



هوش مصنوعی

Artificial Intelligence

نام کتاب : هوش مصنوعی رهیاتی نوین

مؤلف : راسل و نورویگ

مهدی بازرگانی
دانشگاه آزاد اسلامی زنجان



فصل چهارم

روش های جستجوی آگاهانه

<http://inbaa.com> , <http://csiz.ir>

مقدمه

3

- 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

4

- 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

5

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

6

▶ ایده: برای هر گره از یک تابع ارزیابی (evaluation function) استفاده کن.
 • تخمین "میزان مطلوب بودن" گره
 ◀ هر بار مطلوب ترین گره گسترش نیافته را بسط می دهد.

▶ پیاده سازی:

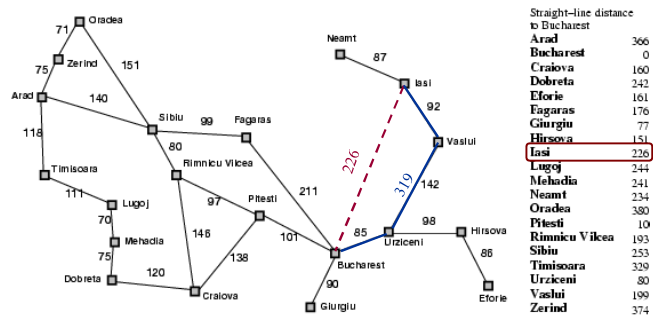
fringe یک صف می باشد که بر اساس میزان مطلوب بودن گره ها مرتب می باشد.

▶ موارد خاص:

- جستجوی حریصانه (Greedy search)
- جستجوی A*

نقشه رومانی به همراه هزینه مراحل بر حسب km

7



جستجوی اول - بهترین حریصانه

8

□ تابع ارزیابی $f(n) = h(n)$ (هیوریستیک)

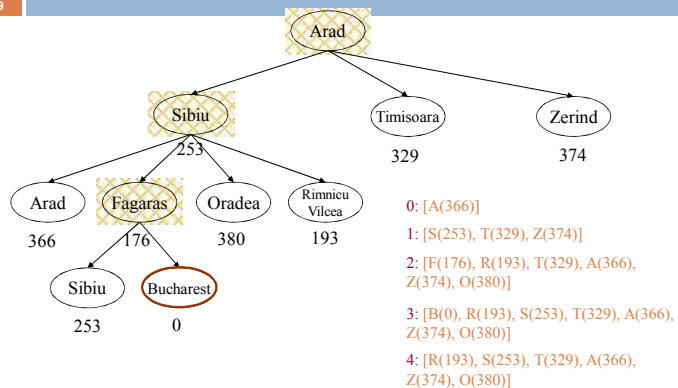
□ = هزینه تخمینی از گره n تا نزدیکترین هدف

□ 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

9



خواص جستجوی حریصانه

10

- ▶ **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.
- ▶ **بهبوده؟** خیر، (با توجه به مثال قبل)

جستجوی A*

11

- ▶ 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.

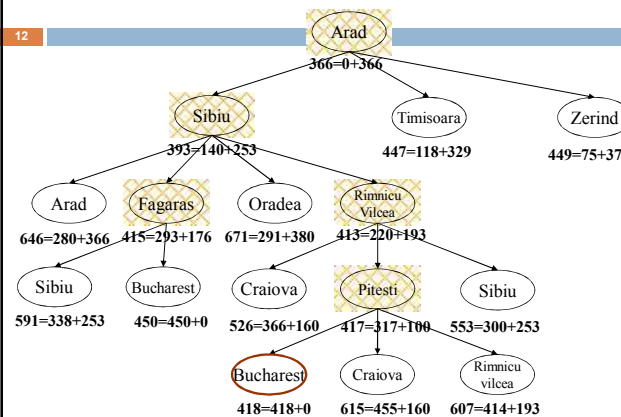
▶ تابع ارزیابی

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

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

مثال جستجوی A*

12



هیوریستیک قابل قبول

14

- یک هیوریستیک مانند $h(n)$ قابل قبول است اگر برای هر گره n داشته باشیم: $h(n) \leq h^*(n)$ 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.

