بسم الله الرحمن الرحیم

هوش مصنوعی پیشرفته

درس ۳

عامل‌های جستجوی برخط و محیط‌های ناشناخته

Online Search Agents and Unknown Environments

کاظم فولادی
دانشکده مهندسی برق و کامپیوتر
دانشگاه تهران

http://courses.fouladi.ir/aai

روش‌های جستجو

برون‌خط/برخط

روش‌های جستجو	
برون‌خط *Offline*	**برخط** *Online*
راه‌حل پیش از اجرا مشخص می‌شود.	محاسبات و کنش‌ها یک درمیان انجام می‌شوند.

نخست انجام یک کنش،
سپس مشاهده‌ی محیط و
محاسبه‌ی کنش بعدی

ضروری برای محیط‌های پویا و نیمه‌پویا
و
محیط‌های ناشناخته
(مسائل اکتشافی)

امکان در نظر گرفتن همه‌ی اقتضائات وجود ندارد.

مسائل جستجوی برخط

ONLINE SEARCH PROBLEMS

مسائل جستجوی برخط تنها توسط یک عامل اجرا کننده‌ی کنش‌ها می‌تواند حل شود
(به‌جای پردازش‌های محاسباتی محض)

دانایی عامل		
$\diamondsuit$ ACTION(s)	تابع کنش	کنش‌های مجاز در حالت s
$\diamondsuit$ $c(s,a,s')$	تابع هزینه‌ی گام	تا مشخص نشدن s' قابل استفاده نیست.
$\diamondsuit$ GOAL-TEST(s)	آزمون هدف	مشخص‌کننده‌ی حالت هدف

فرضیات:

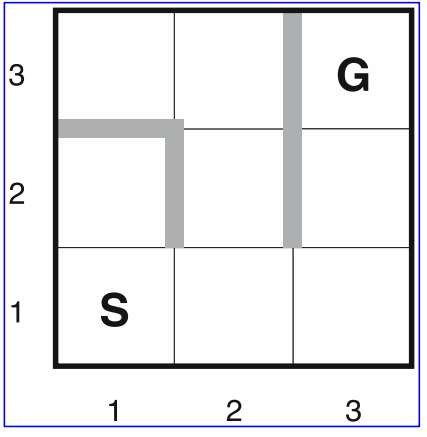
- عامل می‌تواند حالت‌های قبلاً ملاقات‌شده را بازشناسی کند
- کنش‌ها قطعی هستند
- عامل به یک تابع هیوریستیک $h(s)$ دسترسی دارد که فاصله از حالت جاری تا یک حالت هدف را تخمین می‌زند (مانند: فاصله شهری)

مسائل جستجوی برخط

مثال

ONLINE SEARCH PROBLEMS

هدف عامل: شروع از S و رسیدن به هدف G با کمترین هزینه‌ی ممکن



یک مسئله‌ی ماز ساده.

عامل هیچ چیز در مورد محیط نمی‌داند.

مثال: رباتی که در یک ساختمان جدید قرار داده شده است و باید نقشه‌ای بسازد که او را از S به G ببرد.

مسائل جستجوی برخط

ONLINE SEARCH PROBLEMS

هدف عامل: رسیدن به هدف با کمترین هزینه‌ی ممکن

هزینه: هزینه‌ی کل مسیر پیمایش شده

معیار کارآیی

نسبت رقابتی
Competitive Ratio

نسبت هزینه به هزینه‌ی مسیر راه‌حل، اگر فضای جستجو از قبل معلوم باشد.

ممکن است بی‌نهایت باشد (وقتی عامل تصادفاً به بن‌بست می‌رسد و نمی‌تواند کنش خود را معکوس کند.)

مسائل جستجوی برخط

اجتناب‌ناپذیری بن‌بست

ONLINE SEARCH PROBLEMS

هیچ الگوریتمی نمی‌تواند در همه‌ی فضاهای حالت از بن‌بست اجتناب کند.

