هوش مصنوعی

فصل ۴

فراسوی جستجوی کلاسیک

Beyond Classical Search

کاظم فولادی
دانشکده مهندسی برق و کامپیوتر
دانشگاه تهران

http://courses.fouladi.ir/ai

هوش مصنوعی پیشرفته

درس ۱

جستجو با کنش‌های غیرقطعی

Searching with Nondeterministic Actions

کاظم فولادی

دانشکده مهندسی برق و کامپیوتر

دانشگاه تهران

http://courses.fouladi.ir/aai

هوش مصنوعی پیشرفته

فراسوی جستجوی کلاسیک

۳

جستجو با کنش‌های غیرقطعی

انواع مسئله

تک‌حالتی *Single-State*	چندحالتی *Multiple-State* *Conformant*	اقتضائی *Contingency*	اکتشافی *Exploration*
قطعی مشاهده‌پذیر کامل	مشاهده‌ناپذیر	غیرقطعی و / یا مشاهده‌پذیر جزئی	فضای حالت ناشناخته
عامل دقیقاً می‌داند در کدام حالت قرار دارد. و اثر کنش‌هایش را می‌داند.	عامل هیچ چیزی در مورد اینکه در کدام حالت است نمی‌داند. اما اثر کنش‌ها را می‌داند.	ادراک‌های عامل، اطلاعات جدیدی در مورد حالت فعلی فراهم می‌کنند.	مثل مسافر غریب بدون نقشه لزوم تجربه کردن عامل * **یک‌درمیان‌سازی** **جستجو و کنش**
راه حل = یک دنباله	راه حل = یک دنباله (در صورت وجود)	راه حل = یک درخت (یا سیاست)	برخط (online)

طرح اقتضائی (استراتژی)

راه‌حل برای محیط‌های غیرقطعی و مشاهده‌ی پذیر جزئی

CONTINGENCY PLAN (STRATEGY)

نقش **ادراک‌ها**	
در محیط مشاهده‌پذیر جزئی	در محیط غیرقطعی
کمک به **باریک کردن مجموعه‌ی حالت‌های ممکن** که عامل می‌تواند در آن باشد	تعیین اینکه **کدام‌یک از برآمدهای ممکن یک کنش** واقعاً اتفاق افتاده است

ادراک‌ها نمی‌توانند از قبل تعیین شوند،
اما کنش‌های عامل‌ها وابسته به ادراک‌های آینده خواهد بود.

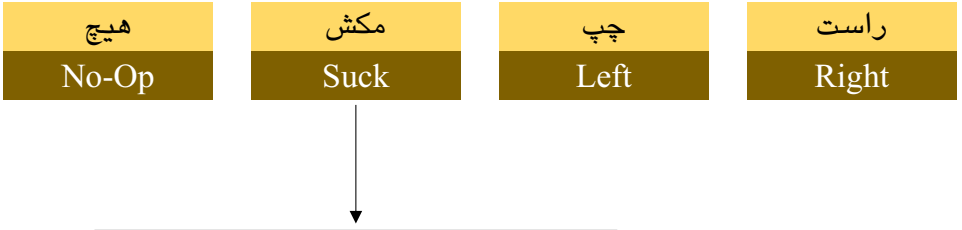
راه‌حل مسئله یک دنباله از کنش‌ها نیست، بلکه یک طرح اقتضائی (استراتژی) [درخت] است:

تعیین می‌کند که در هر گام با توجه به ادراکی که در آن انجام می‌شود، چه کاری باید انجام شود.

مثال: دنیای جاروبرقی خطادار

کنش‌ها

THE ERRATIC VACUUM WORLD



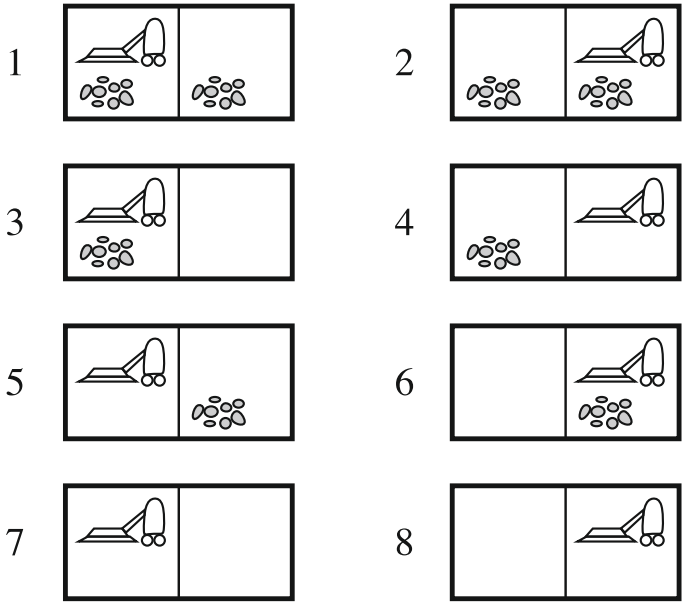
روی خانه‌ی آلوده:
- آن خانه تمیز می‌شود.
- گاهی خانه‌ی مجاور نیز تمیز می‌شود!

روی خانه‌ی تمیز:
- گاهی آن خانه را آلوده می‌کند!

مثال: دنیای جاروبرقی خطادار

۸ حالت ممکن

THE ERRATIC VACUUM WORLD



حالت‌های 7 و 8 **هدف** هستند.