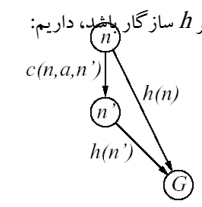
جستجوی A^*

13

- از A^* یک هیوریستیک قابل قبول (admissible) استفاده می کند، یعنی، همواره $h(n) \leq h^*(n)$ که در آن $h^*(n)$ هزینه واقعی n تا هدف می باشد.
- هم چنین داریم $h(n) \geq 0$ و $h(G) = 0$ (G یک هدف می باشد).
- به طور کلی: $0 \leq h(n) \leq h^*(n)$
- 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).

اثبات لم: سازگاری (Consistency)

16

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

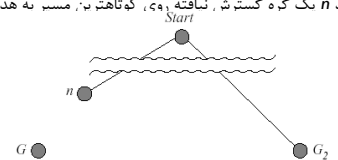
$$= g(n) + c(n, a, n') + h(n')$$

$$\geq g(n) + h(n)$$

$$\geq f(n)$$
- 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^* (اثبات)

15

- فرض کنید یک جواب زیر بهینه مانند G_2 تولید شده و در صف قرار دارد. هم چنین فرض کنید n یک گره گسترش یافته است، که تاهت به مسیری به هدف بهینه G باشد (؟؟)
- 
- چون، $h(G_2) = 0$
 - چون، G_2 زیر بهینه است
 - چون، h قابل قبول است
 - چون $f(G_2) = g(G_2) > g(G) \geq f(n)$
 - چون $f(G_2) \geq f(n)$ ، الگوریتم A^* هیچ وقت G_2 را برای گسترش یافتن انتخاب نمی کند.

خصوصیات A*

18

- 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:

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

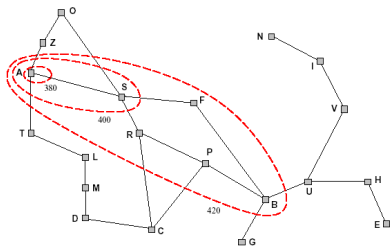
بهینگی A*

17

▶ **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 $f_i < f_{i+1}$



جستجوی هیوریستیک با حافظه محدود

20

- چند راه حل برای مسأله حافظه در A^* (با حفظ خصوصیات کامل بودن و بهینگی):
- جستجوی عمیق کننده تکراری 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

خصوصیات A*

19

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

▶ بهینگی؟؟ 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 کارایی بهینه (optimally efficient) می باشد!

جستوی اول-بهترین بازگشتی (RBFS)

21

```
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
```

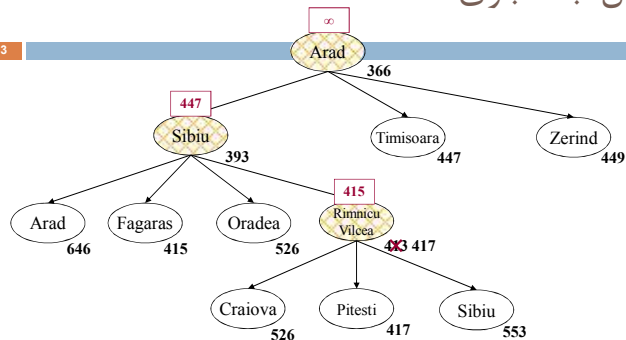
جستجوی RBFS

22

- 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.

