هوش مصنوعی
معرفی

فصل اول
سید ناصر رضوی
Email: razavi@Comp.iust.ac.ir
۱۳۸۶

مراجع و منابع

- هوش مصنوعی: رهیافت نوین (S. Russell & P. Norvig)
- برنامه نویسی پرولوگ برای هوش مصنوعی (I. Bratko)
- هوش مصنوعی (E. Rich)
- هوش مصنوعی (Luger)

وظایف و مسولیت ها

- **مربی**
  - رفتار محترمانه با دانشجو یان
  - رفتار منصفانه
  - در دسترس بودن برای پاسخگویی
  - آموزش آنچه که دانشجو باید بداند (رعایت سرفصل)
  - بیان نحوه ارزیابی
- **دانشجو**
  - رفتار محترمانه
  - اجتناب از کلک زدن، دروغ گویی و سرقت آثار دیگران و یا کمک به افراد دیگر در انجام چنین کارهایی
  - عدم ایجاد مزاحمت و مداخله در فعالیت های دانشگاهی دیگران
- **عواقب: کاهش نمره و یا مردود شدن در درس**

نحوه ارزیابی

- سه الی چهار تمرین و کوئیز (10%)
- دو پروژه
  - پرولوگ (5%)
  - جستجو (15%)
- دو امتحان
  - میان ترم (30%)
  - پایان ترم (40%)
- نحوه ارزیابی تمرین ها و پروژه های معوقه:
  - تأخیر حداکثر یک هفته : ۲۰ درصد جریمه
  - تأخیر بیش از یک هفته : صفر

سرفصل

- مرور درس
- هوش مصنوعی چست؟
- یک تاریخچه مختصر
- وضعیت فعلی هوش مصنوعی

مرور درس

- فصل اول: معرفی
- فصل دوم: عامل ها (agents)
- فصل سوم و چهارم و پنجم و ششم: جستجو
- فصل هفتم و هشتم و نهم: منطق

هوش مصنوعی چیست؟

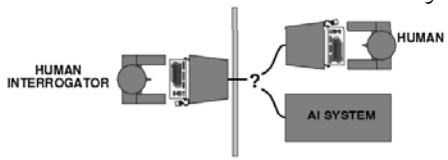
- تعاریف هوش مصنوعی در چهار دسته قرار می گیرند

انسان گونه فکر کردن	منطقی فکر کردن
انسان گونه عمل کردن	منطقی عمل کردن

- کتاب درسی طرفدار «منطقی عمل کردن»

انسان گونه رفتار کردن: تست تورینگ

- تورینگ (۱۹۵۰) ‹ماشین های محاسباتی و هوشمندی›
- ‹آیا ماشین ها می توانند فکر کنند؟› ← ‹آیا ماشین ها می توانند هوشمندانه عمل کنند؟›
- یک آزمون عملی برای رفتار هوشمندانه



- وی پیش بینی نمود که تا سال ۲۰۰۰ ماشین ها تا ۳۰ درصد شانس فریب دادن یک انسان عامی را خواهند داشت.
  - قابلیت های مورد نیاز کامپیوتر
    - پردازش زبان طبیعی
    - بازنمایی دانش
    - استدلال خودکار
    - یادگیری

انسان گونه فکر کردن: مدلسازی شناختی

- درک چگونگی تفکر انسانی و عملکرد مغز
  - درون گرایی
  - تجارب روانشناسی
- به دنبال ایجاد تئوری دقیقی درباره عملکرد ذهن انسان و تبدیل آن به برنامه کامپیوتری
  - GPS: سیمون و نیوول، ۱۹۶۳
    - در پی تعقیب مراحل استدلال برنامه و مقایسه آن با مراحل حل مسائل توسط انسان
    - آنالیز هدف– وسیله

منطقی فکر کردن: ‹ قوانین تفکر›

- ارسطو: « فرآیند استدلال/تفکر درست چیست؟»
- مثال: "سقراط انسان است، تمام انسانها فانی هستند، پس سقراط فانی است."
- پایه ریزی **منطق** (Logic)
- برنامه هایی براساس قوانین تفکر برای ایجاد سیستمهای هوشمند
- موانع اصلی
  - دریافت دانش غیررسمی و تبدیل آن به دانش رسمی : «اکثر انسانها پرتلاش هستند.»
  - تفاوت میان قادر به حل مسأله بودن در تئوری و در عمل (بن بست محاسباتی)

منطقی عمل کردن: عامل منطقی

- رفتار منطقی: انجام عمل درست
- عمل درست: عملی که با توجه به اطلاعات موجود، انتظار می رود شانس رسیدن به هدف را به حداکثر برساند .
- لزوماً شامل تفکر نمی باشد – مانند پلک زدن – اما تفکر باید در خدمت عمل منطقی باشد.

عامل منطقی

- عامل: هر چیزی که قادر به درک نمودن و عمل کردن باشد.
- این درس در مورد طراحی عامل های منطقی می باشد.
- به طور انتزاعی، عامل یک تابع از تاریخچه ادراکی بر روی اعمال می باشد:

$$[f: \mathcal{P}^* \rightarrow \mathcal{A}]$$

- ما برای هر دسته از محیط ها و وظایف مختلف، به دنبال عاملی (یا دسته ای از عامل ها) با بهترین کارایی می باشیم.
- هشدار: محدودیت های محاسباتی باعث شده اند که منطقی بودن به طور کامل، غیر قابل دسترس باشد.

← طراحی بهترین برنامه برای منابع ماشینی داده شده

علوم زیربنایی هوش مصنوعی

•فلسفه	منطق، روش های استدلال، ذهن به عنوان سیستم فیزیکی
	زیربنایی در یادگیری، زبان، منطقی بودن
•ریاضیات	روش های بازنمایی رسمی و الگوریتم های اثبات،
	محاسبات، تصمیم(نا) پذیری، احتمالات
•اقتصاد	بهره وری، نظریه تصمیم
•عصب شناسی	مواد فیزیکی برای فعالیت های ذهنی
•روان شناسی	پدیده ادراک و کنترل، تکنیک های آزمایشگاهی
•مهندسی کامپیوتر	ایجاد کامپیوترهای سریع
•نظریه کنترل	طراحی سیستم هایی به منظور بیشینه سازی یک تابع
	هدف در طول زمان
•زبان شناسی	بازنمایی دانش، گرامر

تاریخچه مختصر هوش مصنوعی

۱۹۴۳	مک کالچ و پیتز: مدل مداری بولی از ذهن
۱۹۵۰	تورینگ: « ماشین های محاسباتی و هوشمندی»
۱۹۵۶	نشت در دارتموث: پیدایش «هوش مصنوعی»
دهه ۵۰	اولین برنامه های هوش مصنوعی شامل: برنامه ساموئل برای انجام بازی چکر، برنامه Logic Theorist سیمون و نیوول و ...
۱۹۶۵	الگوریتم کامل رابینسون برای استدلال منطقی
۷۳-۱۹۶۶	پیدایش نظریه پیچیدگی محاسباتی، توقف تحقیقات بر روی شبکه های عصبی
۷۹-۱۹۶۹	پیدایش سیستم های اولیه مبتنی بر دانش

تاریخچه مختصر هوش مصنوعی (ادامه)

۱۹۸۰	ورود هوش مصنوعی در عرصه های صنعتی
۱۹۸۶	محبوبیت مجدد شبکه های عصبی
۱۹۸۷	تبدیل شدن هوش مصنوعی به یک علم
۱۹۹۵	ظهور عامل های هوشمند

وضعیت فعلی هوش مصنوعی

- شکست قهرمان شطرنج (گری کاسپاروف) توسط Deep Blue در سال ۱۹۹۷
- اثبات یک حدس ریاضیاتی (حدس رابینز) که برای چندین دهه به صورت حل نشده باقی مانده بود.
- کنترل خودکار (هدایت اتوماتیک یک اتومبیل از پیتزبرگ تا سن دیه گو در ۹۸٪ از مسیر – سیستم بینایی ALVINN
- برنامه های زمان بندی خودکار ناسا
- حل بهتر جداول کلمات متقاطع توسط PROVERB نسبت به انسان
- سیستم های خبره پزشکی
- ...

عامل های هوشمند

فصل دوم
سید ناصر رضوی
Email: razavi@Comp.iust.ac.ir
۱۳۸۴

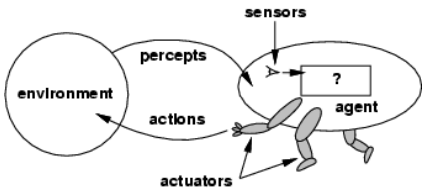
رئوس مطالب

- عامل ها و محیط ها
- منطقی بودن
- PEAS(معیار کارآیی، محیط، اثرکننده ها، حسگرها)
- انواع محیط ها
- انواع عامل ها

عامل ها

- عامل: هر چیزی که بتواند محیط پیرامونش را از طریق حسگرها درک کند و در آن محیط از طریق اثرکننده ها عمل کند.
- عامل انسانی:
  - چشم ها و گوش ها و سایر اندام های حسی به عنوان حسگرها
  - دست ها، پاها، دهان و سایر اعضای بدن به عنوان اثر کننده ها
- عامل روبات:
  - دوربین ها و فاصله یاب مادون قرمز به عنوان حسگرها
  - انواع موتورها به عنوان اثر کننده ها

عامل ها و محیط ها

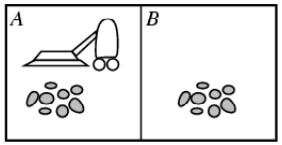


- تابع عامل تاریخچه ادراکی را به اعمال نگاشت می کند:

$$[f: \mathcal{P}^* \rightarrow \mathcal{A}]$$

- برنامه عامل برای ایجاد f بر روی معماری فیزیکی اجرا می شود.
- عامل = معماری + برنامه

دنیای جارو برقی



- ادراک ها: مکان ها و محتویات آنها، مانند $[A, Dirty]$
- اعمال: حرکت به چپ و راست، مکش و *NoOp*

عامل های منطقی

- یک عامل باید بر اساس آنچه که می تواند درک کند و اعمالی که می تواند انجام دهد، «کار درست را انجام دهد». عمل درست آن است که باعث شود عامل بیشترین موفقیت را بدست آورد.
- معیار کارآیی: یک معیار هدف برای سنجش میزان موفقیت رفتار یک عامل
- مثال: معیار موفقیت عامل دنیای جارو برقی:
  - مقدار گرد و خاک تمیز شده
  - میزان زمان مصرف شده
  - مقدار برق مصرف شده
  - میزان سر و صدای تولید شده و ...

عامل های منطقی

- عامل منطقی: برای هر دنباله ادراکی ممکن، یک عامل منطقی باید بر اساس شواهد دریافتی از دنباله ادراکی و دانش درونی عملی را انتخاب کند که انتظار می رود معیار کارآیی اش را به حداکثر برساند.

عامل های منطقی

- منطقی بودن با دانش کل بودن (دانستن همه چیز توسط دانش نامحدود) تفاوت دارد.
- عامل می تواند اعمالی را انجام دهد که از طریق تغییر در ادراک های آتی اطلاعات مفید بدست آورد (جمع آوری دانش، اکتشاف)
- یک عامل خودمختار است اگر رفتارش بر اساس تجربه اش تعیین شود (به همراه قابلیت یادگیری و تطبیق پذیری)

PEAS

- PEAS: Performance measure, Environment, Actuators, Sensors
- در طراحی یک عامل ابتدا باید موارد بالا تعیین گردند.
- مثال: طراحی یک راننده تاکسی اتوماتیک
  - معیار کارآیی: امنیت، سرعت، راحتی، سود و ...
  - محیط: خیابان ها، افراد پیاده، مشتری ها و ...
  - اثرکننده ها: فرمان، شتاب دهنده، ترمزها، بوق، چراغ ها و ...
  - حسگرها: دوربین ها، حسگرهای صوتی (Sonar)، سرعت سنج، GPS، کیلومتر شمار، حسگرهای موتور، صفحه کلید، میکروفون و ...

PEAS

- عامل: سیستم تشخیص پزشکی
  - معیار کارآیی: سلامتی بیمار، به حداقل رساندن هزینه و ...
  - محیط: بیمار، بیمارستان، کارمندان و ...
  - اثر کننده ها: صفحه نمایش (پرسش ها، آزمایش ها، تشخیص ها، مداوا)
  - حسگرها: صفحه کلید (دریافت علایم، یافته ها و پاسخ های بیمار)

PEAS

- عامل: روبات جابه جا کننده اشیاء
  - معیار کارآیی: درصد قطعاتی که در صندوق درست قرار می گیرند
  - محیط: نوار نقاله و اشیاء روی آن، صندوق ها
  - اثرکننده ها: بازوها و دست
  - حسگرها: دوربین، حسگر زاویه مفاصل

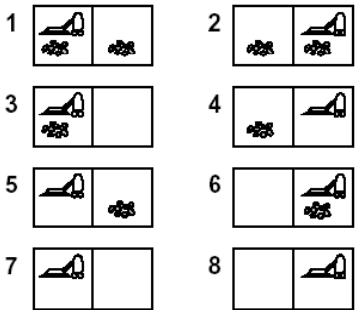
PEAS

- آموزش دهنده زبان به صورت محاوره ای
  - معیار کارآیی: به حداکثر رساندن نمره دانش آموز در امتحان
  - محیط: مجموعه دانش آموزان
  - اثرکننده ها: صفحه نمایش (تمرین ها، پیشنهادات و اصلاحات)
  - حسگرها: صفحه کلید

محیط

- هر محیط دارای مجموعه ای از حالت ها می باشد:
  - محیط در هر لحظه فقط در یکی از این حالت ها می باشد.

- مثال: دنیای مکش



$S = \{1, 2, 3, 4, 5, 6, 7, 8\}$

عامل و محیط

- در لحظه شروع، محیط در یکی از حالت های ممکن می باشد
  - عمل عامل در محیط، باعث تغییر حالت محیط می شود
    - حالت فعلی: S_i
    - عمل عامل: $Action$
    - حالت بعدی: S_j

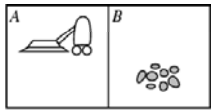
- مثال: دنیای مکش

انواع محیط

- کاملا قابل مشاهده (در مقابل مشاهده پذیر جزئی): محیطی که در آن در هر لحظه از زمان حسگرهای عامل به آن امکان دستیابی به حالت کامل محیط را می دهند.

- مثال: دنیای مکش – حسگرها: [location, status]
  - تشخیص مکان : چپ یا راست
  - تشخیص وضعیت: تمیز یا کثیف

[LEFT, [CLEAN, DIRTY]]



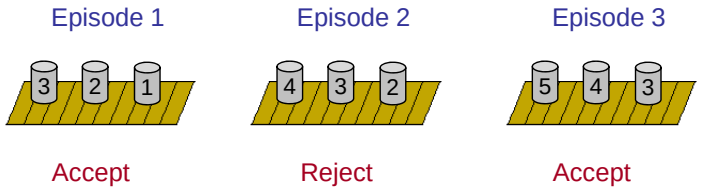
انواع محیط

- قطعی: (در مقابل اتفاقی): حالت بعدی محیط کاملا بوسیله حالت فعلی و عمل انجام شده توسط عامل قابل تعیین می باشد.
  - اگر محیط به جز در مورد عمل عامل های دیگر قطعی باشد، آنگاه محیط استراتژیک می باشد.

انواع محیط

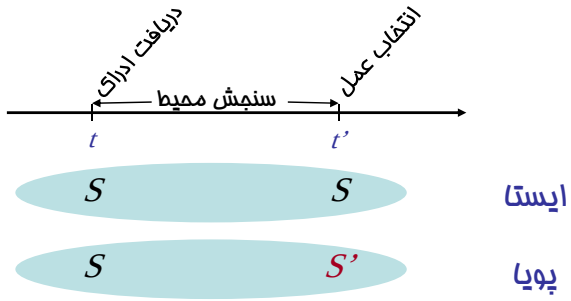
- اپیزودیک (در مقابل ترتیبی): تجربه عامل به «دوره های» غیرقابل تجزیه تقسیم می شود (هر دوره شامل ادراک عامل و سپس انجام یک عمل می باشد) و انتخاب عمل در هر دوره تنها به خود همان دوره بستگی دارد.

- مثال: روبات کنترل کننده کیفیت



انواع محیط

- ایستا (در مقابل پویا): محیط در حین سنجش عامل (برای انتخاب عمل) تغییر نمی کند. اگر خود محیط با گذشت زمان تغییر نکند ولی معیار کارآیی عامل تغییر کند، آنگاه محیط نیمه پویا می باشد.



انواع محیط

- گسسته (در مقابل پیوسته): محیطی که در آن تعداد محدود و متمایزی از درک ها و عمل های کاملا واضح تعریف شده باشد.

- در محیط گسسته، مجموعه حالات محیط یک مجموعه گسسته می باشد و حالات بسادگی قابل تمایز می باشند.
  - مثال: محیط دنیای مکش
    - *State* = {1, 2, …, 8}
    - *Action* = {Left, Right, Suck, NoOp}
    - *Percept* = {[Left, Clean], [Left, Dirty], [Right, Clean], …}

انواع محیط

- تک عاملی (در برابر چند عاملی): یک عامل خودش به تنهایی در محیط عمل می کند.
  - مثال: محیط عامل حل کننده جدول کلمات متقاطع و دنیای مکش

- چند عاملی: تعدادی عامل که با یکدیگر در تعامل می باشند.
  - مثال: شطرنج (رقابتی)، روبوکاپ (بین اعضای یک تیم همیاری و بین اعضای دو تیم رقابتی)، محیط تاکسی خودکار (همیاری جزیی)

انواع محیط

	شطرنج با ساعت	شطرنج بدون ساعت	رانندگی تاکسی
کاملا قابل مشاهده	بله	بله	خیر
قطعی	استراتژیک	استراتژیک	خیر
دوره ای	خیر	خیر	خیر
ایستا	نیمه پویا	بله	خیر
گسسته	بله	بله	خیر
تک عاملی	خیر	خیر	خیر

- نوع محیط به میزان زیادی تعیین کننده طراحی عامل می باشد.
- دنیای واقعی: مشاهده پذیر جزئی، اتفاقی، ترتیبی، پویا، پیوسته و چندعاملی

توابع و برنامه های عامل

- یک عامل کاملا بوسیله تابع عامل مشخص می شود.
  - یادآوری: تابع عامل دنباله ادراکی را به عمل نگاشت می کند.
- یک تابع عامل (یا یک کلاس هم ارزی کوچک) منطقی (rational)می باشد.
- هدف: یافتن روشی به منظور پیاده سازی تابع عامل منطقی به طور مختصر و مفید

عامل مبتنی بر جدول جستجو

- یک روش به منظور توصیف تابع عامل
- نشان دهنده فعالیت مناسب برای هر دنباله ادراکی ممکن
- مثال: جدول دنیای جاروبرقی

Percept Sequence	**Action**
[*A*, *Clean*]	*Right*
[*A*, *Dirty*]	*Suck*
[*B*, *Clean*]	*Left*
[*B*, *Dirty*]	*Suck*
[*A*, *Clean*], [*A*, *Clean*]	*Right*
[*A*, *Clean*], [*A*, *Dirty*]	*Suck*
…	

برنامه عامل مبتنی بر جدول جستجو

function TABLE-DRIVEN-AGENT(*percept*) **returns** an action

static: *percepts*, a sequence, initially empty

table, a table of actions, indexed by percept sequence, initially fully specified

append *percept* to the end of *percepts*

action ← LOOKUP(*percepts*, *table*)

return *action*

عامل مبتنی بر جدول جستجو

- معایب:
  - جدول بسیار عظیم (مثلا در شطرنج 10^{150} سطر)
  - زمان بسیار زیاد برای ایجاد جدول و احتمال بالای خطا
  - عدم خود مختاری
  - حتی با قابلیت یادگیری، نیاز به زمان بسیار زیادی برای یادگیری مداخل جدول دارد.

انواع عامل ها

- چهار نوع اصلی به ترتیب افزایش عمومیت (Generality):
  - عامل های واکنشی ساده (Simple reflex)
  - عامل های واکنشی مبتنی بر مدل (Model-based reflex)
  - عامل های مبتنی بر هدف (Goal-based)
  - عامل های مبتنی بر سودمندی (Utility-based)