اثبات با بحث تخاصمی (Adversary Argument):

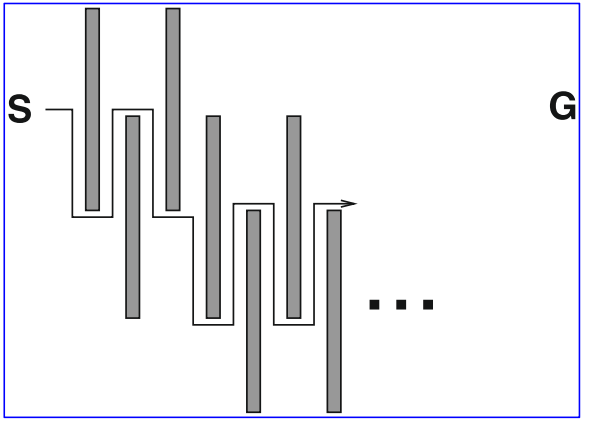
فرض می‌کنیم در حالی که عامل فضای حالت را کشف می‌کند،
یک رقیب وجود دارد که فضای حالت را می‌سازد.
مطابق شکل، حالت‌های S و A ملاقات شده‌اند؛
اما در دو گام دیگر در یکی از فضاهای حالت به بن‌بست برخورد می‌کند.

مسائل جستجوی برخط

مثال

ONLINE SEARCH PROBLEMS

هدف عامل: شروع از S و رسیدن به هدف G با کمترین هزینه‌ی ممکن



یک محیط دوبعدی که می‌تواند باعث شود یک عامل جستجوی برخط،
یک راه دلخواه ناکارآمد را به سمت هدف دنبال کند.

هر انتخابی که عامل انجام بدهد، رقیب راه را با یک دیوار بلند نازک دیگر می‌بندد
در نتیجه مسیر دنبال‌شده بسیار بلندتر از بهترین مسیر ممکن می‌شود.

مسائل جستجوی برخط

فضاهای حالت اکتشاف‌پذیر به‌صورت امن

SAFELY EXPLORABLE STATE-SPACE

بن‌بست‌ها یک دشواری جدی برای مسئله‌ی اکتشاف توسط ربات‌هاست؛
برای همین، فرض می‌کنیم که فضای حالت قابل اکتشاف به‌صورت امن باشد.

فضای حالت اکتشاف‌پذیر به‌صورت امن
Safely Explorable State-Space

فضای حالتی که در آن برخی حالت‌های هدف از همه‌ی حالت‌های دسترس‌پذیر، قابل دسترس باشد.

مثال: فضاهای حالت با کنش‌های وارون‌پذیر (مثل: معمای ۸، ماز، ...)

عامل‌های جستجوی برخط

ONLINE SEARCH AGENTS

عامل یک نقشه از محیط را ایجاد و نگهداری می‌کند.

- این نقشه بر اساس ورودی ادراکی به‌هنگام می‌شود.
- این نقشه برای تصمیم‌گیری در مورد کنش بعدی استفاده می‌شود.

تفاوت مهم با عامل‌های جستجوی برون‌خط:

نسخه‌ی برخط تنها گره‌ای را می‌تواند گسترش دهد که به صورت فیزیکی در آن قرار دارد (ترتیب گسترش محلی).
(جستجوی عمق–اول این ویژگی را دارد)

ONLINE DFS AGENTS

نقشه‌ی محیط در جدول $\text{RESULT}[s, a]$

حالت حاصل از اجرای کنش a در حالت s را ثبت می‌کند.

- هرگاه یک کنش از حالت جاری کشف نشده باشد، عامل آن کنش را آزمایش می‌کند.
- هرگاه عامل همه‌ی کنش‌های یک حالت را آزمایش کرده باشد،
  - در **DFS برون‌خط:** آن حالت به‌سادگی از صف حذف می‌شود.
  - در **DFS برخط:** عامل باید به‌صورت فیزیکی عقب‌گرد کند.

عقب‌گرد (backtracking): برگشتن به حالتی که عامل اخیراً از آن وارد حالت فعلی شده است.

(لازم است عامل در یک جدول برای هر حالت، حالت‌های ماقبلی که تاکنون به آنها عقب‌گرد نکرده است را نگهداری کند.)
اگر حالت دیگری باقی نمانده باشد که عامل به آن عقب‌گرد کند، جستجو کامل می‌شود.