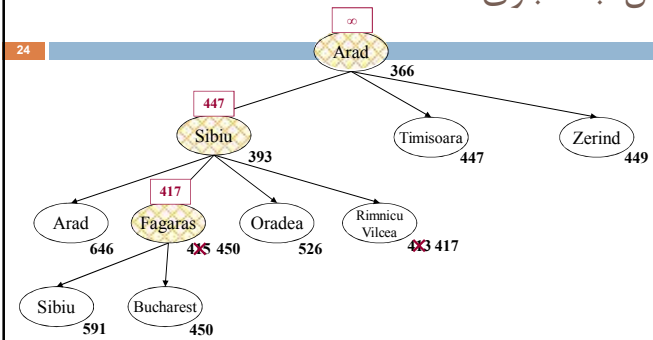
مثال جستجوی RBFS

23



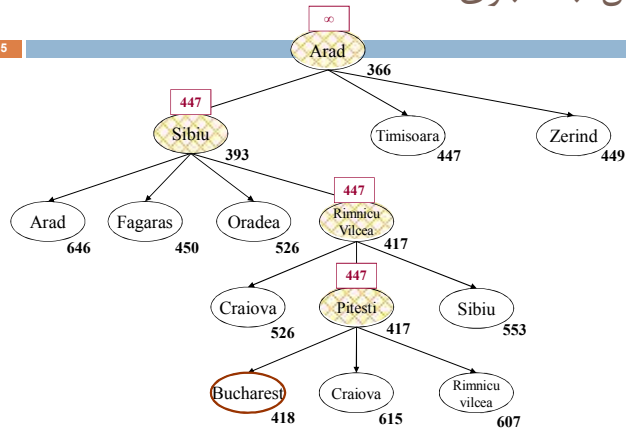
مثال جستجوی RBFS

24



مثال جستجوی RBFS

26



ارزیابی RBFS

26

- 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 پیچیدگی 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 پیچیدگی 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 آن استفاده کنند.

(Simplified) Memory Bounded A*

27

- ایده: استفاده از تمامی حافظه موجود
- یعنی، گسترش بهترین گره های برگ 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

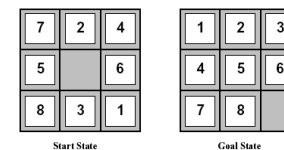
هیوریستیک های قابل قبول

28

مثال برای معمای هشت:

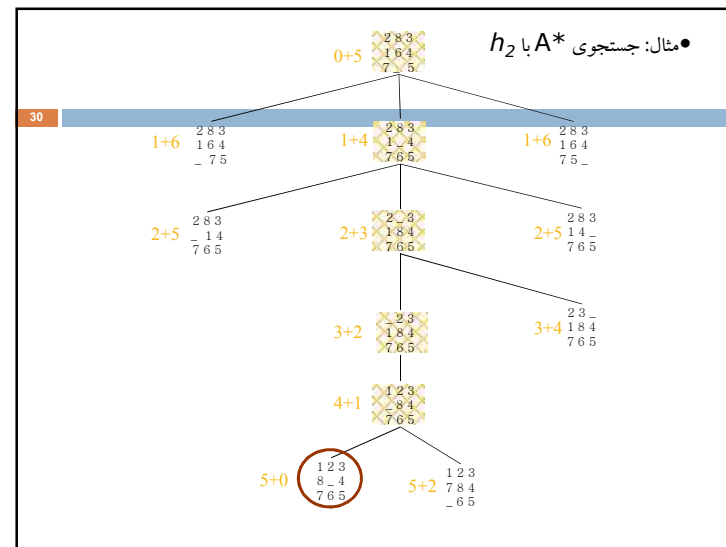
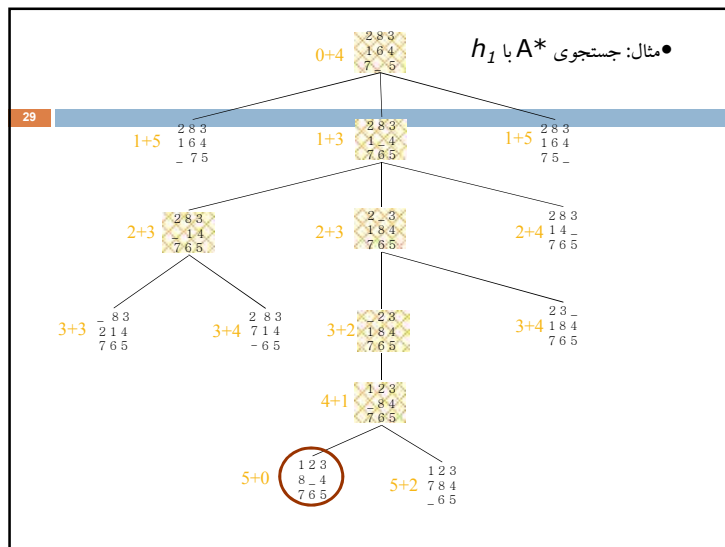
$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance



$$h_1(S) = 6$$

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



31

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:

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

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

32

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:

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

- مثال: اگر A^* راه حلی را در عمق 5 با استفاده از 52 گره پیدا کند، فاکتور انشعاب موثر را محاسبه کنید.

پاسخ:

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

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

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.

مقایسه بین هزینه جستجو و فاکتور انشعاب موثر

d	Search Cost			Effective Branching Factor		
	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

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)

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 مسأله 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.

تپه نوردی (یا گرادیان صعودی/نزولی)

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

```

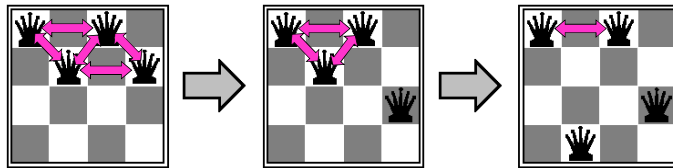
function HILL-CLIMBING(problem) returns a state that is a local maximum
inputs: problem, a problem
local variables: current, a node
                 neighbor, a node

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
    
```

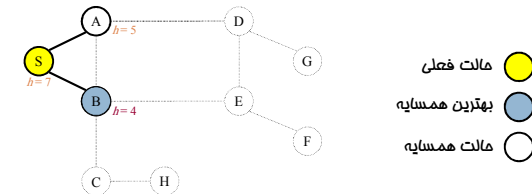
مثال: n -وزیر

□ 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.

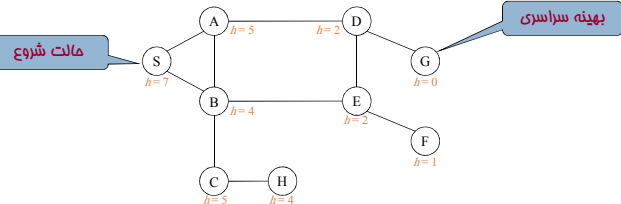
□ یکی از وزیرها را در ستون خودش به گونه ای جابه جا کن که تعداد برخوردها کاهش یابد.