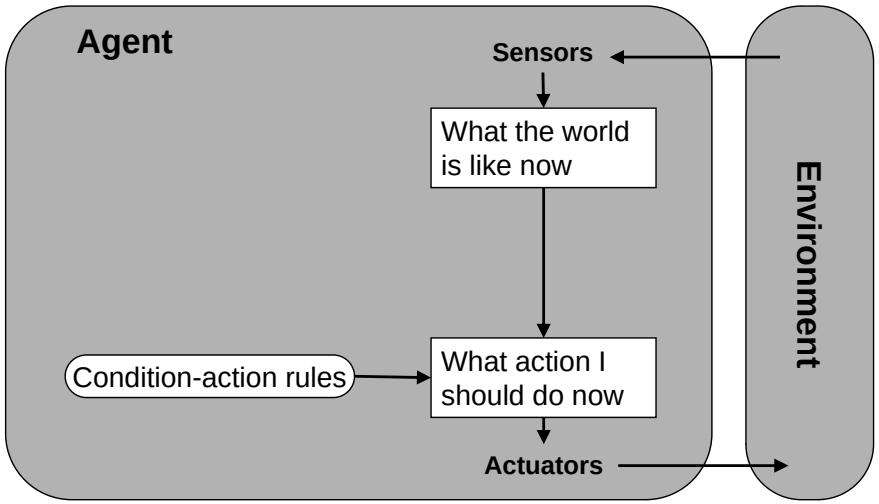
عامل های واکنشی ساده

- ساده ترین نوع عامل
- در هر لحظه، عمل تنها بر اساس درک فعلی انتخاب می شود
- مثال:

```
function REFLEX-VACCUM-AGENT( [location, status]) returns an action

    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

- شامل قوانین شرط–عمل مانند:
  - "اگر چراغ ترمز اتوموبیل جلویی روشن شد، آنگاه ترمز کن"

ساختار عامل های واکنشی ساده



برنامه عامل واکنشی ساده

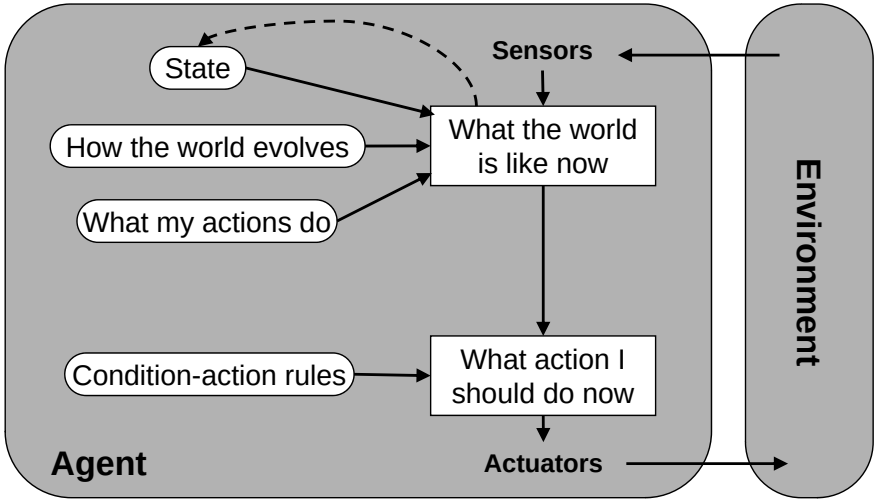
function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
static: *rules*, a set of condition-action rules

state ← INTERPRET-INPUT(*percept*)
rule ← RULE-MATCH(*state*, *rules*)
action ← RULE-ACTION[*rule*]
return *action*

عامل های واکنشی مبتنی بر مدل (حافظه دار)

- عامل واکنشی ساده در صورتی کار می کند که محیط کاملا قابل مشاهده باشد
- اگر محیط مشاهده پذیر جزئی باشد، پیگیری تغییرات دنیا لازم است
- مثال: تاکسی اتوماتیک
- مستلزم دو نوع دانش
  - نحوه تغییر دنیا
  - تاثیر اعمال عامل بر دنیا

عامل های واکنشی مبتنی بر مدل



برنامه عامل های واکنشی مبتنی بر مدل

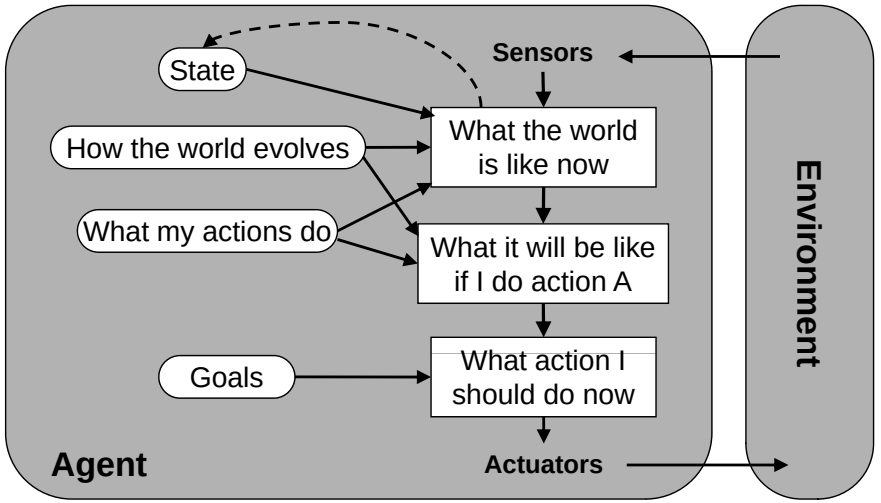
function REFLEX-AGENT-WITH-STATE(*percept*) **returns** an action
static: *state*, a description of the current world state
rules, a set of condition-action rules
action, the most recent action, initially none

state ← UPDATE-STATE(*state*, *action*, *percept*)
rule ← RULE-MATCH(*state*, *rules*)
action ← RULE-ACTION[*rule*]
return *action*

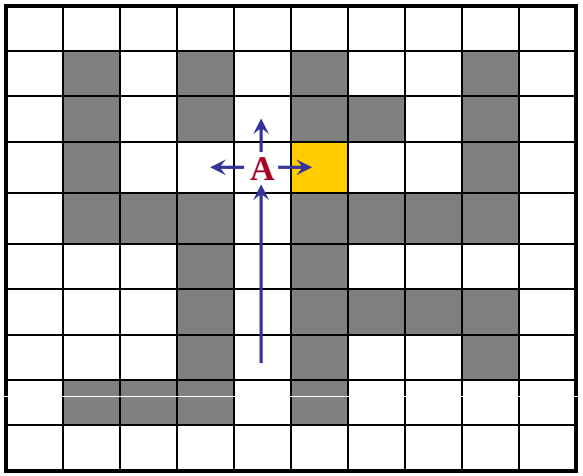
عامل های مبتنی بر هدف

- اطلاعات لازم برای تصمیم گیری در مورد عملی که باید انجام شود:
  - اطلاعات مربوط به حالت فعلی
  - اطلاعات هدف (توصیف موقعیت مطلوب)
  - مثال: عمل مناسب برای تاکسی اتوماتیک در یک چهار راه کدام است؟ (بالا، پایین چپ، راست)
- اگر برای رسیدن به هدف نیاز به چندین عمل باشد
  - جستجو (search)
  - برنامه ریزی (planning)

عامل های مبتنی بر هدف



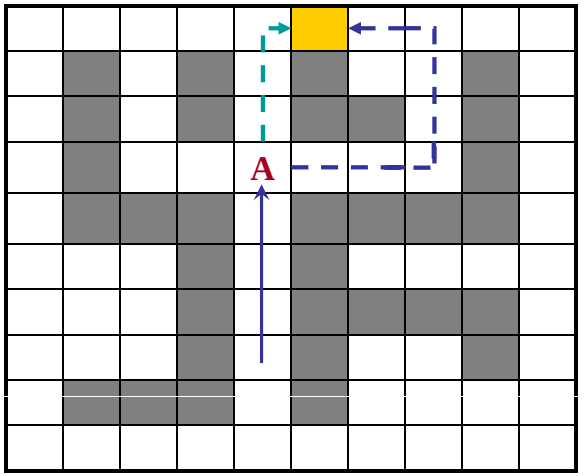
مثال: عامل هدف گرا



مثال: عامل هدف گرا

[UP, UP, UP, RIGHT]

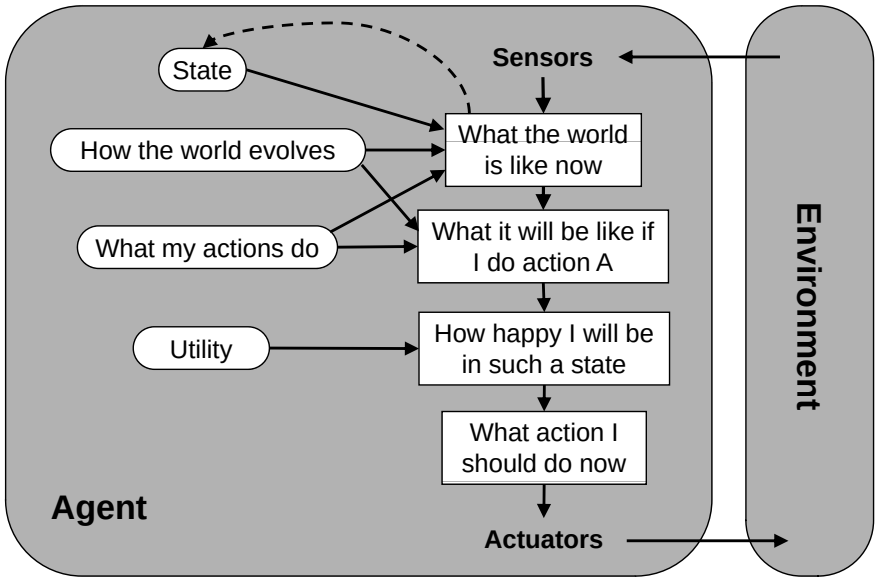
[RIGHT, RIGHT, RIGHT, UP, UP, UP, LEFT, LEFT]



عامل های سودمند

- در بسیاری از محیط ها اهداف برای تولید رفتاری با کیفیت بالا مناسب نیستند
- مثال: تاکسی اتوماتیک
  - ممکن است چندین مسیر برای رسیدن به مقصد موجود باشد، اما بعضی از آنها سریعتر، امن تر، مطمئن تر و یا ارزانتر از بقیه می باشند
- اهداف ملاکی خام برای توصیف وضعیت ها هستند (مطلوب و نامطلوب)
- تابع سودمندی: حالت (یا دنباله ای از حالات) را به یک عدد حقیقی نگاشت می کند که درجه مطلوبیت آن را توصیف می کند
- امکان تصمیم گیری در مواردی که:
  - اهداف متناقض باشند
  - چندین هدف وجود دارد ولی رسیدن به هیچ یک قطعی نیست

عامل های مبتنی بر سودمندی



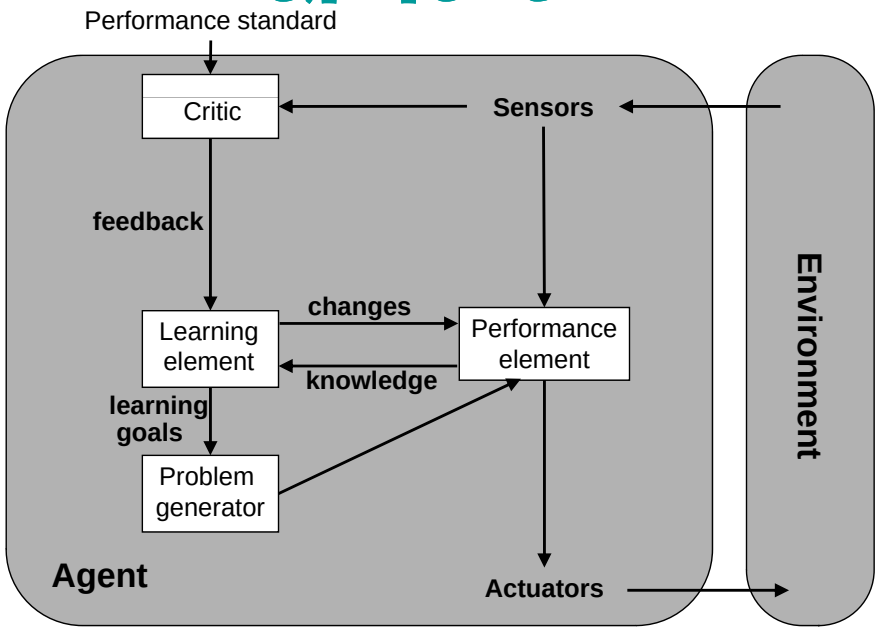
عامل های یادگیرنده

- تورینگ (۱۹۵۰): ایده برنامه نویسی واقعی هوشمند به صورت دستی
  - ← نیاز به روش های سریعتر
  - ← ساخت ماشین های یادگیرنده و آموزش به آنها
- مولفه های عامل یادگیرنده
  - عنصر یادگیرنده : برای ایجاد بهبود
  - عنصر کارآیی: انتخاب فعالیت های خارجی
  - منتقد: تولید بازخورد با توجه به استاندارد کارآیی برای عنصر یادگیرنده
  - مولد مساله: پیشنهاد فعالیت های اکتشافی
- مثال: تاکسی اتوماتیک
  - عنصر کارآیی: حرکت سریع از خط ۳ به خط ۱
  - منتقد: دریافت شکایت راننده های دیگر
  - ایجاد قانونی بیانگر بد بودن این عمل و اصلاح عنصر کارآیی

عامل های یادگیرنده

- انواع دانشی که عنصر یادگیرنده می تواند یاد بگیرد:
  - یادگیری مستقیم از دنباله ادراکی
  - یادگیری نحوه تغییرات دنیا: مشاهده دو حالت متوالی
  - یادگیری در مورد تاثیر عمل عامل: مشاهده نتایج فعالیت عامل
    - مثال: نحوه ترمز کردن در جاده های خیس
- پاداش و جریمه

عامل های یادگیری



حل مسائل توسط جستجو

فصل سوم
سید ناصر رضوی
Email: razavi@Comp.iust.ac.ir
۱۳۸۴

مقدمه

- عامل های حل مسأله
- انواع مسأله
- فرموله سازی مسأله
- مسائل نمونه
- الگوریتم های ابتدایی جستجو

عامل های حل مسأله

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation
  state ← UPDATE-STATE(state, percept)
  if seq is empty then do
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

فرضیات در مورد محیط: ایستا، قابل مشاهده، گسسته و قطعی

مثال: رومانی

- یک روز تعطیل در رومانی؛ مکان فعلی شهرآراد
- پرواز فردا، بخارست را ترک می کند.

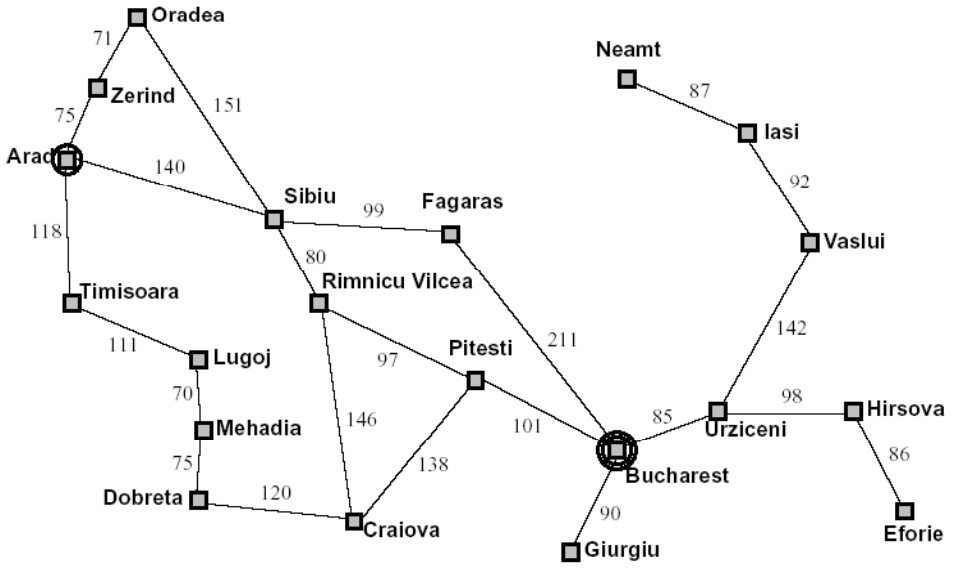
- فرموله سازی هدف:
  - بودن در بخارست
- فرموله سازی مسأله:
  - حالت ها: شهرهای مختلف
  - عملیات: رفتن از شهری به شهر دیگر
- یافتن پاسخ:
  - دنباله ای از شهرها، مانند:

Arad → Sibiu → Fagaras → Bucharest

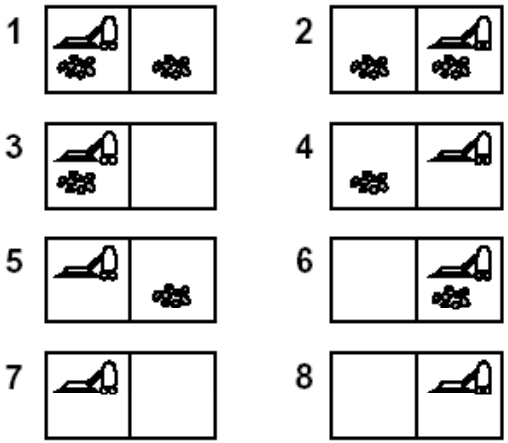
انواع مسأله

- قطعی، کاملا مشاهده پذیر ← مسائل تک – حالته
  - عامل دقیقا می داند در چه حالتی خواهد بود؛ راه حل یک دنباله می باشد.
- قطعی، مشاهده پذیر جزئی ← مسائل چند–حالته
  - ممکن است عامل ایده ای درباره اینکه کجاست نداشته باشد؛ راه حل یک دنباله است.
- غیر قطعی و/یا مشاهده پذیر جزئی ← مسائل احتمالی
  - ادراک اطلاعات جدیدی درباره حالت فعلی فراهم می کند.
  - در حین اجرا باید از حسگرها استفاده کند.
  - راه حل به صورت یک درخت
  - اغلب جستجو و اجرا به صورت یک در میان (interleave)
- فضای حالت ناشناخته ← مسائل اکتشافی (online)

مثال: رومانی



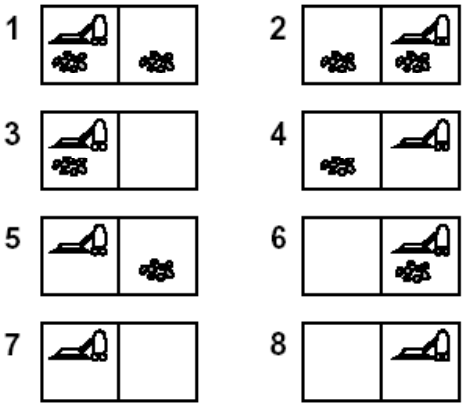
مثال: دنیای مکش



- تک–حالته، شروع در #5.

راه حل ؟

مثال: دنیای مکش



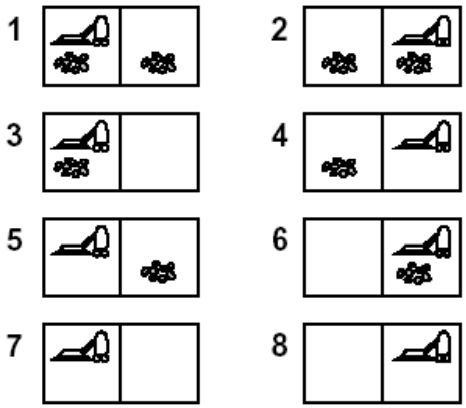
- تک–حالته، شروع در #5.

راه حل؟ [*Right, Suck*]

- چند–حالته، شروع در {1, 2, 3, 4, 5, 6, 7, 8} مثال عمل *Right* به {2, 4, 6, 8}.

راه حل؟

مثال: دنیای مکش



- چند-حالته، شروع در {1, 2, 3, 4, 5, 6, 7, 8}مثال عمل *Right* به {2, 4, 6, 8}.

راه حل؟

[*Right, Suck, Left, Suck*]

- احتمالی

- غیر قطعی: مکش می تواند یک فرش تمیز را کثیف کند.
- درک محلی: گرد و خاک در محل فعلی
- ادراک: [L, Clean] یعنی شروع در #5 یا #7

راه حل؟

[*Right, **if** dirt **then** Suck*]

فرموله سازی مسائل تک-حالته

یک مسأله بوسیله چهار مورد تعریف می شود:

۱. حالت اولیه مثلاً بودن در شهر Arad

۲. عمل ها یا تابع حالت بعدی

S(x)= مجموعه ای از زوج های عمل-حالت

- S(Arad) = {<Arad → Zerind, Zerind>, …}

۳. تابع تست هدف

- صریح: x = “at Bucharest”
- ضمنی: NoDirt(x)

۴. تابع هزینه مسیر :

- مثال: مجموع فواصل، تعداد عمل های انجام شده و ...
- هزینه گام (Step cost): $c(x, a, y) \geq 0$

- راه حل: دنباله ای از عملیات که از حالت اولیه شروع و به حالت هدف ختم می شود.

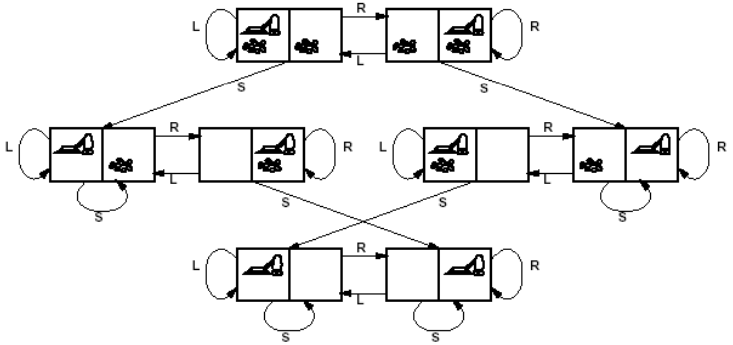
انتخاب یک فضای حالت

- دنیای واقعی به شدت پیچیده می باشد

← بنابراین، برای حل مسأله باید فضای حالت انتزاعی باشد.

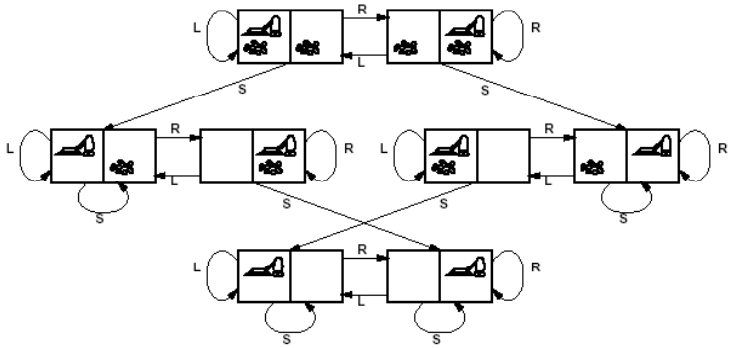
- حالت (انتزاعی) = مجموعه ای از حالت های واقعی
- عمل (انتزاعی) = ترکیبی پیچیده از عمل های واقعی
  - مثلا عمل Arad → Zerind می تواند مجموعه ای پیچیده از اعمال باشد.
- راه حل (انتزاعی) =
  - مجموعه ای از مسیرهای واقعی که در دنیای واقعی راه حل می باشند.
- هر عمل انتزاعی باید از مسأله اصلی ساده تر باشد!

مثال: گراف فضای حالت دنیای مکش



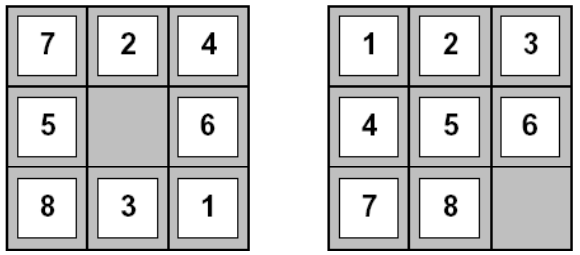
- حالات؟
- اعمال؟
- تست هدف؟
- هزینه مسیر؟

مثال: گراف فضای حالت دنیای مکش



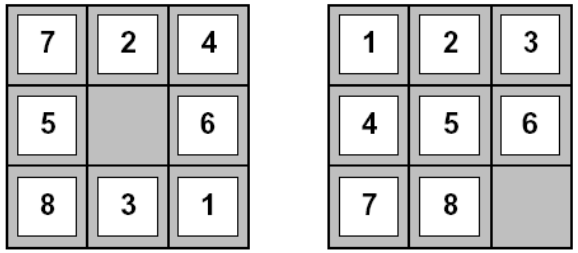
- حالات؟ وجود گرد و خاک و مکان های عامل (بدون در نظر گرفتن مقدار گرد و خاک)
- اعمال؟ *Left, Right, Suck*
- تست هدف؟ نبودن گرد و خاک
- هزینه مسیر؟ بازاء هر عمل ۱

مثال: معمای هشت



- حالات؟
- اعمال؟
- تست هدف؟
- هزینه مسیر؟

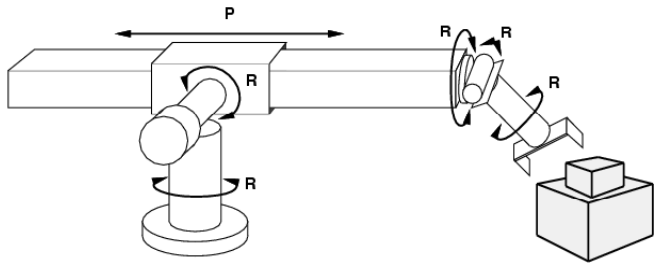
مثال: معمای هشت



- حالات؟ اعداد صحیح بیانگر محل کاشی ها
- اعمال؟ حرکت خانه خالی به چپ، بالا، راست و پایین
- تست هدف؟ حالت هدف (داده شده)
- هزینه مسیر؟ بازاء هر حرکت ۱

[توجه: راه حل بهینه خانواده معمای n یک مسأله NP-hard می باشد]

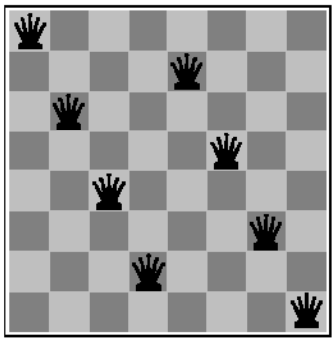
مثال: روبات اسمبل کننده



- حالات؟ زاویه مفاصل روبات، مختصات قطعات
- اعمال؟ حرکت پیوسته مفاصل روبات
- تست هدف؟ سرهم بندی کامل
- هزینه مسیر؟ زمان اجرا

مسأله هشت وزیر

- قراردادن هشت وزیر در صفحه شطرنج به طوریکه هیچ وزیری نتواند به وزیر دیگری حمله کند.



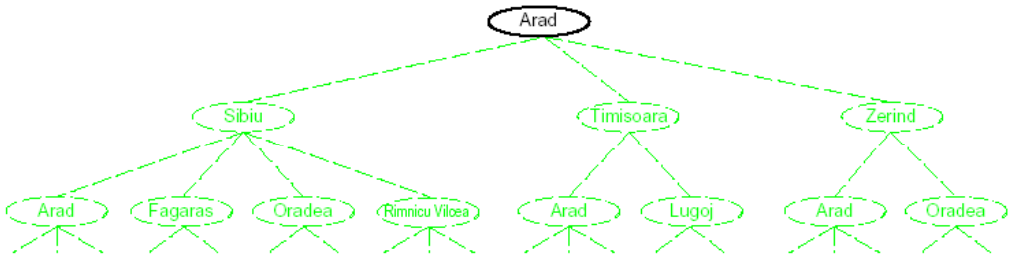
- آزمون هدف: ۸ وزیر روی صفحه شطرنج که با هم برخورد ندارند.
- هزینه مسیر: صفر
- حالات: ترتیب ۸ وزیر هر کدام در یک ستون
  مثال روبرو: {8, 6, 4, 2, 7, 5, 3, 1}
- عملگرها: انتقال یک وزیر دارای برخورد به مربع دیگری در همان ستون

الگوریتم های جستجوی درخت

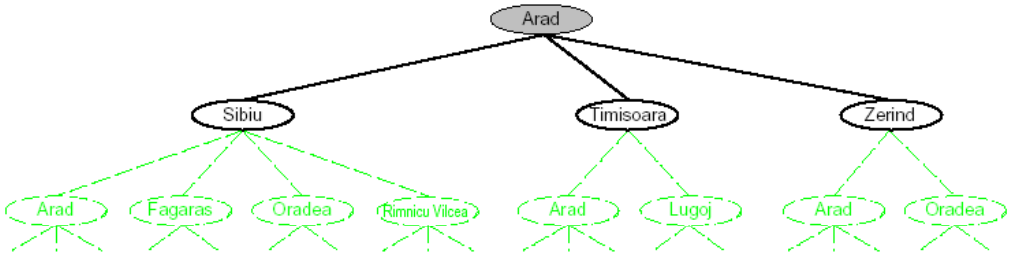
- **ایده اصلی:** کاوش offline و شبیه سازی شده فضای حالت بوسیله تولید حالات بعدی حالت هایی که تا کنون تولید شده اند.

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

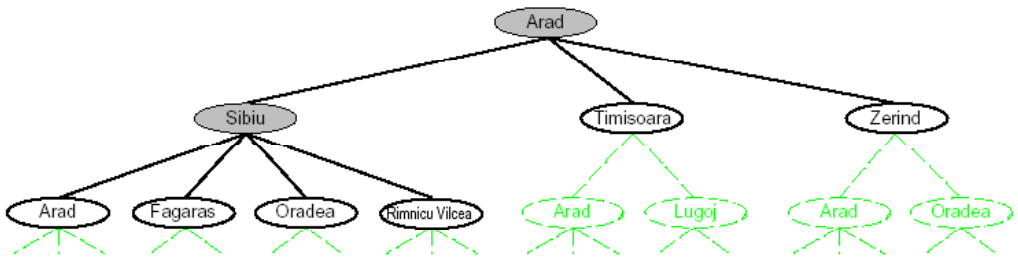

مثال جستجوی درخت



مثال جستجوی درخت



مثال جستجوی درخت



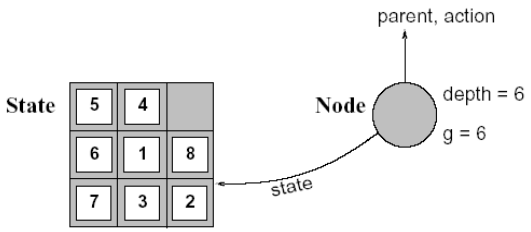
پیاده سازی: جستجوی عمومی درخت

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND( node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
    s ← a new NODE
    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
    add s to successors
  return successors
```


پیاده سازی: حالت و گره

- یک حالت (بیانگر) یک پیکره بندی فیزیکی می باشد
- یک گره یک ساختار داده ای تشکیل دهنده بخشی از درخت جستجو شامل: پدر، فرزندان، عمق و هزینه مسیر $g(x)$ است.
- حالت ها :

پدر، فرزند، عمق و هزینه مسیر ندارند!



- تابع EXPAND گره های جدید ایجاد می کند، فیلدهای مختلف را مقدار می دهد و با استفاده از تابع SUCCESSORS-FN مسأله، حالت های مربوطه ایجاد می شود.

استراتژی های جستجو

- یک استراتژی بوسیله ترتیب گسترش گره ها تعریف می شود.
- ابعاد ارزیابی استراتژی ها:
  - کامل بودن– آیا در صورت وجود راه حل، همیشه راه حلی پیدا می کند؟
  - پیچیدگی زمانی – تعداد گره های تولید شده/گسترش یافته
  - پیچیدگی حافظه – حداکثر تعداد گره ها در حافظه
  - بهینگی – آیا همیشه کم هزینه ترین راه حل را پیدا می کند؟

- پیچیدگی زمان و فضا برحسب پارامترهای زیر سنجیده می شوند:
  - b : حداکثر فاکتور انشعاب درخت جستجو
  - d : عمق کم هزینه ترین راه حل
  - m : حداکثر عمق فضای حالت (ممکن است ∞ باشد)

استرتژی های جستجوی ناآگاهانه

- استراتژی های **ناآگاهانه** تنها از اطلاعات موجود در تعریف مسأله استفاده می کنند.

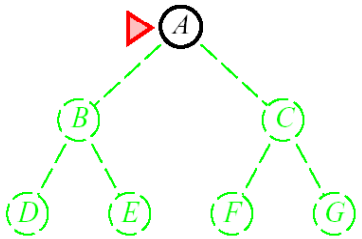
- جستجوی اول-سطح (BFS)
- جستجوی هزینه-یکنواخت (UCS)
- جستجوی اول-عمق (عمقی) (DFS)
- جستجوی با عمق محدود (DLS)
- جستجوی عمیق کننده تکراری (IDS)

جستجوی سطحی

- هربار سطحی ترین گره گسترش نیافته را گسترش می دهد.
- **پیاده سازی:**

– *fringe* یک صف FIFO می باشد. یعنی، فرزندان جدید به انتهای صف اضافه می شوند.

0: [A]

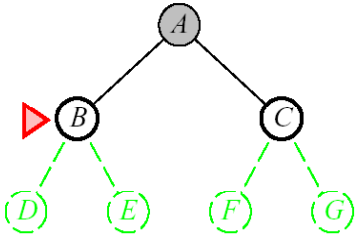


جستجوی سطحی

- هربار سطحی ترین گره گسترش نیافته را گسترش می دهد.
- **پیاده سازی:**

– *fringe* یک صف FIFO می باشد. یعنی، فرزندان جدید به انتهای صف اضافه می شوند.

1: [B, C]

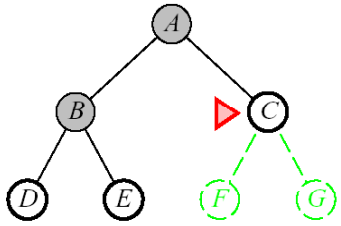


جستجوی سطحی

- هربار سطحی ترین گره گسترش نیافته را گسترش می دهد.
- **پیاده سازی:**

– *fringe* یک صف FIFO می باشد. یعنی، فرزندان جدید به انتهای صف اضافه می شوند.

2: [C, D, E]

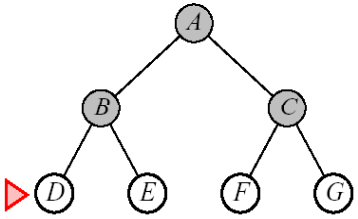


جستجوی سطحی

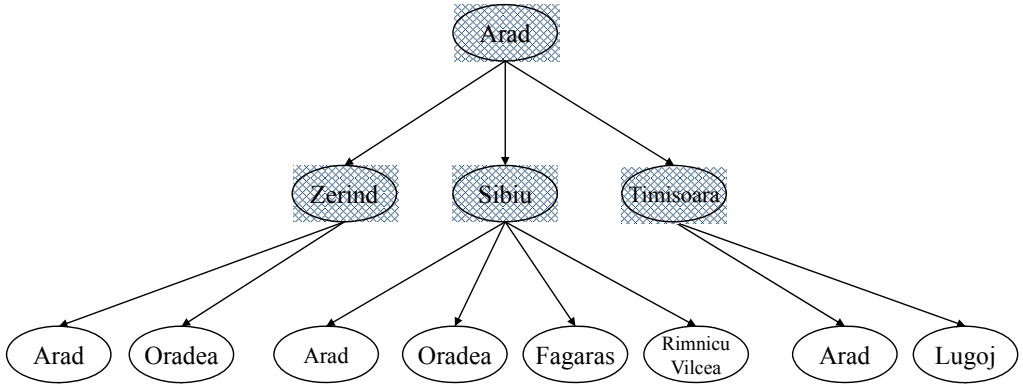
- هربار سطحی ترین گره گسترش نیافته را گسترش می دهد.
- **پیاده سازی:**

– *fringe* یک صف FIFO می باشد. یعنی، فرزندان جدید به انتهای صف اضافه می شوند.

3: [D, E, F, G]



مثال: جستجوی سطحی

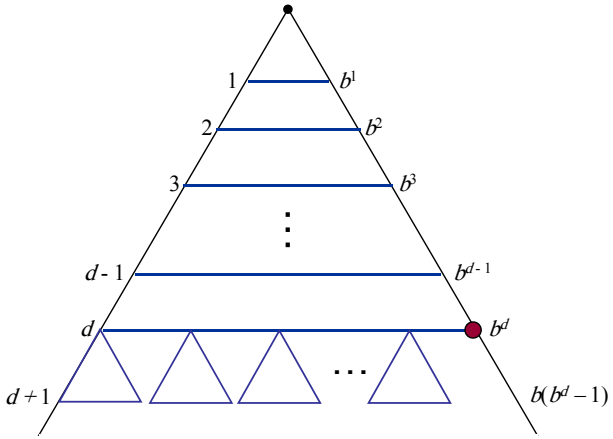


خصوصیات جستجوی سطحی

- کامل؟ بله (به شرط محدود بودن b)
- پیچیدگی زمانی؟ $b + b^2 + \ldots + b^d + b(b^d\text{-}1) = O(b^{d+1})$
- پیچیدگی حافظه؟ $O(b^{d+1})$ چون همه گره ها را در حافظه نگه می دارد
- بهینه؟ بله (مثلا اگر بازاء هر عمل، هزینه = ۱)

- مشکل اصلی **حافظه** می باشد. (نسبت به زمان)

پیچیدگی زمانی و حافظه جستجوی سطحی



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

زمان و فضای لازم در جستجوی سطحی

عمق	تعداد گره ها	زمان	حافظه
۲	۱۱۰۰	۱۱/۰ ثانیه	۱ مگا بایت
۴	۱۱۱۱۰۰	۱۱ ثانیه	۱۰۶ مگا بایت
۶	$۱۰^{۷}$	۱۹ دقیقه	۱۰ گیگا بایت
۸	$۱۰^{۹}$	۳۱ ساعت	۱ ترا بایت
۱۰	$۱۰^{۱۱}$	۱۲۹ روز	۱۰۱ ترا بایت
۱۲	$۱۰^{۱۳}$	۳۵ سال	۱۰ پتا بایت
۱۴	$۱۰^{۱۵}$	۳۵۲۳ سال	۱ هگزا بایت

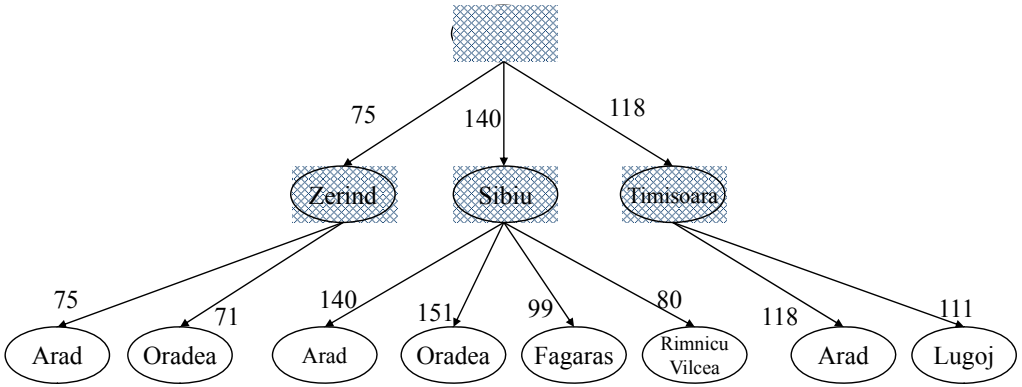
- $b = ۱۰$
- ۱۰۰۰۰ گره در هر ثانیه
- هر گره ۱۰۰۰ بایت

جستجوی هزینه-یکنواخت

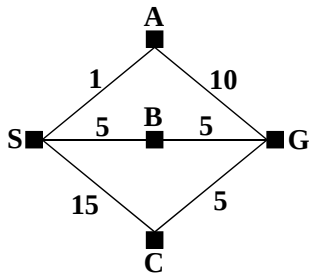
- هربار کم هزینه ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = صفی که براساس هزینه مسیر مرتب شده باشد.
- معادل جستجوی سطحی اگر هزینه گام ها مساوی باشند.

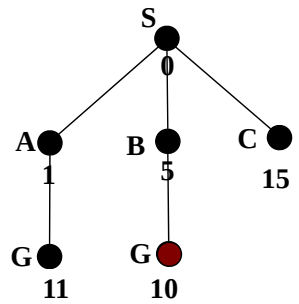
- کامل؟ بله اگر هزینه گامها $\geq \varepsilon$
- پیچیدگی زمانی؟ تعداد گره هایی که هزینه مسیر آنها کوچکتر یا مساوی هزینه راه حل بهینه باشد.
  $O(b^{\lceil C^*/\varepsilon \rceil})$
- پیچیدگی حافظه؟ مانند پیچیدگی زمانی
- بهینه؟ بله (گره ها به ترتیب صعودی $g(n)$ گسترش می یابند).

مثال: جستجوی هزینه یکنواخت



مثال: جستجوی هزینه یکنواخت



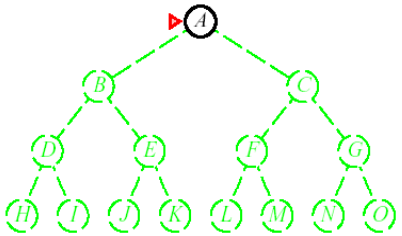


0: [S(0)]
1: [A(1), B(5), C(15)]
2: [B(5), G(11), C(15)]
3: [G(10), G(11), C(15)]
4: [G(11), C(15)]

جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

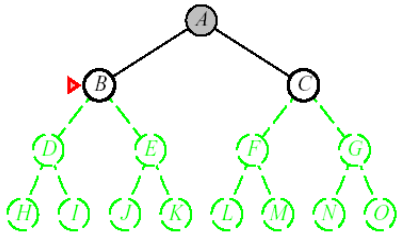
0: [A]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

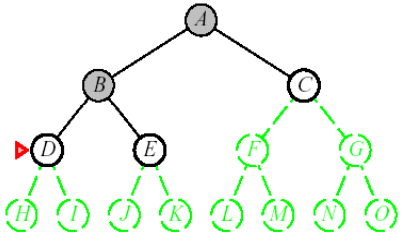
1: [B, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

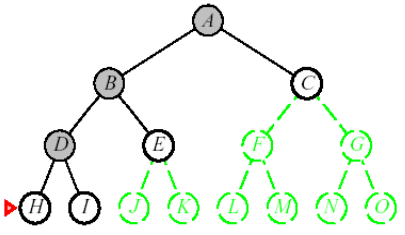
2: [D, E, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

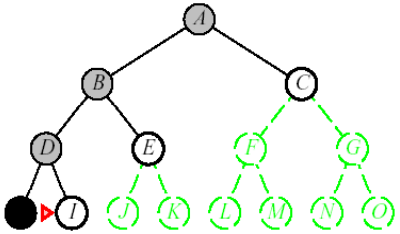
3: [H, I, E, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

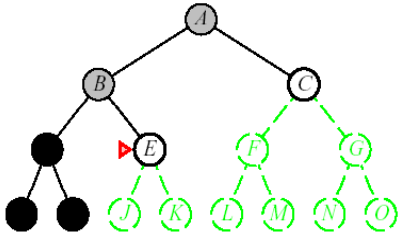
4: [I, E, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

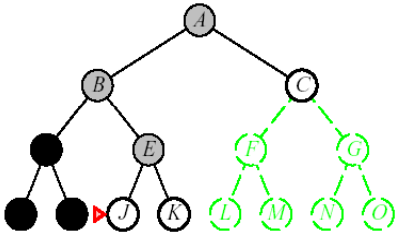
5: [E, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

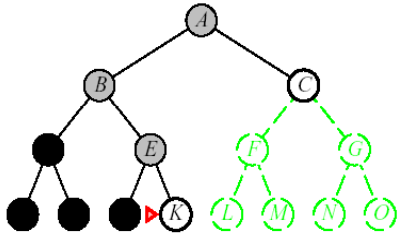
6: [J, K, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

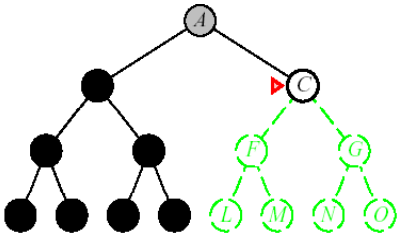
7: [K, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

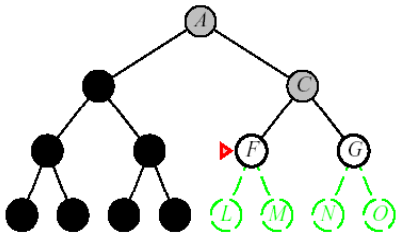
8: [C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

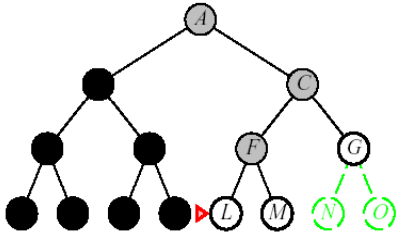
9: [F, G]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

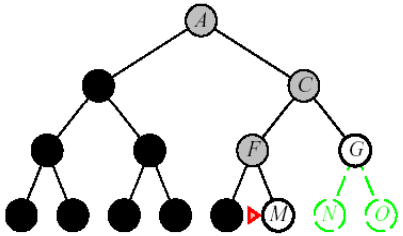
10: [L, M, G]



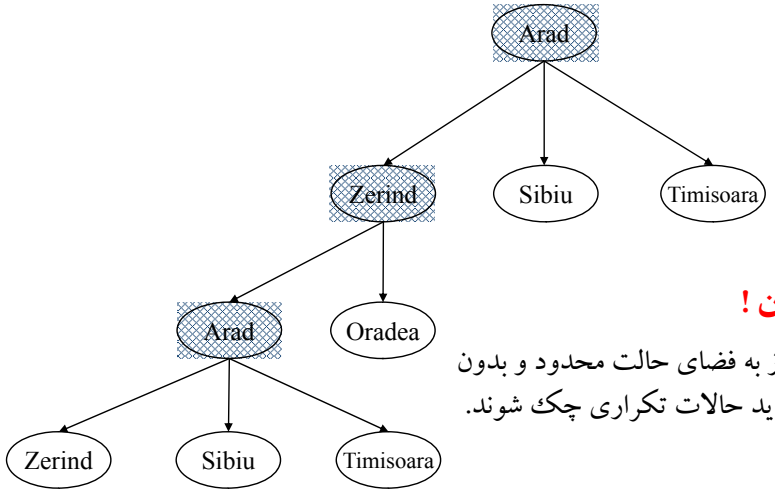
جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

11: [M, G]



مثال: جستجوی عمقی



- **حلقه بی پایان !**

در این جستجو نیاز به فضای حالت محدود و بدون چرخه داریم، یا باید حالات تکراری چک شوند.

خصوصیات جستجوی عمقی

- کامل؟
  - خیر (در فضاهای حالت با عمق نامحدود، دارای حلقه)
  - برای اجتناب از حالات تکرای در طول مسیر، نیاز به اصلاح دارد.
  - بنابراین، در فضای حالت محدود کامل است.
- پیچیدگی زمانی؟ $O(b^m)$
  - در بدترین حالت تمام گره های درخت جستجو تولید می شوند
  - اگر m خیلی بیشتر از d باشد، بسیار زیاد
  - اگر تعداد راه حل ها زیاد باشد، می تواند بسیار سریعتر از جستجوی سطحی باشد
- پیچیدگی حافظه؟
  - $O(bm)$، به صورت خطی!
- بهینه؟ خیر

جستجوی با عمق محدود

= جستجوی عمقی با محدوده عمقی l
یعنی، فرزندان گره های واقع در عمق l تولید نخواهند شد.

- در این استراتژی با در نظر گرفتن یک محدوده عمقی مانند l از به دام افتادن جستجوی عمقی در یک حلقه بی پایان جلوگیری می شود.(برش روی درخت جستجو)
- مثلا در نقشه رومانی چون ۲۰ شهر وجود دارد بنابراین طول راه حل باید حداکثر ۱۹ باشد.
- بنابراین هیچ وقت گره ای با عمق بیش از ۱۹ بررسی نخواهد شد.
- اگر در محدوده عمقی l راه حلی وجود داشته باشد، بالاخره پیدا خواهد شد، اما هیچ تضمینی برای یافتن راه حل بهینه وجود ندارد.

جستجوی با عمق محدود

- کامل؟
  - بله (اگر $l \geq d$)
- پیچیدگی زمانی؟
  - $O(b^l)$
- پیچیدگی حافظه؟
  - $O(bl)$
- بهینه؟
  - خیر

جستجوی با عمق محدود
پیاده سازی بازگشتی

```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
      result ← RECURSIVE-DLS(successor, problem, limit)
      if result = cutoff then cutoff-occurred? ← true
      else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
```


جستجوی عمیق کننده تکراری

- مشکل اصلی در جستجوی عمقی با عمق محدود شده (DLS) انتخاب یک محدوده عمقی مناسب است.
- در نقشه رومانی طول بزرگترین مسیر بین دو شهر ۹ می باشد(قطر)، و این محدوده عمقی مناسب تر از ۱۹ می باشد. اما در بیشتر فضاهای حالت انتخاب محدوده مناسب قبل از حل مسأله میسر نمی باشد.
- جستجوی عمیق کننده تکراری روشی برای تعیین محدوده عمقی مناسب با امتحان کردن تمامی محدوده ها (از صفر به بالا) می باشد. یعنی اول عمق صفر، بعد عمق ۱، بعد عمق ۲ و ...