فرمول‌بندی مسئله

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله (مسائل اقتضائی)

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله

حالت آغازین Initial state	حالتی که عامل از آن شروع می‌کند.
کنش‌های ممکن Available actions	کنش‌های ممکن عامل در هر حالت s : $\text{ACTIONS}(s)$
مدل گذر Transition model	در هر حالت s ، هر کنش a چه می‌کند؟ : $\text{RESULT}(s,a)$ (مجموعه‌ی برآمدهای ممکن به‌جای یک برآمد واحد)
آزمون هدف Goal test	تعیین کننده‌ی اینکه حالت جاری هدف است یا خیر: صریح (explicit): بیان مجموعه حالات هدف ضمنی (implicit): بیان ویژگی هدف
تابع هزینه‌ی مسیر Path cost function	تابعی که به هر مسیر یک هزینه‌ی عددی نسبت می‌دهد. هزینه‌ی گام (step cost): هزینه‌ی گذر از یک حالت به حالت دیگر با یک کنش $c(s,\ a,\ s')$

فضای حالت مسئله *problem state-space*

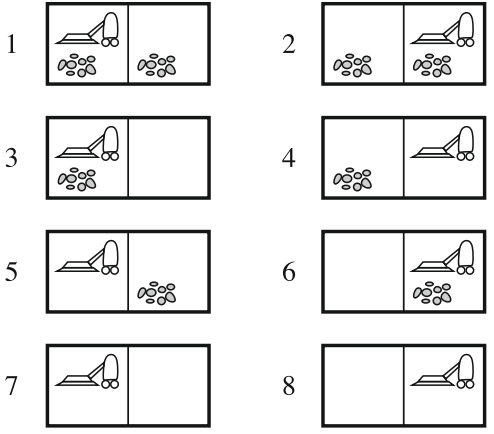
مجموعه‌ی حالت‌های دسترس‌پذیر از حالت آغازین با حداقل یک دنباله از کنش‌ها

راه‌حل: یک طرح اقتضائی (استراتژی) [درخت]

مثال: دنیای جاروبرقی خطادار

طرح اقتضائی

THE ERRATIC VACUUM WORLD



تعمیم تابع نتیجه (مدل گذر):

مثال: کنش مکش (Suck) در حالت 1 مجموعه‌ی $\{5,7\}$ را برمی‌گرداند.

تعمیم مفهوم راه‌حل:

مثال: اگر از حالت 1 شروع کنیم، هیچ دنباله‌ی واحدی از کنش‌ها وجود ندارد که مسئله را حل کند.

به‌جای آن به یک طرح اقتضائی نظیر زیر نیاز داریم:

$$[Suck, \textbf{if } State = 5 \textbf{ then } [Right, Suck] \textbf{ else } [\,]]$$

راه‌حل برای مسائل غیرقطعی، می‌تواند حاوی جملات شرطی if-then-else تودرتو باشد:
⇐ امکان انتخاب کنش‌ها بر اساس اقتضائات ظاهر شده در حین اجرا

درخت جستجوی AND-OR

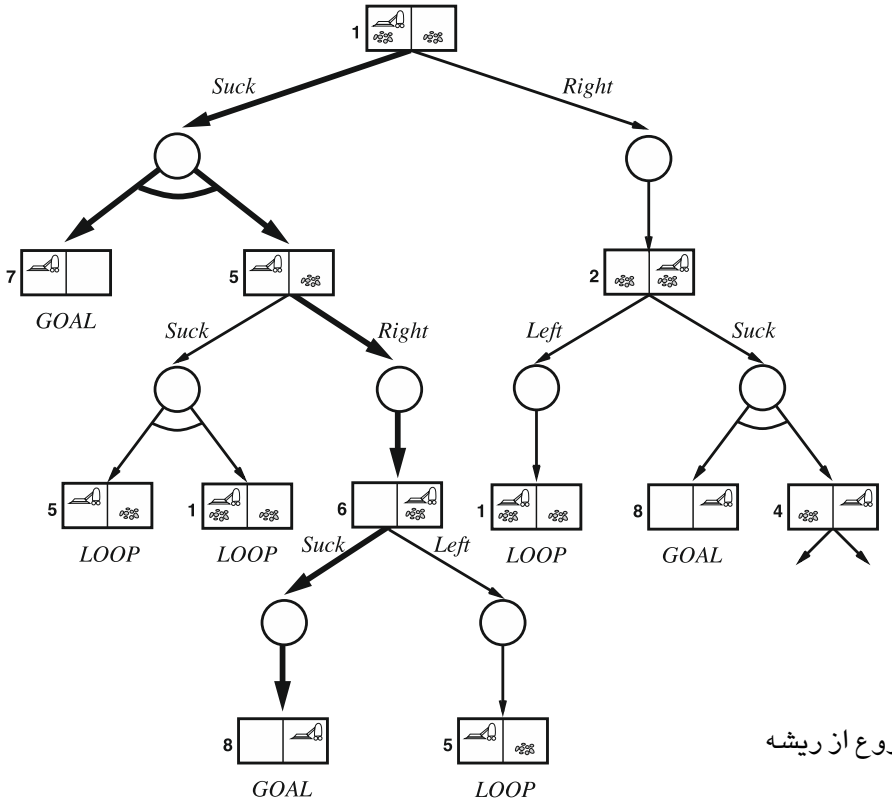
AND-OR Search Tree

درخت جستجوی AND-OR AND-OR Search Tree	
گره‌های AND AND Nodes	**گره‌های OR** OR Nodes
شاخه‌های خارج شده از آن انتخاب‌های محیط (از برآمد هر کنش عامل) را نشان می‌دهد.	شاخه‌های خارج شده از آن انتخاب‌های خود عامل را نشان می‌دهد.