مثال: جستجوی تپه نوردی

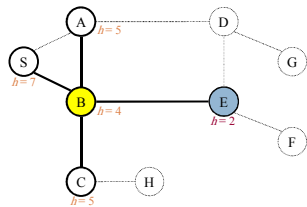


مثال: جستجوی تپه نوردی



مثال: جستجوی تپه نوردی

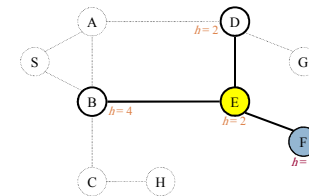
41



- حالت فعلی
- بهترین همسایه
- حالت همسایه

مثال: جستجوی تپه نوردی

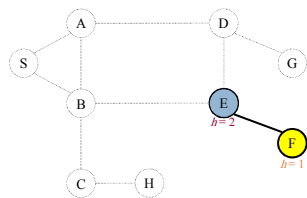
42



- حالت فعلی
- بهترین همسایه
- حالت همسایه

مثال: جستجوی تپه نوردی

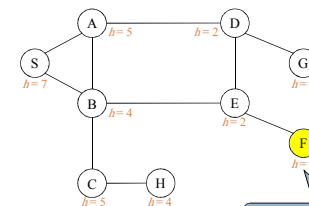
43



- حالت فعلی
- بهترین همسایه
- حالت همسایه

مثال: جستجوی تپه نوردی

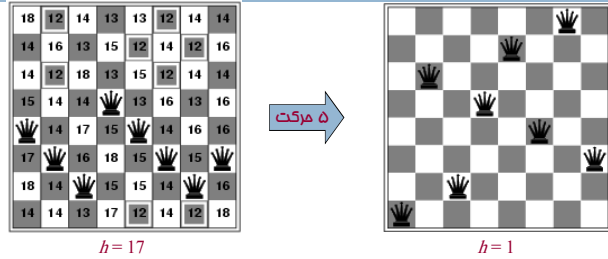
44



- حالت فعلی
- بهترین همسایه
- حالت همسایه

جستوی تپه نوردی : مثال ۸ وزیر

45

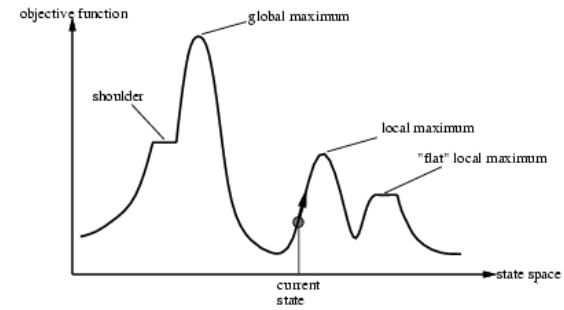


h = تعداد جفت وزیرهایی که بطور مستقیم و یا بطور غیر مستقیم یکدیگر را تهدید می کنند.

تپه نوردی (ادامه)

46

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

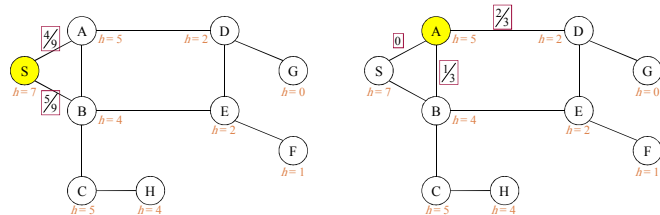


انواع دیگر تپه نوردی (تپه نوردی اتفاقی)

47

تپه نوردی اتفاقی

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

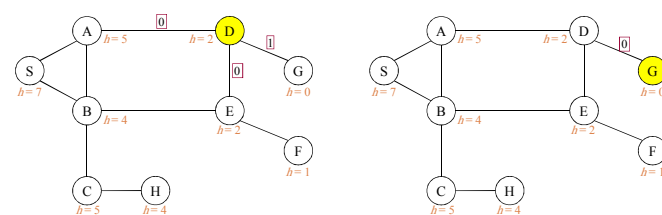


انواع دیگر تپه نوردی (تپه نوردی اتفاقی)

48

تپه نوردی اتفاقی

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

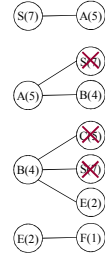
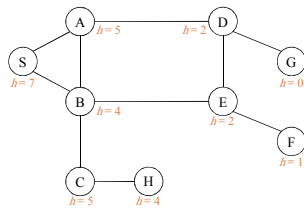


انواع دیگر تپه نوردی (تپه نوردی اولین انتخاب)

49

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

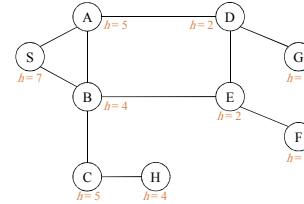


انواع دیگر تپه نوردی (تپه نوردی با شروع مجدد تصادفی)

50

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



$S \rightarrow B \rightarrow E \rightarrow F$ ✗
 $C \rightarrow B \rightarrow E \rightarrow F$ ✗
 H ✗
 $C \rightarrow H$ ✗
 $A \rightarrow D \rightarrow G$

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

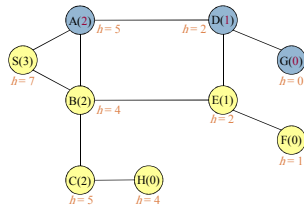
راه حل: تکرار تپه نوردی تا زمانی که وقت داریم.