جستجوی عمیق کننده تکراری

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or fail-
ure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

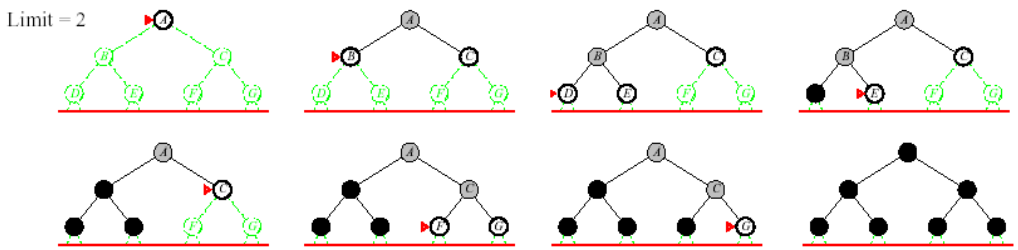

جستجوی عمیق کننده تکراری

- جستجوی عمیق کننده تکراری مزایای جستجوی سطحی و عمقی را با هم ترکیب می کند:
  - مانند جستجوی سطحی کامل (اگر فاکتور انشعاب محدود باشد) و بهینه (اگر هزینه مسیر یک تابع غیر نزولی برحسب عمق باشد)است.
  - جستجوی عمقی دارای مصرف حافظه خطی $O(bd)$ می باشد.
- این جستجو از نظر پیچیدگی زمانی مانند جستجوی محدود شده می باشد، به جز اینکه برخی حالات چند بار بسط داده می شوند.

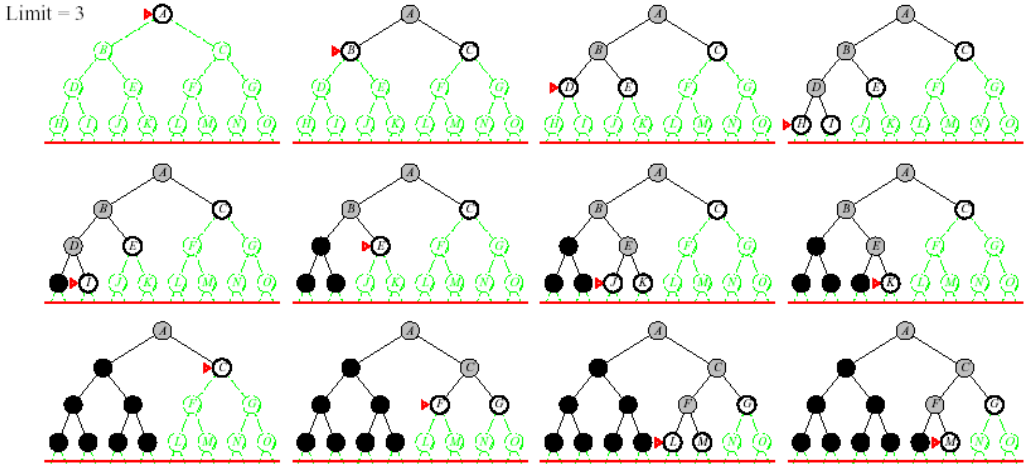
جستجوی عمیق کننده تکراری (Limit = 0)

جستجوی عمیق کننده تکراری (Limit = 1)

جستجوی عمیق کننده تکراری (Limit = 2)



جستجوی عمیق کننده تکراری (Limit = 3)



خواص جستجوی عمیق کننده تکراری

- کامل؟؟ بله (مانند جستجوی سطحی)
- پیچیدگی زمانی ؟؟

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

- پیچیدگی حافظه؟؟ $O(bd)$
- بهینه ؟؟ بله، (مانند جستجوی سطحی)

- می تواند برای کاوش درخت جستجوی هزینه یکنواخت اصلاح شود!!!

پیچیدگی زمانی عمیق کننده تکراری

DLS (/= 0)	0
DLS (/= 1)	b^1
DLS (/= 2)	$b^1 + b^2$
.	.
.	.
.	.
DLS (/= d - 1)	$b^1 + b^2 + b^3 + \ldots + b^{d-1}$
DLS (/= d)	$b^1 + b^2 + b^3 + \ldots + b^{d-1} + b^d$

سربار $\leq N_{DLS\ (/=d)}$

$$N_{IDS} = db^1 + (d-1)b^2 + (d-2)b^3 + \ldots + 2b^{d-1} + b^d$$

کارآیی IDS

- تعداد گره های تولید شده توسط DLS در عمق d با فاکتور انشعاب b:

$N_{DLS} = b + b^2 + \ldots + b^{d-1} + b^d$

- تعداد گره های تولید شده توسط IDS در عمق d با فاکتور انشعاب b:

$N_{IDS} = db^1 + (d-1)b^2 + \ldots + 2b^{d-1} + 1b^d$

- اگر $b = 10$ و $d = 5$:

$N_{DLS} = 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}110$

$N_{IDS} = 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$

- محاسبه میزان سربار:

$((123450 - 111110)/111110) * 100 = 11\%$

  - با افزایش فاکتور انشعاب b میزان سربار کاهش می یابد.
  - در بدترین حالت $b = 2$، سربار 100% است و این جستجو دو برابر زمان می برد.

جستجوی دوطرفه

- ایده: انجام جستجو در دو جهت به طور همزمان
  - رو به جلو: از حالت اولیه به سمت حالت هدف
  - رو به عقب: از حالت هدف به سمت حالت اولیه
- انگیزه: $b^{d/2} + b^{d/2}$ بسیار کمتر از b^d می باشد

- مثال: اگر راه حل یک مسأله در عمق $d = 6$ باشد و $b = 10$ آنگاه
  - جستجوی دو طرفه (در هر دو طرف جستجوی سطحی) ← 22,200 گره
  - جستجوی سطحی ← 11,111,000 گره

جستجوی دوطرفه

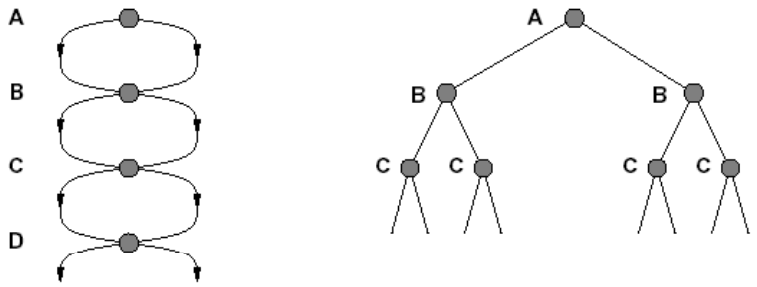
- پیچیدگی زمانی: $O(b^{d/2})$
- پیچیدگی حافظه: $O(b^{d/2})$
  - به منظور بررسی تعلق حداقل یکی از درخت ها باید در حافظه نگهداری شود
  - مصرف حافظه نمایی، بزرگترین ضعف جستجوی دوطرفه می باشد
- کامل بودن و بهینگی (برای هزینه های گام یکسان):
  - اگر در هر دو طرف از جستجوی سطحی استفاده شود

خلاصه الگوریتم ها

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

حالات تکراری

- شکست در تشخیص حالت های تکرای می تواند یک مسأله خطی را به یک مسأله نمایی تبدیل کند!



جستجوی گراف

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

خلاصه

- فرموله سازی مسأله اغلب نیاز به انتزاع جزییات مسأله دارد، تا بتوان فضای حالتی بدست آوردکه به صورت مقرون به صرفه ای قابل کاوش کرد ن و جستجو باشد.

- انواع استراتژی های ناآگاهانه وجود دارد.

- مصرف حافظه جستجوی عمیق کننده تکراری دارای مرتبه خطی می باشد و زمان خیلی بیشتری نسبت به سایر روشهای ناآگاهانه مصرف نمی کند.

روش های جستجوی آگاهانه

فصل چهارم
سید ناصر رضوی
Email:razavi@Comp.iust.ac.ir
۱۳۸۶

مقدمه

- جستجوی اول-بهترین (Best-First Search)
  - جستجوی حریصانه (Greedy search)
  - جستجوی A* و خصوصیات آن
- جستجوی عمیق کننده تکراری A* و SMA*
- جستجوی اول بهترین بازگشتی (RBFA*)
- هیوریستیک ها (Heuristics)
- الگوریتم های جستجوی محلی
  - جستجوی تپه نوردی (Hill Climbing)
  - جستجوی Simulated Annealing
  - الگوریتم های ژنتیک

مشکلات روش های جستجوی ناآگاهانه

- معیار انتخاب گره بعدی برای گسترش دادن تنها به شماره سطح آن بستگی دارد.
- از ساختار مسأله بهره نمی برند.
- درخت جستجو را به یک روش از پیش تعریف شده گسترش می دهند. یعنی قابلیت تطبیق پذیری با آنچه که تا کنون در مسیر جستجو دریافته اند و نیز حرکتی که می تواند خوب باشد ندارند.

مرور: جستجوی درخت

function GENERAL-SEARCH(*problem*, *strategy*) **returns** a solution , or failure

initialize the search tree using the initial state of *problem*

loop do

if there are no candidates for expansion **then return** failure

choose a leaf node for expansion according to *strategy*

if the node contains the goal state **then return** the corresponding solution

else expand the node and add the resulting node to the search tree

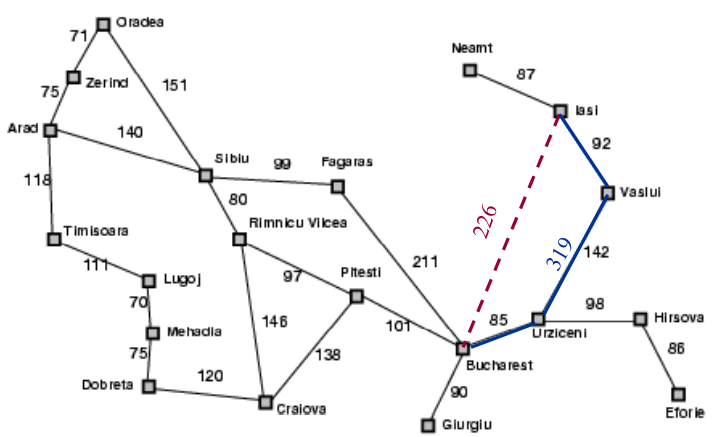
end

- استراتژی توسط ترتیب گسترش یافتن گره ها تعریف می شود.

جستجوی اول- بهترین

- ایده: برای هر گره از یک تابع ارزیابی (evaluation function) استفاده کن.
  - تخمین "میزان مطلوب بودن" گره
  - ←هربار مطلوب ترین گره گسترش نیافته را بسط می دهد.
- پیاده سازی:
  fringe یک صف می باشد که بر اساس میزان مطلوب بودن گره ها مرتب می باشد.
- موارد خاص:
  - جستجوی حریصانه (Greedy search)
  - جستجوی A*

نقشه رومانی به همراه هزینه مراحل برحسب *km*

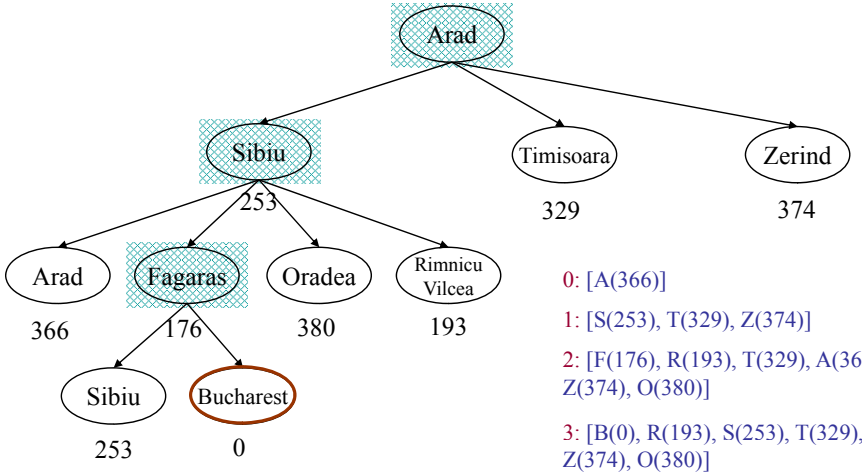


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

جستجوی اول-بهترین حریصانه

- تابع ارزیابی $f(n) = h(n)$ (هیوریستیک)
- = هزینه تخمینی از گره n تا نزدیکترین هدف
- مثال: $h_{SLD}(n)$ = فاصله مستقیم از n تا بخارست.
- جستجوی حریصانه گره ای را گسترش می دهد که به نظر می رسد نزدیکترین گره به هدف (بخارست) باشد.

مثال جستجوی حریصانه



0: [A(366)]

1: [S(253), T(329), Z(374)]

2: [F(176), R(193), T(329), A(366), Z(374), O(380)]

3: [B(0), R(193), S(253), T(329), A(366), Z(374), O(380)]

4: [R(193), S(253), T(329), A(366), Z(374), O(380)]

خواص جستجوی حریصانه

- **کامل؟** خیر، ممکن است در حلقه گیر کند.
  - مثال: حالت اولیه Iasi، حالت هدف Fagaras

Iasi → Neamt → Iasi → Neamt → …

- **پیچیدگی زمانی؟** $O(b^m)$، اما با یک هیوریستیک خوب می تواند بسیار بهبود یابد.
- **پیچیدگی حافظه؟** $O(b^m)$، تمام گره ها را در حافظه نگه می دارد.
- **بهینه؟** خیر، (با توجه به مثال قبل)

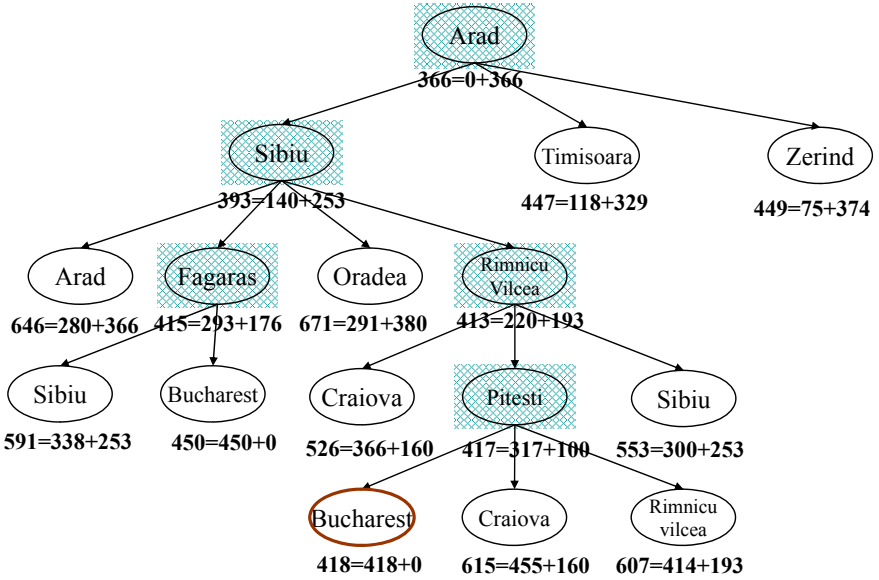
جستجوی A*

- ایده: از گسترش مسیرهایی که تاکنون مشخص شده پرهزینه می باشند، اجتناب کن.
- A*: ترکیب مزایای UCS و جستجوی حریصانه:
  - جستجوی حریصانه $h(n)$ را حداقل می کند، نه کامل و نه بهینه
  - جستجوی UCS، هزینه مسیر را حداقل می کند؛ کامل و بهینه است؛ می تواند بسیار زمانبر باشد.
- تابع ارزیابی

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

- $g(n)$: هزینه مسیر پیموده شده تا n.
- $h(n)$: هزینه تخمینی ارزانترین مسیر راه حل از n تا **هدف**.
- $f(n)$: هزینه تخمینی ارزانترین راه حل که از n می گذرد.

مثال جستجوی A*



جستجوی A*

A* از یک هیوریستیک قابل قبول (admissible) استفاده می کند، یعنی، همواره $h(n) \le h^*(n)$ که در آن $h^*(n)$ هزینه واقعی n تا هدف می باشد.

هم چنین داریم

$h(n) \ge 0$ و

$h(G) = 0$ (G یک هدف می باشد).

به طور کلی :

$$0 \le h(n) \le h^*(n)$$

مثال: در هیوریستیک $h_{SLD}(n)$ هیچگاه هزینه تخمینی بیشتر از هزینه واقعی نخواهد بود. (کوتاهترین فاصله بین دو نقطه، فاصله مستقیم آن دو نقطه می باشد).

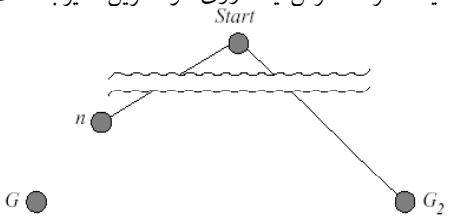
هیوریستیک قابل قبول

- یک هیوریستیک مانند $h(n)$ قابل قبول است اگر برای هر گره n داشته باشیم: $h(n) \le h^*(n)$ که $h^*(n)$ هزینه واقعی برای رسیدن به هدف از گره n می باشد.
- یک هیوریستیک قابل قبول هرگز هزینه رسیدن به هدف را بیش از حد تخمین نمی زند، یعنی خوش بینانه است.
- مثال: هیوریستیک $h_{SLD}(n)$ (هیچگاه فاصله واقعی را بیش از حد تخمین نمی زند).
- قضیه: اگر $h(n)$ قابل قبول باشد، A* با استفاده از TREE-SEARCH بهینه است.

بهینگی A* (اثبات)

- فرض کنید یک جواب زیر بهینه مانند G_2 تولید شده و در صف قرار دارد.

هم چنین فرض کنید n یک گره گسترش نیافته روی کوتاهترین مسیر به هدف بهینه G باشد(؟؟)



$f(G_2) = g(G_2)$ چون، $h(G_2) = 0$

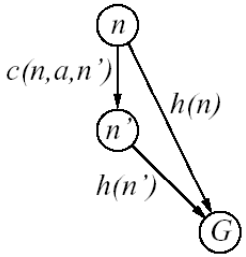
$> g(G)$ چون، G_2 زیر بهینه است

$\ge f(n)$ چون، h قابل قبول است

چون $f(G_2) \ge f(n)$، الگوریتم A* هیچ وقت G_2 را برای گسترش یافتن انتخاب نمی کند.

اثبات لم: سازگاری (Consistency)

- یک هیوریستیک سازگار است اگر : $h(n) \le c(n, a, n') + h(n')$
- اگر h سازگار باشد، داریم:

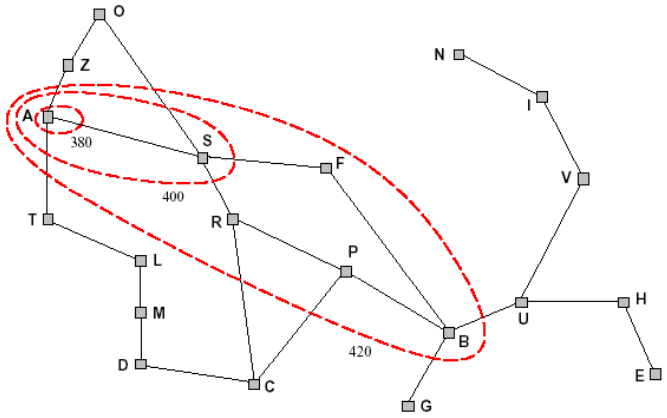


$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\ge g(n) + h(n) \\ &\ge f(n) \end{aligned}$$

- یعنی، $f(n)$ در طول هر مسیری غیر کاهشی می باشد. (یکنوایی، monotonicity)
- قضیه: اگر $h(n)$ سازگار باشد، A* با استفاده از GRAPH-SEARCH بهینه است.

بهینگی A*

- **لم:** A* گره ها را براساس ترتیب صعودی مقادیر f گسترش می دهد.
- A* به تدریج f-contour ها را اضافه می کند.
- کانتور i شامل تمام گره ها با $f = f_i$ می باشد، که $f_i < f_{i+1}$



خصوصیات A*

- کامل؟ بله، مگر اینکه تعدادی نامحدود گره با $f \leq f(G)$ وجود داشته باشد. A* در گراف های متناهی محلی (با فاکتور انشعاب محدود) کامل است به شرط آنکه هزینه تمام عملگرها مثبت باشد.
  - گره ای با فاکتور انشعاب نامحدود وجود داشته باشد.
  - مسیری با هزینه محدود اما با تعداد گره های نامحدود وجود داشته باشد.

- پیچیدگی زمانی؟ نمایی بر حسب [خطای نسبی h * طول راه حل]
مگر اینکه خطا در تابع کشف کننده رشدی سریعتر از لگاریتم هزینه مسیر واقعی نداشته باشد، به زبان ریاضی:

$$| h(n) - h^*(n)| \leq O(\log h^*(n))$$

خصوصیات A*

- پیچیدگی فضا؟؟ تمام گره ها را در حافظه نگه می دارد.

- بهینه؟؟ بله – نمی تواند f_{i+1} را گسترش دهد مگر f_i تمام شده باشد.
  - A* تمام گره ها با $f(n) < f^*$ را گسترش می دهد.
  - A* برخی از گره ها با $f(n) = f^*$ را گسترش می دهد.
  - A* گره ای با $f(n) > f^*$ را هرگز گسترش نمی دهد.

- A* دارای کارآیی بهینه (optimally efficient) می باشد!

جستجوی هیوریستیک با حافظه محدود

- چند راه حل برای مسأله حافظه در A* (با حفظ خصوصیات کامل بودن و بهینگی):
  - جستجوی عمیق کننده تکرای A* (IDA*)
    - مانند IDS ولی به جای محدوده عمقی از محدوده *f-cost* ($g + h$) استفاده می شود
  - جستجوی اول-بهترین بازگشتی (RBFS)
    - یک الگوریتم بازگشتی با فضای خطی که سعی می کند از جستجوی اول-بهترین استاندارد تقلید کند
  - جستجوی (ساده شده) A* با حافظه محدود ((S)MBA*)
    - حذف بدترین گره وقتی که حافظه پر می باشد

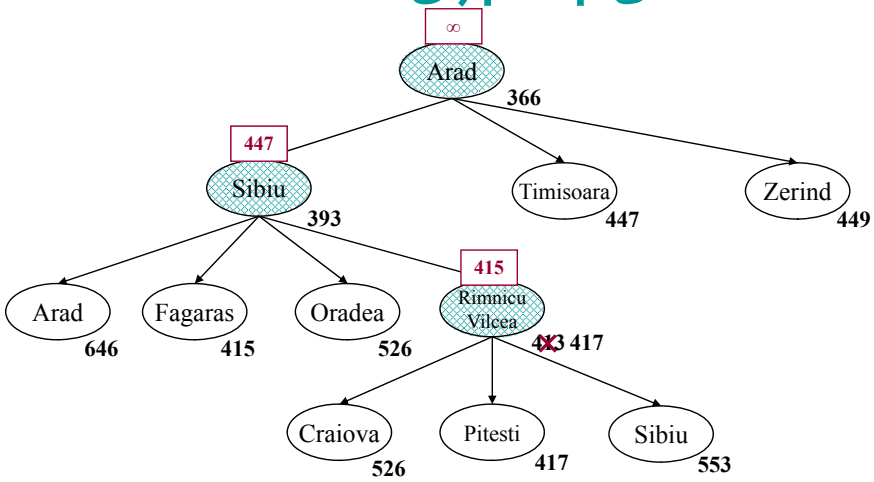
جستوی اول-بهترین بازگشتی(RBFS)

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
    return RFBS(problem,MAKE-NODE(INITIAL-STATE[problem]),∞)

function RFBS( problem, node, f_limit) return a solution or failure and a new f-cost limit
    if GOAL-TEST[problem](STATE[node]) then return node
    successors ← EXPAND(node, problem)
    if successors is empty then return failure, ∞
    for each s in successors do
        f[s] ← max(g(s) + h(s), f[node])
    repeat
        best ← the lowest f-value node in successors
        if f[best] > f_limit then return failure, f[best]
        alternative ← the second lowest f-value among successors
        result, f[best] ← RBFS(problem, best, min(f_limit, alternative))
        if result ≠ failure then return result
```

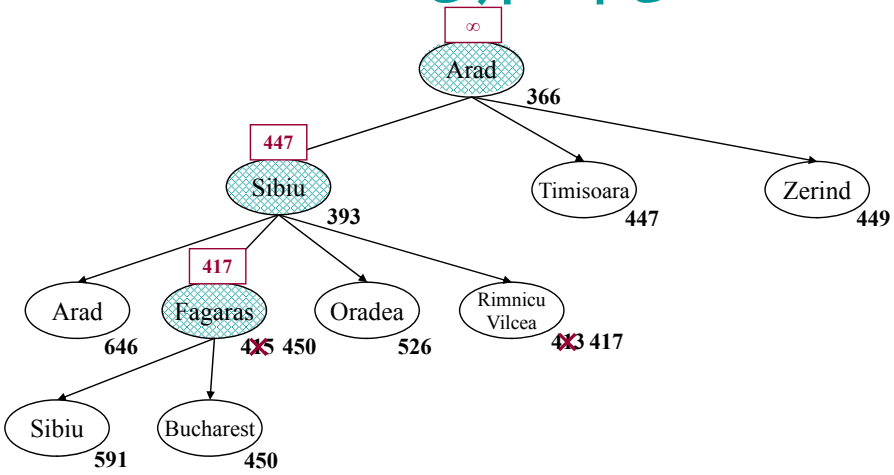
جستجوی RBFS

- مقدار f-value مربوط به بهترین مسیر جایگزین موجود را نگهداری می کند.
  - اگر f-value فعلی از f-value مسیر جایگزین تجاوز کند، آنگاه به این مسیر جایگزین بازگشت می کند.
  - در بازگشت به عقب مقدار f-value مربوط به هر گره موجود در مسیر را با بهترین مقدار f-value فرزندانش جایگزین می کند.
  - بنابراین گسترش مجدد نتیجه فعلی هنوز ممکن می باشد.

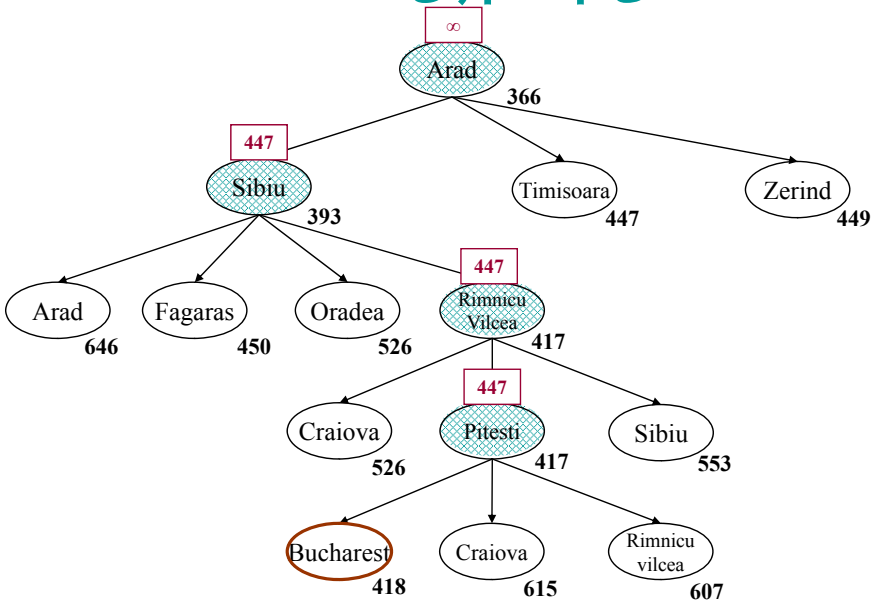
مثال جستجوی RBFS



مثال جستجوی RBFS



مثال جستجوی RBFS



ارزیابی RBFS

- RBFS کمی کارآتر از IDA* می باشد
  - هنوز گسترش اضافی گره ها وجود دارد(تغییر عقیده)
- RBFS هم مانند A*، اگر $h(n)$ قابل قبول باشد بهینه است
- پیچیدگی حافظه $O(bd)$
  - IDA* فقط یک عدد را نگهداری می کند (حد فعلی *f-cost*)
- تعیین پیچدگی زمانی مشکل است
  - به دقت تابع هیوریستیک و میزان تغییر بهترین مسیر در اثر بسط گره ها بستگی دارد.
  - مانند IDA* در معرض افزایش نمایی پیچدگی زمانی قرار دارد
- RBFS و IDA* هر دو از میزان بسیار کم حافظه رنج می برند.
  - با اینکه حافظه زیادی وجود دارد، نمی توانند از ان استفاده کنند.

(Simplified) Memory Bounded A*

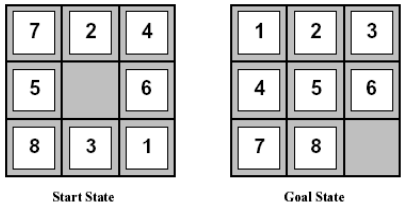
- ایده: استفاده از تمامی حافظه موجود
  - یعنی، گسترش بهترین گره های برگی تا زمانی که حافظه موجود پر شود
  - در صورت پر شدن حافظه، SMA* بدترین گره برگ (با بیشترین مقدار *f-value*) را از حافظه حذف می کند
  - مانند RBFS اطلاعات گره های فراموش شده را در پدرشان ذخیره می کند
- اگر تمام برگ ها دارای f-value برابر باشند چه می شود؟
  - ممکن است یک گره را هم برای گسترش و هم برای حذف انتخاب کند
  - راه حل SMA*:
    - گسترش: بهترین گره برگ که از همه جدید تر است
    - حذف: بدترین گره که از همه قدیمی تر است
  - SMA* کامل است اگر راه حل قابل دستیابی وجود داشته باشد و بهینه است اگر راه حل بهینه قابل دستیابی باشد

هیوریستیک های قابل قبول

مثال برای معمای هشت:

$h_1(n)$ = number of misplaced tiles

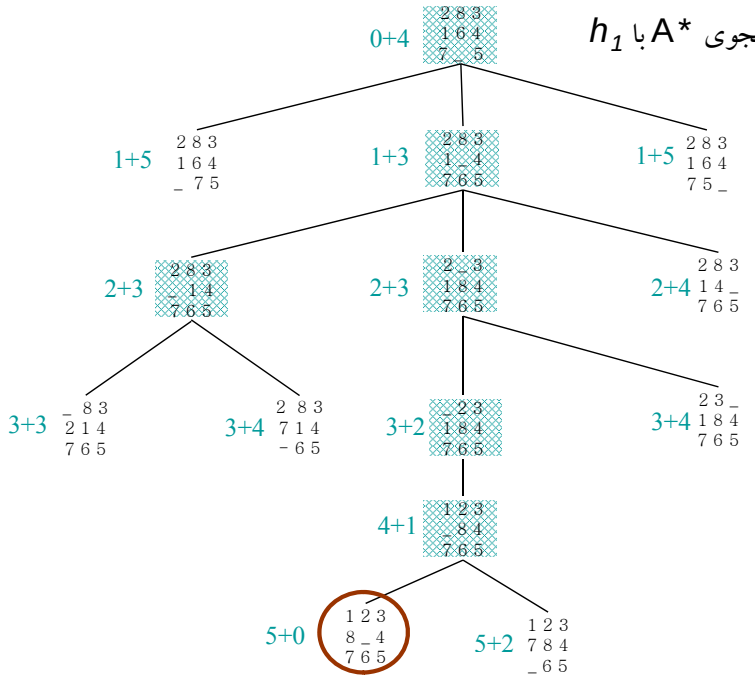
$h_2(n)$ = total Manhattan distance



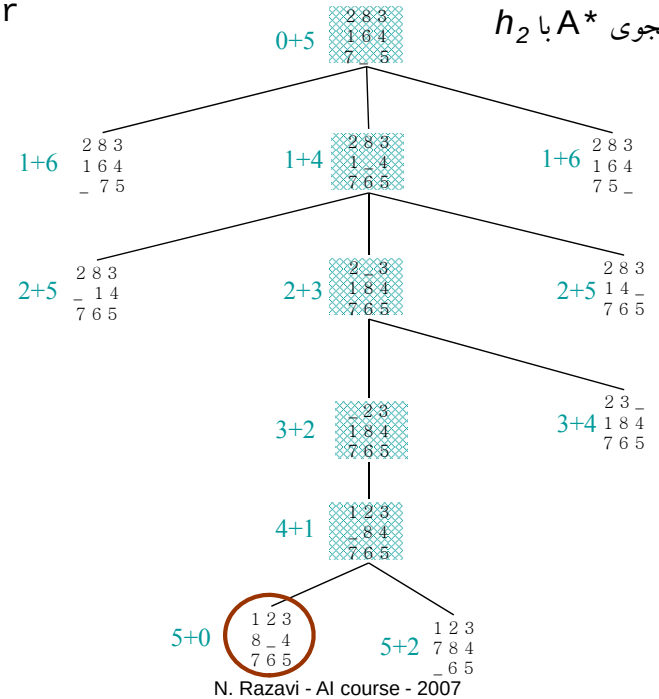
$h_1(S) = 6$

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

- مثال: جستجوی A* با h_1



- مثال: جستجوی A* با h_2



تسلط (Dominance)

- اگر بازاء هر n داشته باشیم، $h_2(n) \geq h_1(n)$
  (و هر دو هیوریستیک قابل قبول باشند)
- آنگاه h_2 بر h_1 تسلط دارد و برای جستجو بهتر می باشد.
- مثال هزینه جستجو: تعداد گره های تولید شده در عمق های مختلف:

$d = 14 \rightarrow$ IDS = 3,473,941
A*(h_1) = 539
A*(h_2) = 113

$d = 24 \rightarrow$ IDS = 54,000,000,000
A*(h_1) = 39,135
A*(h_2) = 1641

فاکتور انشعاب موثر (b*)

- فاکتور انشعاب موثر (EBF):
  اگر تعداد گره های گسترش یافته توسط روال جستجو N باشد و عمق راه حل d باشد، b^* به صورت زیر محاسبه می شود:

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

- مثال: اگر A* راه حلی را در عمق ۵ با استفاده از ۵۲ گره پیدا کند، فاکتور انشعاب موثر را محاسبه کنید.

پاسخ:

$$53 = 1 + b^* + (b^*)^2 + \ldots (b^*)^5 \rightarrow b^* = 1.92$$

- در یک هیوریستیک هر چه فاکتور انشعاب به یک نزدیکتر باشد، بهتر است و آن هیوریستیک کیفیت کشف کنندگی بیشتری دارد.

مقایسه بین هزینه جستجو و فاکتور انشعاب موثر

	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	6,384	39	25	2.80	1.33	1.24
10	47,127	93	39	2.79	1.38	1.22
12	364,404	227	73	2.78	1.42	1.24
14	3,473,941	539	113	2.83	1.44	1.23
16	-	1,301	211	-	1.45	1.25
18	-	3,056	363	-	1.46	1.26
20	-	7,276	679	-	1.47	1.27
22	-	18,094	1,219	-	1.48	1.28
24	-	39,135	1,641	-	1.48	1.26

ابداع توابع هیوریستیک

- می توان هیوریستیکهای قابل قبول را برای یک مسأله، از *هزینه دقیق* راه حل یک نسخه راحت (relaxed) از مسأله بدست آورد.
- مثال: قانون معمای هشت. یک کاشی می تواند از خانه A به خانه B برود، اگر A مجاور B باشد و B خالی باشد.
- اگر قوانین معمای هشت به گونه ای راحت شوند که یک کاشی بتواند به هر خانه ای حرکت کند، هیوریستیک $h_1(n)$ کوتاهترین راه حل را می دهد.
- اگر قوانین به گونه ای راحت شوند که یک کاشی بتواند به هر خانه مجاور حرکت کند، هیوریستیک $h_2(n)$ کوتاهترین راه حل را می دهد.
- نکته کلیدی: هزینه راه حل بهینه یک مسأله راحت، بیشتر از هزینه راه حل بهینه در مسأله واقعی نیست.

الگوریتم های جستجوی محلی و مسائل بهینه سازی

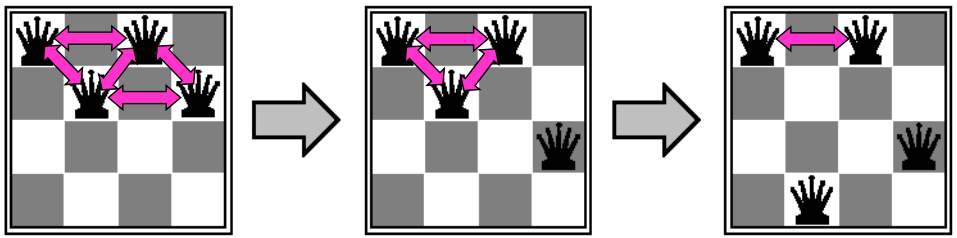
- در بسیاری از مسائل بهینه سازی، مسیر راه حل اهمیت ندارد؛ خود حالت هدف پاسخ مسأله می باشد.
- فضای حالت = مجموعه پیکره بندی های«کامل»؛
- یافتن پیکره بندی بهینه، مانند TSP
- یافتن یک پیکره بندی که محدودیت های مسأله را ارضاء کند، مانند مساله n–وزیر
- در چنین مواردی می توان از الگوریتم های جستجوی محلی بهره گرفت.
- یک حالت«فعلی» را به تنهایی در نظر بگیر؛ سعی کن آن را بهبود بخشی.

الگوریتم های جستجوی محلی و مسائل بهینه سازی

- **جستجوی محلی** = استفاده از یک حالت فعلی و حرکت به حالت های همسایه
- مزایا:
  - استفاده از حافظه بسیار کم
  - یافتن راه حل های معقول در اغلب موارد در فضاهای حالت بزرگ و یا نامحدود
- مفید برای مسائل بهینه سازی محض
  - یافتن بهترین حالت بر طبق تابع هدف (objective function)

مثال: n–وزیر

- مسأله : n وزیر را در یک صفحه شطرنج $n * n$ به گونه ای قرار بده که هیچ دو وزیری در یک سطر، ستون و یا قطر قرار نگیرند.
- یکی از وزیرها را در ستون خودش به گونه ای جابه جا کن که تعداد برخورد ها کاهش یابد.

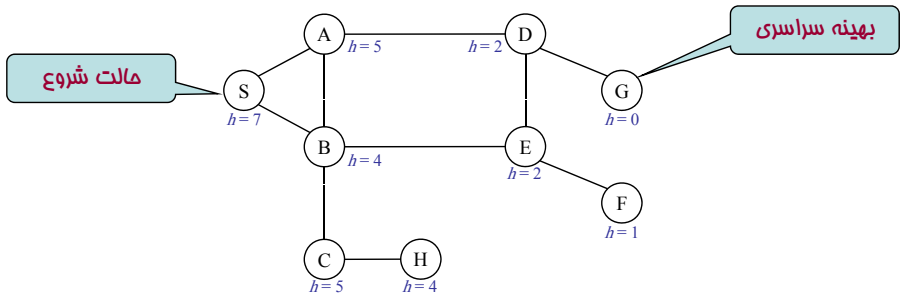


تپه نوردی(یا گرادیان صعودی/نزولی)

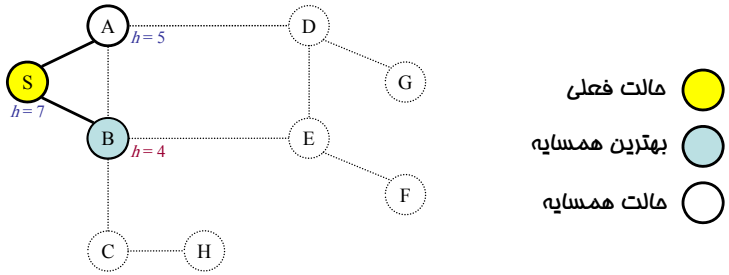
«مانند بالا رفتن از کوه اورست در مه غلیظ با ضعف حافظه»

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node
   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
```