درخت جستجوی AND-OR

مثال: دنیای جاروبرقی خطادار

AND-OR SEARCH TREE



راه‌حل: زیردرخت پررنگ‌تر با شروع از ریشه

درخت جستجوی AND-OR

راه‌حل

AND-OR SEARCH TREE

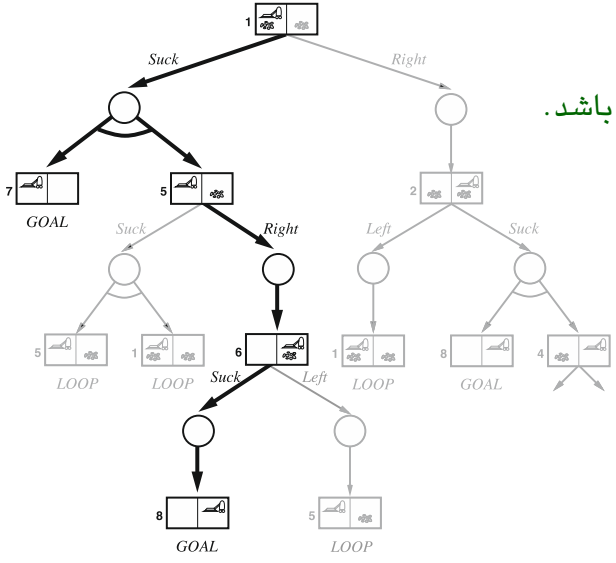
راه‌حل (برای یک مسئله‌ی جستجوی AND-OR)

یک زیردرخت که

۱) هر برگ حاوی یک حالت هدف باشد.

۲) در هر یک از گره‌های OR یک کنش را مشخص کند.

۳) در هر یک از گره‌های AND همه‌ی شاخه‌های برآمد، موجود باشد.



طرح (plan) حاصل از نمادگذاری **if-then-else** برای اداره کردن گره‌های AND استفاده می‌کند.

جستجوی AND-OR

الگوریتم

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
  OR-SEARCH(problem.INITIAL-STATE, problem, [])

function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
  if problem.GOAL-TEST(state) then return the empty plan
  if state is on path then return failure
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
      if plan ≠ failure then return [action | plan]
  return failure