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

51

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 + 1 * تعداد متوسط مراحل موفقیت



احتمال موفقیت = $\frac{1}{3}$
 تعداد تکرارها = 3
 تعداد متوسط مراحل موفقیت = 1
 تعداد متوسط مراحل شکست = $\frac{8}{6}$
 $(3 - 1) \times \frac{8}{6} + 1 \times 1 \cong 4$

Simulated Annealing

52

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  $e^{\Delta E/T}$ 
    
```

خصوصیات آنلینگ شبیه سازی شده

54

- 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 اندازه کافی آرام کاهش بیاید، 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

آنلینگ شبیه سازی شده

53

- ▶ 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:

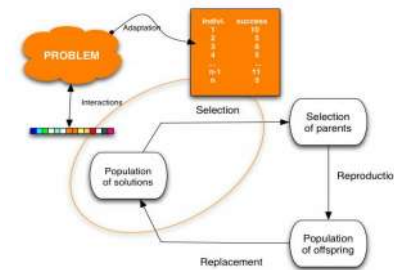
$$e^{-\Delta E/T}$$

T : تعداد مراحل بهبود راه حل
 ΔE : میزان کاهش در هر مرحله

الگوریتم ژنتیک

56

- نوعی از local beam search به همراه ترکیب جنسی



(جستجوی محلی دسته ای) Local beam search

55

- 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.

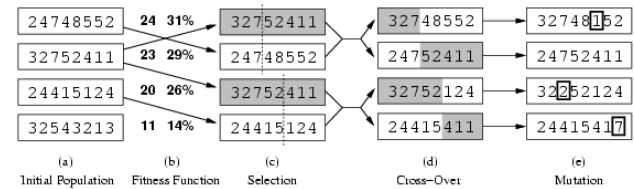
الگوریتم های ژنتیک

57

- 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

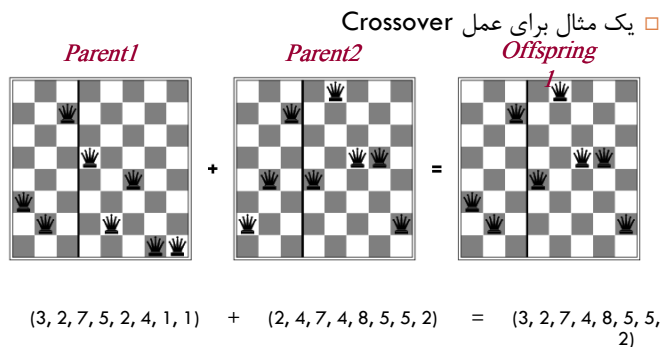
58



- 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

59



الگوریتم های ژنتیک

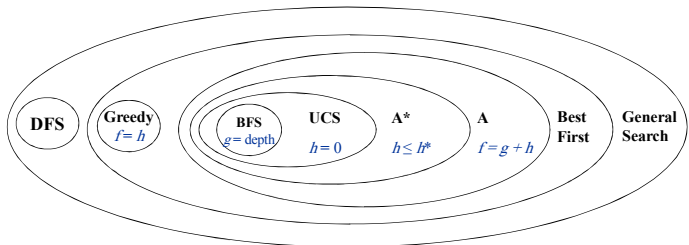
60

```
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

61



مسائل اکتشافی

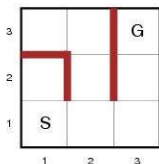
62

- 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

63

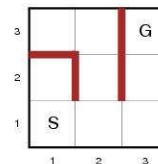
- دانش عامل:
- عمل (ها): لیست اعمال مجاز در حالت s
- $c(s, a, s')$: تابع هزینه گام (پس از تعیین s')
- **GOAL-TEST(s)**
- عامل می تواند حالت قبلی را تشخیص دهد
- اعمال قطعی هستند
- دستیابی به هیوربستیک قابل قبول $h(s)$
- مانند فاصله مانهاتانی