عامل‌های جستجوی عمق-اول برخط

الگوریتم

ONLINE DFS AGENTS

```
function ONLINE-DFS-AGENT(s') returns an action
  inputs: s', a percept that identifies the current state
  persistent: result, a table indexed by state and action, initially empty
              untried, a table that lists, for each state, the actions not yet tried
              unbacktracked, a table that lists, for each state, the backtracks 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 (not in untried) then untried[s'] ← ACTIONS(s')
  if s is not null then
      result[s, a] ← s'
      add s to the front of unbacktracked[s']
  if untried[s'] is empty then
      if unbacktracked[s'] is empty then return stop
      else a ← an action b such that result[s', b] = POP(unbacktracked[s'])
  else a ← POP(untried[s'])
  s ← s'
  return a
```

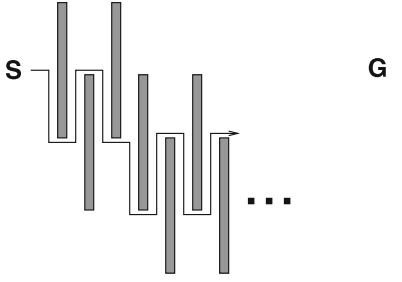
عامل‌های جستجوی عمق-اول برخط

خصوصیات

ONLINE DFS AGENTS

در بدترین حالت، هر پیوند در فضای حالت، دو مرتبه پیمایش می‌شود.

ممکن است عامل یک گشت طولانی را بپیماید،
حتی هنگامی که به هدف نزدیک است.
(حل این مشکل با روی‌کرد عمیق‌کننده‌ی تکراری برخط)

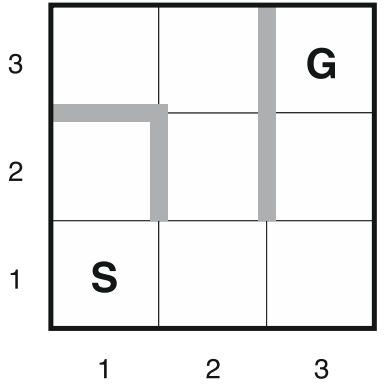


جستجوی عمق-اول برخط تنها زمانی کار می‌کند که کنش‌ها وارون‌پذیر باشند.

عامل‌های جستجوی عمق-اول برخط

مثال (۱ از ۷)

ONLINE DFS AGENTS



$s' = (1, 1)$

Assume maze problem on 3x3 grid.

$s' = (1, 1)$ is initial state

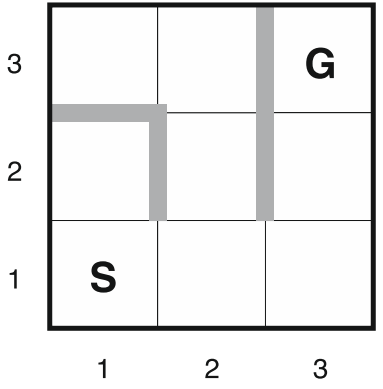
Result, $untried$ (UX), $unbacktracked$ (UB), ... are empty

s ,a are also empty

عامل‌های جستجوی عمق-اول برخط

مثال (۲ از ۷)

ONLINE DFS AGENTS



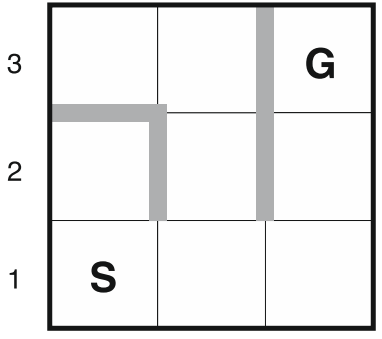
$s' = (1,1)$

- GOAL-TEST$((1,1))$?
  - $s' \neq G$ 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$(UX[(1,1)]) \rightarrow a$
  - $a =$ UP
- $s = (1,1)$
- return a

عامل‌های جستجوی عمق-اول برخط

مثال (۳ از ۷)

ONLINE DFS AGENTS



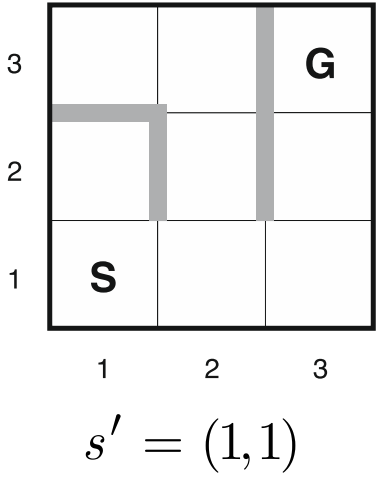
$s' = (2,1)$

- GOAL-TEST$((2,1))$?
  - $s' \neq G$ thus $false$
- $(2,1)$ a new state?
  - $true$
  - ACTION$((2,1)) \rightarrow UX[(2,1)]$:={DOWN}
- s is null?
  - $false$ $(s = (1,1))$
  - $result[\text{UP},(1,1)] := (2,1)$
  - $UB[(2,1)] := (1,1)$
- $UX[(2,1)]$ empty?
  - $false$
- POP$(UX[(2,1)]) \rightarrow a$
  - $a =$ DOWN
- $s = (2,1)$
- return a