function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
  for each s_i in states do
      plan_i ← OR-SEARCH(s_i, problem, path)
      if plan_i = failure then return failure
  return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n-1} then plan_{n-1} else plan_n]
```

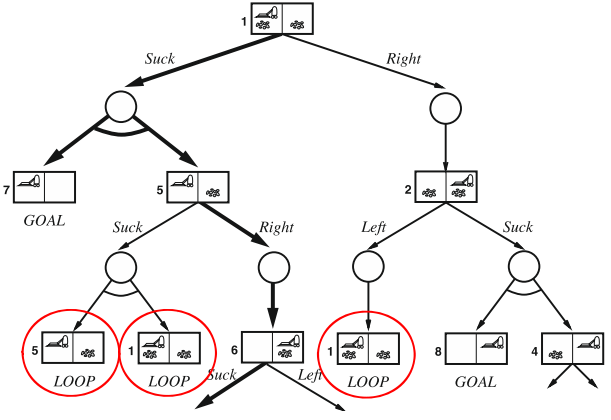
مبنای این الگوریتم روش DFS است.

جستجوی AND-OR

برخورد با دورها

یک موضوع کلیدی،
روش برخورد با دورها (چرخه‌ها)یی است که در اغلب مسائل غیرقطعی ظاهر می‌شود.

الگوریتم جستجوی ارائه شده ،شکست (failure) را برمی‌گرداند
اگر حالت جاری با یکی از حالت‌ها بر روی مسیر آغاز شده از ریشه تا آن گره یکسان باشد.
این بدان معنی نیست که راه‌حلی وجود ندارد،
فقط اینکه: اگر راه‌حل «بدون دور» وجود دارد، باید از یک حالت قبلی‌تر دسترس‌پذیر باشد.



جستجوی AND-OR

پیمایش‌های عرض–اول / بهترین–اول

گراف‌های AND-OR می‌توانند با روش‌های **عرض–اول** و **بهترین–اول** نیز پیمایش شوند.

برای این منظور توابع هیوریستیک باید به‌گونه‌ای تغییر کنند که
هزینه‌ی یک راه‌حل اقتضائی (درخت) را به‌جای یک **دنباله** برآورد کنند.
مفهوم قابل‌قبول (پذیرفتنی) بودن تابع هیوریستیک نیز روی آن اعمال می‌شود.

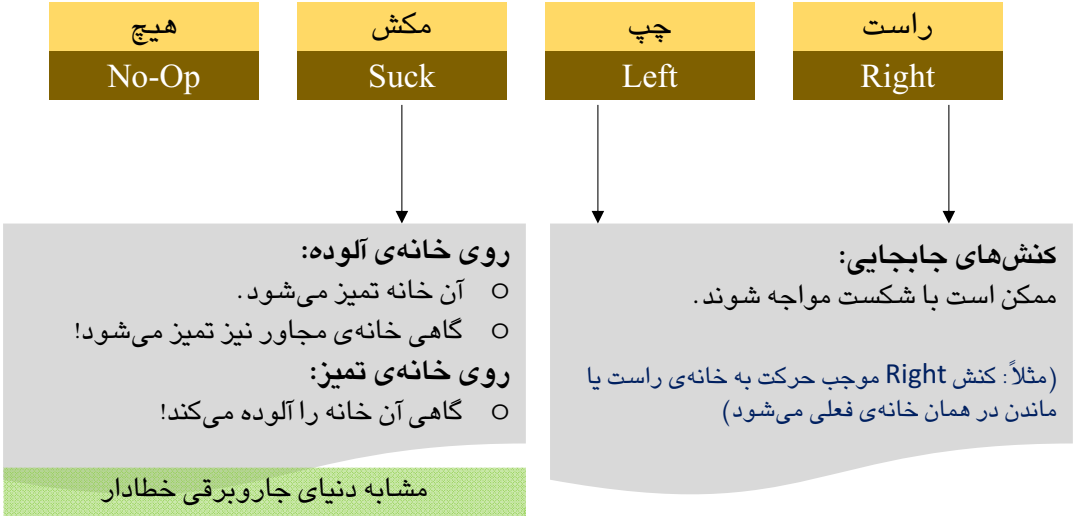
⇓

یک نسخه‌ی قابل مقایسه با جستجوی A* به‌دست می‌آید.

مثال: دنیای جاروبرقی خطادار لغزنده

کنش‌ها

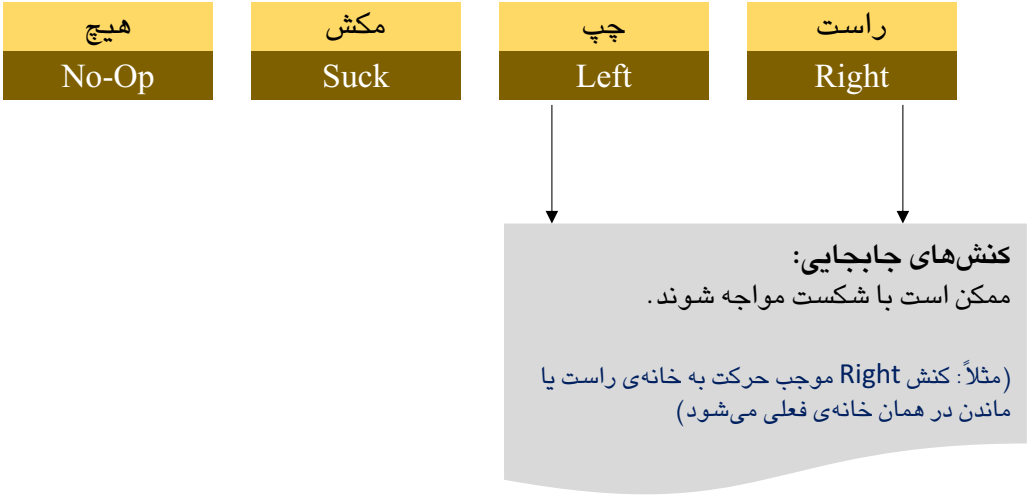
THE ERRATIC SLIPPERY VACUUM WORLD



مثال: دنیای جاروبرقی لغزنده

کنش‌ها

THE SLIPPERY VACUUM WORLD



مثال: دنیای جاروبرقی لغزنده

راه‌حل

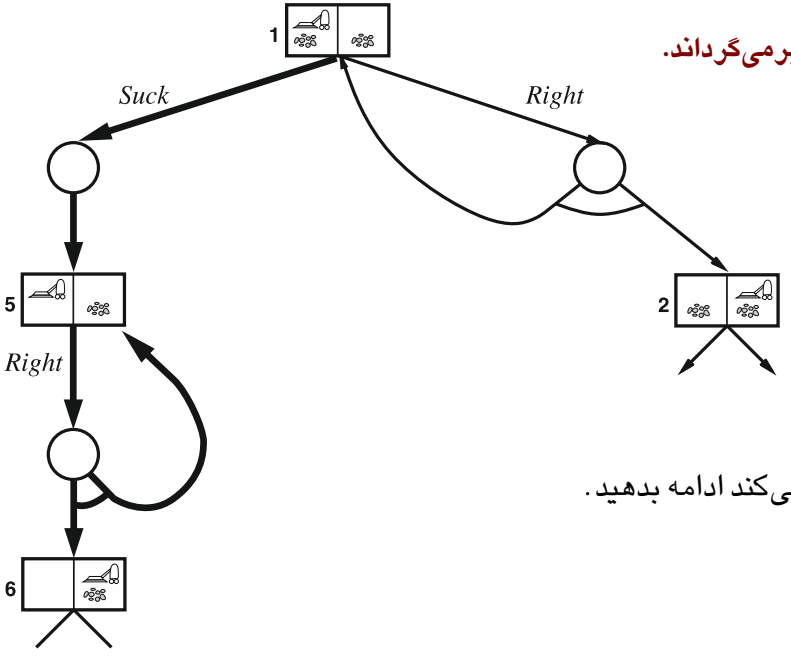
THE SLIPPERY VACUUM WORLD

راه‌حل «بدون دور»

هیچ راه‌حل بدون دوری با شروع از حالت 1 وجود ندارد

⇓

الگوریتم جستجوی گرافی AND-OR، شکست را برمی‌گرداند.



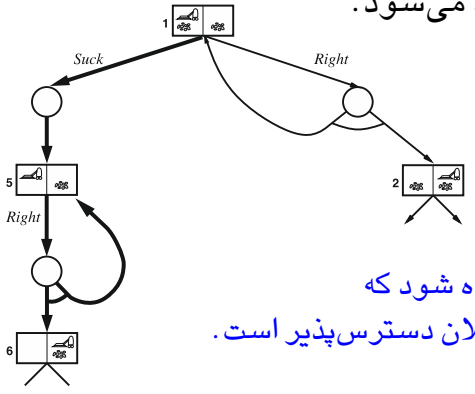
راه‌حل «دور دار»

بعد از *Suck* آزمون *Right* را تا زمانی که کار می‌کند ادامه بدهید.

جستجوی AND-OR

راه‌حل دور دار

راه‌حل دوردار را می‌توان با اضافه کردن یک برچسب که به بخشی از پلان اشاره می‌کند، بیان کرد به طوری‌که از آن برچسب متعاقباً استفاده می‌شود.

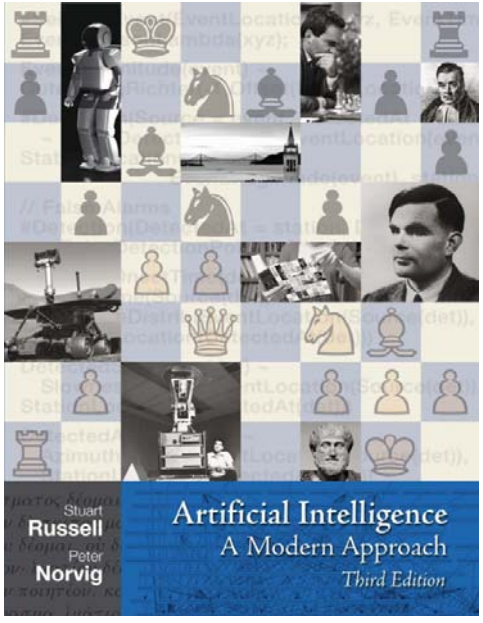


$$[Suck, \text{L1: } Right \textbf{ if } State = 5 \textbf{ then } \text{L1} \textbf{ else } Suck]$$