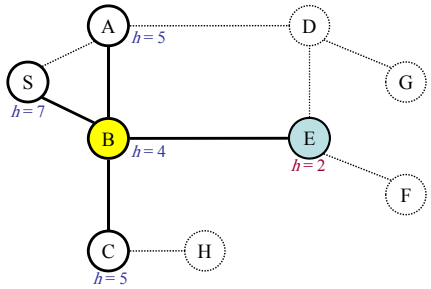

مثال: جستجوی تپه نوردی

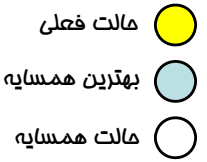


مثال: جستجوی تپه نوردی

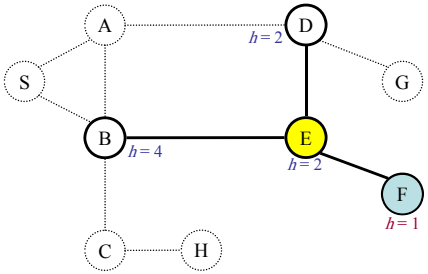


مثال: جستجوی تپه نوردی





مثال: جستجوی تپه نوردی

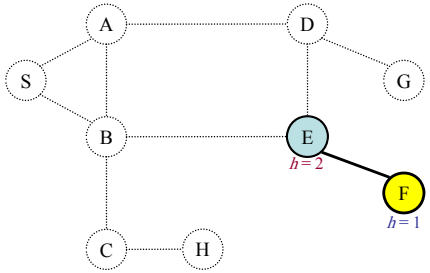


حالت فعلی

بهترین همسایه

حالت همسایه

مثال: جستجوی تپه نوردی

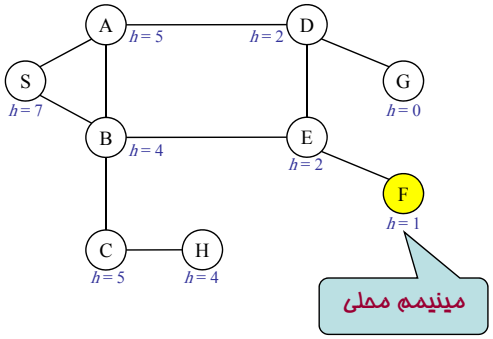


حالت فعلی

بهترین همسایه

حالت همسایه

مثال: جستجوی تپه نوردی

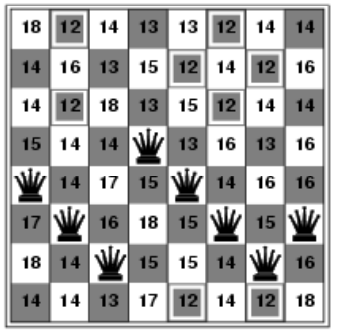


حالت فعلی

بهترین همسایه

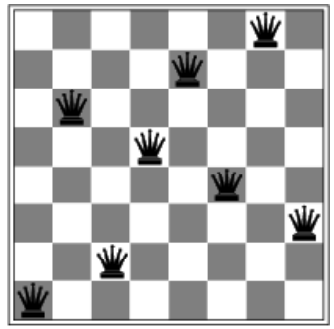
حالت همسایه

جستوی تپه نوردی : مثال ۸ وزیر



$h = 17$

۵ حرکت

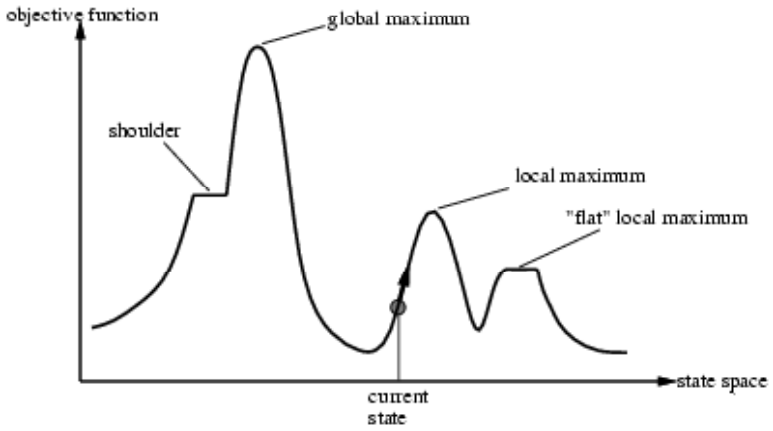


$h = 1$

h = تعداد جفت وزیرهایی که بطور مستقیم و یا بطور غیر مستقیم یکریگر را تهدید می کنند.

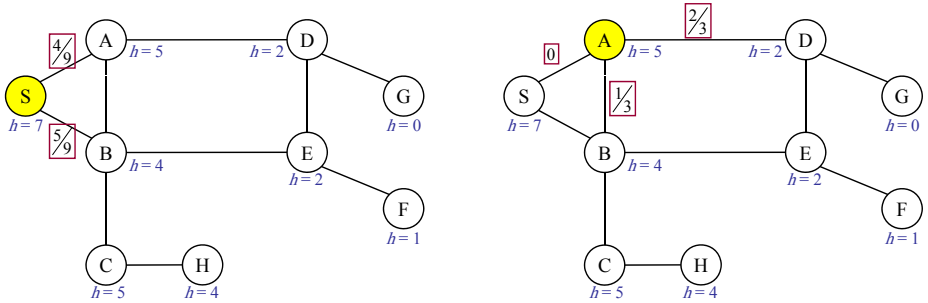
تپه نوردی (ادامه)

- **مشکل**: بسته به حالت اولیه مسأله ممکن است در ماکزیمم محلی گیر کند.



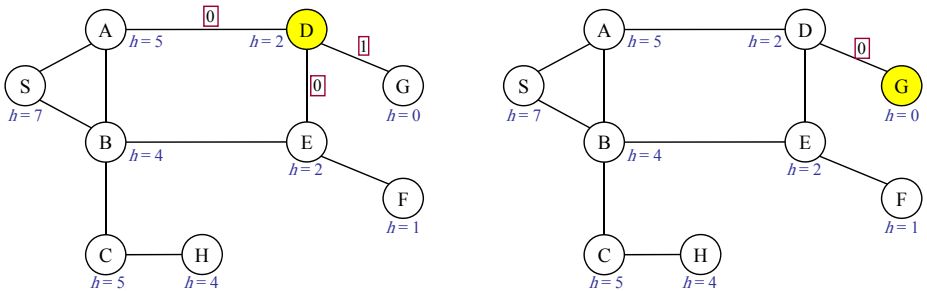
انواع دیگر تپه نوردی (تپه نوردی اتفاقی)

- تپه نوردی اتفاقی
  - انتخاب تصادفی در میان حرکت های رو به بالا
  - احتمال انتخاب می تواند متناسب با شیب حرکت تغییر کند



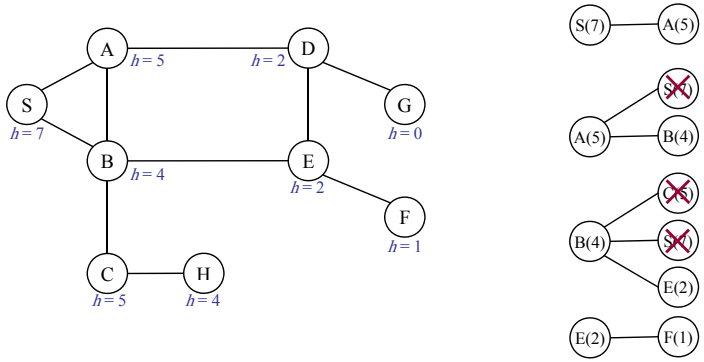
انواع دیگر تپه نوردی (تپه نوردی اتفاقی)

- تپه نوردی اتفاقی
  - انتخاب تصادفی در میان حرکت های رو به بالا
  - احتمال انتخاب می تواند متناسب با شیب حرکت تغییر کند



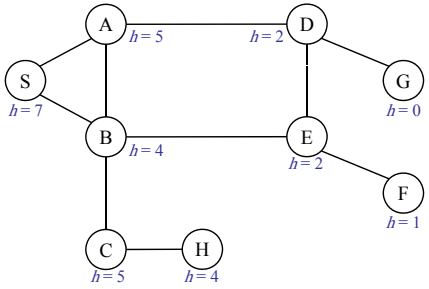
انواع دیگر تپه نوردی (تپه نوردی اولین انتخاب)

- تپه نوردی اولین انتخاب
  - همان تپه نوردی اتفاقی که حالت های بعدی را به طور تصادفی تولید می کند تا یکی از آنها بهتر از حالت فعلی باشد



انواع دیگر تپه نوردی (تپه نوردی با شروع مجدد تصادفی)

- تپه نوردی با شروع مجدد تصادفی (ایده: اگر شکست خوردی دوباره سعی کن)
  - سعی می کند که از گیر افتادن در ماکزیمم محلی اجتناب کند



S → B → E → F ✗

C → B → E → F ✗

H ✗

C → H ✗

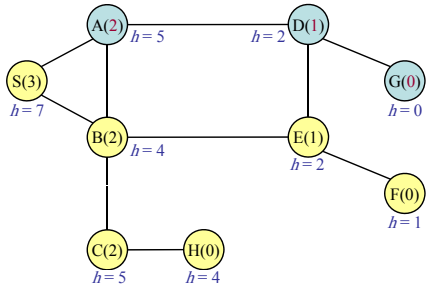
A → D → G

مشکل: معمولاً در یک مسأله در دنیای واقعی، قبل از حل مسأله جواب بهینه را نمی دانیم !

راه حل: تکرار تپه نوردی تا زمانی که وقت داریم.

انواع دیگر تپه نوردی (تپه نوردی با شروع مجدد تصادفی)

- محاسبه تعداد متوسط مراحل در تپه نوردی با شروع تصادفی مجدد
  - تعداد تکرارها = عکس احتمال موفقیت
  - تعداد متوسط مراحل = تعداد شکست ها * تعداد متوسط مراحل شکست + ۱ * تعداد متوسط مراحل موفقیت



احتمال موفقیت = $1/3$

تعداد تکرارها = 3

تعداد متوسط مراحل موفقیت = 1

تعداد متوسط مراحل شکست = $8/6$

$(3-1)\times 8/6 + 1\times 1 \cong 4$

Simulated Annealing

- **ایده**: از ماکزیمم های محلی با انجام حرکت های« بد» فرار کن

اما به تدریج اندازه و تعداد حرکات بد (به سمت پایین) را کم کن

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps
  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(Δ E/T)
```


آنلینگ شبیه سازی شده

- پیشروی مانند تپه نوردی می باشد، اما در هر مرحله حالت بعدی به طور تصادفی انتخاب می شود.
- اگر حالت بعدی انتخاب شده بهتر باشد، همواره به آن حالت بعدی خواهیم رفت.
- در غیر این صورت، تنها با یک احتمال به آن حالت خواهیم رفت و این احتمال به صورت نمایی کاهش می یابد.
- تابع احتمال:

$e^{\Delta E/T}$

T: تعداد مراحل بهبود راه حل

ΔE: میزان کاهش در هر مرحله

خصوصیات آنلینگ شبیه سازی شده

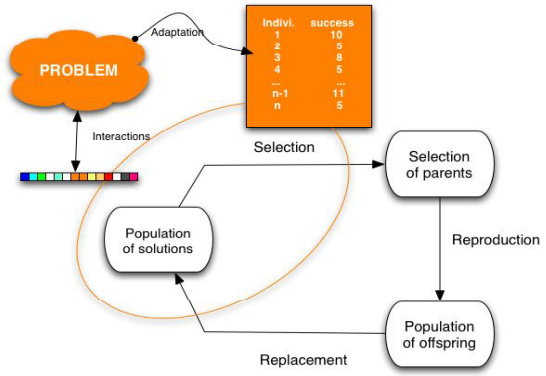
- در مقادیر بالاتر T، احتمال انجام حرکات بد (رفتن به سمت پایین)بیشتر است. (مانند جستجوی تصادفی رفتار می کند)
- با کاهش T این احتمال کاهش یافته و در $T=0$ این احتمال به صفر می رسد. (مانند تپه نوردی رفتار می کند)
- می توان ثابت کرد که اگر T به به اندازه کافی آرام کاهش بیابد، جستجوی SA یک پاسخ بهینه (global optimum) با احتمالی که به سمت یک میل می کند خواهد یافت.
- کاربردها:
  - حل مسائل VLSI
  - برنامه ریزی
  - اعمال بهینه سازی
  - زمانبندی خطوط هوایی

Local beam search
(جستجوی محلی دسته ای)

- شروع با k حالت که به طور تصادفی ایجاد شده اند.
- در هر تکرار، تمام فرزندان برای هر k حالت تولید می شوند.
- اگر یکی از آنها حالت هدف بود جستجو متوقف می شود و در غیر این صورت از میان لیست کامل فرزندان k تا از بهترین ها انتخاب می شوند و مرحله بالا تکرار می شود.

الگوریتم ژنتیک

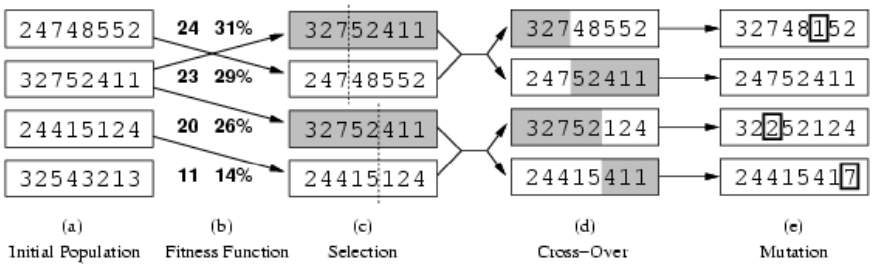
- نوعی از local beam search به همراه ترکیب جنسی



الگوریتم های ژنتیک

- یک حالت بعدی با ترکیب دو حالت پدر ایجاد می شود.
- شروع با k حالت تصادفی (جمعیت)
- یک حالت با یک رشته بر روی یک مجموعه محدود بازنمایی می شود (اغلب رشته ای از صفر و یک) – (کروموزوم)
- تابع ارزیابی (تابع برازندگی – Fitness function) : مقادیر بالاتری را برای حالات بهتر ایجاد می کند.
- نسل بعدی جمعیت با انجام اعمال زیر روی جمعیت فعلی تولید می شود:
  - انتخاب (Selection)
  - آمیزش (Crossover)
  - جهش (Mutation)

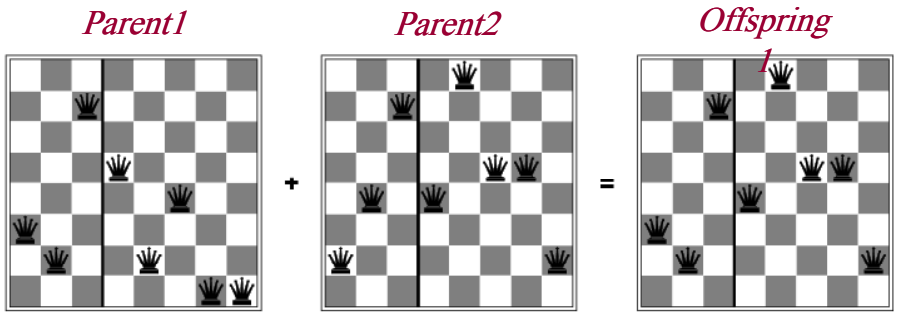
الگوریتم های ژنتیک



- تابع برازندگی : تعداد جفت وزیر هایی که یکدیگر را تهدید نمی کنند
  (مینیمم: صفر، ماکزیمم: 8 * 7/2 = 28)
- 24 / (24+23+20+11) = 31%
- 23 / (24+23+20+11) = 29%

الگوریتم های ژنتیک

- یک مثال برای عمل Crossover

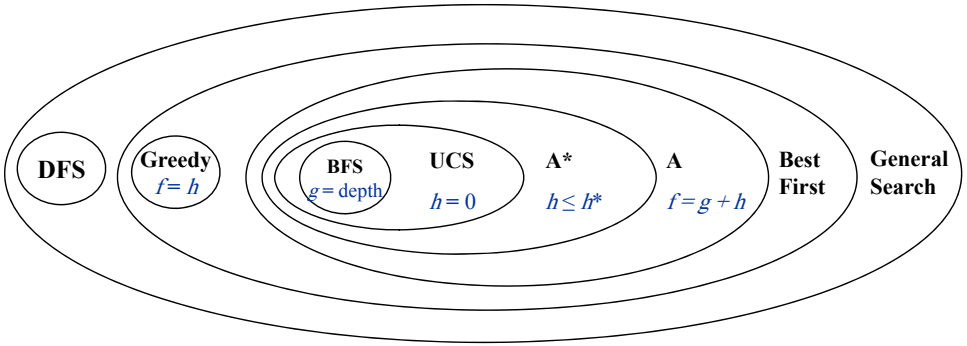


(3, 2, 7, 5, 2, 4, 1, 1) + (2, 4, 7, 4, 8, 5, 5, 2) = (3, 2, 7, 4, 8, 5, 5, 2)

الگوریتم ژنتیک

```
function GENETIC-ALGORITHM( population, FITNESS-FN) returns an individual
    input: population, a set of individuals
           FITNESS-FN, a function that measures the fitness of an individual

    repeat
        new_population ← empty set
        loop for i from 1 to SIZE(population) do
            x ← RANDOM_SELECTION(population, FITNESS_FN)
            y ← RANDOM_SELECTION(population, FITNESS_FN)
            child ← REPRODUCE(x,y)
            if (small random probability) then child ← MUTATE(child)
            add child to new_population
        population ← new_population
    until some individual is fit enough or enough time has elapsed
    return the best individual in population, according to FITNESS-FN
```


دسته بندی استراتژی های جستجو

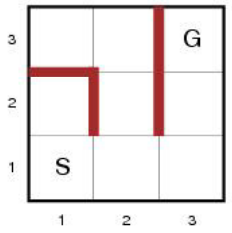


مسائل اکتشافی

- تا کنون تمام الگوریتم ها offline بودند
  - Offline = راه حل قبل از اجرای آن معین است
  - Online = محاسبه و عمل بصورت یک در میان
  - جستجوی online برای محیط های پویا و نیمه پویا ضروری می باشد
    - در نظر گرفتن تمامی امکان های مختلف غیر ممکن است
- استفاده شده در مسائل اکتشافی
  - حالت ها و اعمال ناشناخته
  - مثال: یک روبات در یک محیط جدید، یک نوزاد و ...

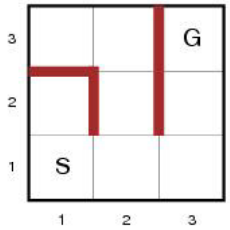
مسائل جستجوی online

- دانش عامل:
  - عمل (ها): لیست اعمال مجاز در حالت s
  - $c(s, a, s')$: تابع هزینه گام (پس از تعیین s')
  - GOAL-TEST(s)
- عامل می تواند حالت قبلی را تشخیص دهد
- اعمال قطعی هستند
- دستیابی به هیوریستیک قابل قبول $h(s)$
- مانند فاصله مانهاتانی

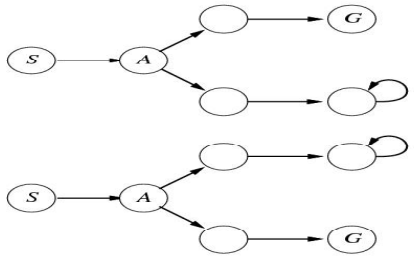


مسائل جستجوی online

- هدف: رسیدن به حالت هدف با حداقل هزینه
  - هزینه = کل هزینه مسیر پیموده شده
  - نسبت رقابتی = مقایسه هزینه با هزینه مسیر راه حل در حالتی که فضای جستجو شناخته شده باشد
  - می تواند نا محدود باشد
  - در مواردی که عامل به طور تصادفی به یک بن بست می رسد



استدلال دشمنانه



- یک دشمن را تصور کنید که قادر است در حین کاوش عامل فضای حالت را بسازد
- حالت های ملاقات شده S و A. حالت بعدی؟
  - در یکی از فضاهای حالت شکست می خورد
  - هیچ الگوریتمی نمی تواند از بن بست ها در تمامی فضاهای حالت اجتناب کند

عامل های جستجوی Online

- عامل یک نقشه از محیط نگهداری می کند
  - نقشه بر اساس ورودی ادراکی بهنگام می شود
  - از این نقشه برای انتخاب عمل بعدی استفاده می شود
- به تفاوت مثلا با A* دقت کنید
  - یک نسخه online تنها می تواند گرهی را گسترش دهد که به طور فیزیکی در آن قرار داشته باشد

Online DFS-Agent

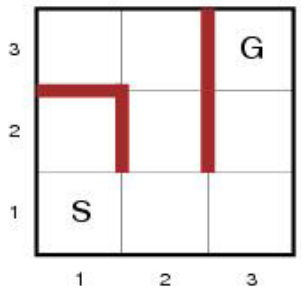
function ONLINE-DFS-AGENT(*s'*) **returns** an action
  input: *s'*, a percept identifying current state
  static: *result*, a table indexed by action and state, initially empty
    unexplored, a table that lists for each visited state, the action not yet tried
    unbacktracked, a table that lists for each visited state, the backtrack not yet tried
    s,a, the previous state and action, initially null

  if GOAL-TEST(*s'*) **then return** stop
  if *s'* is a new state **then** *unexplored*[*s'*] ← ACTIONS(*s'*)
  if *s* is not null **then do**
    result[*a,s*] ← *s'*
    add *s* to the front of *unbackedtracked*[*s'*]
  if *unexplored*[*s'*] is empty **then**
    if *unbacktracked*[*s'*] is empty **then return** stop
    else *a* ← an action *b* such that *result*[*b*, *s'*]=POP(*unbacktracked*[*s'*])
  else *a* ← POP(*unexplored*[*s'*])
  s ← s'
  return *a*

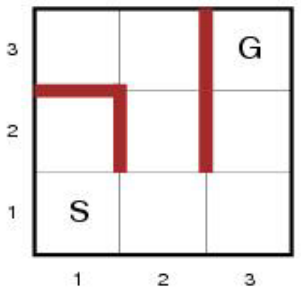
جستجوی عمقی online ، مثال

- مساله مسیر پر پیچ و خم بر روی یک صفحه ۳*۳
- حالت اولیه: $s'=(1, 1)$
- Result و Unexplored(UX) و Unbacktracked(UB) و...

  تهی هستند
- S و a نیز تهی هستند



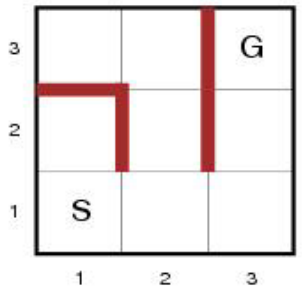
جستجوی عمقی online، مثال



- GOAL-TEST((1, 1))?
  - *S* not = *G* thus false
- (1,1) a new state?
  - True
  - ACTION((1,1)) -> UX[(1,1)]
    - {RIGHT,UP}
- *s* is null?
  - True (initially)
- UX[(1,1)] empty?
  - False
- POP(UX[(1,1)]) -> *a*
  - *a* = UP
- *s* = (1,1)
- Return *a*

جستجوی عمقی online، مثال

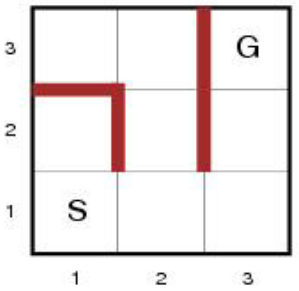
S'=(1,1)



- GOAL-TEST((1,1))?
  - S not = G thus false
- (1,1) a new state?
  - false
- *s* is null?
  - false (*s* = (2, 1))
  - result [DOWN, (2, 1)] <- (1,1)
  - UB[(1,1)] = {(2,1)}
- UX[(1,1)] empty?
  - False
- *a* = RIGHT, *s* = (1,1)
- return *a*

جستجوی عمقی online، مثال

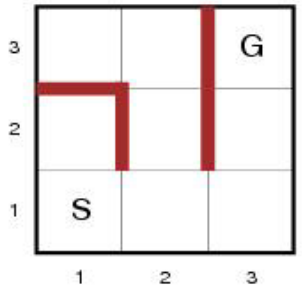
S'=(1,2)



- GOAL-TEST((1,2))?
  - S not = G thus false
- (1,2) a new state?
  - True, UX[(1,2)]={RIGHT,UP,LEFT}
- *s* is null?
  - false (s = (1,1))
  - result [RIGHT, (1,1)] <- (1, 2)
  - UB[(1,2)] = {(1,1)}
- UX[(1, 2)] empty?
  - False
- *a* = LEFT, *s* =(1, 2)
- return *a*

جستجوی عمقی online، مثال

S'=(1,1)



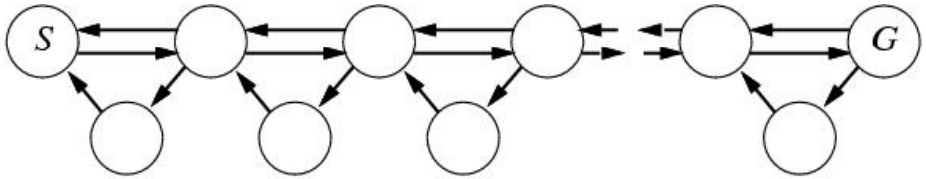
- GOAL-TEST((1, 1))?
  - S not = G thus false
- (1, 1) a new state?
  - false
- *s* is null?
  - false (*s* = (1, 2))
  - result [LEFT, (1, 2)] <- (1, 1)
  - UB[(1, 1)]={(1, 2), (2, 1)}
- UX[(1, 1)] empty?
  - True
  - UB[(1, 1)] empty? False
- *a* = *b* for *b* in result [*b*, (1,1)] = (1, 2)
  - *b* = RIGHT
- *a* = RIGHT, s = (1, 1) …

جستجوی عمقی online، مثال

- بدترین حالت: هر گره دو بار ملاقات شده است
- یک عامل ممکن است در حالی که به پاسخ نزدیک است، یک راه طولانی را بپیماید
- یک روش عمیق کننده تکرای online می تواند این مشکل را حل کند
- جستجوی عمقی online فقط وقتی کار می کند که اعمال قابل برگشت باشند

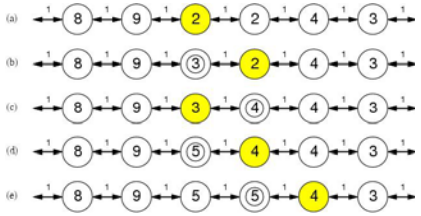
جستجوی محلی online

- تپه نوردی online می باشد
  - یک حالت ذخیره شده است
- عملکرد بد به دلیل وجود ماکزیمم های محلی
  - شروع مجدد تصادفی غیر ممکن است
- راه حل: حرکت تصادفی باعث کاوش می شود (می تواند حالات بسیاری را به صورت نمایی تولید کند)



جستجوی محلی online

- راه حل ۲: اضافه نمودن حافظه به تپه نورد
- ذخیره بهترین تخمین فعلی $H(s)$ از هزینه رسیدن به هدف
- $H(s)$ در ابتدا تخمین هیوریستیک $h(s)$ می باشد
- پس از آن بر اساس تجربه بهنگام می شود (شکل پایین)



Learning real-time A*