مسائل جستجوی online

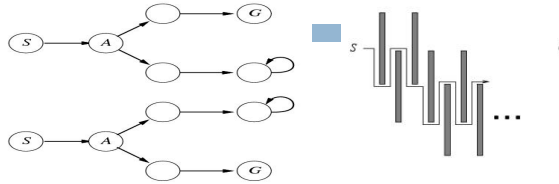
64

- 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 نامحدود باشد
- در مواردی که عامل به طور تصادفی به یک بن بست می رسد



استدلال دشمنانه

66



- 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 این بست ها در تمامی فضاهای حالت اجتناب کند

عامل های جستجوی Online

66

- 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 مثلاً 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

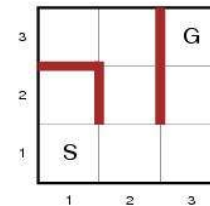
67

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'] \leftarrow ACTIONS(s')
if s is not null **then do**
result[a, s] \leftarrow s'
 add s to the front of *unbacktracked*[s']
if *unexplored*[s'] is empty **then**
if *unbacktracked*[s'] is empty **then return** stop
else $a \leftarrow$ an action b such that *result*[b, s'] = POP(*unbacktracked*[s'])
else $a \leftarrow$ POP(*unexplored*[s'])
 $s \leftarrow s'$
return a

جستجوی عمقی online ، مثال

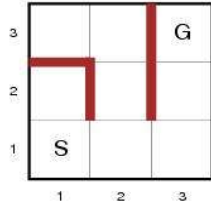
68



- مساله مسیر پر پیچ و خم بر روی یک صفحه 3×3
- حالت اولیه: $s' = (1, 1)$
- *Result* و *Unexplored*(UX) و *Unbacktracked*(UB) ... و تهی هستند
- نیز تهی هستند و S

جستجوی عمقی online، مثال

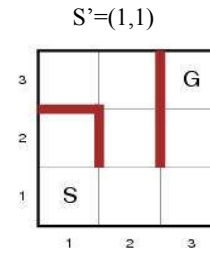
69



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

جستجوی عمقی online، مثال

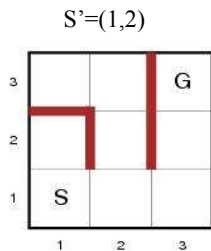
70



- GOAL-TEST((1,1))?
 - $S \text{ not} = G$ 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
- $a = \text{RIGHT}$, $s = (1,1)$
- return a

جستجوی عمقی online، مثال

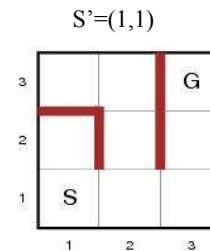
71



- GOAL-TEST((1,2))?
 - $S \text{ not} = G$ 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
- $a = \text{LEFT}$, $s = (1, 2)$
- return a

جستجوی عمقی online، مثال

72



- GOAL-TEST((1, 1))?
 - $S \text{ not} = G$ 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)
 - $b = \text{RIGHT}$
- $a = \text{RIGHT}$, $s = (1, 1) \dots$

جستجوی عمقی online، مثال

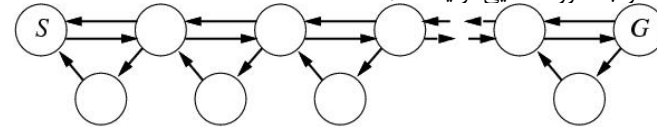
73

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

جستجوی محلی online

74

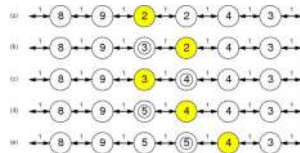
- ❑ تپه نوردی **online** می باشد
- ❑ یک حالت ذخیره شده است
- ❑ عملکرد بد به دلیل وجود ماکزیمم های محلی
- ❑ شروع مجدد تصادفی غیر ممکن است
- ❑ راه حل: حرکت تصادفی باعث کاوش می شود (می تواند حالات بسیاری را به صورت نمایی تولید کند)



جستجوی محلی online

75

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



Learning real-time A*

76

```

function LRTA*-COST (s, a, s', H) return a cost estimate
  if s' is undefined then 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
    
```