یک پلان دوردار می‌تواند به‌عنوان راه‌حلی دیده شود که
هر برگ آن یک حالت هدف است و این برگ از هر نقطه در پلان دسترس‌پذیر است.

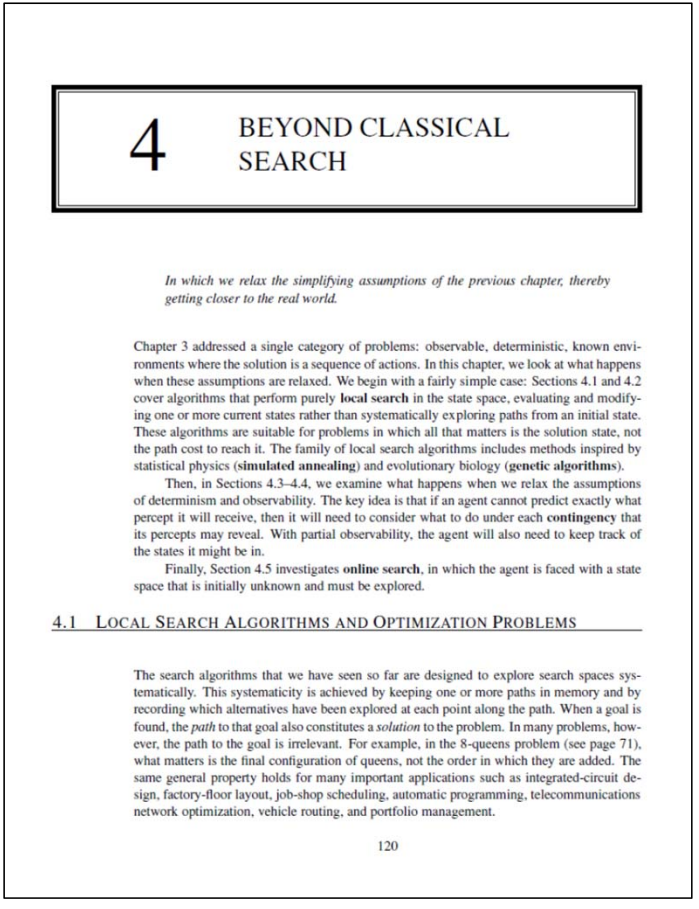
با داشتن تعریف یک راه‌حل دوردار، عاملی که این راه‌حل را اجرا می‌کند سرانجام به هدف می‌رسد، با احتمال یک،
تنها اگر عدم قطعیت ناشی از یک فرآیند اتفاقی باشد
و توسط یک خصوصیت پنهان محیط که از برخی گذرهای حالت جلوگیری می‌کند، ایجاد نشده باشد.

منبع اصلی



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

Chapter 4 (4.3)



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

هوش مصنوعی پیشرفته

درس ۲

جستجو با مشاهدات جزئی

Searching with Partial Observations

کاظم فولادی

دانشکده مهندسی برق و کامپیوتر

دانشگاه تهران

http://courses.fouladi.ir/aai

هوش مصنوعی پیشرفته

فراسوی جستجوی کلاسیک

۴

جستجو با مشاهدات جزئی

جستجو با مشاهدات جزئی

SEARCHING WITH PARTIAL OBSERVATIONS

در مسائلی با مشاهده‌پذیری جزئی، ادراکات عامل برای تعیین حالت دقیق ناکافی است.
یک کنش می‌تواند منجر به یکی از چندین حالت ممکن شود، حتی اگر محیط **قطعی** باشد.

مفهوم کلیدی = **حالت باور (belief state)**
بازنمایی باور فعلی عامل در مورد حالت‌های فیزیکی که عامل ممکن است در آنها باشد
[با داشتن دنباله‌ی کنش‌ها و ادراکات]

عامل بدون حسگر	عامل با حسگر	
بدون مشاهده	با مشاهده‌ی جزئی	با مشاهده‌ی کامل

جستجو بدون مشاهده

SEARCHING WITH NO OBSERVATION

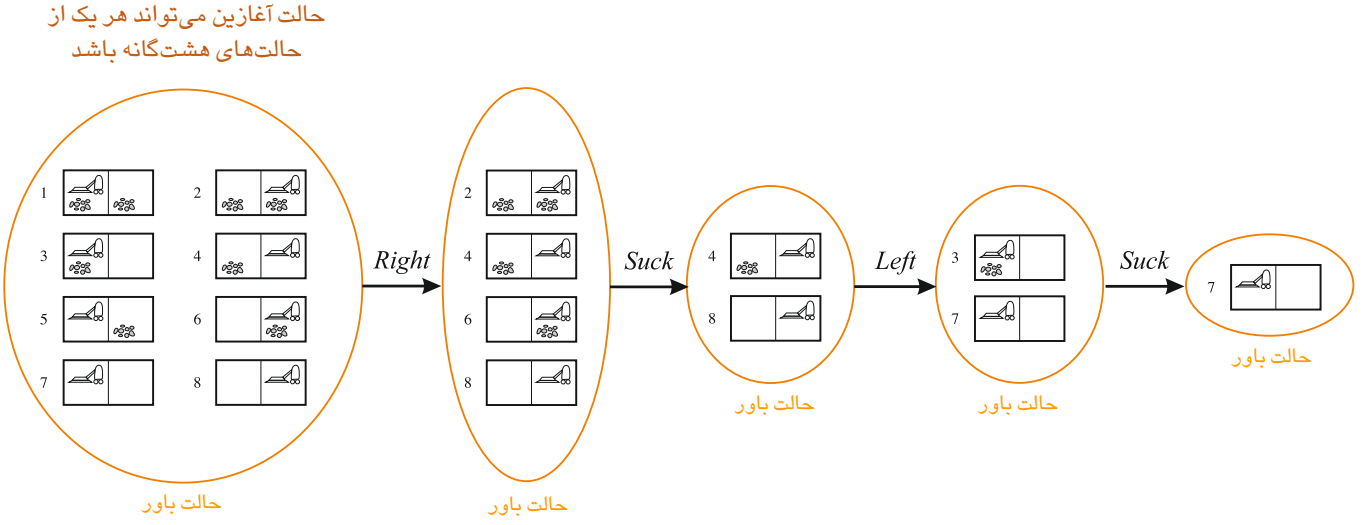
عامل‌های بدون حسگر نیز می‌توانند اغلب به طور موفق کنش کنند
و دارای **برتری** هستند
(به دلیل عدم تکیه بر اطلاعات حسگری بالقوه غیرقابل اتکا و پرهزینه)

جستجو بدون مشاهده

مثال: دنیای جاروبرقی بدون حسگر

SENSORLESS VACUUM WORLD

عامل **جغرافیای خودش** را می‌شناسد، اما **موقعیت** خود یا توزیع **آلودگی** را نمی‌داند.



با استفاده از دنباله کنش‌های [*Right*, *Suck*, *Left*, *Suck*] تضمین می‌شود که به حالت هدف 7 برسیم
صرف‌نظر از اینکه حالت آغازین چه باشد!

اصطلاحاً می‌گوییم عامل می‌تواند دنیا را به حالت 7 **ناگزیر** کند (**coerce**).

جستجو بدون مشاهده

جستجو در فضای باور

SEARCHING IN BELIEF STATE

برای حل مسائل بدون حسگر در فضای حالت‌های باور جستجو می‌کنیم
(به‌جای جستجو در فضای حالت‌های فیزیکی)