```
function LRTA*-COST (s, a, s',H) return an cost estimate
    if s' is undefined the return h(s)
    else return c(s, a, s') + H[s']

function LRTA*-AGENT (s') return an action
    input: s', a percept identifying current state
    static: result, a table indexed by action and state, initially empty
            H, a table of cost estimates indexed by state, initially empty
            s,a, the previous state and action, initially null

    if GOAL-TEST (s') then return stop
    if s' is a new state (not in H) then H[s'] ← h(s')
    unless s is null
            result[a,s] ← s'
            H[s] ← MIN  LRTA*-COST (s, b, result [b, s], H)
                    b ∈ ACTIONS (s)
     a ← an action b in ACTIONS (s') that minimizes LRTA*-COST (s', b, result [b, s'], H)
    s ← s'
    return a
```

مسائل ارضاء محدودیت
(CSP)

فصل پنجم
سید ناصر رضوی
Email: razavi@Comp.iust.ac.ir
۱۳۸۴

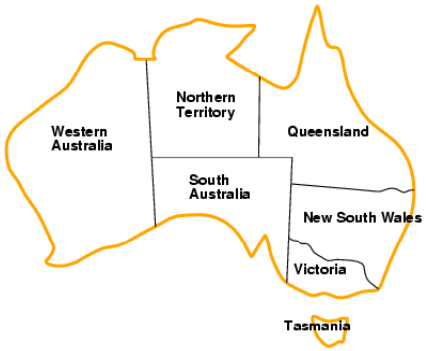
مقدمه

- چند مثال از مسائل ارضاء محدودیت
- کاربرد General Search در حل CSP
- جستجوی عقبگرد (backtracking) برای CSP
- کنترل روبه جلو (forward checking)
- استفاده از هیوریستیک ها در CSP ها
- جستجوی محلی برای مسائل ارضاء محدودیت

مسائل ارضاء محدودیت

- مسأله جستجوی استاندارد
  - حالت یک " جعبه سیاه" است – هر ساختار داده ای که از تابع حالات بعدی، تابع هیوریستیک و تست هدف پشتیبانی کند.
- CSP:
  - حالت توسط متغیرهای X_i تعریف می شود که هر یک مقادیرشان را از یک دامنه D_i اختیار می کنند.
  - تست هدف مجموعه ای از محدودیت ها می باشد که ترکیبات مجاز مقادیر برای زیرمجموعه ای از متغیرها را مشخص می کند.
- مثال ساده ای از یک زبان بازنمایی رسمی
- امکان استفاده از الگوریتم های همه منظوره که نسبت به الگوریتم های استاندارد قدرت بیشتری دارند.

مثال: رنگ آمیزی نقشه

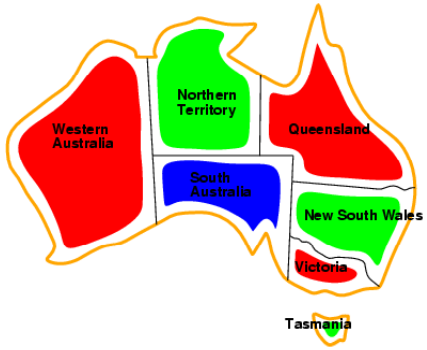


- متغیرها: WA, NT, Q, NSW, V, SA, T
- دامنه ها: {red, green, blue}
- محدودیت ها: نواحی همسایه باید رنگ متفاوتی داشته باشند

مثلا: $WA \neq NT$ ، به بیان دیگر:

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

مثال: رنگ آمیزی نقشه

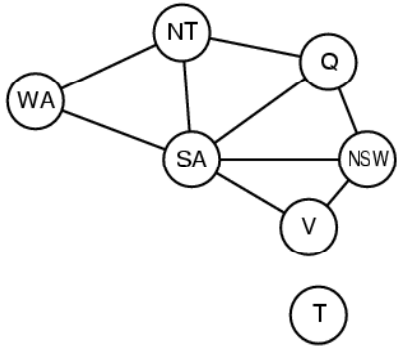


- راه حل ها انتساب های **کامل** و **سازگار** می باشند.
- مثال:

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

گراف محدودیت

- گراف محدودیت:
  - گره ها متغیرها را نشان می دهند.
  - یالهای گراف محدودیت ها ی بین متغیرها را نشان می دهند.



مزایای بیان مسأله به صورت *CSP*

- به دلیل نمایش استاندارد حالت ها (مجموعه متغیرهایی با مقدارشان)، می توان تابع Successor و آزمون هدف را به شکل کلی نوشت به طوریکه برای هر CSP قابل اعمال باشد.

- می توان هیوریستیک های کلی و کارایی ایجاد کرد که نیاز به تخصص اضافی در دامنه خاص مسأله نداشته باشند.

انواع مسائل CSP

- متغیرهای گسسته
  - دامنه های محدود:
    - n متغیر، اندازه هر دامنه d ← تعداد انتساب های کامل $O(d^n)$
    - مثال: CSP های بولین ، n- وزیر ، رنگ آمیزی نقشه و ...
  - دامنه های نامحدود:
    - اعداد صحیح، رشته ها و ...
    - مثال: زمانبندی کارها- متغیرها، زمان شروع/پایان هر کار هستند.
    - نیاز به زبان محدودیت دارند. مثال: $StartJob_1 + 5 \leq StartJob_3$
- متغیرهای پیوسته
  - مثال: زمان های شروع و پایان مشاهدات تلسکوپ فضایی هابل
  - محدودیت های خطی که توسط برنامه ریزی خطی قابل حل در زمان چندجمله ای می باشند.

انواع محدودیت ها

- **یکانی (Unary) :** روی یک متغیر تعریف می شود،
  - مثال: $SA \neq green$
- **دودویی (Binary):** محدودیت شامل یک زوج از متغیرها می باشد،
  - مثال: $SA \neq WA$
- **مرتبه بالاتر (Higher-order):** محدودیت شامل سه یا بیشتر متغیر است،
  - مثال: محدودیت های موجود در ستون های مسائل ریاضیات رمزی
  - در صورت متناهی بودن دامنه، می تواند به تعدادی محدودیت دودیی کاهش یابد.

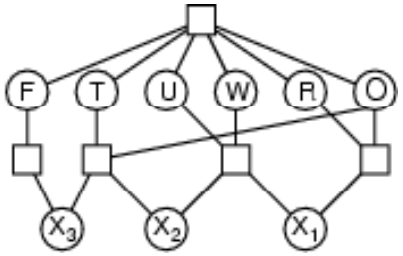
- محدودیت های مثال های بالا همگی از نوع **مطلق** می باشند. نقض یک محدودیت مطلق به معنای حذف یک راه حل بالقوه می باشد.

محدودیت اولویت دار (نرم):

مثلا، قرمز بر سبز ارجحیت دارد. اغلب بوسیله در نظر گرفتن هزینه برای انتساب متغیرها قابل بازنمایی می باشند. ← Constraint Optimization Problem

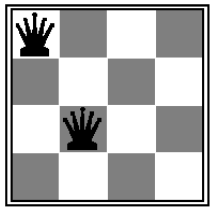
مثال: ریاضیات رمزی

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



- Variables: $FTUW\ R\ O\ X_1\ X_2\ X_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints: $Alldiff(F,T,U,W,R,O)$
-
  - $O + O = R + 10 \cdot X_1$
  -
  - $X_1 + W + W = U + 10 \cdot X_2$
  -
  - $X_2 + T + T = O + 10 \cdot X_3$

مثال: ۴-وزیر به عنوان *CSP*



در هر ستون یک وزیر فرض کنید.

متغیرها : Q_1, Q_2, Q_3, Q_4

دامنه ها: $D_i=\{1, 2, 3, 4\}$

محدودیت ها:

$Q_i \neq Q_j$ (دو وزیر نمی توانند در یک سطر قرار بگیرند $Q_1 = 1 \quad Q_2 = 3$

$|Q_i - Q_j| \neq |i - j|$ (یا در یک قطر)

هر محدودیت به مجموعه ای از مقادیر مجاز برای متغیرهایش ترجمه می شود، مثلا:

مقادیر $(Q1, Q2)$ می توانند مانند زیر باشند:

$(1, 3), (1, 4), (2, 4), (3, 1), (4, 1), (4, 2)$

مسائل ارضاء محدودیت در دنیای واقعی

- مسائل انتسابی (assignment)
  - مثلا، چه کسی چه کلاسی را درس می دهد.
- مسائل تعیین جدول زمانبندی (Timetabling)
  - مثلا، کدام کلاس، کجا و کی ارائه می شود؟
- زمانبندی حمل و نقل (transportation)
- زمانبندی کارخانه (factory scheduling)

- توجه: بسیاری از مسائل در دنیای واقعی شامل متغیرهایی با مقادیر حقیقی می باشند.

فرموله سازی جستجوی استاندارد(افزایشی)

- حالات توسط مقادیری که تا کنون انتساب یافته اند، تعریف می شوند.
- حالت اولیه: تمام متغیرها بدون مقدار، یعنی انتساب تهی {}
- عملگرها: انتساب مقدار به یک متغیر بدون مقدار به طوری که با انتساب فعلی درگیری ایجاد نکند.← در صورت عدم وجود انتساب های مجاز شکست می خورد.
- تست هدف: انتساب فعلی کامل باشد.
- هزینه مسیر: هزینه یکسان برای تمام مراحل

۱) برای تمام CSP ها یکسان است!
۲) هر پاسخ در عمق n با n متغیر ظاهر می شود.← استفاده از جستجوی عمقی
۳) در عمق l، فاکتور انشعاب (b) برابر است با $d*(n-l)$.
بنابراین تعداد برگهای درخت جستجو برابر است با $n! * d^n$!!!!
$(nd) * [(n-1)d] * [(n-2)d] * \ldots d = n! d^n$
مثلا در ۸-وزیر: $8!8^8$

پیاده سازی

datatype CSP-STATE
components: UNASSIGNED, a list of variables not yet assigned
ASSIGNED, a list of variable that have value

datatype CSP-VAR
components: NAME, for i/o purpose
DOMAIN, a list of possible values
VALUE, current value (if any)

- محدودیت ها را می توان به دو روش نمایش داد:
  - **صریح:** به عنوان مجموعه ای از مقادیر مجاز، یا
  - **ضمنی:** بوسیله تابعی که ارضاء محدودیت ها را بررسی می کند.

پیچیدگی این رهیافت

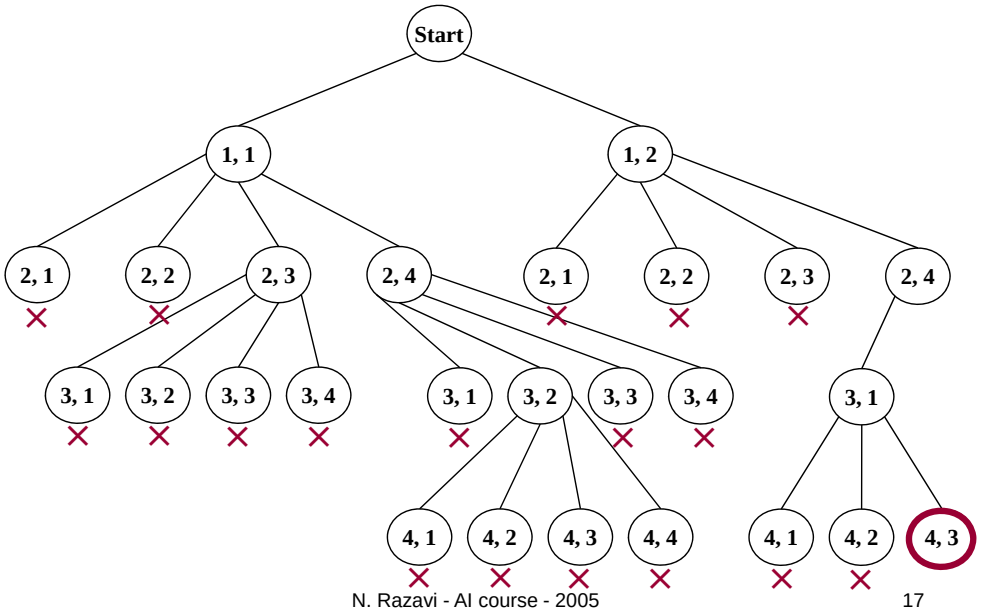
- حداکثر عمق فضا (m) = ؟؟ n (تعداد متغیرها)
- عمق حالت پاسخ (d) = ؟؟ n (تمام متغیرها مقدار دارند)
- الگوریتم جستجو؟؟ جستجوی اول-عمق
- فاکتور انشعاب b ؟؟ $\Sigma_i|D_i|$ (در بالای درخت)

- این ها می توانند با توجه به نکات زیر بهبود یابند:
۱) ترتیب انتساب مقادیر به متغیرها اهمیت ندارد، پس بسیاری از مسیرها معادل یکدیگر می باشند.(در ۸-وزیر اندازه فضای حالت از $8!8^8$ به 8^8 کاهش می یابد)
۲) انتساب های بعدی نمی توانند یک محدودیت نقض شده را تصحیح کنند.(مثلا اگر در ۸-وزیر دو وزیر اول در یک ردیف قرار بگیرند، جستجوی عمقی 8^6 حالت باقیمانده را امتحان می کند تا بفهمد پاسخی وجود ندارد)← جستجوی عقب گرد.

Backtracking

- انتساب متغیرها جابه جایی پذیر است مثلا $[Q_1=1,Q_2=3]$ برابر است با $[Q_2=3,Q_1=1]$
- در هر گره تنها باید انتساب های یک متغیر در نظر گرفته شود؛پس، b=d و تعداد برگها=d^n
- جستجوی اول-عمق با انتساب های یک متغیر جداگانه در CSP ها، backtracking نام دارد.
- قرار دادن یک آزمون قبل از بسط گره ها که بررسی می کند آیا تاکنون محدودیتی نقض شده یا خیر. اگر محدودیتی نقض شده باشد، دیگر این گره بسط داده نمی شود و جستجو به عقب باز می گردد.
- می تواند n-وزیر را تا حدود n=25 حل کند.

مثال: جستجوی عقبگرد برای ۴-وزیر



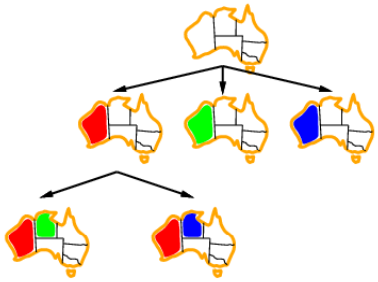
جستجوی عقبگرد

```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
   return RECURSIVE-BACKTRACKING({}, csp)
function RECURSIVE-BACKTRACKING( assignment,csp) returns a solution, or
failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE( Variables[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment according to Constraints[csp] then
         add { var = value } to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failue then return result
         remove { var = value } from assignment
   return failure
```

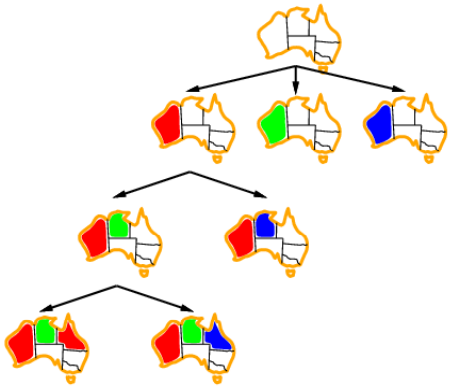

جستجوی عقبگرد

جستجوی عقبگرد

جستجوی عقبگرد



جستجوی عقبگرد



بهبود کارآیی *backtracking*

- افزایش سرعت توسط روش های جستجوی همه منظوره:

۱- در مرحله بعد باید به کدام متغیر مقدار داده شود؟

۲- مقادیر آن باید به چه ترتیبی امتحان شوند؟

۳- آیا می توان شکست های حتمی را زودتر تشخیص داد؟

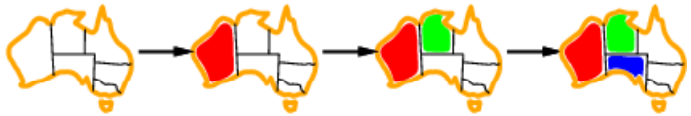
فرض کنید در عقبگرد در مسأله ۸-وزیر، ۶ وزیر اول به گونه ای قرار گرفته اند که قرار دادن هشتمین وزیر را غیر ممکن می سازند. عقب گرد تمام ممکانهای ممکن برای وزیر هفتم را چک می کند، اگرچه مسأله غیر قابل حل می باشد.

۴- آیا می توان از ساختار مسأله بهره گرفت؟

متغیر با بیشترین محدودیت

- متغیر با بیشترین محدودیت :

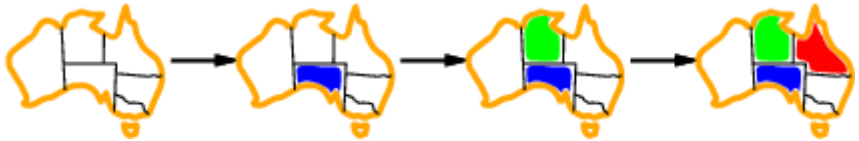
– متغیری را انتخاب کن که کمترین مقادیر معتبر را دارد.



– هیوریستیک **کمترین مقادیر باقیمانده** (MRV)

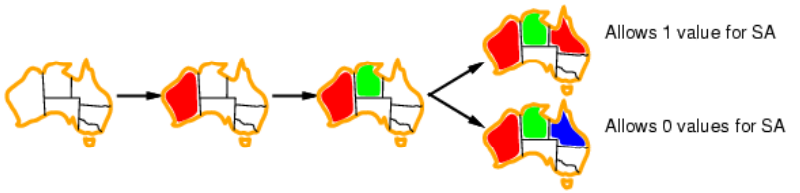
متغیر با بیشترین محدودیت

- حل درگیری (tie) میان متغیرهایی که بیشترین محدودیت را دارند
- متغیر با بیشترین محدودیت: هیوریستیک درجه
  - متغیر با بیشترین محدودیت را روی مقادیر باقیمانده انتخاب کن



مقدار با کمترین محدودیت (Least constraining value)

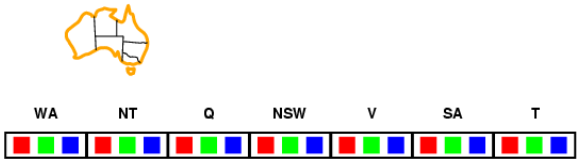
- برای یک متغیر، مقداری را که کمترین محدودیت را ایجاد می کند انتخاب کن.
  - مقداری که کمترین مقادیر را از متغیرهای باقیمانده حذف می کند



- با ترکیب این هیوریستیک ها مسأله ۱۰۰۰-وزیر قابل حل می شود.

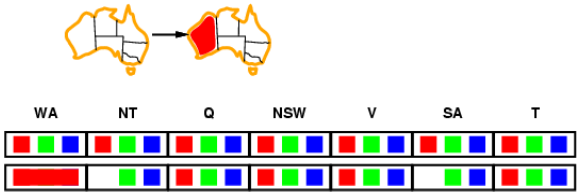
بررسی رو به جلو (Forward checking)

- **ایده:**
  - مراقب مقادیر مجاز باقیمانده برای متغیرهای بدون مقدار باش.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده ای ندارد، به جستجو پایان بده.



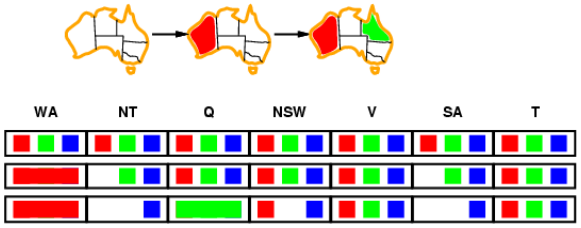
بررسی رو به جلو (Forward checking)

- **ایده:**
  - مراقب مقادیر مجاز باقیمانده برای متغیرهای بدون مقدار باش.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده ای ندارد، به جستجو پایان بده.



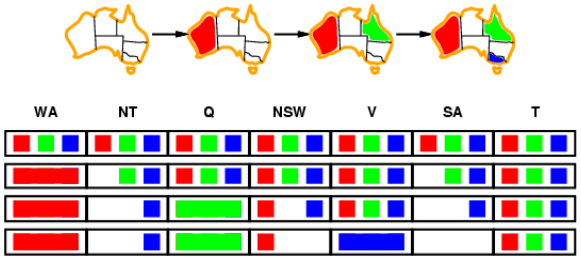
بررسی رو به جلو
(Forward checking)

- **ایده:**
  - مراقب مقادیر مجاز باقیمانده برای متغیرهای بدون مقدار باش.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده ای ندارد، به جستجو پایان بده.



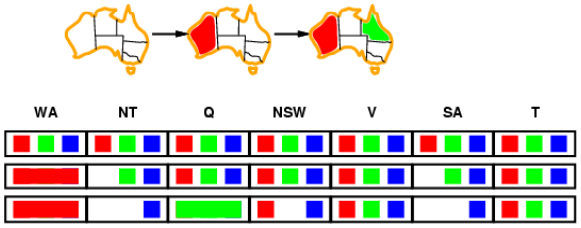
بررسی رو به جلو
(Forward checking)

- **ایده:**
  - مراقب مقادیر مجاز باقیمانده برای متغیرهای بدون مقدار باش.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده ای ندارد، به جستجو پایان بده.



انتشار محدودیت

- بررسی رو به جلو محدودیت ها را از متغیرهای انتساب یافته به متغیر های انتساب نیافته منتشر می کند، اما تمام شکست ها را نمی تواند در زود ترین زمان ممکن تشخیص دهد.

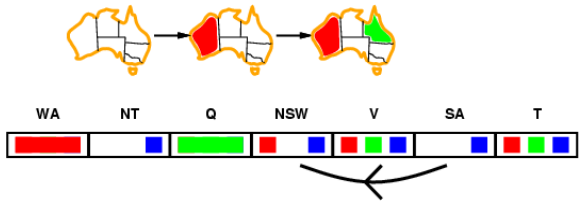


- NT و SA هر دو نمی توانند آبی باشند!
- انتشار محدودیت به طور مکرر بر محدودیت ها به طور محلی تأکید دارد.

سازگاری کمان
(Arc consistency)

- ساده ترین شکل انتشار، هر کمان را سازگار می کند.
- $X \rightarrow Y$ سازگار است اگر و فقط اگر

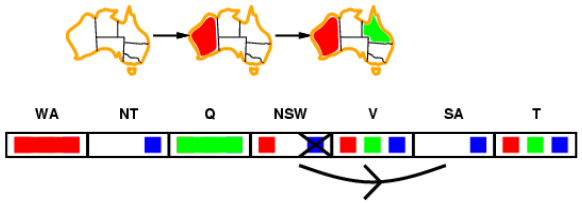
بازاء هر مقدار x برخی مقادیر مجاز y موجود باشند.



سازگاری کمان
(Arc consistency)

- ساده ترین شکل انتشار، هر کمان را سازگار می کند.
- X→Y سازگار است اگر و فقط اگر

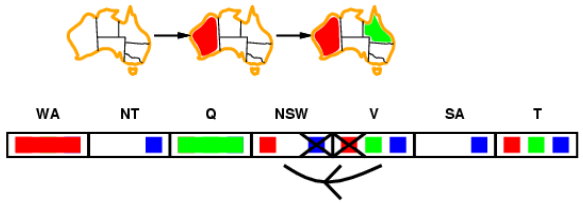
بازاء هر مقدار x برخی مقادیر مجاز y موجود باشند.



سازگاری کمان
(Arc consistency)

- ساده ترین شکل انتشار، هر کمان را سازگار می کند.
- X→Y سازگار است اگر و فقط اگر

بازاء هر مقدار x برخی مقادیر مجاز y موجود باشند.



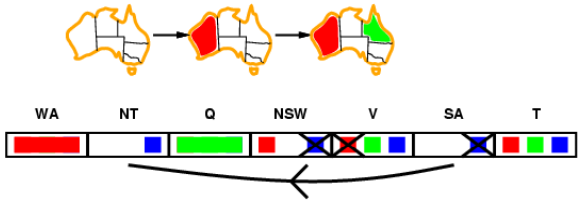
اگر X مقداری را از دست بدهد، همسایه های X نیاز به بررسی مجدد دارند.

www.txt.ir

سازگاری کمان
(Arc consistency)

- ساده ترین شکل انتشار، هر کمان را سازگار می کند.
- X→Y سازگار است اگر و فقط اگر

بازاء هر مقدار x برخی مقادیر مجاز y موجود باشند.



سازگاری کمان شکست ها را زودتر از FC تشخیص می دهد.

و می تواند به عنوان پیش پردازنده و یا بعد از هر انتساب اجرا شود.

www.txt.ir

الگوریتم سازگاری کمان AC-3

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

while *queue* is not empty **do**
$(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
if RM-INCONSISTENT-VALUES(X_i, X_j) **then**
for each X_k **in** NEIGHBORS[X_i] **do**
add (X_k, X_i) to *queue*

function RM-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff remove a value
removed ← *false*
for each x **in** DOMAIN[X_i] **do**
if no value y in DOMAIN[X_j] allows (x,y) to satisfy constraint(X_i, X_j)
then delete x from DOMAIN[X_i]; *removed* ← *true*
return *removed*

- پیچیدگی زمانی: $O(n^2d^3)$

جستجوی محلی برای CSP

- تپه نوردی و SA با حالات کامل کار می کنند؛ یعنی تمام متغیرها باید دارای مقدار باشند.

- برای اعمال آنها بر مسائل CSP:
  - داشتن حالاتی با محدودیتهای نقض شده باید مجاز باشد.
  - عملگرها باید بتوانند به متغیرها مقادیر جدید بدهند.

- انتخاب متغیر: به صورت تصادفی یک متغیر درگیر را انتخاب کن
- هیوریستیک *min-conflicts*:
  - مقداری را انتخاب کن که کمترین تعداد محدودیت ها را نقض می کند،

یعنی، تپه نوردی با هیوریستیک " تعداد کل محدودیت های نقض شده"

الگوریتم *Min-Conflicts*

function MIN-CONFLICTS(*csp*, *max_steps*) **returns** a solution or failure
  inputs: *csp*, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for *csp*
  for *i* = 1 to *max_steps* **do**
    if *current* is a solution for *csp* **then return** *current*
    var ← a randomly chosen, conflicted variable from VARIABLES[*csp*]
    value ← the value *v* for *var* that minimizes CONFLICTS(*var*, *v*, *current*, *csp*)
    set *var* = *value* in *current*
  return *failure*

Figure 5.8 The MIN-CONFLICTS algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.

مثال : ۸-وزیر

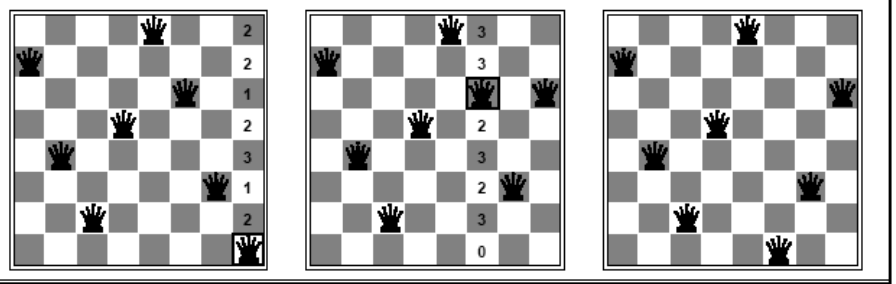


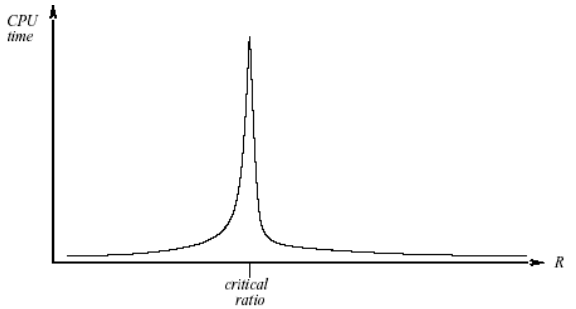
Figure 5.9 A two-step solution for an 8-queens problem using min-conflicts. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflict square, breaking ties randomly.

کارآیی *min-conflicts*

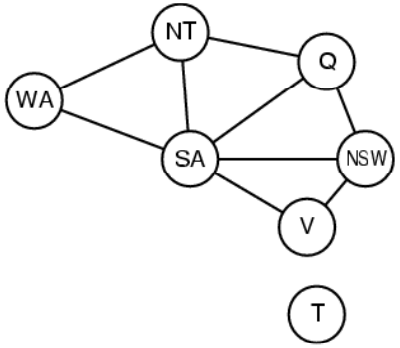
حل n- وزیر با n دلخواه (مثلا 10,000,000) در یک زمان تقریبا ثابت با شروع از یک وضعیت تصادفی با احتمال بالا(میانگین ۵۰ مرحله)

همین موضوع به نظر می رسد برای هر CSP با شروع تصادفی درست باشد، به غیر از یک ناحیه باریک از نسبت R.

R = (num. of constraints)/(num. of variables)



ساختار مسأله



- Tasmania و جزیره اصلی زیرمسایل مستقل هستند.
- و با توجه به مولفه های همبند در گراف محدودیت قابل شناسایی هستند.

ساختار مسأله (ادامه)

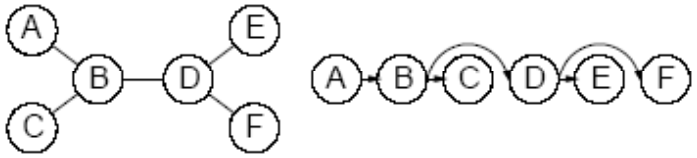
- فرض کنید هر زیرمسأله شامل c متغیر از کل n متغیر باشد
- هزینه راه حل در بدترین حالت خطی(برحسب n) و برابر $d^c \cdot n/c$ می باشد

- مثال: $n = 80,\ d = 2,\ c = 20$

- 2^{80} = 4 billion years at 10 million nodes/sec
- 4.2^{20} = 0.4 seconds at 10 million nodes/sec

الگوریتم برای *CSP* های دارای ساختار درختی

۱. یک متغیر را به عنوان ریشه انتخاب کن و سپس متغیرها را از ریشه تا برگها به گونه ای مرتب کن که والد هر گره قبل از آن گره قرار بگیرد.



۲. بازاء j از n تا 2 عمل زیر را انجام بده

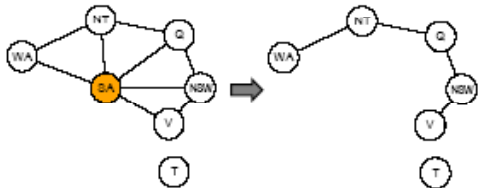
REMOVEINCONSISTENT($Parent(X_j)$, X_j)

۳. برای j از یک تا n ، مقدار X_j را به طور سازگار با $Parent(X_j)$ تعیین کن

پیچیدگی زمانی: $O(n \cdot d^2)$

CSP های دارای ساختار شبه درختی

- شرطی سازی: به یک متغیر مقدار بده؛ دامنه همسایه های آن را هرس کن



- شرط برشی: مجموعه ای از متغیرها (در تمام حالات ممکن) را مقدار دهی کن به طوریکه گراف گراف محدودیت باقیمانده یک درخت شود.

اندازه برش = c ← زمان اجرا $O(d^c \cdot (n - c)\ d^2)$

رهیافت دوم: تجزیه درختی

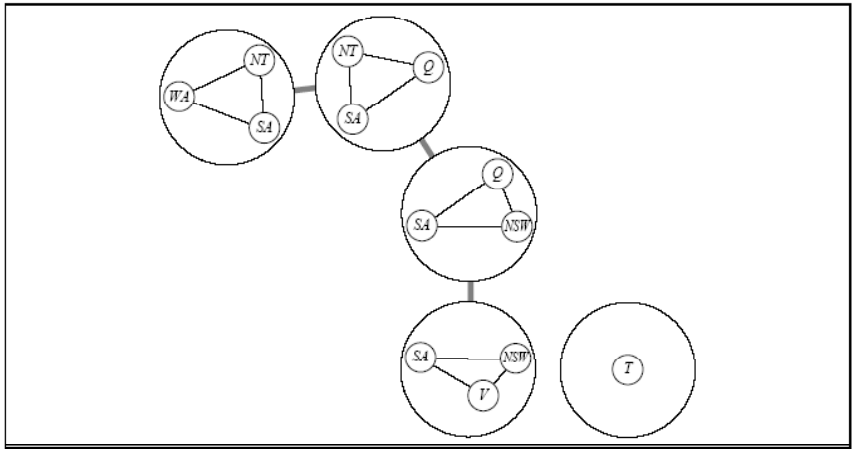


Figure 5.12 A tree decomposition of the constraint graph in Figure 5.11(a).

رهیافت دوم: تجزیه درختی

- شرایط تجزیه درختی:
  - هر متغیر در مسأله اصلی باید حداقل در یک زیرمسأله ظاهر شود
  - اگر دو متغیر به وسیله محدودیتی در مسأله اصلی متصل شده باشند، آنگاه آن دو متغیربا هم(به همراه محدودیت) باید حداقل در یک زیر مسأله ظاهر شوند.
  - اگر متغیری در درخت در دو زیر مسأله ظاهر شده باشد، آنگاه باید در هر زیرمسأله در طول مسیری که زیرمسایل را متصل می کند، ظاهر شود.

رهیافت دوم: تجزیه درختی

- حل تجزیه درختی:
  - هر زیر مسأله را به صورت یک mega-variable در نظر می گیریم که دامنه آن مجموعه تمام راه حل های ممکن آن زیر مسأله می باشد.
  - سپس محدودیت های میان زیر مسایل را با استفاده از الگوریتم کارای درخت حل می کنیم. مثلا اگر پاسخ اولین زیر مسأله انتساب زیر باشد:

$$\{WA = red, SA = blue, NT = green\}$$

آنگاه تنها پاسخ سازگار برای زیر مسأله بعدی انتساب زیر می باشد:

$$\{SA = blue, NT = green, Q = red\}$$

پیچیدگی زمانی: $O(n \cdot d^{w+1})$

خلاصه

- مسائل CSP نوع خاصی از مسائل می باشند:
  - حالات توسط مقادیر مجموعه ای از متغیرها تعریف می شوند.
  - تست هدف توسط محدودیت ها روی مقادیر متغیرها تعریف می شود.
- جستجوی عقبگرد = جستجوی اول-عمق که در هر گره یک متغیر مقدار گرفته باشد.
- هیوریستیک های ترتیب متغیرها و انتخاب مقادیر بسیار کمک کننده می باشند.
- بررسی رو به جلو از انتساب های منجر به شکست جلوگیری می کند.
- انتشار محدودیت کارهای بیشتری برای محدود کردن مقادیر و تشخیص ناسازگاری ها انجام می دهد.
- در عمل معمولا جستجوی محلی min-conflicts مؤثر می باشد.

جستجوی رقابتی

فصل ششم
سید ناصر رضوی
Email: razavi@Comp.iust.ac.ir
۱۳۸۴

مقدمه

- تصمیم های بهینه
- هرس آلفا-بتا
- تصمیم های بلادرنگ و غیر کامل

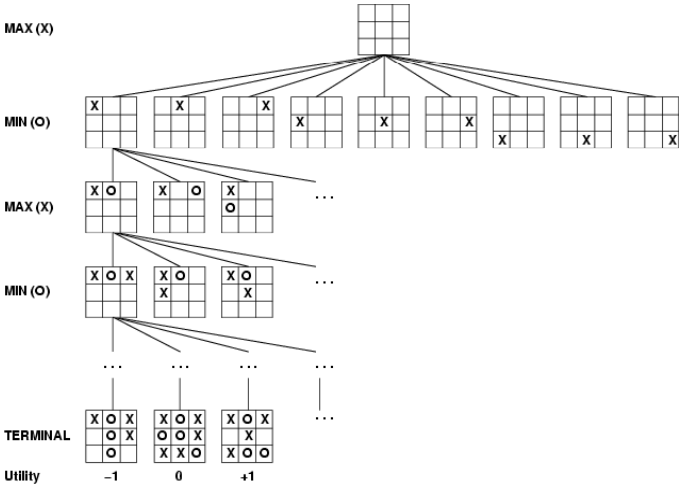
بازی در مقایسه با مسائل جستجو

- رقیب «غیر قابل پیش بینی» ← مشخص کردن یک حرکت برای هر پاسخ ممکن از طرف رقیب
- محدودیت های زمانی ← متاسفانه برای یافتن هدف باید تقریب زد

تصمیم های بهینه در بازی ها

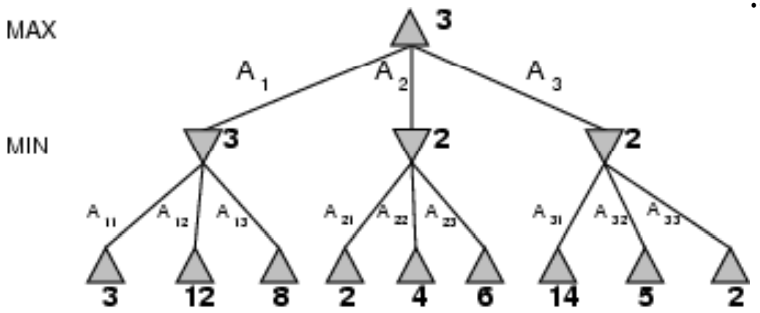
- بازی های دو نفره (MAX و MIN)
- ابتدا MAX بازی می کند، سپس MIN و ...
- بازی به عنوان یک نوع از مسأله جستجو
  - حالت اولیه: شامل موقعیت صفحه و نوبت بازیکن
  - تابع جانشین: لیستی از زوج های (*move, state*) را بر می گرداند
  - تست ترمینال: تعیین کننده پایان بازی (حالت های ترمینال)
  - تابع سودمندی: به هر حالت پایانی یک مقدار عددی می دهد
- درخت بازی: توسط حالت اولیه و حرکت های قانونی برای هر طرف مشخص می شود

درخت بازی (۲-نفره، قطعی)



Minimax

- بازی عالی در بازی های قطعی
- ایده: انتخاب حرکت به موقعیتی با بیشترین مقدار minimax
- = بهترین امتیاز قابل دستیابی در برابر بهترین بازی
- مثال:



استراتژی بهینه

- با داشتن درخت بازی، استراتژی بهینه را می توان با در نظر گرفتن مقدار minimax گره ها تعیین نمود:

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

الگوریتم Minimax

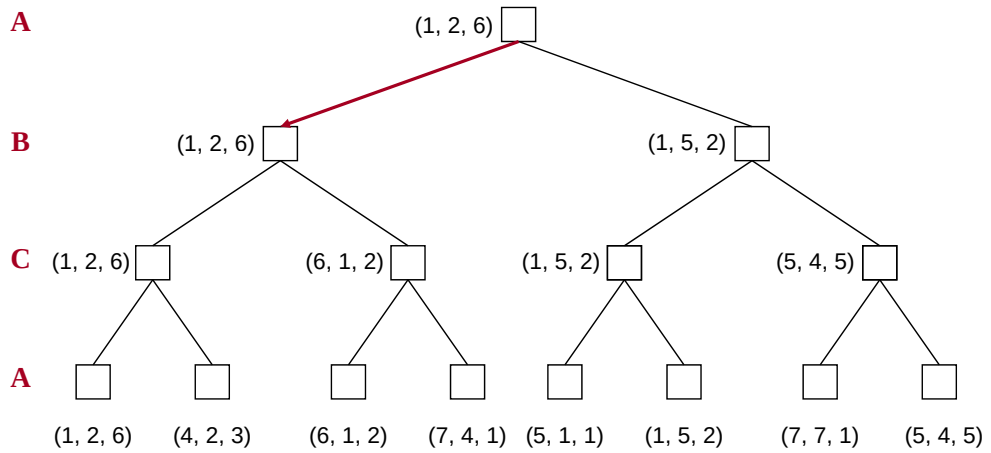
```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

خصوصیات Minimax

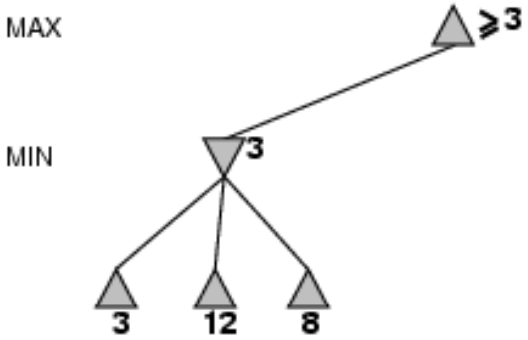
- کامل؟ بله (به شرط محدود بودن درخت)
- بهینه؟ بله (در مقابل رقیبی که بهینه بازی می کند)
- پیچیدگی زمانی؟ $O(b^m)$
- پیچیدگی حافظه؟ $O(bm)$ (کاوش اول-عمق)

- در شطرنج $b \approx 35$ و $m \approx 100$ (در بازی های معقول)
  ← راه حل دقیق کاملا مقرون به صرفه نمی باشد.