عامل‌های جستجوی عمق-اول برخط

مثال (۴ از ۷)

ONLINE DFS AGENTS



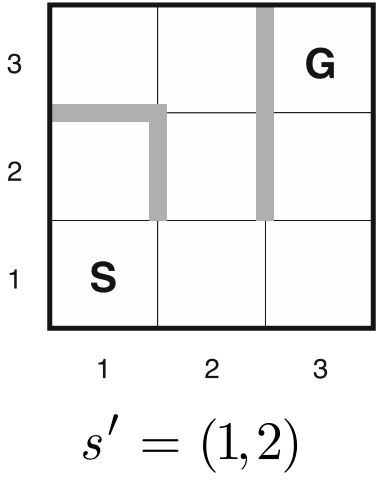
$s' = (1,1)$

- GOAL-TEST$((1,1))$?
  - $s' \neq G$ thus $false$
- $(1,1)$ a new state?
  - $false$
- s is null?
  - $false$ $(s = (2,1))$
  - $result[\text{DOWN}, (2,1)] := (1,1)$
  - $UB[(1,1)] := (2,1)$
- $UX[(1,1)]$ empty?
  - $false$
- POP$(UX[(1,1)]) \rightarrow a$
  - $a =$ RIGHT
- $s = (1,1)$
- return a

عامل‌های جستجوی عمق-اول برخط

مثال (۵ از ۷)

ONLINE DFS AGENTS



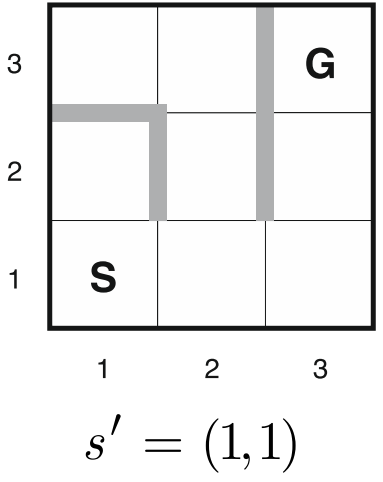
$s' = (1,2)$

- GOAL-TEST$((1,2))$?
  - $s' \neq G$ thus $false$
- $(1,2)$ a new state?
  - $true$
  - ACTION$((1,2)) \rightarrow UX[(1,2)]:=$ { RIGHT, UP, LEFT }
- s is null?
  - $false$ $(s = (1,1))$
  - $result[\text{RIGHT}, (1,1)] := (1,2)$
  - $UB[(1,2)] := (1,1)$
- $UX[(1,2)]$ empty?
  - $false$
- POP$(UX[(1,2)]) \rightarrow a$
  - $a =$ LEFT
- $s = (1,2)$
- return a

عامل‌های جستجوی عمق-اول برخط

مثال (۶ از ۷)

ONLINE DFS AGENTS



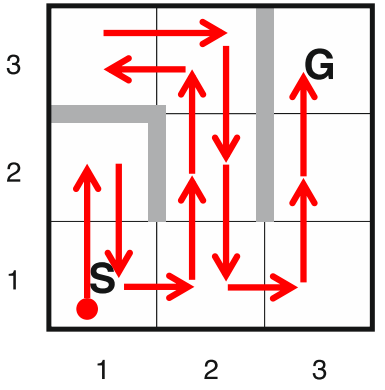
$s' = (1,1)$

- GOAL-TEST$((1,1))$?
  - $s' \neq G$ thus $false$
- $(1,1)$ a new state? $false$
- s is null?
  - $false$ $(s = (1,2))$
  - $result[\text{LEFT}, (1,2)] := (1,1)$
  - $UB[(1,1)] := (1,2), (2,1)$
- $UX[(1,1)]$ empty?
  - $true$
  - $UB[(1,1)]$ empty? $false$
- $a = b$ for b in $result[b, (1,1)] = (1,2)$
  - $b =$ RIGHT
- $a =$ RIGHT
- $s = (1,1)$
- return a

عامل‌های جستجوی عمق-اول برخط

مثال (۷ از ۷)

ONLINE DFS AGENTS



عامل‌های جستجوی برخط

جستجوی محلی برخط

ONLINE LOCAL SEARCH

تپه‌نوردی، هم‌اکنون یک الگوریتم جستجوی برخط است.
(در هر گام، یک حالت ذخیره می‌شود)

مشکلات:

- کارآیی پایین در اثر ماکزیمم‌های محلی
- شروع مجدد تصادفی برای جستجوی محلی برخط ناممکن است.