فضای باور مسئله **مشاهده‌پذیر کامل** است
(زیرا عامل همیشه حالت باور خود را به طور کامل می‌داند)

راه‌حل در صورت وجود، همیشه دنباله‌ای از کنش‌هاست
(زیرا ادراکات همیشه تهی هستند ⇐ کاملاً قابل پیش‌بینی هستند ⇐ هیچ اقتضائاتی برای طرح / plan وجود ندارد)

* این چهارچوب برای محیطهای غیرقطعی هم درست است.

فرمول‌بندی مسئله

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله‌ی جستجو در فضای حالت‌های باور

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله

حالت‌های باور
Belief states

کل فضای حالت باور: شامل همه‌ی مجموعه‌ی ممکن از حالت‌های فیزیکی
اگر مسئله‌ی فیزیکی P دارای N حالت باشد، مسئله‌ی بدون حسگر حداکثر 2^N حالت دارد.

حالت آغازین
Initial state

حالتی که عامل از آن شروع می‌کند.
(معمولاً مجموعه‌ی همه‌ی حالت‌ها در P: گاهی عامل دانایی بیشتری دارد)

کنش‌های ممکن
Available actions

کنش‌های ممکن عامل در هر حالت s: $\text{ACTIONS}(s)$

مدل گذر
Transition model

در هر حالت s، هر کنش a چه می‌کند؟: $\text{RESULT}(s,a)$
(مجموعه‌ی برآمدهای ممکن به‌جای یک برآمد واحد)

آزمون هدف
Goal test

تعیین کننده‌ی اینکه حالت جاری هدف است یا خیر:
صریح (explicit): بیان مجموعه حالات هدف ضمنی (implicit): بیان ویژگی هدف

تابع هزینه‌ی مسیر
Path cost function

تابعی که به هر مسیر یک هزینه‌ی عددی نسبت می‌دهد.
هزینه‌ی گام (step cost): هزینه‌ی گذر از یک حالت به حالت دیگر با یک کنش $c(s,\ a,\ s')$

فضای حالت مسئله
problem state-space

مجموعه‌ی حالت‌های دسترس‌پذیر از حالت آغازین با حداقل یک دنباله از کنش‌ها

راه‌حل: یک دنباله از کنش‌ها

فرمول‌بندی مسئله

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله‌ی جستجو در فضای حالت‌های باور: کنش‌ها

فرض کنید عامل در فضای باور $b = \{s_1, s_2\}$ قرار دارد، اما

$$\text{ACTIONS}_P(s_1) \neq \text{ACTIONS}_P(s_2)$$

در این صورت عامل مطمئن نیست که کدام کنش‌ها مجاز است.

اگر کنش‌های غیرمجاز تأثیری بر محیط نداشته باشد از **اجتماع** استفاده می‌کنیم

$$\text{ACTIONS}(b) = \bigcup_{s \in b} \text{ACTIONS}_P(s)$$

اگر یک کنش در یک حالت صدمه‌زننده باشد، از **اشتراک** استفاده می‌کنیم

$$\text{ACTIONS}(b) = \bigcap_{s \in b} \text{ACTIONS}_P(s)$$

فرمول‌بندی مسئله