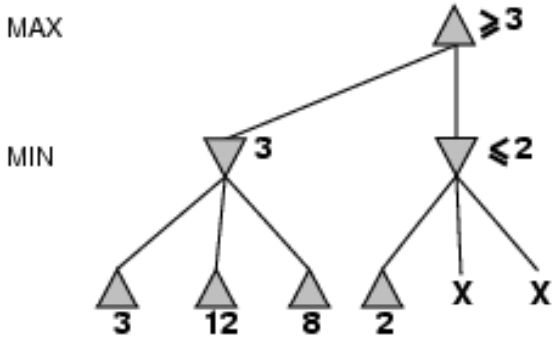
تصمیمات بهینه در بازی های چند نفره



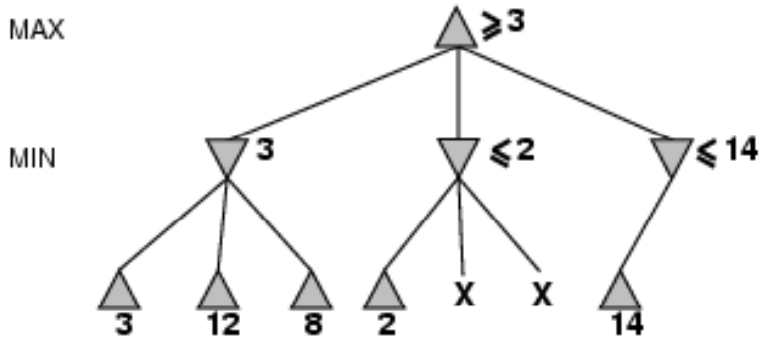
مثال هرس آلفا-بتا



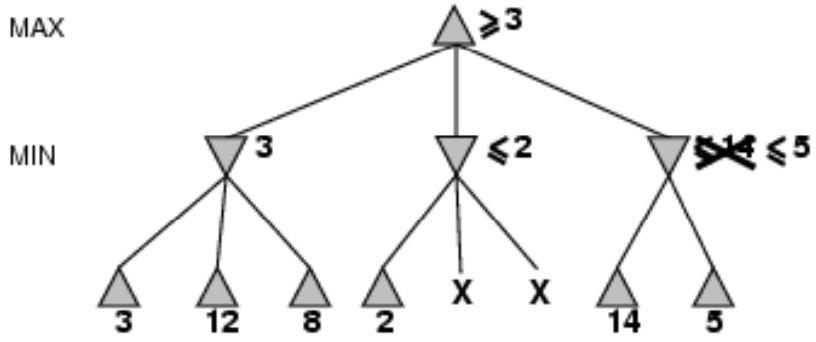
مثال هرس آلفا-بتا



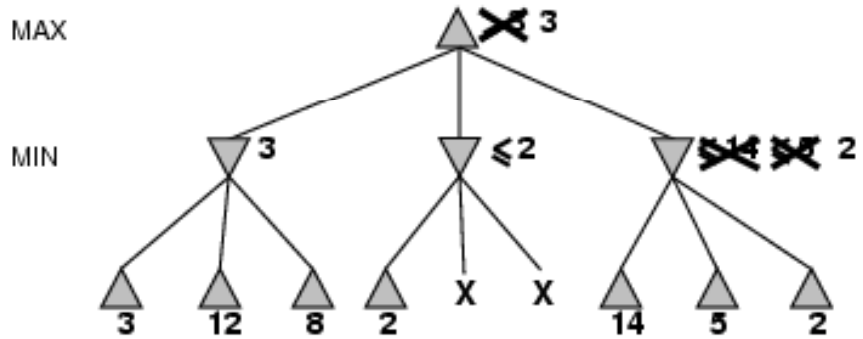
مثال هرس آلفا-بتا



مثال هرس آلفا-بتا



مثال هرس آلفا-بتا



MINIMAX-VALUE (*root*) = max (min (3, 12, 8), min (2, *x*, *y*), min (14, 5, 2))

= max (3, min (2, *x*, *y*), 2)

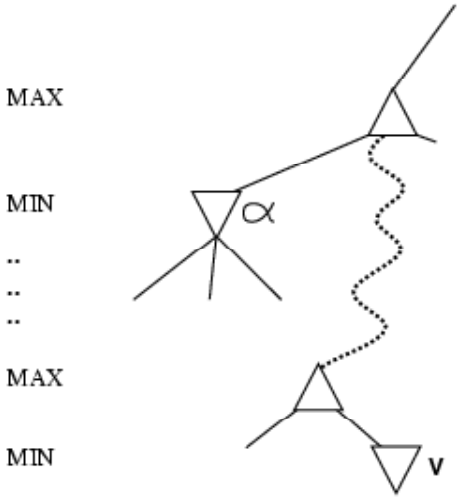
= max(3, *z*, 2) where $z \leq 2$

= 3

خواص آلفا-بتا

- هرس کردن بر روی نتیجه نهایی تاثیر ندارد
- ترتیب خوب حرکت ها میزان تاثیر الگوریتم را بهبود می بخشد
- با «بهترین ترتیب»، پیچیدگی زمانی = $O(b^{m/2})$
- ← عمق جستجوی دو برابر
- یک مثال ساده از ارزش استدلال در مورد محاسبات مرتبط (شکلی از ابر استدلال)

وجه تسمیه



- آلفا مقدار بهترین انتخاب (یعنی بالاترین مقدار) یافته شده تا کنون در هر نقطه انتخاب در طول مسیر برای *max* می باشد
- اگر v بدتر از آلفا باشد، *max* از آن اجتناب می کند

← آن شاخه را هرس می کند

- بتا نیز برای *min* مانند آلفا تعریف می شود

الگوریتم آلفا-بتا

```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```


الگوریتم آلفا-بتا

```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```


محدودیت های منابع

فرض کنید ۱۰۰ ثانیه زمان داریم و در هر ثانیه می توان 10^4 گره گسترش داد.

← در هر حرکت 10^6 گره

روش استاندارد:

- تست برش: مانند محدوده عمقی
- تابع ارزیابی:

= میزان تخمینی مطلوب بودن موقعیت

تابع ارزیابی

- در شطرنج، معمولا مجموع وزن دار ویژگی ها

$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- مثال: $w_1=9$ و

$$f_1(s) = (\#\text{ of white queens}) - (\#\text{ of black queens})$$

برش جستجو

- MinimaxCutoff مانند MinimaxValue می باشد جز اینکه:
  ۱. Terminal? با Cutoff? جایگزین شده و
  ۲. Utility با Eval جایگزین شده است

- آیا در عمل کار می کند؟

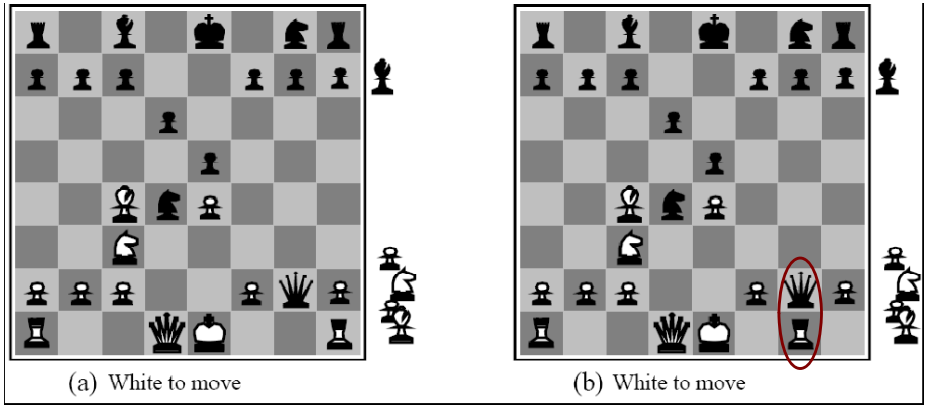
$b^m = 10^6 \rightarrow m = 4$

در شطرنج پیش بینی ۴ لایه نا امید کننده است!
- ۴ لایه = انسان مبتدی
- ۸ لایه = کامپیوترهای معمولی و انسان های حرفه ای
- ۱۲ لایه = Deep Blue و کاسپاروف

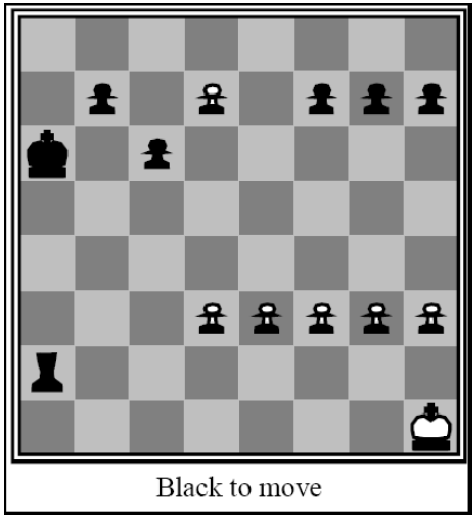
پیاده سازی

- تعیین یک محدوده عمقی مانند d
  - d باید طوری انتخاب شود که زمان لازم از قوانین بازی تجاوز نکند
- روش بهتر: استفاده از IDS تا وقتی که زمان داریم
- مشکلات:
  - خطا به دلیل تقریبی بودن تابع ارزیابی (شکل 6.8b)
- به تابع تست Cutoff پیچیده تری نیاز داریم
  - مشکل حالت های ناآرام ← جستجوی انتظار برای آرامش
  - مشکل اثر افق (شکل 6.9) ←
    - بهبود سخت افزار برای انجام جستجوهای عمیق تر
    - گسترش های انفرادی

شکل ۶-۸



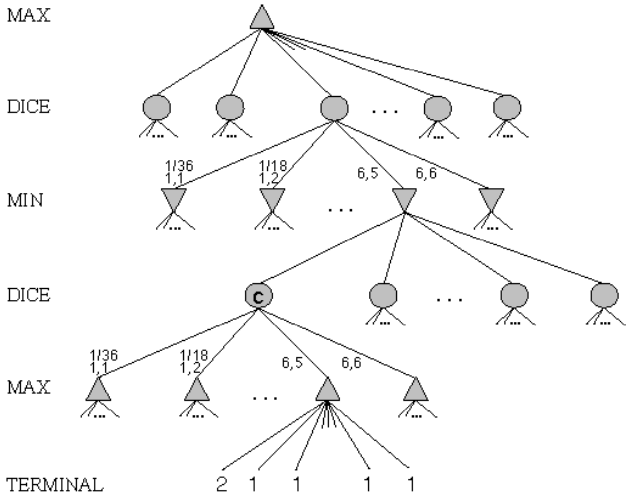
اثر افق



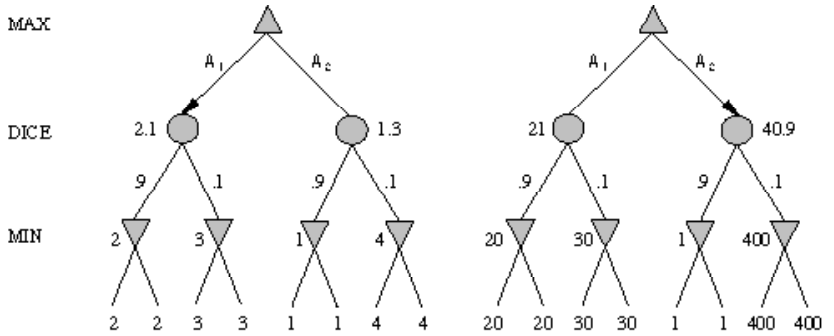
بازی های قطعی در عمل

- Chess: Deep Blue در سال ۱۹۹۷ قهرمان دنیای انسان ها (کاسپاروف) را شکست داد. Deep blue در یک ثانیه ۲۰۰ میلیون موقعیت را جستجو می کند و از توابع ارزیابی بسیار پیچیده ای استفاده می کند.
- Othello: قهرمان های انسانی از رقابت با کامپیوترها (که خیلی خوب هستند) امتناع می ورزند.
- Go: قهرمان های انسانی از رقابت با کامپیوترها (که بسیار بد هستند) امتناع می ورزند. در بازی Go فاکتور انشعاب بیشتر از ۳۰۰ می باشد، بنابراین اکثر برنامه ها برای ارائه حرکت های معقول از پایگاه های دانش الگویی استفاده می کنند.

بازی هایی که دارای عناصر شانس هستند



معنای تابع ارزیابی



- تغییری که ترتیب نسبی مقادیر را حفظ می کند، در تصمیم minimax تأثیری ندارد، اما می تواند تصمیم در مورد گره های شانس را تغییر دهد
- راه حل: تبدیلات خطی

خلاصه

- کار کردن بر روی بازی ها لذت بخش است!
- بازی ها نکات مهم متعددی را در AI به نمایش می گذارند
- عالی بودن ممکن نیست ← باید تقریب زد
- فکر کردن در مورد اینکه به چه چیزی فکر کنی ایده خوبی می باشد

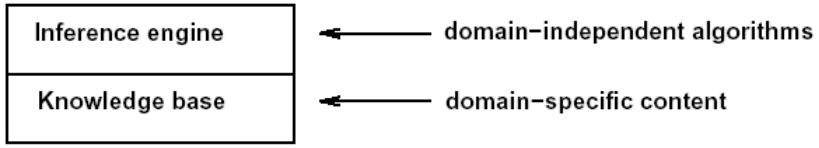
فصل هفتم

فصل هفتم
سید ناصر رضوی
E-mail:razavi@Comp.iust.ac.ir
۱۳۸۴

مقدمه

- عامل های مبتنی بر دانش
- محیط Wumpus
- منطق – مدل ها و استلزام
- منطق گزاره ای (بولین)
- هم ارزی، اعتبار و صدق پذیری
- قوانین استنتاج و اثبات تئوری
  - زنجیره استنتاج رو به جلو (forward chaining)
  - زنجیره استنتاج رو به عقب (backward chaining)
  - رزولوشن

پایگاه دانش



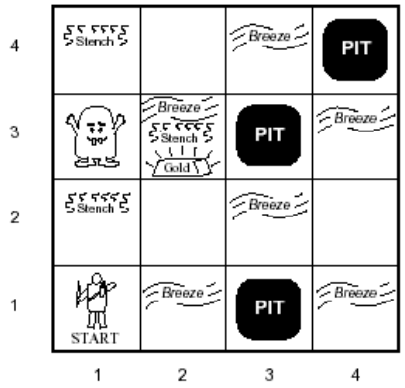
- پایگاه دانش = مجموعه ای از جملات در یک زبان رسمی
- رهیافت توصیفی برای ایجاد یک عامل (یا سیستم های دیگر):
  - به عامل آنچه را که نیاز دارد بداند، بگو (TELL)
  - سپس عامل می تواند از خود بپرسد (ASK) که چه عملی انجام دهد – پاسخ ها باید از KB **پیروی** کنند.
- می توان عامل ها را در سطح دانش در نظر گرفت:
  - یعنی، چه می دانند، بدون توجه به چگونگی پیاده سازی
- یا در سطح پیاده سازی:
  - یعنی، ساختارهای داده ای در KB و الگوریتم هایی که بر روی آنها کار می کنند.

یک عامل ساده مبتنی بر دانش

```
function KB-AGENT( percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL( KB, MAKE-PERCEPT-SENTENCE( percept, t))
    action ← ASK( KB, MAKE-ACTION-QUERY( t))
    TELL( KB, MAKE-ACTION-SENTENCE( action, t))
    t ← t + 1
    return action
```

- عامل باید قادر باشد:
  - حالات و اعمال و ... را بازنمایی کند.
  - ادراک جدید دریافت کند.
  - بازنمایی داخلی دنیا را بهنگام سازد.
  - خواص پنهان دنیا و اعمال مناسب را نتیجه گیری کند.

دنیای *Wumpus*

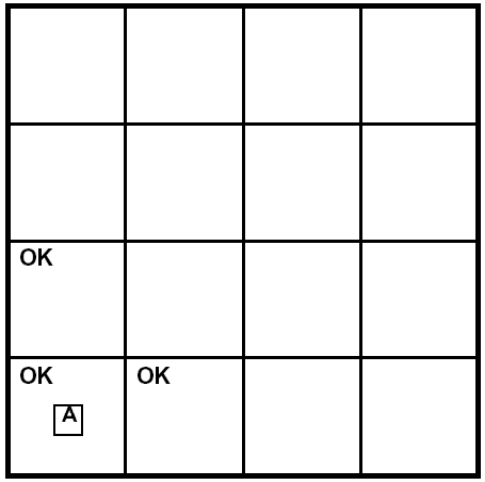


- **معیار کارآیی**
  - طلا 10000+، مرگ 1000-
  - هر عمل 1-، شلیک 10-
- **محیط**
  - خانه های مجاور وامپوس دارای بو هستند
  - خانه های مجاور چاله ها دارای نسیم هستند
  - در خانه حاوی طلا، درخشش وجود دارد
  - شلیک وامپوس را می کشد، اگر عامل رو به وامپوس باشد
  - تنها یک شلیک موثر است
  - اگر در خانه عامل طلا باشد، می تواند آنرا بردارد
  - عامل می تواند طلا را رها کند
- **حسگرها**
  - نسیم، درخشش، بو، ضربه و جیغ
- **محرک ها**
  - چرخش به چپ و راست
  - حرکت به جلو
  - برداشتن و رها کردن طلا و شلیک تیر

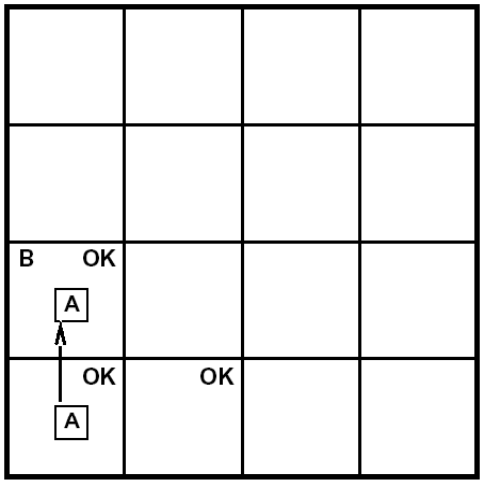
مشخصات محیط وامپوس

- دسترس پذیر؟؟ خیر-- تنها ادراک محلی میسر می باشد
- قطعی؟؟ بله– نتیجه اعمال کاملا مشخص است
- اپیزودیک؟؟ خیر
- ایستا؟؟ بله – وامپوس و چاله ها حرکت نمی کنند
- گسسته؟؟ بله
- تک-عاملی؟؟ بله – وامپوس اساساً یک ویژگی طبیعی است

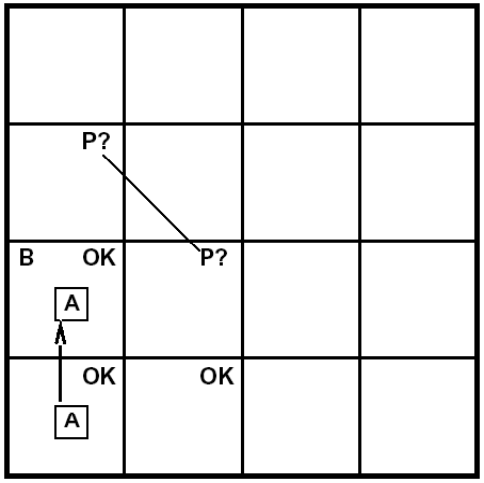
کاوش دنیای وامپوس



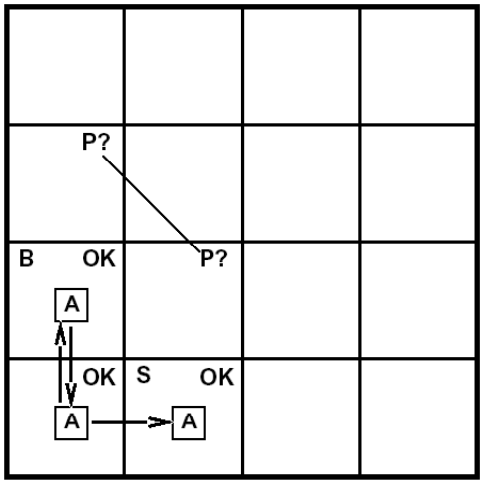
کاوش دنیای وامپوس



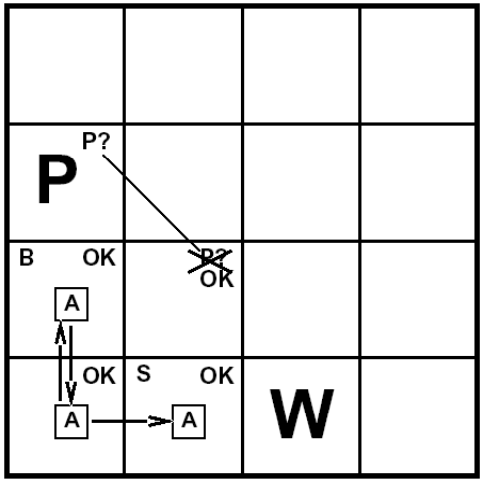
کاوش دنیای وامپوس



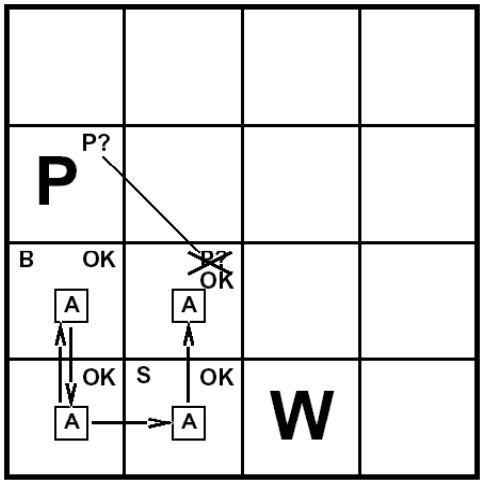
کاوش دنیای وامپوس



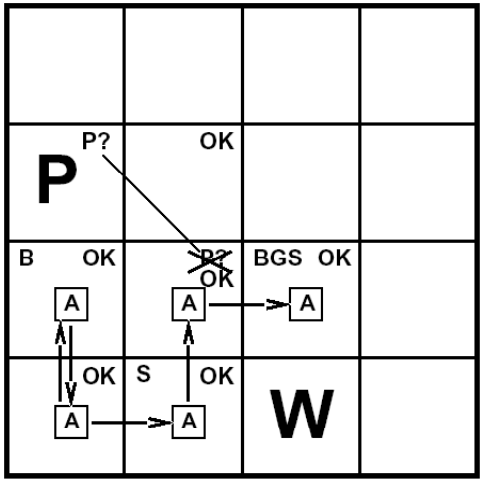
کاوش دنیای وامپوس



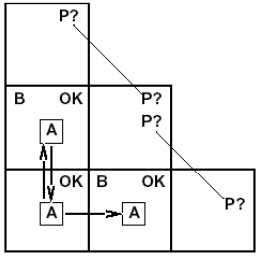
کاوش دنیای وامپوس

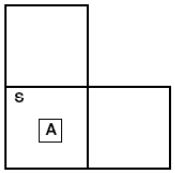


کاوش دنیای وامپوس



موقعیت های تصمیم گیری دیگر





- در خانه های (1, 2) و (2, 1) نسیم احساس می شود ← عمل مطمئنی وجود ندارد
- با فرض توزیع یکنواخت چاله ها، احتمال چاله در (2, 2) بیشتر

- احساس بو در (1, 1) ← قادر به حرکت نیست
- استفاده از استراتژی اجبار:
  ۱) به خانه روبرو شلیک کن
  ۲) اگر وامپوس آنجا بوده ← مرده ← امن
  ۳) اگر وامپوس آنجا نبوده ← امن

منطق

- منطق یک زبان رسمی برای بازنمایی دانش بطوری که بتوان از آن نتیجه گیری نمود.

- دستور ساختاری (syntax) : ساختار جملات زبان را تعریف می کند
- معنا (semantic): معنای جملات را تعریف می کند
  - یعنی، تعریف درستی یک جمله در یک دنیا

- مثال: زبان ریاضی
  - $x + 2 \geq y$ جمله
  - $x2 + y \geq$ جمله نیست
- جمله $x + 2 \geq y$ در دنیایی با $x=7, y=1$ درست و در دنیایی با $x=0$ و $y=6$ نادرست می باشد.

استلزام (entailment)

- استلزام بدین معناست که چیزی از چیز دیگری پیروی می کند:

$$KB \models \alpha$$

- پایگاه دانش KB مستلزم جمله α است، اگر و فقط اگر
- α در تمام دنیاهایی که در آن KB درست است، درست باشد.

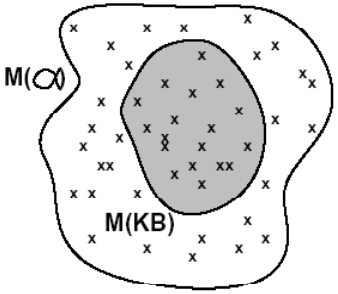
- مثال: $x + y = 4$ مستلزم $x + y = 4$ می باشد.

- استلزام رابطه ایست که بین ساختار جملات (syntax) و بر مبنای معنای جملات تعریف می شود.

مدل ها

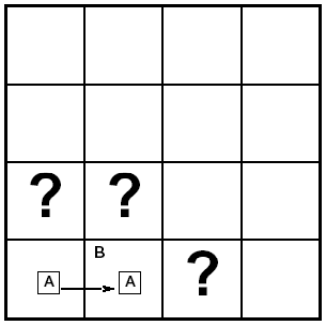
- منطق دانان عموماً بر حسب مدل ها فکر می کنند، که بطور رسمی دنیاهای ساخت یافته ای می باشند که درستی را می توان نسبت به آنها ارزیابی کرد.

- می گوییم m مدلی از جمله α می باشد اگر α در m درست باشد
- $M(\alpha)$ مجموعه تمام مدل های α می باشد
- بنابراین $KB \models \alpha$ اگر وفقط اگر $M(KB) \subseteq M(\alpha)$



استلزام در دنیای وامپوس

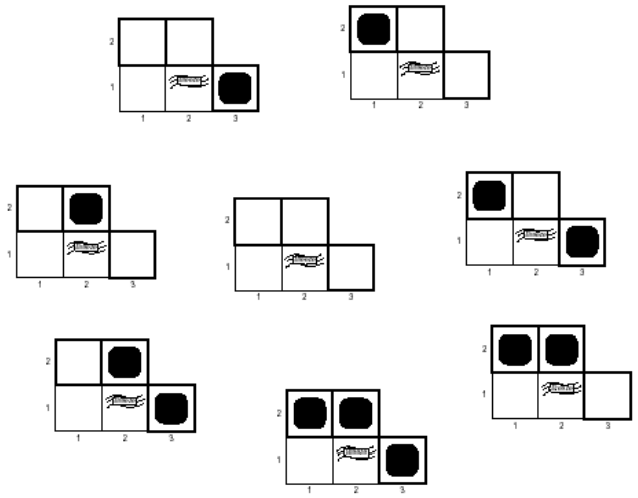
- موقعیت پس از [1, 1]، رفتن به راست، دریافت نسیم در [2, 1]



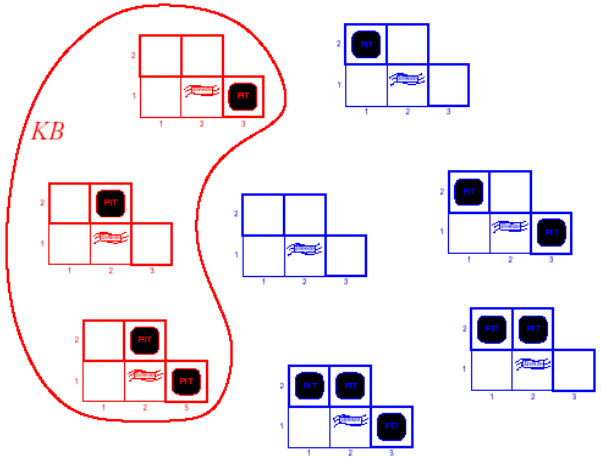
- مدلهای ممکن برای ? ها را تنها با فرض چاله ها در نظر بگیرید

- سه انتخاب بولین ← هشت مدل مختلف

مدل های وامپوس

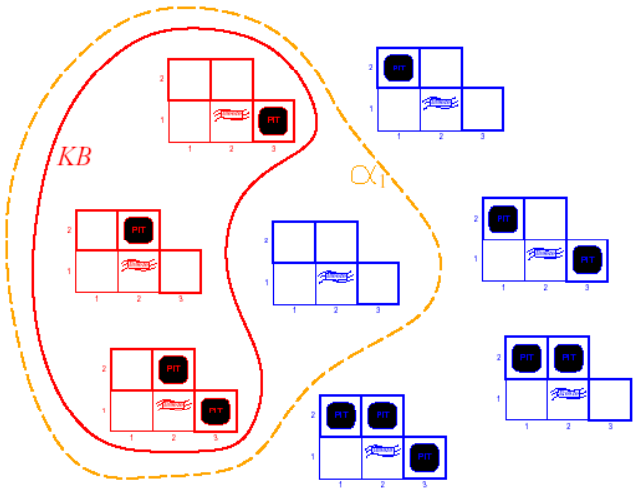


مدل های وامپوس



KB = Wumpus-world rules + observations

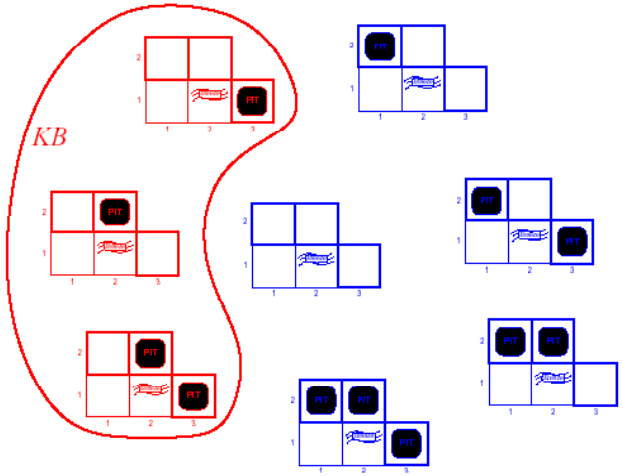
مدل های وامپوس



KB = Wumpus-world rules + observations

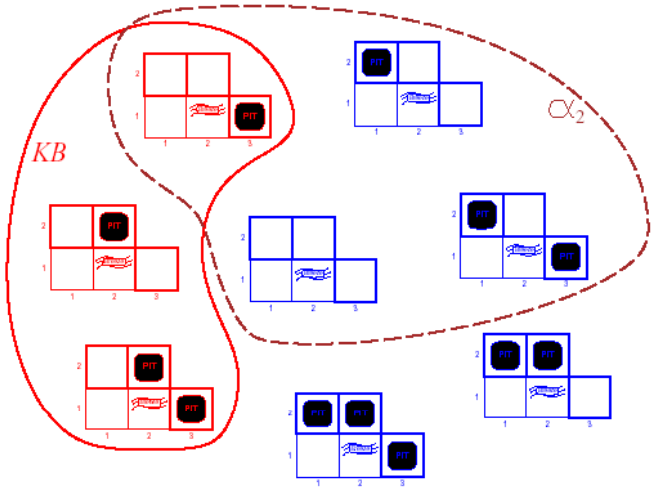
α_1 = "[1, 2] is safe", KB $\models \alpha_1$, proved by model checking

مدل های وامپوس



KB = Wumpus-world rules + observations

مدلهای وامپوس



KB = Wumpus-world rules + observations

α_2 = "[2, 2] is safe", KB $\not\models \alpha_2$

استنتاج (Inference)

- $KB \vdash_i \alpha$ = جمله α بوسیله رویه i از KB قابل اشتقاق می باشد.
- نتایج KB مانند یک انبار کاه می باشد، و α مانند یک سوزن
- استلزام = سوزن در انبار کاه؛ استنتاج = یافتن سوزن
- صحت (soundness): رویه i صحیح است اگر

$$KB \vdash_i \alpha \Rightarrow KB \models \alpha$$

- کامل بودن (completeness): رویه استنتاج i کامل است اگر

$$KB \models \alpha \Rightarrow KB \vdash_i \alpha$$

- مثال: در منطق مرتبه اول (First Order Logic) یک رویه استناج کامل و صحیح وجود دارد.