راه‌حل‌ها:

- (۱) گام‌برداری تصادفی
- (۲) اضافه کردن حافظه به تپه‌نوردی

عامل‌های جستجوی برخط

جستجوی محلی برخط: گام‌برداری تصادفی

ONLINE LOCAL SEARCH: RANDOM WALK

گام‌برداری تصادفی
Random Walk

به طور ساده، یکی از کنش‌های موجود از حالت فعلی به‌طور تصادفی انتخاب می‌شود.

می‌توان به کنش‌هایی که هنوز آزمایش نشده‌اند، ترجیح‌هایی را نسبت داد.

* اکتشاف وارد حل مسئله می‌شود (ممکن است تعداد نمایی گام ایجاد کند)

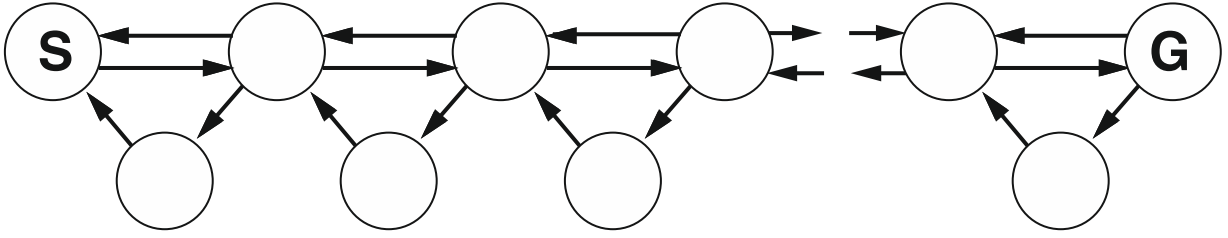
ثابت می‌شود که گام‌برداری تصادفی سرانجام یک هدف را می‌یابد اگر فضای حالت متناهی باشد.

عامل‌های جستجوی برخط

جستجوی محلی برخط: گام‌برداری تصادفی: مثال

ONLINE LOCAL SEARCH: RANDOM WALK

یک فضای حالت که احتمال حرکت پس‌رو در آن ۲ برابر حرکت پیش‌رو است:



یک محیط که گام‌برداری تصادفی در آن برای پیدا کردن هدف تعدادی نمایی گام را برخواهد داشت.

عامل‌های جستجوی برخط

جستجوی محلی برخط: اضافه کردن حافظه به تپه‌نوردی (A* بی‌درنگ یادگیرنده)

LEARNING REAL-TIME A* (LRTA*)

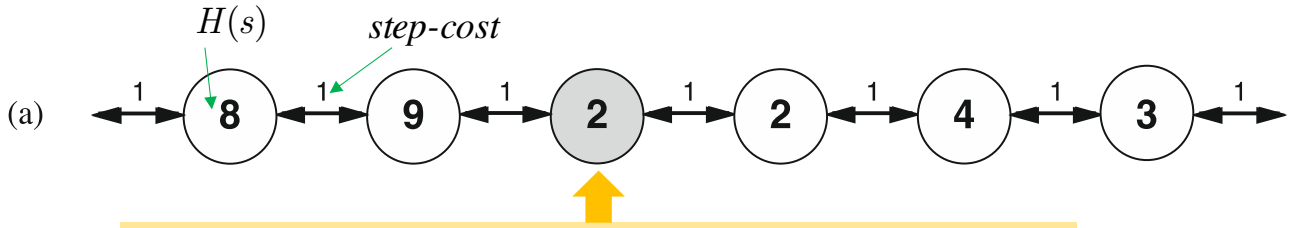
یک «بهترین تخمین فعلی» $H(s)$ از هزینه‌ی رسیدن به هدف
با شروع از هر یک از حالات ملاقات شده، ذخیره می‌شود.

- $H(s)$ در ابتدا تخمین هیوریستیک $h(s)$ است.
- در ادامه $H(s)$ به‌هنگام می‌شود هرگاه عامل در فضای حالت فعلی به تجربه‌ای دست می‌یابد.