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله‌ی جستجو در فضای حالت‌های باور: مدل گذر

نتیجه‌ی یک کنش، مجموعه‌ی همه‌ی حالت‌های فیزیکی حاصل از انجام آن کنش بر روی همه‌ی حالت‌های فیزیکی در حالت باور فعلی است.

برای کنش‌های **قطعی**، این مجموعه عبارت است از:

$$b' = \text{RESULT}(b,a) = \{s' : s' = \text{RESULT}_P(s,a) \text{ and } s \in b\}$$

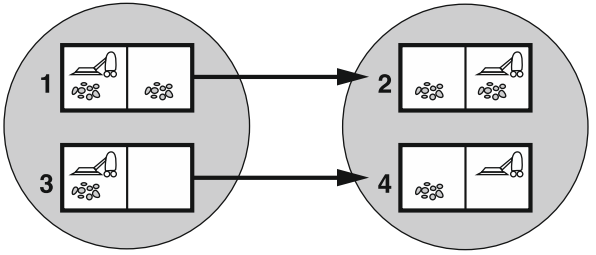
برای کنش‌های **غیرقطعی**، این مجموعه عبارت است از:

$$\begin{aligned} b' = \text{RESULT}(b,a) &= \{s' : s' \in \text{RESULTS}_P(s,a) \text{ and } s \in b\} \\ &= \bigcup_{s \in b} \text{RESULTS}_P(s,a) \ , \end{aligned}$$

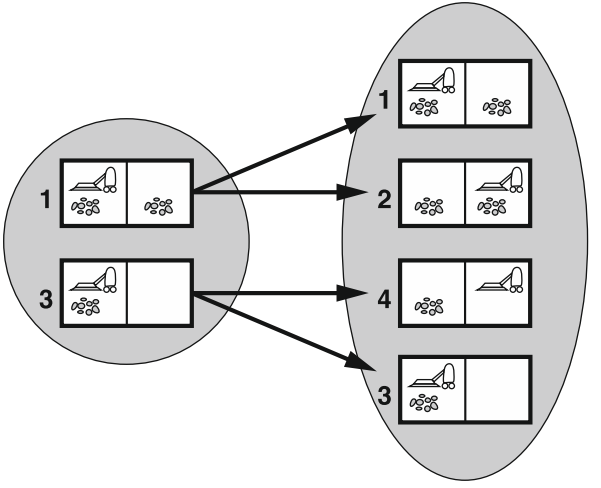
فرآیند تولید یک حالت باور جدید پس از یک کنش، گام پیش‌بینی $\text{PREDICT}_P(b,a)$ نام دارد.

فرمول‌بندی مسئله

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله‌ی جستجو در فضای حالت‌های باور: مدل گذر: مثال



پیش‌بینی حالت بعدی
برای دنیای جاروبرقی بدون حسگر
با کنش قطعی *Right*



پیش‌بینی حالت بعدی
برای دنیای جاروبرقی لغزنده
با کنش غیرقطعی *Right*

فرمول‌بندی مسئله

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله‌ی جستجو در فضای حالت‌های باور: آزمون هدف

حالت‌های هدف در فضای باور تنها حالت‌هایی هستند که در آن
همه‌ی حالت‌های فیزیکی GOAL-TEST_P را ارضا می‌کنند.

یک عامل ممکن است تصادفاً زودتر به هدف برسد، اما این را نخواهد دانست.

فرمول‌بندی مسئله

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله‌ی جستجو در فضای حالت‌های باور: هزینه‌ی مسیر

فرض می‌کنیم هر کنش در همه‌ی حالت‌ها هزینه‌ی یکسانی دارد.

اگر یک کنش یکسان برای حالت‌های فیزیکی متفاوت داخل یک حالت باور،
هزینه‌های مختلفی داشته باشد،
تعریف هزینه کمی دشوار می‌شود.

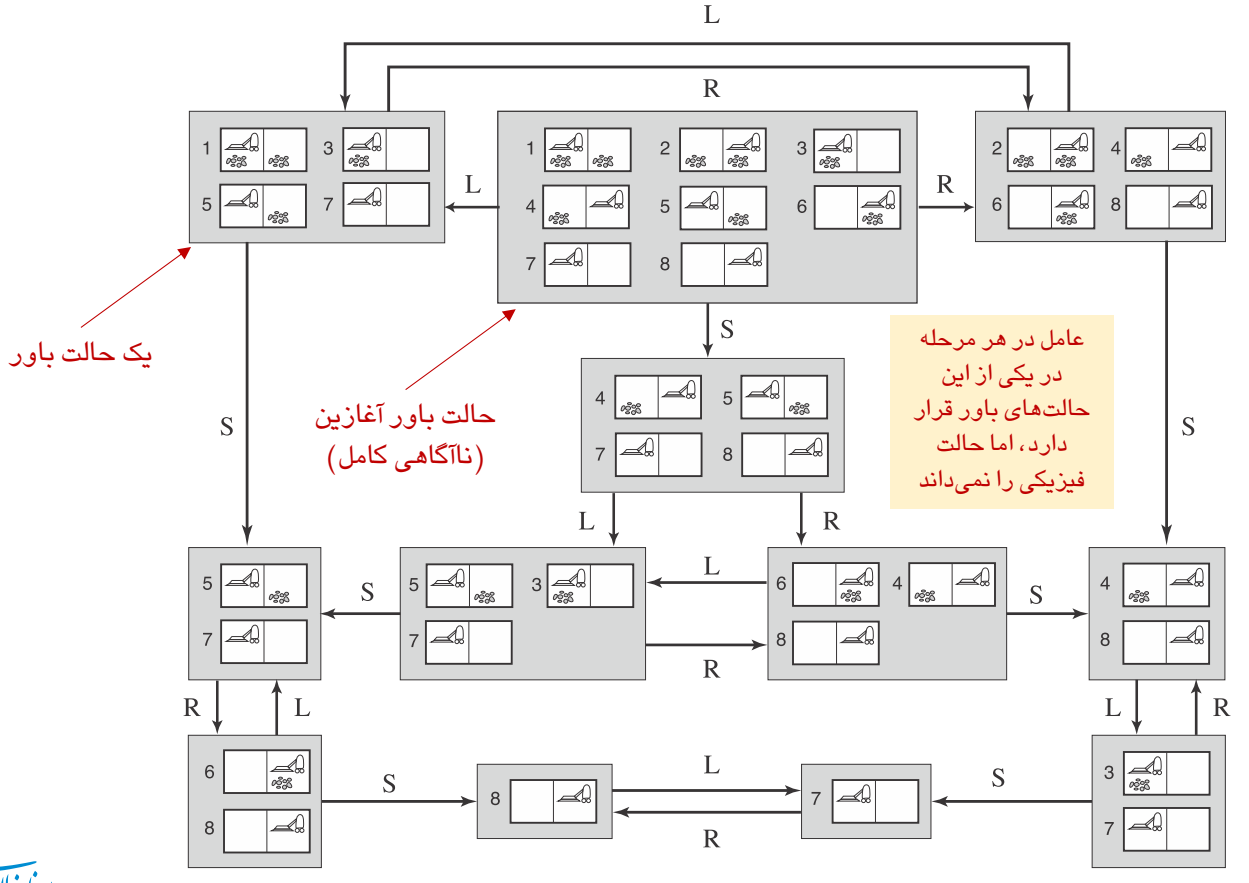
جستجو در فضای باور

تعریف‌های انجام شده، امکان ساخت خودکار مسئله‌ی فضای باور را ایجاد می‌کند.

در این مرحله می‌توان از هر روش جستجوی موجود برای حل این مسئله استفاده کرد.

جستجو در فضای باور

مثال: بخش دسترس‌پذیر فضای حالت باور برای دنیای جاروبرقی بدون حسگر قطعی



جستجو در فضای باور

کارآیی

حل مسئله‌ی بدون حسگر، به‌ندرت با الگوریتم‌های تاکنون تشریح‌شده امکان‌پذیر است.
(زیرا اندازه‌ی فضای باور بسیار بزرگ است.)

