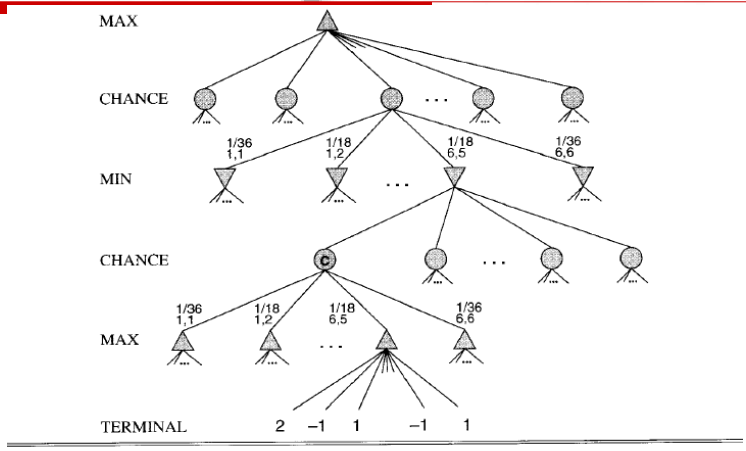


Figure 6.11 Schematic game tree for a backgammon position.

بازیهای دارای عامل شانس

- در این درخت موقعیت ها دارای مقادیر بیشینه کمینه مشخصی نیستند.
- میتوان مقادیر بیشینه کمینه مورد انتظار را برای احتمالهای مختلف پرتاب تاس ها در یک گره محاسبه کرد.
- بدین شکل مقادیر بیشینه کمینه در بازیهای معین، به مقدار بیشینه کمینه مورد انتظار در بازیهای دارای شاس می انجامد.
- گره های شانس با محاسبه میانگین وزنی مقادیر حاصل از تمامی حالتهای ممکن با فرمول ذیل ارزیابی میشود:

بازیهای دارای عامل شانس

$$\text{EXPECTIMINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in Successors(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a chance node} \end{cases}$$

- P(S) احتمال وقوع پرتاب تاس میباشد.

- در این رابطه تابع جایگزین گره شانس n، حالت n را با تمامی پرتاب های ممکن تاس ها گسترش میدهد تا تمامی جایگزین ها مانند S را تولید کند.

بازیهای دارای عامل شانس:ارزیابی

- چون روش بیشینه کمینه مورد انتظار، تمامی توالیهای ممکن پرتاب تاسها را نیز در نظر میگیرد راه حل در $O(b^m n^m)$ تکرار حاصل خواهد شد.
- n، تعداد حالات مختلف پرتاب تاسها میباشد.
- با اعمال تغییراتی در چیدمان مقادیر ارزیابی، رفتار الگوریتم متفاوت خواهد بود. برای اجتناب از این حساسیت، بوسیله تکنیکهایی باید تابع ارزیاب تغییر کند. (ورود عدم قطعیت)

۲٤۷

بازیهای دارای عامل شانس:مثال

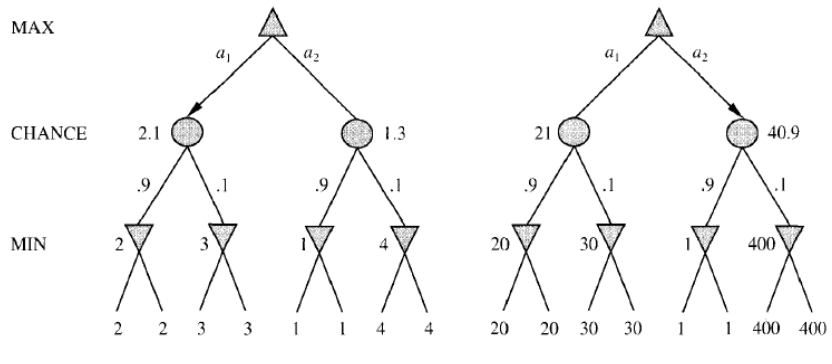


Figure 6.12 An order-preserving transformation on leaf values changes the best move.

۲٤۸

٢٤٩

amirjalili.ir

٢٥٠

۲۵۱

amirjalili.ir

۲۵۲