عامل‌های جستجوی برخط

جستجوی محلی برخط: اضافه کردن حافظه به تپه‌نوردی (A* بی‌درنگ یادگیرنده): مثال (۱ از ۳)

LEARNING REAL-TIME A* (LRTA*)



به نظر می‌رسد عامل در یک می‌نیمم محلی گیر کرده است.

هزینه‌ی رسیدن به هدف از طریق یک همسایه =
هزینه‌ی رسیدن به همسایه + هزینه‌ی تخمینی رسیدن به هدف از آن همسایه:

$$\text{cost-estimate}(s,a,G) = c(s,a,s') + H(s')$$

برای این حالت دو کنش ممکن است:

- Left: با هزینه‌ی $1 + 9 = 10$
- Right: با هزینه‌ی $1 + 2 = 3$ (گزینه‌ی بهتر)

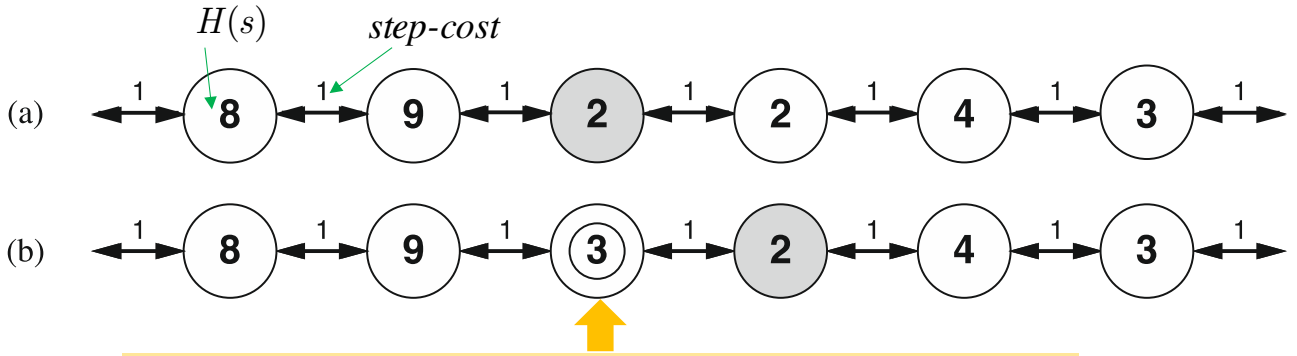
$\Leftarrow$

$H(s) = 2$ بیش از حد خوش‌بینانه بود و باید به $H(s) = 3$ به‌هنگام شود.

عامل‌های جستجوی برخط

جستجوی محلی برخط: اضافه کردن حافظه به تپه‌نوردی (A* بی‌درنگ یادگیرنده): مثال (۱ از ۳)

LEARNING REAL-TIME A* (LRTA*)