منطق گزاره ای : ساختار

- منطق گزاره ای ساده ترین نوع منطق است – برای بیان ایده های ساده و مبنایی

- سیمبولهای گزاره ای P_1، P_2 و ... هر کدام یک جمله می باشند
- True و False ثابت های گزاره ای می باشند و هر کدام به تنهایی یک جمله اند

- اگر S جمله باشد، آنگاه $\neg$ S نیز یک جمله است (نقیض)
- اگر S_1 و S_2 جمله باشند، $S_1 \wedge S_2$ نیز یک جمله است (ترکیب عطفی)
- اگر S_1 و S_2 جمله باشند، $S_1 \vee S_2$ نیز یک جمله است (ترکیب فصلی)
- اگر S_1 و S_2 جمله باشند، $S_1 \Rightarrow S_2$ نیز یک جمله است (ترکیب شرطی)
- اگر S_1 و S_2 جمله باشند، $S_1 \Leftrightarrow S_2$ نیز یک جمله است (ترکیب دوشرطی)

منطق گزاره ای: معنا

- هر مدل درست بودن/غلط بودن سیمبول های گزاره ای را مشخص می کند
- مثلا $P_{1,2}$ (درست)، $P_{2,2}$ (درست) ، $P_{3,1}$(نادرست)
- قوانین ارزیابی درستی نسبت به یک مدل *m*

P	Q	$\neg$P	P$\wedge$Q	P $\vee$ Q	P $\Rightarrow$ Q	P $\Leftrightarrow$ Q
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

جملات دنیای وامپوس

- اجازه دهید $P_{i,j}$ درست باشد، اگر و فقط اگر در خانه [i, j] چاله باشد.
- اجازه دهید $B_{i,j}$ درست باشد، اگر و فقط اگر در خانه [i, j] نسیم باشد.

$\neg P_{1,1}$

$\neg B_{1,1}$

$B_{2,1}$

- " چاله ها باعث وزش نسیم در خانه های مجاور می شوند".

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

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

- " در یک خانه نسیم می وزد اگر و فقط اگر چاله ای مجاور آن باشد"

استفاده از جدول درستی برای استنتاج

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

استنتاج بوسیله شمارش

- شمارش تمام مدل ها به روش اول – عمق صحیح و کامل است

```
function TT-ENTAILS?(KB, α) returns true or false
    symbols ← a list of the proposition symbols in KB and α
    return TT-CHECK-ALL(KB, α, symbols, [])

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
    if EMPTY?( symbols) then
        if PL-TRUE(KB, model) then return PL-TRUE(α, model)
        else return true
    else do
        P ← FIRST( symbols); rest ← REST( symbols)
        return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
               TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

برای n سیمبول $O(2^n)$

هم ارزی منطقی

- دو جمله هم ارز منطقی می باشند، اگر و فقط اگر هر دو در مدلهای یکسانی درست باشند.
- $\alpha \equiv ß$ iff $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

اعتبار و صدق پذیری

- یک جمله معتبر (valid) است اگر در تمام مدل ها درست باشد
  - مثال: True، $A \vee \sim A$ ،$A \Rightarrow A$ ،$(A \wedge (A \Rightarrow B) \Rightarrow B)$
- ارتباط معتبر بودن با استنتاج:

$KB \models \alpha$ iff $(KB \Rightarrow \alpha)$ is ***valid***

- یک جمله صدق پذیر (satisfiable) اگر در بعضی از مدل ها درست باشد
  - مثال: $A \vee B$
- یک جمله صدق ناپذیر است اگر در هیچ مدلی درست نباشد
  - مثال: $A \wedge \sim A$
- ارتباط صدق پذیری با استنتاج:

$KB \models \alpha$ iff $(KB \wedge \sim\alpha)$ is ***unsatisfiabe***

روش های اثبات

- روش های اثبات به دو نوع تقسیم می شوند:
- اعمال قوانین استنتاج:
  - تولید صحیح جملات جدید از جملات قدیمی
  - اثبات = دنباله ای از اعمال قوانین استنتاج

  می توان از قوانین استنتاج به عنوان عملگرها در الگوریتم استاندارد جستجو استفاده کرد.
  - اغلب نیاز به تبدیل جملات به یک شکل نرمال دارند.
- بررسی مدل:
  - شمارش جدول درستی (برحسب n نمایی)
  - عقبگرد بهبود یافته (DPLL)
  - جستجوی هیوریستیک در فضای مدل (صحیح اما نا کامل)

استنتاج رو به جلو و رو به عقب

- شکل نرمال Horn (HNF):
  KB = ترکیب عطفی عبارت های Horn
- عبارت Horn =
  - سیمبول گزاره ای
  - (سیمبول گزاره ای) ⇒(ترکیب عطفی سیمبول های گزاره ای)

$$P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$$

  - مثال :

$$C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$$

- قانون استنتاج Modes Ponens برای شکل Horn:

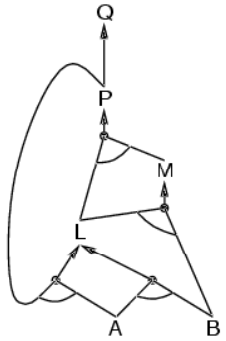
$$\frac{\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n, \quad \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- می تواند در هر دو روش روبه جلو و روبه عقب بکار رود.
- این روشها بسیار طبیعی هستند و در زمان خطی (برحسب اندازه KB) اجرا می شوند.

استنتاج روبه جلو

- ایده : هر قانونی که بخش شرایط آن در KB ارضاء شده را اعمال کن (fire) و نتیجه قانون را به KB اضافه کن، تا زمانیکه پاسخ پیدا شود و یا استنتاج دیگری ممکن نباشد.

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

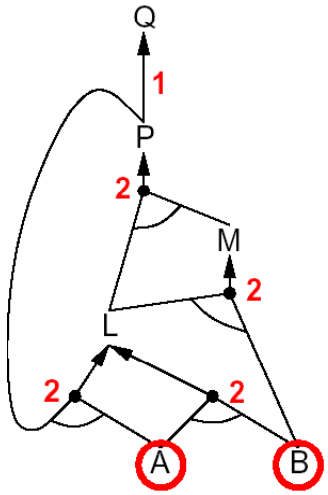


الگوریتم استنتاج روبه جلو

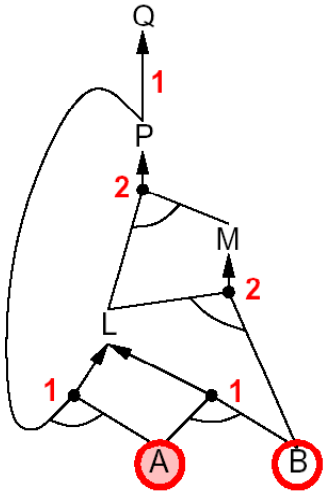
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known to be true
  while agenda is not empty do
      p ← POP(agenda)
      unless inferred[p] do
          inferred[p] ← true
          for each Horn clause c in whose premise p appears do
              decrement count[c]
              if count[c] = 0 then do
                  if HEAD[c] = q then return true
                  PUSH(HEAD[c], agenda)
  return false
```

- استنتاج روبه جلو برای پایگاه دانش در شکل Horn کامل و صحیح است.

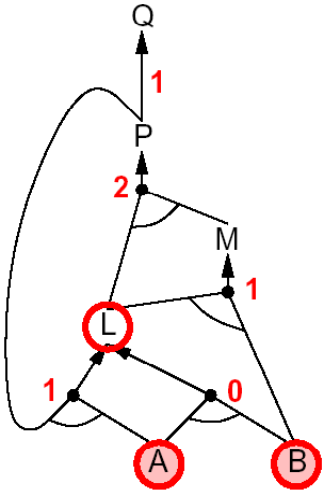
مثال استنتاج رو به جلو



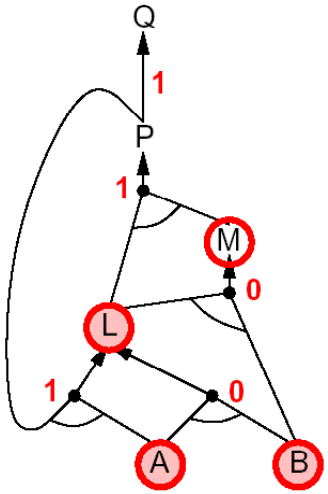
مثال استنتاج رو به جلو



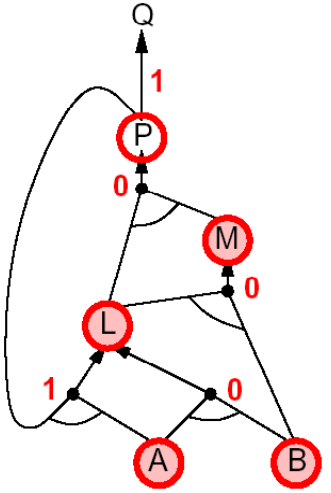
مثال استنتاج رو به جلو



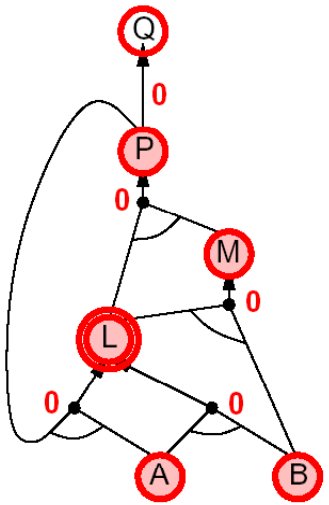
مثال استنتاج رو به جلو



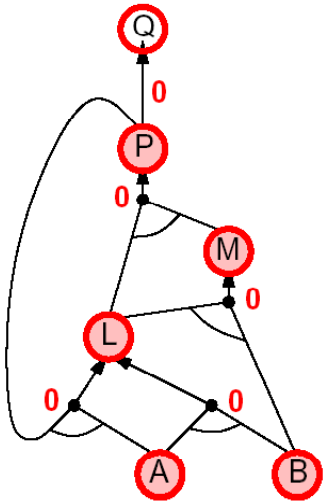
مثال استنتاج رو به جلو



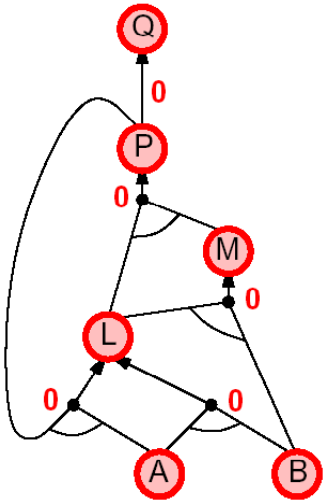
مثال استنتاج رو به جلو



مثال استنتاج رو به جلو



مثال استنتاج رو به جلو



استنتاج رو به عقب

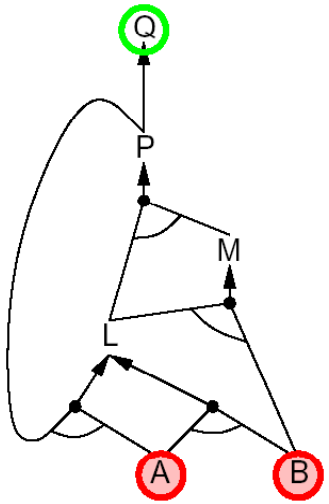
- ایده: برای اثبات q به سمت عقب حرکت کن
- برای اثبات q بوسیله BC،
  - بررسی کن که آیا q اکنون ثابت شده می باشد، یا
  - بوسیله BC تمام شرایط برخی از قوانین را که نتیجه آنها q است اثبات کن

- اجتناب از حلقه: بررسی قرار داشتن زیرهدف جدید روی پشته هدف
- اجتناب از اعمال تکراری: بررسی اینکه آیا زیرهدف جدید

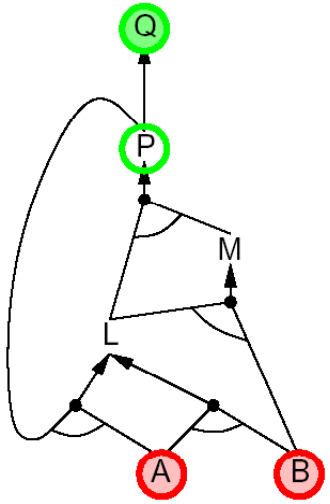
۱) قبلا درستی اش اثبات شده، یا

۲) قبلا شکست خورده است (fail)

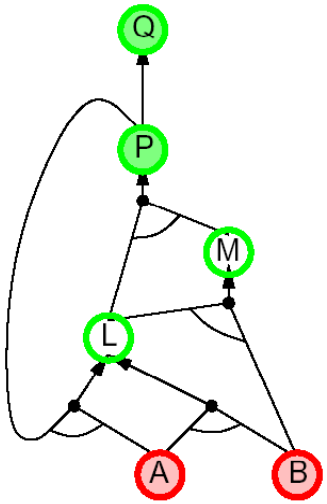
مثال از استنتاج رو به عقب



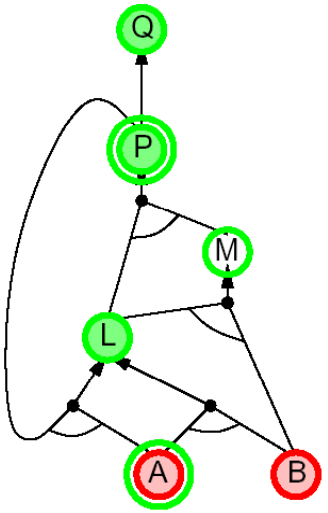
مثال از استنتاج رو به عقب



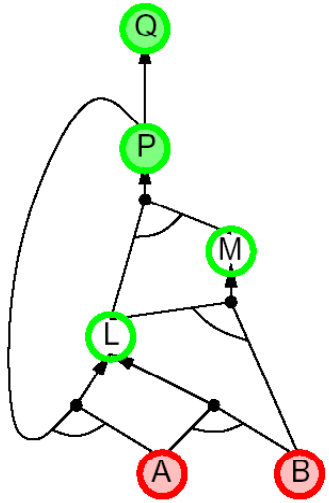
مثال از استنتاج رو به عقب



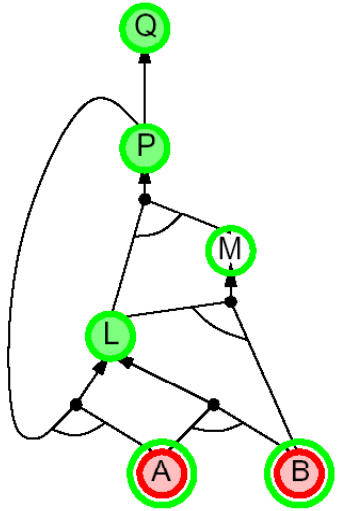
مثال از استنتاج رو به عقب



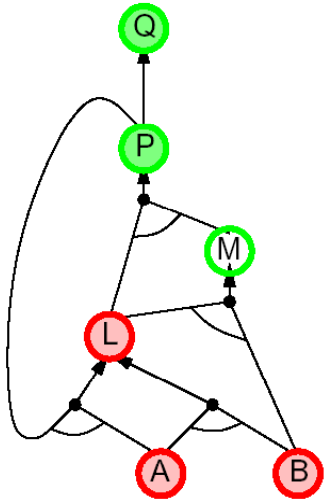
مثال از استنتاج رو به عقب



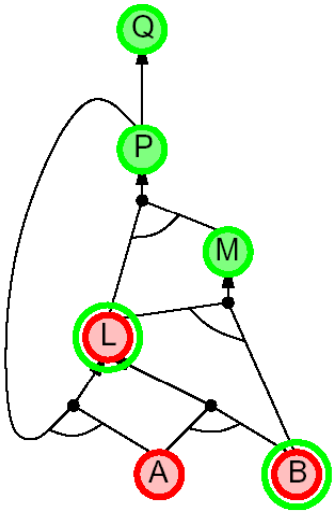
مثال از استنتاج رو به عقب



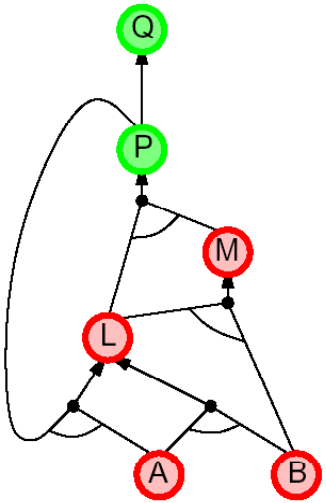
مثال از استنتاج رو به عقب



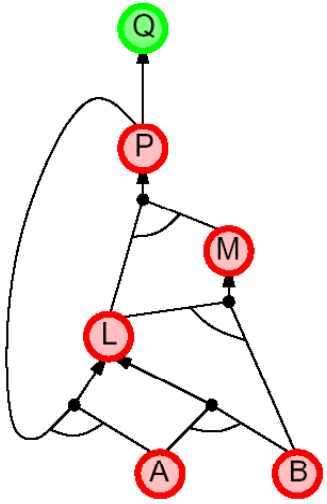
مثال از استنتاج رو به عقب



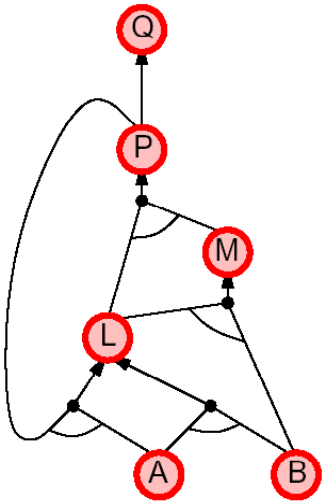
مثال از استنتاج رو به عقب



مثال از استنتاج رو به عقب



مثال از استنتاج رو به عقب



مقایسه دو روش

- FC:
  - بر مبنای داده (data driven)
  - ممکن است کارهای بسیاری انجام دهد که به هدف مربوط نمی شوند
- BC:
  - بر مبنای هدف (goal driven)
  - پیچیدگی BC می تواند بسیار بهتر از خطی نسبت به اندازه KB باشد.

Resolution

- شکل نرمال عطفی (CNF)

conjunctions of *disjunctions* of *literals*

Clauses

- مثال:

$$(A \vee \sim B) \wedge (B \vee \sim C \vee \sim D)$$

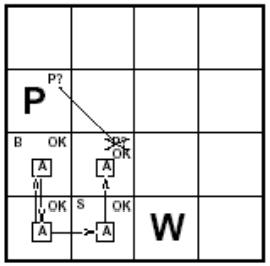
- قانون استنتاج رزولوشن (برای CNF): صحیح و کامل برای منطق گزاره ای

$$\frac{l_1 \vee \cdots \vee l_i \vee \cdots l_k, \quad m_1 \vee \cdots m_j \vee \cdots \vee m_n}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

l_i و m_j نقیض یکدیگرند.

Resolution

- مثال :



$$\frac{P_{1,3} \vee P_{2,2}, \quad \neg P_{2,2}}{P_{1,3}}$$

- رزولوشن برای منطق گزاره ای صحیح و کامل می باشد.

Resolution

- صحت قانون استنتاج رزولوشن

$$\frac{\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow l_i \qquad \neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}{\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

تبدیل به CNF

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inwards using de Morgan's rules and double-negation:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply distributivity law ($\wedge$ over $\vee$) and flatten:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

الگوریتم *Resolution*

- اثبات بوسیله تناقض، یعنی نشان بده $KB \wedge \sim\alpha$ صدق ناپذیر است

function PL-RESOLUTION(*KB*, α) **returns** *true* or *false*

clauses ← the set of clauses in the CNF representation of $KB \wedge \sim\alpha$

new ← {}

loop do

for each C_i, C_j **in** *clauses* **do**

resolvents ← PL-RESOLVE(C_i, C_j)

if *resolvents* contains the empty clause **then return** *true*

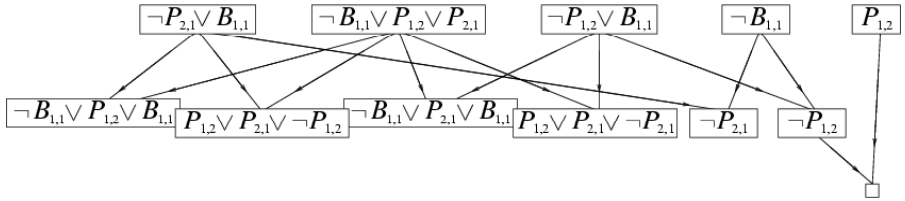
new ← *new* ∪ *resolvents*

if *new* ⊆ *clauses* **then return** *false*

clauses ← *clauses* ∪ *new*

مثال برای رزولوشن

- $KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$



الگوریتم های استنتاج کارآ در منطق گزاره ای

- دو خانواده از الگوریتم های استنتاج کارآ برای منطق گزاره ای
- الگوریتم های کامل جستجوی عقبگرد:
  - الگوریتم DPLL (Davis, Putnam, Logemann, Loveland)
- الگوریتم ناکامل جستجوی محلی
  - الگوریتم WalkSAT

الگوریتم DPLL

- تعیین کن که آیا یک جمله ورودی در زبان منطق گزاره ای (در شکل نرمال CNF) صدق پذیر است یا خیر.
- بهبودها نسبت به روش شمارش جدول درستی:

۱- خاتمه زود هنگام

- یک فراکرد (Clause) درست است اگر هر یک از لیترال ها درست باشد.
- یک جمله نادرست است اگر هر یک از فراکردهای آن نادرست باشد.

۲- هیوریستیک سیمبول محض

- سیمبول محض: سیمبولی که در تمام فراکرد ها با یک علامت ظاهر شود.
- مثال: در سه فراکرد $(A \vee \neg B), (\neg B \vee \neg C), (C \vee A)$ سیمبولهای A و B سیمبول محض می باشند، اما C یک سیمبول محض نیست.
- لیترال یک سیمبول محض را درست تلقی کن.

۳- هیوریستیک فراکرد واحد

- فراکرد واحد: تنها شامل یک لیترال می باشد. یا فراکردی که تمام لیترال های آن غیر از یک لیترال، نادرست می باشند.
- تنها لیترال موجود در یک فراکرد واحد باید درست باشد.

الگوریتم DPLL

```
function DPLL-SATISFIABLE?(s) returns true or false
  inputs: s, a sentence in propositional logic
  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(clauses, symbols, [])

function DPLL(clauses, symbols, model) returns true or false
  if every clause in clauses is true in model then return true
  if some clause in clauses is false in model then return false
  P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
  if P is non-null then return DPLL(clauses, symbols–P, [P = value|model])
  P, value ← FIND-UNIT-CLAUSE(clauses, model)
  if P is non-null then return DPLL(clauses, symbols–P, [P = value|model])
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, [P = true|model]) or
         DPLL(clauses, rest, [P = false|model])
```


الگوریتم WalkSAT

- الگوریتم جستجوی محلی و ناکامل
- تابع ارزیابی: هیوریستیک حداقل درگیری برای کمینه کردن تعداد فراکردهای ارضاء نشده
- تعادل میان میزان حریصانه بودن و تصادفی بودن

الگوریتم WalkSAT

```
function WALKSAT(clauses, p, max-flips) returns a satisfying model or failure
  inputs: clauses, a set of clauses in propositional logic
          p, the probability of choosing to do a "random walk" move
          max-flips, number of flips allowed before giving up
  model ← a random assignment of true/false to the symbols in clauses
  for i = 1 to max-flips do
    if model satisfies clauses then return model
    clause ← a randomly selected clause from clauses that is false in model
    with probability p flip the value in model of a randomly selected symbol
        from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
  return failure
```


مسائل ارضاءپذیری سخت

- جملات 3-CNF تصادفی را در نظر بگیرید، مثلا:

$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$

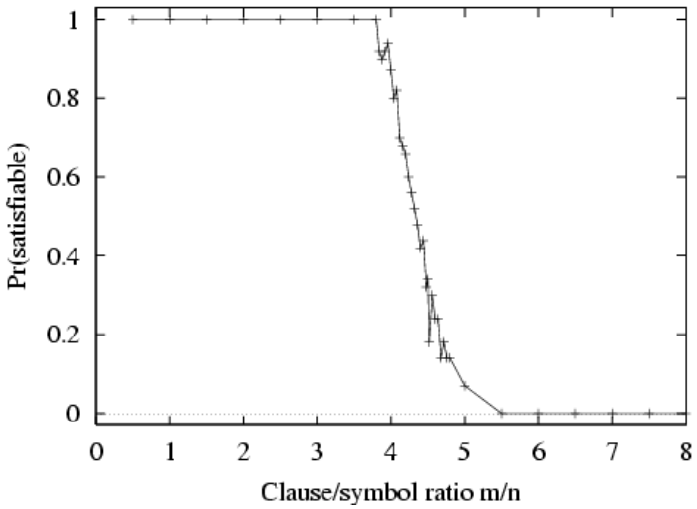
5 سیمبول و 5 فراکرد – ۳۲ انتساب و ۱۶ مدل – به طور متوسط دو حدس کافی می باشد (برای یافتن مدل)

m = تعداد فراکردها

n = تعداد سیمبول ها

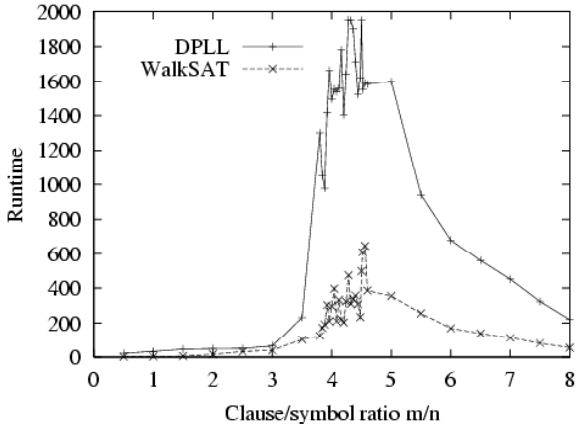
به نظر می رسد مسائل سخت نزدیک $m/n = 4.3$ باشند (نسبت بحرانی)

مسائل ارضاءپذیری سخت



$n = 50$

مسائل ارضاءپذیری سخت



زمان اجرا برای 100 جمله 3-CNF ارضاء پذیر، $n = 50$

مسائل ارضاءپذیری سخت

- سه نکته واضح در شکل قبل:
  - مسائل نزدیک نقطه بحرانی بسیار سخت از دیگر مسائل تصادفی هستند.
  - حتی در مسائل سخت، الگوریتم DPLL نسبتا کارآمد است – چند هزار مرحله به طور میانگین در مقایسه با $2^{50} \approx 10^{15}$ برای شمارش جدول درستی.
  - در کل محدوده، الگوریتم WalkSAT بسیار سریعتر از DPLL می باشد.

عامل های مبتنی بر استنتاج در دنیای وامپوس

- یک عامل دنیای وامپوس با استفاده از منطق گزاره ای:

$\neg P_{1,1}$
$\neg W_{1,1}$
$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$
$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$
$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$
$\neg W_{1,1} \vee \neg W_{1,2}$
$\neg W_{1,1} \vee \neg W_{1,3}$
…

- ۶۴ سیمبول گزاره ای متفاوت
- ۱۵۵ جمله

```
function PL-WUMPUS-AGENT(percept) returns an action
  inputs: percept, a list, [stench, breeze, glitter]
  static: KB, initially containing the "physics" of the wumpus world
          x, y, orientation, the agent's position (init. [1,1]) and orient. (init. right)
          visited, an array indicating which squares have been visited, initially false
          action, the agent's most recent action, initially null
          plan, an action sequence, initially empty

  update x,y,orientation, visited based on action
  if stench then TELL(KB, S_{x,y}) else TELL(KB, ¬ S_{x,y})
  if breeze then TELL(KB, B_{x,y}) else TELL(KB, ¬ B_{x,y})
  if glitter then action ← grab
  else if plan is nonempty then action ← POP(plan)
  else if for some fringe square [i,j], ASK(KB, (¬ P_{i,j} ∧ ¬ W_{i,j})) is true or
          for some fringe square [i,j], ASK(KB, (P_{i,j} ∨ W_{i,j})) is false then do
      plan ← A*-GRAPH-SEARCH(ROUTE-PB([x,y], orientation, [i,j], visited))
      action ← POP(plan)
  else action ← a randomly chosen move
  return action
```


محدودیت های منطق گزاره‌ای

- به طور کلی منطق گزاره ای از قدرت بیان کافی برخوردار نیست:
  - برای هر مربع شامل جملات "فیزیکی" آن مربع
  - برای هر زمان t و هر مکان [x, y]

$$L_{x,y}^{t} \wedge FacingRight^{t} \wedge Forward^{t} \Rightarrow L_{x+1,y}^{t+1}$$

خلاصه

- عامل های منطقی از استنتاج بر روی یک پایگاه دانش برای اشتقاق دانش جدید و تصمیم گیری استفاده می کنند
- مفاهیم مبنایی منطق:
  - ساختار (syntax)
  - معنا (semantics)
  - استلزام (entailments)
  - استنتاج (inference)
  - صحت (soundness)
  - کامل بودن (completeness)
- استنتاج رو به جلو و رو به عقب برای عبارت های Horn دارای زمان خطی هستند و کامل می باشند.
- رزولوشن برای منطق گزاره ای کامل است
- منطق گزاره ای در بازنمایی دانش ضعیف می باشد

منطق مرتبه اول

فصل هشتم
سید ناصر رضوی
E-mail: razavi@Comp.iust.ac.ir
۱۳۸۴

مقدمه

- چرا FOL
- بررسی ساختار و معانی جملات در FOL
- استفاده از FOL
- دنیای وامپوس در FOL
- مهندسی دانش در FOL

مزایا و معایب منطق گزاره ای

☺ منطق گزاره ای توصیفی است --- دانش و استنتاج مستقل

☺ منطق گزاره ای ترکیبی است --- در غیر اینصورت کار سیستم استدلال دشوار است
یعنی، معنای یک جمله مرکب با توجه به جملات سازنده آن تعیین می شود

☺ معنای جملات در منطق گزاره ای مستقل از متن می باشد
(برخلاف زبان طبیعی)

☹ منطق گزاره ای قدرت بیان بسیار محدودی دارد
(برخلاف زبان طبیعی)
مثلا، در منطق گزاره ای نمی توان گفت " چاله ها باعث وزش نسیم در خانه های مجاور می شوند"
-مگر آنکه برای هر خانه یک جمله نوشته شود.

FOL

- در حالیکه منطق گزاره ای فرض می کند دنیا از **حقایق** تشکیل شده است، منطق مرتبه اول (مانند زبان طبیعی) فرض می کند دنیا شامل موارد زیر است:
  - اشیاء (Objects) : دنیا از اشیایی تشکیل شده که بواسطه خصوصیاتشان (Properties) از یکدیگر قابل تمایز می‌باشند
  - روابط (Relations): در بین اشیاء روابط مختلفی می تواند وجود داشته باشد که بعضی از این روابط به صورت تابع (Function) می باشند.
  - توابع (functions): پدرِ ، برادرِ، یکی بیشتر از و ...

- حقایق به صورت ارجاع به اشیاء، خصوصیات و روابط بین اشیاء بیان می شوند.
- مثال :
  - Sum(1, 2, 3), Even(2), Odd(3), …
  - Parent (Bob, Jim), Male(Bob), …
  - Add(1, 2), LeftLegof(John), …

منطق ها به طور کلی

Language	Ontology	Epistemology
Propositional Logic	facts	true/false/unknown
First Order Logic	facts, objects, relations	true/false/unknown
Temporal Logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief∈[0, 1]
Fuzzy logic	degree of truth ∈[0, 1]	known interval value

ساختار *FOL*: عناصر ابتدایی

- ثابت ها (KingJohn, 2, …)
- مسندها (Brother, >, …)
- توابع (Sqrt, LeftLegOf, …)
- متغیرها (x, y, a, b, …)
- رابط های منطقی (¬, ⇒, ∧, ∨, ⇔)
- تساوی (=)
- سورها (∃ ,∀)

ساختار جملات در *FOL*

Sentence → AtomicSentence
| Sentence Connective Sentence
| Quantifier Variable, … Sentence
| ~ Sentence
| (Sentence)

AtomicSentence → Predicate(Term, …) | Term = Term

Term → Function(Term, …)
| Constant
| Variable

ساختار جملات در *FOL*

Connective → ⇒ | ∧ | ∨ | ⇔
Quantifier → ∀ | ∃
Constant → A | X_1 | John | …
Variable → a | x | …
Predicate → Before | HasColor | …
Function → MotherOf | LeftLegOf

ثابت ها، مسندها و توابع

- سیمبولهای ثابت: به یک شی خاص در مدل رجوع می کنند
  - A, B, John, …
- مسندها (Predicates) : به یک رابطه ویژه در مدل رجوع می کنند.
  - Brother, Mother, …
- توابع : بعضی از روابط تابع هستند، یعنی هر شی توسط رابطه دقیقا به یک شیء دیگر رجوع می کند.
  - MotherOf, Cos, LeftLegOF,…

ترم (*Term*)

- ترم: یک عبارت منطقی که به یک شیء رجوع می کند.
  - ثابت ها (A, B, C, John, ...)
  - متغیرها(a, b, x, …)
  - توابع

MotherOf(Richard),
Length(LeftLegOf(John)),
Cos(30),…

جملات ساده (*Atomic Sentences*)

- یک جمله اتمی به صورت زیر می باشد:

Predicate(Term$_1$, … , Term$_n$)

- Brother(KingJohn,RichardTheLionheart)
- >(Length(LeftLegOf(Richard)),Length(LeftLegOf(KingJohn)))
- Married(FatherOf(Richard), MotherOF(John))

- یک جمله ساده زمانی درست است که رابطه Predicate بین اشیایی که Term$_1$ و ... و Term$_n$ به آنها اشاره می کنند برقرار باشد.