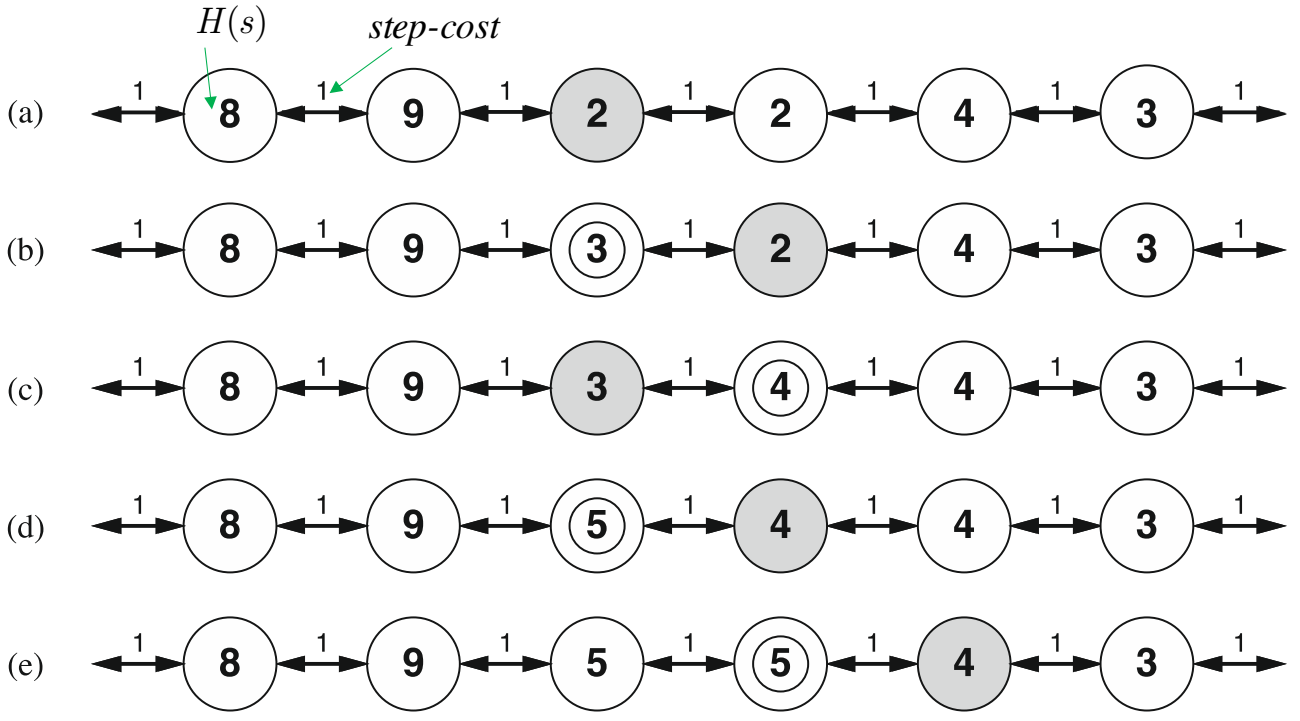


$H(s) = 2$ بیش از حد خوش‌بینانه بود و باید به $H(s) = 3$ به‌هنگام شود.

عامل‌های جستجوی برخط

جستجوی محلی برخط: اضافه کردن حافظه به تپه‌نوردی (A* بی‌درنگ یادگیرنده): مثال (۱ از ۳)

LEARNING REAL-TIME A* (LRTA*)



با ادامه دادن این روند، عامل چند بار جلو و عقب می‌رود و هر بار H را به‌هنگام می‌کند.
اکنون عامل می‌نیمم محلی را پشت سر می‌گذارد.

عامل‌های جستجوی برخط

جستجوی محلی برخط: اضافه کردن حافظه به تپه‌نوردی (A* بی‌درنگ یادگیرنده): الگوریتم

LEARNING REAL-TIME A* (LRTA*)

function LRTA*-AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: $result$, a table, indexed by state and action, 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** $H[s'] \leftarrow h(s')$
  if s is not null
    $result[s, a] \leftarrow s'$
    $H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)}$ LRTA*-COST($s, b, result[s, b], H$)
  $a \leftarrow$ an action b in ACTIONS(s') that minimizes LRTA*-COST($s', b, result[s', b], H$)
  $s \leftarrow s'$
  return a

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

عامل‌های جستجوی برخط

جستجوی محلی برخط: اضافه کردن حافظه به تپه‌نوردی (A* بی‌درنگ یادگیرنده): ویژگی‌های الگوریتم

LEARNING REAL-TIME A* (LRTA*)

```
function LRTA*-AGENT(s') returns an action
  inputs: s', a percept that identifies the current state
  persistent: result, a table, indexed by state and action, 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')
  if s is not null
      result[s, a] ← s'
      H[s] ← min_{b ∈ ACTIONS(s)} LRTA*-COST(s, b, result[s, b], H)
  a ← an action b in ACTIONS(s') that minimizes LRTA*-COST(s', b, result[s', b], H)
  s ← s'
  return a

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

عامل LRTA* مشابه ONLINE-DFS-AGENT یک نقشه از محیط را در جدول *result* می‌سازد.

تخمین هزینه‌ی هر حالت را زمانی به‌هنگام می‌کند که آن را ترک می‌کند و
سپس بهترین حرکت ظاهری را به‌عنوان حرکت بعدی انتخاب می‌کند.

کنش‌هایی که آزمایش نشده‌اند، همیشه فرض می‌شوند که به هدف با کمترین هزینه $h(s)$ منجر می‌شوند.

عامل‌های جستجوی برخط

جستجوی محلی برخط: اضافه کردن حافظه به تپه‌نوردی (A* بی‌درنگ یادگیرنده): خصوصیات

LEARNING REAL-TIME A* (LRTA*)

خوش‌بینی تحت عدم اطمینان
Optimism under Unceratinty

عامل را تشویق می‌کند که کنش‌های جدید ممکن را اکتشاف کند.

عامل LRTA* تضمین می‌کند که هدف را بیابد
در هر محیط متناهی اکتشاف‌پذیر به‌صورت امن

عامل LRTA* برخلاف A* برای فضاهای حالت نامتناهی کامل نیست.
(مواردی وجود دارد که در آن می‌تواند به سرگردانی نامتناهی بینجامد)

عامل LRTA* می‌تواند یک محیط با n حالت را در $O(n^2)$ گام اکتشاف کند.
(اما معمولاً بسیار بهتر عمل می‌کند.)

در LRTA* اگر تخمین‌های اولیه پذیرفتنی باشند، در این صورت با تلاش‌های تکراری برای حل مسئله، مقادیر یادگرفته شده سرانجام به فاصله‌های واقعی آنها تا هدف در امتداد هر مسیر بهینه همگرا می‌شود.

عامل‌های جستجوی برخط

یادگیری در جستجوی برخط

LEARNING IN ONLINE SEARCH

ناآگاهی اولیه‌ی عامل‌های جستجوی برخط ⇐ فرصت‌هایی برای **یادگیری**

یادگیری نقشه‌ی محیط (با فرض محیط قطعی)

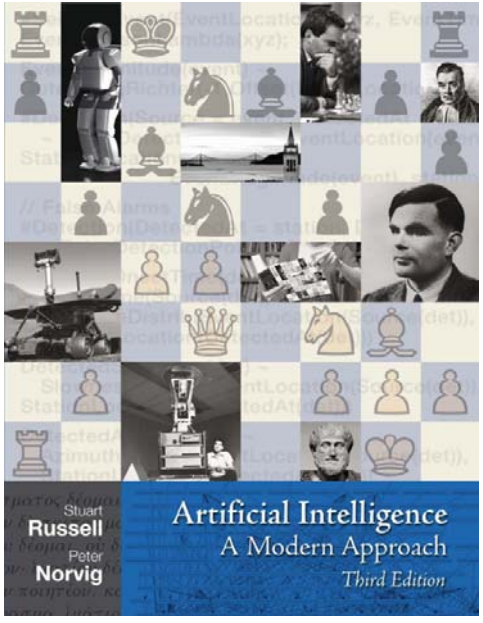
یادگیری تخمین مقدار هر حالت (تا فهمیدن مقدار دقیق آن)

یادگیری معنای کنش‌ها (مثلاً: UP به y یک واحد اضافه می‌کند.)

ملزومات:

- یک بازنمایی صوری و صریح برای دستیابی به قواعد عمومی (بازنمایی اعلانی)
- الگوریتم‌های سازنده‌ی قواعد عمومی مناسب از روی مشاهدات خاص انجام شده توسط عامل (یادگیری قواعد)

منبع اصلی



Stuart Russell and Peter Norvig,
Artificial Intelligence: A Modern Approach,
3rd Edition, Prentice Hall, 2010.

Chapter 4 (4.5)

4 BEYOND CLASSICAL SEARCH

In which we relax the simplifying assumptions of the previous chapter, thereby getting closer to the real world.

Chapter 3 addressed a single category of problems: observable, deterministic, known environments where the solution is a sequence of actions. In this chapter, we look at what happens when these assumptions are relaxed. We begin with a fairly simple case: Sections 4.1 and 4.2 cover algorithms that perform purely **local search** in the state space, evaluating and modifying one or more current states rather than systematically exploring paths from an initial state. These algorithms are suitable for problems in which all that matters is the solution state, not the path cost to reach it. The family of local search algorithms includes methods inspired by statistical physics (**simulated annealing**) and evolutionary biology (**genetic algorithms**).

Then, in Sections 4.3–4.4, we examine what happens when we relax the assumptions of determinism and observability. The key idea is that if an agent cannot predict exactly what percept it will receive, then it will need to consider what to do under each **contingency** that its percepts may reveal. With partial observability, the agent will also need to keep track of the states it might be in.

Finally, Section 4.5 investigates **online search**, in which the agent is faced with a state space that is initially unknown and must be explored.

4.1 Local Search Algorithms and Optimization Problems

The search algorithms that we have seen so far are designed to explore search spaces systematically. This systematicity is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each point along the path. When a goal is found, the *path* to that goal also constitutes a *solution* to the problem. In many problems, however, the path to the goal is irrelevant. For example, in the 8-queens problem (see page 71), what matters is the final configuration of queens, not the order in which they are added. The same general property holds for many important applications such as integrated-circuit design, factory-floor layout, job-shop scheduling, automatic programming, telecommunications network optimization, vehicle routing, and portfolio management.

120