جملات مرکب (Complex Sentences)

- جملات مرکب از ترکیب جملات ساده بوسیله رابط های منطقی بدست می آیند.

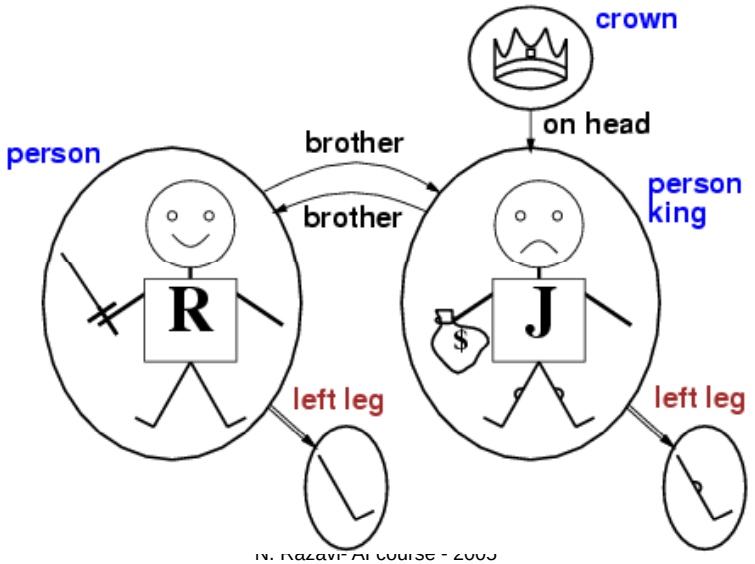
$$\neg S_1,\ S_1 \wedge S_2,\ S_1 \vee S_2,\ S_1 \Rightarrow S_2,\ S_1 \Leftrightarrow S_2$$

- Brother(John, Richard) $\wedge$ Brother(Richard, John)
- Older(John, 30) $\vee$ Younger(John, 30)
- Older(John, 30) $\Rightarrow$ ~Younger(John, 30)
- ~Brother(Robin, John)

درستی در منطق مرتبه اول

- جملات نسبت به یک مدل و یک تفسیر درست می باشند.
- مدل ها شامل اشیا (عناصر دامنه) و روابط میان آنها می باشند.
- تفسیر، مورد مراجعه را برای موارد زیر مشخص می کند:
  - سیمبول های ثابت ← اشیاء
  - سیمبول های مسندی ← روابط
  - سیمبول های تابع ← روابط تابعی

مدل ها در منطق مرتبه اول: مثال



سورعمومی

$$\forall \langle Variables \rangle \langle Sentence \rangle$$

Everyone at the class is smart

$$\forall x\ at(x, Class) \Rightarrow Smart(x)$$

- سور عمومی برابر است با **ترکیب عطفی** تمام جملاتی که با جایگذاری متغیرها در Sentence بدست می آیند.

$$\begin{aligned} & at(KingJohn, Class) \Rightarrow Smart(KingJohn) \\ \wedge\ & at(Richard, Class) \Rightarrow Smart(Richard) \\ \wedge\ & at(Class, Class) \Rightarrow Smart(Class) \\ \wedge\ & \cdots \end{aligned}$$

سوروجودی

$$\exists \langle Variables \rangle \langle Sentence \rangle$$

Someone at class is smart

$$\exists x\ at(x, class) \wedge Smart(x)$$

- سور وجودی برابر است با **ترکیب فصلی** تمام جملاتی که با جایگذاری متغیرها در Sentence بدست می آیند.

$$\begin{aligned} & at(KingJohn, Class) \wedge Smart(KingJohn) \\ \vee\ & at(Richard, Class) \wedge Smart(Richard) \\ \vee\ & at(Class, Class) \wedge Smart(Class) \\ \vee\ & \cdots \end{aligned}$$

یک اشتباه متداول

- سور عمومی اغلب با ترکیب شرطی و سور وجودی اغلب با ترکیب عطفی بکار می رود

$$\forall x\ Human(x) \Rightarrow Mortal(x)$$
$$\exists x\ Sister(x, Spot) \wedge Cat(x)$$

- مثال : جمله زیر به معنای '' هر کسی در کلاس است و هر کسی باهوش است '' می باشد

$$\forall x\ at(x, Class) \wedge Smart(x)$$

ویا جمله زیر:

$$\exists x\ at(x, Class) \Rightarrow Smart(x)$$

سورهای تودرتو (*Nested Quntifiers*)

$$\forall x\ \forall y\ Parent(x, y) \Rightarrow Child(y, x)$$
$$\exists x\ \exists y\ Owner(x, y)$$

- در جملات بالا می توان جای سورها را با هم تعویض نمود
  اما $\exists x \forall y$ معادل $\forall y \exists x$ نمی باشد .

$$\forall x \exists y\ mother(y, x)$$
$$\exists y \forall x\ mother(y, x)$$

ارتباط بین سورها

- سور عمومی و وجودی از طریق تناقض با یکدیگر در ارتباط هستند:

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

- مثال: دوگانی سور

$$\forall x\ Likes(x, IceCream) \qquad \neg\exists x\ \neg Likes(x, IceCream)$$
$$\exists x\ Likes(x, Broccoli) \qquad \neg\forall x\ \neg Likes(x, Broccoli)$$

تساوی

- $Term_1 = Term_2$ تحت یک تفسیر مشخص درست است اگر و فقط اگر $Term_1$ و $Term_2$ هر دو به شیء یکسانی مراجعه کنند.

- مثال: تعریف *Sibling* بر حسب *Parent*

$$\forall x,y\ Sibling(x,y) \Leftrightarrow [\neg(x = y) \wedge \exists m,f\ \neg (m = f) \wedge Parent(m,x) \wedge Parent(f,x) \wedge Parent(m,y) \wedge Parent(f,y)]$$

استفاده از FOL

دامنه روابط خانوادگی:

- برادرها با یکدیگر Sibling هستند

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

- مادر هر شخص والد مونث آن شخص می باشد

$\forall m,c\ Mother(c) = m \Leftrightarrow (Female(m) \wedge Parent(m,c))$

- رابطه "Sibling" دارای خاصیت تقارنی می باشد

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

استفاده از FOL

دامنه مجموعه:

- $\forall s\ Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x,s_2\ Set(s_2) \wedge s = \{x|s_2\})$
-
- $\neg\exists x,s\ \{x|s\} = \{\}$
-
- $\forall x,s\ x \in s \Leftrightarrow s = \{x|s\}$
-
- $\forall x,s\ x \in s \Leftrightarrow [\exists y,s_2\ (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$
-
- $\forall s_1,s_2\ s_1 \subseteq s_2 \Leftrightarrow (\forall x\ x \in s_1 \Rightarrow x \in s_2)$
- $\forall s_1,s_2\ (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$

محاوره با پایگاه دانش در FOL

- فرض کنید عاملی در دنیای وامپوس از پایگاه دانش FOL استفاده می کند و در زمان $t = 5$ یک نسیم و بو (درخشش خیر) دریافت می کند.

```
Tell(KB,Percept([Smell,Breeze,None],5))
Ask(KB,∃a BestAction(a,5))
```

- یعنی، آیا KB مستلزم بهترین عمل در $t = 5$ می باشد؟
- پاسخ: بله ، $\{a/shoot\}$ → لیست جانشینی
- با داشتن جمله S و لیست جانشینی σ :
  - $S\sigma$ به نتیجه حاصل از جایگذاری σ در S اشاره دارد

S = Smarter(x,y)
σ = {x/Hillary,y/Bill}
$S\sigma$ = Smarter(Hillary,Bill)

- ASK(KB, σ) همه یا برخی از σ ها را بر می گرداند به طوری که $KB \models \sigma$

پایگاه دانش برای دنیای وامپوس

- ادراک
  - $\forall t,s,b\ Percept([s,b,Glitter],t) \Rightarrow Glitter(t)$
  -
- واکنش
  - $\forall t\ Glitter(t) \Rightarrow BestAction(Grab,t)$

استنتاج خواص پنهانی

- $\forall x,y,a,b\ Adjacent([x,y],[a,b]) \Leftrightarrow [a,b] \in \{[x+1,y], [x-1,y],[x,y+1],[x,y-1]\}$

- خصوصیات مکان ها:
- $\forall s,t\ At(Agent,s,t) \wedge Breeze(t) \Rightarrow Breezy(s)$

- خانه های مجاور با چاه ها دارای نسیم می باشند:
  - قانون تشخیصی --- علت را از روی اثر آن نتیجه می گیرد

$\forall s\ Breezy(s) \Rightarrow \exists\ r\ Adjacent(r,s) \wedge Pit(r)$

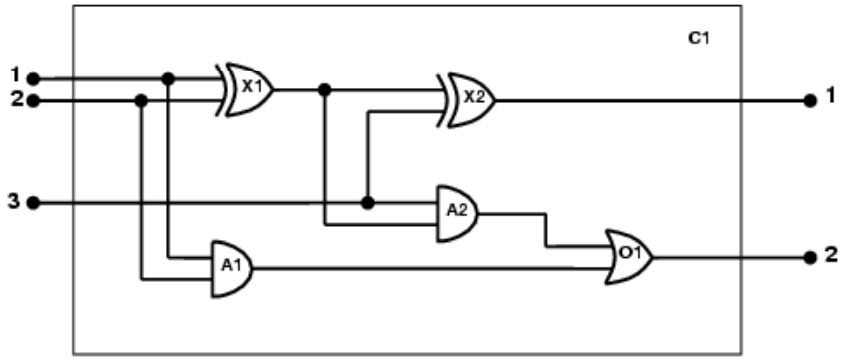
  - قانون سببی --- اثر را از روی علت آن نتیجه می گیرد

مهندسی دانش در *FOL*

۱. مشخص نمودن وظیفه
۲. جمع آوری دانش مربوطه
۳. تصمیم گیری در مورد مسندها، توابع و ثابت ها
۴. کد نمودن دانش عمومی دامنه
۵. کد نمودن توصیف نمونه مساله خاص
۶. اعمال پرس و جو به رویه استنتاج و در یافت پاسخ
۷. اشکال زدایی پایگاه دانش

دامنه مدارهای الکتریکی

- جمع کننده یک بیتی



دامنه مدارهای الکتریکی

۱. مشخص نمودن وظیفه
   - آیا مدار واقعا به درستی جمع می کند؟ (وارسی مدار)
۲. جمع آوری دانش مربوطه
   - ترکیب سیم ها و گیت ها؛ انواع گیت ها (AND, OR, XOR, NOT)
   - دانش نامربوط: اندازه، شکل، رنگ، قیمت گیت ها
۳. تصمیم گیری در مورد لغات
   - راه های مختلف:

$Type(X_1) = XOR$

$Type(X_1, XOR)$
$XOR(X_1)$

دامنه مدارهای الکتریکی

۴. کد نمودن دانش عمومی دامنه

$-\forall t_1,t_2\ Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$

$-\forall t\ Signal(t) = 1 \vee Signal(t) = 0$

$-1 \neq 0$

$-\forall t_1,t_2\ Connected(t_1, t_2) \Rightarrow Connected(t_2, t_1)$

$-\forall g\ Type(g) = OR \Rightarrow Signal(Out(1,g)) = 1 \Leftrightarrow \exists n\ Signal(In(n,g)) = 1$

$-\forall g\ Type(g) = AND \Rightarrow Signal(Out(1,g)) = 0 \Leftrightarrow \exists n\ Signal(In(n,g)) = 0$

$-\forall g\ Type(g) = XOR \Rightarrow Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g))$

دامنه مدارهای الکتریکی

۵. کد نمودن دانش مربوط به یک نمونه مساله خاص

$Type(X_1) = XOR$ $Type(X_2) = XOR$
$Type(A_1) = AND$ $Type(A_2) = AND$
$Type(O_1) = OR$

$Connected(Out(1,X_1),In(1,X_2))$ $Connected(In(1,C_1),In(1,X_1))$
$Connected(Out(1,X_1),In(2,A_2))$ $Connected(In(1,C_1),In(1,A_1))$
$Connected(Out(1,A_2),In(1,O_1))$ $Connected(In(2,C_1),In(2,X_1))$
$Connected(Out(1,A_1),In(2,O_1))$ $Connected(In(2,C_1),In(2,A_1))$
$Connected(Out(1,X_2),Out(1,C_1))$ $Connected(In(3,C_1),In(2,X_2))$
$Connected(Out(1,O_1),Out(2,C_1))$ $Connected(In(3,C_1),In(1,A_2))$

دامنه مدارهای الکتریکی

۶. اعمال پرس و جو بر رویه استنتاج

چه ترکیبی از ورودی ها باعث می شوک که اولین خروجی مدار C_1 (بیت جمع) به صفر و خروجی دوم مدار C_1 (بیت نقلی) به یک تبدیل شود؟

$\exists i_1,i_2,i_3\ Signal(In(1,C_1)) = i_1 \wedge Signal(In(2,C_1)) = i_2 \wedge Signal(In(3,C_1)) = i_3 \wedge Signal(Out(1,C_1)) = 0 \wedge Signal(Out(2,C_1)) = 1$

پاسخ های ممکن:

$\{i_1/1, i_2/1, i_3/0\}$,
$\{i_1/1, i_2/0, i_3/1\}$,
$\{i_1/0, i_2/1, i_3/1\}$

دامنه مدارهای الکتریکی

۶. اعمال پرس و جو بر رویه استنتاج

$\exists i_1,i_2,i_3,o_1,\ o_2\ Signal(In(1,C_1)) = i_1 \wedge Signal(In(2,C_1)) = i_2 \wedge Signal(In(3,C_1)) = i_3 \wedge Signal(Out(1,C_1)) = o^1 \wedge Signal(Out(2,C_1)) = o_2$

پاسخ: جدول ورودی/خروجی کامل که می تواند برای بررسی مدار استفاده شود.

دامنه مدارهای الکتریکی

۷. اشکال زدایی پایگاه دانش

اگر حقیقت $1 \neq 0$ را در پایگاه دانش نداشته باشیم:

$\exists i_1,i_2,o \quad Signal(In(1,C_1)) = i_1 \wedge Signal(In(2,C_1)) = i_2 \wedge Signal(Out(1,X_1)) = o$

این پرسش مشخص می کند که برای ورودی های ۱۰ و ۰۱ هیچ خروجی ای در X_1 مشخص نمی باشد. با توجه به اصل موضوعی در مورد XOR

$\forall g \; Type(g) = XOR \Rightarrow Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g))$

اگر ورودی هاصفر و یک بلشند:

$Signal(Out(1, X_1)) = 1$

$\Leftrightarrow 1 \neq 0$

خلاصه

- منطق مرتبه اول
  - اشیاء و روابط عناصر اولیه معنایی هستند
  - ساختار: ثابت ها، توابع، مسندها، تساوی، سورها
- قدرت بیان بیشتر: کافی برای تعریف دنیای وامپوس

استنتاج در منطق مرتبه اول

فصل نهم
سید ناصر رضوی
E-mail: razavi@Comp.iust.ac.ir
۱۳۸۴

مقدمه

- تقلیل استنتاج مرتبه-اول به استنتاج گزاره ای
- یکسان سازی (Unification)
- MP تعمیم یافته (GMP)
- زنجیره استنتاج روبه جلو
- زنجیره استنتاج رو به عقب
- رزولوشن

نمونه سازی عمومی (UI)

- هر نمونه از یک جمله دارای سور عمومی، توسط آن جمله قابل استلزام است.

$$\frac{\forall v\, \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

- مثال: $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

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

نمونه سازی وجودی (EI)

- برای هر جمله α، متغیر v و سیمبول ثابت k که در جای دیگری از پایگاه دانش ظاهر نشده باشد:

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

- مثال: $\exists x\ Crown(x) \wedge OnHead(x,John)$

$Crown(C_1) \wedge OnHead(C_1,John)$

به شرطی که C_1 یک سیمبول جدید باشد، که به آن ثابت اسکولم می گویند.

تقلیل به استنتاج گزاره ای

- فرض کنید که KB فقط شامل جملات زیر باشد:

$\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
$King(John)$
$Greedy(John)$
$Brother(Richard,John)$

- با نمونه سازی جمله دارای سور عمومی به تمام طرق ممکن، داریم:

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
$King(John)$
$Greedy(John)$
$Brother(Richard,John)$

- KB جدید گزاره ای سازی شده است. سیمبول های گزاره ای عبارتند از:

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

تقلیل به استنتاج گزاره ای (ادامه)

- هر پایگاه دانش در FOL می تواند به گونه ای گزاره ای سازی شود که استلزام را حفظ کند
- یک جمله توسط KB جدید قابل استلزام است اگر و فقط اگر توسط KB اصلی قابل استلزام باشد
- ایده: پایگاه دانش و کوئری را گزاره ای سازی کن، رزولوشن را اعمال و نتیجه را برگردان
- مشکل: در مورد سیمبول های تابعی تعداد نامحدودی ترم زمینی وجود دارد، مثلاً:

$Father(Father(Father(John)))$

تقلیل به استنتاج گزاره ای (ادامه)

- تئوری Herbrand (۱۹۳۰): اگر جمله α توسط پایگاه دانش FOL قابل استلزام باشد، آنگاه این جمله توسط زیرمجموعه محدودی از پایگاه دانش گزاره ای سازی شده قابل استلزام است
- ایده:

For $n = 0$ to ∞ do
create a propositional KB by instantiating with depth-n terms
see if α is entailed by this KB

- مشکل: اگر KB مستلزم α باشد کار می کند، در غیر اینصورت در حلقه بی نهایت می افتد

- تئوری تورینگ و چرچ (۱۹۳۶): استلزام برای FOL **نیمه تصمیم پذیر** است
  - الگوریتم هایی وجود دارند که برای هر جمله قابل استلزام پاسخ بله را تولید می کنند، ولی الگوریتمی وجود ندارد که برای هر جمله ای که قابل استلزام نباشد پاسخ خیر تولید کند.

مشکلات گزاره ای سازی نمودن

- به نظر می رسد که گزاره ای سازی کردن جملات نامربوط زیادی تولید می کند. برای مثال از جملات زیر:

$\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
$King(John)$
$\forall y\ Greedy(y)$
$Brother(Richard,John)$

- بدیهی است که Evil(John). اما گزاره ای سازی کردن حقایق زیادی مانند Greedy(Richard) تولید می کند که نامربوط می باشند

یکسان سازی (Unification)

- اگر بتوانیم یک جانشینی مانند θ بیابیم به طوری که King(x) و Greedy(x) با King(John) و Greedy(y) تطبیق یابند، آنگاه استنتاج بلافاصله بدست می آید.

  θ = {x/John,y/John}

- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

یکسان سازی (Unification)

- اگر بتوانیم یک جانشینی مانند θ بیابیم به طوری که King(x) و Greedy(x) با King(John) و Greedy(y) تطبیق یابند، آنگاه استنتاج بلافاصله بدست می آید.

  θ = {x/John,y/John}

- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

www.txt.ir

یکسان سازی (Unification)

- اگر بتوانیم یک جانشینی مانند θ بیابیم به طوری که King(x) و Greedy(x) با King(John) و Greedy(y) تطبیق یابند، آنگاه استنتاج بلافاصله بدست می آید.

  θ = {x/John,y/John}

- Unify(α,β) = θ if αθ = βθ

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))	
Knows(John,x)	Knows(x,OJ)	

www.txt.ir

یکسان سازی (Unification)

- اگر بتوانیم یک جانشینی مانند θ بیابیم به طوری که King(x) و Greedy(x) با King(John) و Greedy(y) تطبیق یابند، آنگاه استنتاج بلافاصله بدست می آید.

  θ = {x/John,y/John}

- Unify(α,β) = θ if αθ = βθ

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)	

یکسان سازی (Unification)

- اگر بتوانیم یک جانشینی مانند θ بیابیم به طوری که King(x) و Greedy(x) با King(John) و Greedy(y) تطبیق یابند، آنگاه استنتاج بلافاصله بدست می آید.
  θ = {x/John,y/John}
- Unify(α,β) = θ if αθ = βθ

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 سازی متغیرها: مثلاً Knows(z, OJ)

یکسان سازی

- برای یکسازی سازی Knows(John, x) و Knows(y, z) دو جانشینی وجود دارد یا

1) θ = {y/John, x/z }
2) θ = {y/John, x/John, z/John}

- اولین یکسان ساز عمومی تر از دومی می باشد
- فقط یک عمومی ترین یکسان ساز (MGU) وجود دارد که منحصر به فرد می باشد.

MGU = { y/John, x/z }

الگوریتم یکسان سازی

```
function UNIFY(x, y, θ) returns a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound
          y, a variable, constant, list, or compound
          θ, the substitution built up so far
  if θ = failure then return failure
  else if x = y then return θ
  else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) and COMPOUND?(y) then
      return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  else if LIST?(x) and LIST?(y) then
      return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  else return failure
```


الگوریتم یکسان سازی

```
function UNIFY-VAR(var, x, θ) returns a substitution
  inputs: var, a variable
          x, any expression
          θ, the substitution built up so far
  if {var/val} ∈ θ then return UNIFY(val, x, θ)
  else if {x/val} ∈ θ then return UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) then return failure
  else return add {var/x} to θ
```

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$ where $p_i'\theta = p_i\theta$ for all i

p_1' is $King(John)$ $\quad$ p_1 is $King(x)$
p_2' is $Greedy(y)$ $\quad$ p_2 is $Greedy(x)$
θ is {x/John,y/John} $\quad$ q is $Evil(x)$
q θ is $Evil(John)$

- GMP با پایگاه دانشی از فراکردهای معین (دقیقاً یک لیترال مثبت) کار می کند
- فرض می شود که تمام متغیرها دارای سور عمومی هستند.

GMP صحت

- باید نشان دهیم:

$$p_1', \ldots, p_n', (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\theta$$

به شرطی که برای هر i داشته باشیم: $p_i'\theta = p_i\theta$

- لم: برای هر جمله p توسط UIداریم: $p \models p\theta$

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.
2. $p_1', \backslash; \ldots, \backslash;p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$
3. From 1 and 2, $q\theta$ follows by ordinary Modus Ponens

پایگاه دانش نمونه

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
-
- Prove that Colonel West is a criminal

پایگاه دانش نمونه (ادامه)

... it is a crime for an American to sell weapons to hostile nations:
$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
Nono … has some missiles, i.e., $\exists x$ Owns(Nono,x) $\wedge$ Missile(x):

$Owns(Nono,M1)$ and $Missile(M1)$
… all of its missiles were sold to it by Colonel West
$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$
Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$
An enemy of America counts as "hostile“:
$Enemy(x,America) \Rightarrow Hostile(x)$
West, who is American …

$American(West)$
The country Nono, an enemy of America …

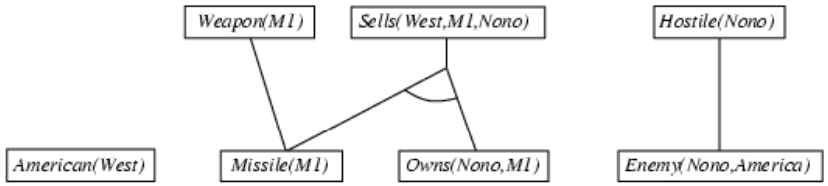
الگوریتم استنتاج رو به جلو

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

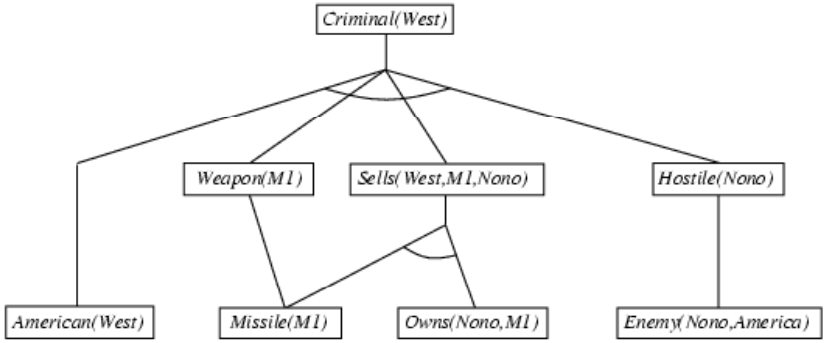
repeat until *new* is empty
  new $\leftarrow \{\}$
  for each sentence r **in** KB **do**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
    for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
        for some $p'_1, \ldots, p'_n$ in KB
      $q' \leftarrow$ SUBST(θ, q)
      if q' is not a renaming of a sentence already in KB or *new* **then do**
        add q' to *new*
        $\phi \leftarrow$ UNIFY(q', α)
        if ϕ is not *fail* **then return** ϕ
  add *new* to KB
return *false*

اثبات توسط استنتاج رو به جلو

اثبات توسط استنتاج رو به جلو



اثبات توسط استنتاج رو به جلو



خصوصیات استنتاج رو به جلو

- برای فراکردهای معین FOL ، کامل و صحیح است
- Datalog = فراکردهای معین FOL بدون سیمبول تابعی
- FC برای Datalog با تعداد تکرارهای محدودی متوقف می شود.
- در حالت کلی، اگر پایگاه دانش مستلزم α نباشد، ممکن است متوقف نشود
- این اجتناب ناپذیر است: استلزام با فراکردهای معین نیمه تصمیم پذیر است

کارآیی استنتاج رو به جلو

- استنتاج رو به جلوی افزایشی: در تکرار k اُم نیاز به تطبیق قانونی که هیچ کدام از شرایطش در تکرار k-1 اضافه نشده اند، نیست

- خود انطباق می تواند پرهزینه باشد:
  – شاخص بندی پایگاه داده اجازه می دهد که حقایق شناخته شده در زمان O(1) بازیابی شوند.

- استنتاج رو به جلو به طور گسترده ای در پایگاه های داده استنتاجی به کار می رود

الگوریتم استنتاج رو به عقب

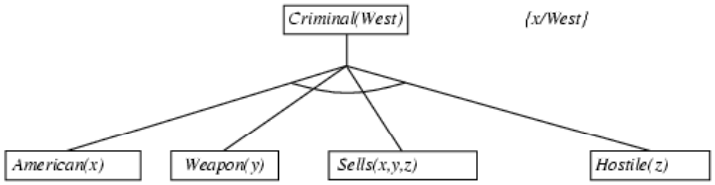
function FOL-BC-ASK(KB, goals, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    goals, a list of conjuncts forming a query
    θ, the current substitution, initially the empty substitution $\{\}$
  local variables: *ans*, a set of substitutions, initially empty

  if *goals* is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST(*goals*))
  for each r **in** KB where STANDARDIZE-APART(r) = $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
    ans $\leftarrow$ FOL-BC-ASK(KB, $[p_1, \ldots, p_n | \text{REST}(goals)]$, COMPOSE($\theta, \theta'$)) $\cup$ *ans*
  return *ans*

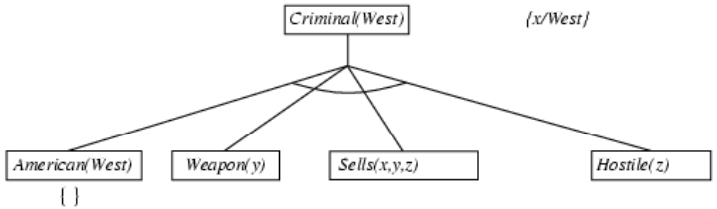
SUBST(COMPOSE(θ1, θ2), p) = SUBST(θ2, SUBST(θ1, p))

مثال استنتاج رو به عقب

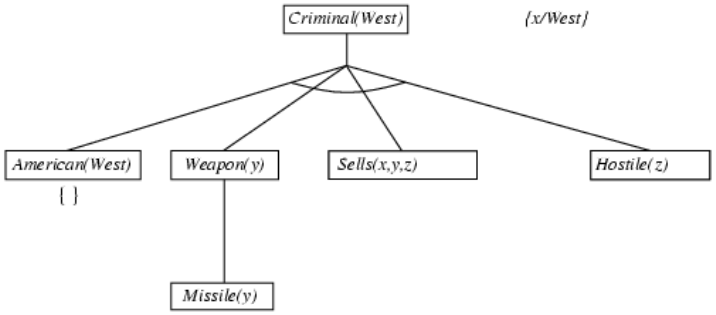
مثال استنتاج رو به عقب



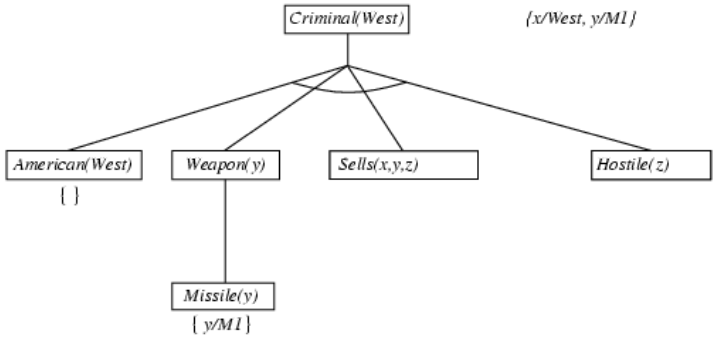
مثال استنتاج رو به عقب



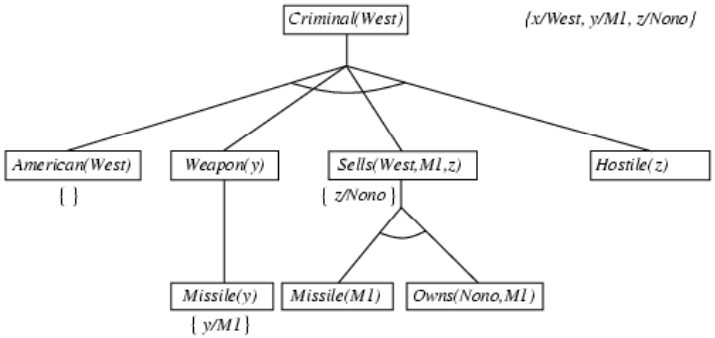
مثال استنتاج رو به عقب



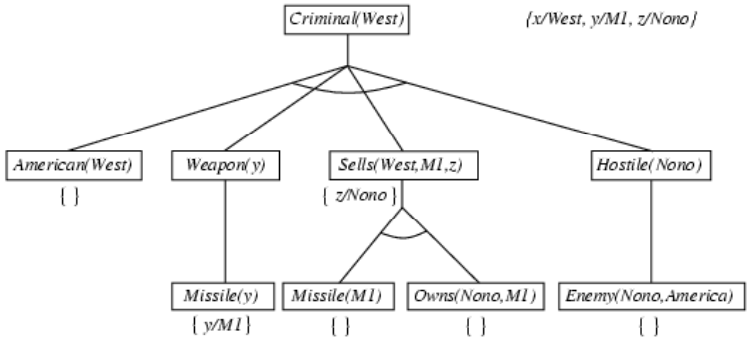
مثال استنتاج رو به عقب



مثال استنتاج رو به عقب



مثال استنتاج رو به عقب



خصوصیات استنتاج روبه عقب

- اثبات: جستجوی اول-عمق بازگشتی – فضای حالت بر حسب اثبات خطی
- ناکامل به دلیل وجود حلقه ها
  - راه حل: مقایسه هدف فعلی با تمام اهداف موجود در پشته
- ناکارآ به دلیل زیرهدف های تکرای (چه موفقیت و چه شکست)
  - راه حل: ذخیره نتایج قبلی (با مصرف حافظه بیشتر) – نوعی memoization
- به طور گسترده ای در **برنامه نویسی منطقی** استفاده می شود

برنامه نویسی منطقی: پرولوگ

- Algorithm = Logic + Control
- اصول: عقبگرد با فراکردهای هورن- به طور گسترده ای در اروپا و ژاپن استفاده شده است (در پروژه نسل پنجم کامپیوترها)
- برنامه = مجموعه ای از فراکردها
- فراکرد =

```
head :- literal1, … literaln.
```

- مثال:

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

خصوصیات پرولوگ

- استنتاج روبه عقب، اول-عمق و از چپ به راست
- عملگرهایی برای محاسبات ریاضی مثلاً: X is Y * Z + 3
- مسندهایی با عوارض جانبی (مثلاً ورودی و خروجی)
- فرض دنیای بسته (‹ نقیض به عنوان شکست ›)
  - مثال: با داشتن

alive(X) :- not dead(X).

alive(Joe) موفق می شود اگر not dead(Joe) شکست بخورد

پرولوگ

- اضافه کردن یک لیست به انتهای لیست دیگر برای تولید یک لیست جدید (عمل Append)

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

- کوئری:

```
append(A,B,[1,2]) ?
```

- پاسخ ها:

```
A=[]    B=[1,2]

A=[1]   B=[2]

A=[1,2] B=[]
```


رزولوشن: خلاصه

- رزولوشن در منطق مرتبه اول

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

که $\text{Unify}(l_i, \neg m_j) = \theta$

- فرض می شود که دو فراکرد استاندارد شده اند به گونه ای که شامل متغیر یکسانی نباشند
- مثال:

$$\frac{\neg Rich(x) \vee Unhappy(x) \qquad Rich(Ken)}{Unhappy(Ken)}$$

with $\theta = \{x/Ken\}$

تبدیل به CNF

- Everyone who loves all animals is loved by someone:
  $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
- 1. Eliminate biconditionals and implications
- $\forall x\ [\neg\forall y\ \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\ Loves(y,x)]$
- 2. Move $\neg$ inwards: $\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg\ \exists x\ p \equiv \forall x\ \neg p$
- $\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)]$
  $\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]$

تبدیل به CNF (ادامه)

3. Standardize variables: each quantifier should use a different one
4. $\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(z,x)]$

4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

   $\forall x\ [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

5. Drop universal quantifiers:
6.
6. $[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

6. Distribute $\vee$ over $\wedge$:

اثبات بوسیله رزولوشن: مثال