برای مثال:

حالت باور آغازین برای یک دنیای جاروبرقی 10×10:
حاوی $100 \times 2^{100} \cong 10^{32}$ حالت فیزیکی است.

راه حل:

الف) بازنمایی حالت باور در یک توصیف متراکم‌تر (مثل بازنمایی صوری/منطق)

ب) اجتناب از الگوریتم‌های جستجوی استاندارد و عدم برخورد با حالت باور به‌صورت جعبه سیاه

⇐ **الگوریتم‌های جستجوی حالت باور افزایشی** (incremental belief-state search algorithms)

جستجو با مشاهدات

SEARCHING WITH OBSERVATIONS

برای یک مسئله‌ی مشاهده‌پذیر جزئی عمومی
باید مشخص کنیم که محیط چگونه ادراکات را برای عامل تولید می‌کند:

PERCEPT(s) تابع ادراک

جستجو با مشاهدات

تابع ادراک $\text{PERCEPT}(s)$

SEARCHING WITH OBSERVATIONS

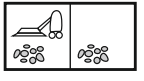
تابع ادراک $\text{PERCEPT}(s)$	
حس کردن قطعی *Deterministic Sensing*	**حس کردن غیرقطعی** *Non-deterministic Sensing*
ادراک عامل برای حالت s	مجموعه‌ی ادراک‌های ممکن عامل برای حالت s

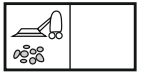
برای مسائل مشاهده‌پذیر کامل: $\text{PERCEPT}(s) = s$

برای مسائل بدون حسگر: $\text{PERCEPT}(s) = null$

وقتی مشاهدات جزئی هستند، یک ادراک می‌تواند توسط حالت‌های متعدد تولید شود.

مثال: ادراک $[A, dirty]$ در دنیای جاروبرقی می‌تواند توسط دو حالت 1 و 3 تولید شود:

1 

3 

جستجو با مشاهدات

گذر در فضای باور

گذر از یک حالت باور به یک حالت باور دیگر به **سه مرحله** تقسیم می‌شود:

۱ پیش‌بینی *Prediction*

با داشتن یک کنش a در حالت باور b، حالت باور پیش‌بینی شده:

$$\hat{b} = \text{PREDICT}(b, a)$$

۲ پیش‌بینی مشاهده *Observation Prediction*

تعیین مجموعه‌ی ادراک‌های ممکن در حالت باور پیش‌بینی شده

$$\text{POSSIBLE-PERCEPTS}(\hat{b}) = \{o : o = \text{PERCEPT}(s) \text{ and } s \in \hat{b}\}$$

۳ به‌هنگام‌سازی *Update*

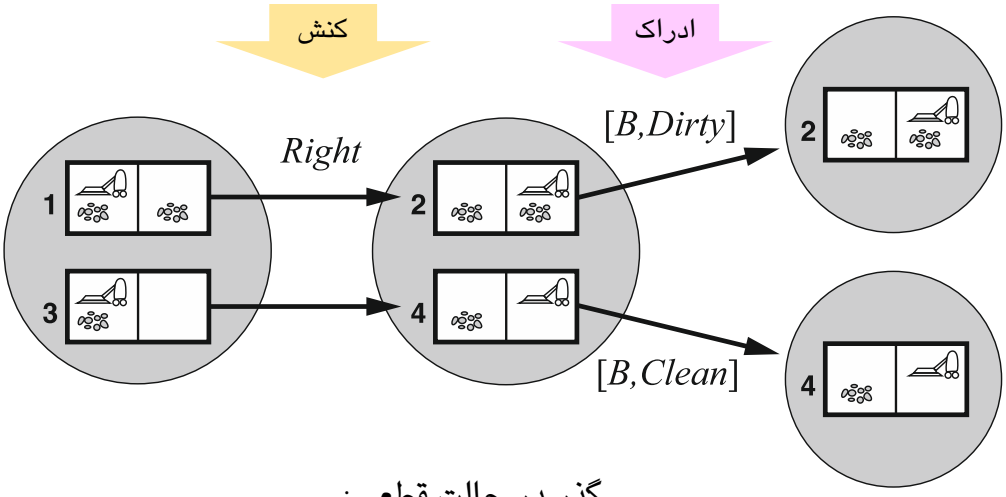
تعیین حالت باور جدید حاصل از یک ادراک

$$b_o = \text{UPDATE}(\hat{b}, o) = \{s : o = \text{PERCEPT}(s) \text{ and } s \in \hat{b}\}$$

حالت باور جدید، مجموعه‌ی حالت‌ها در حالت باور پیش‌بینی شده است که می‌تواند آن ادراک را تولید کند.

جستجو با مشاهدات

گذر در فضای باور: مثال (گذر در حالت قطعی)



گذر در حالت قطعی:

کنش *Right* در حالت باور اولیه اعمال می‌شود

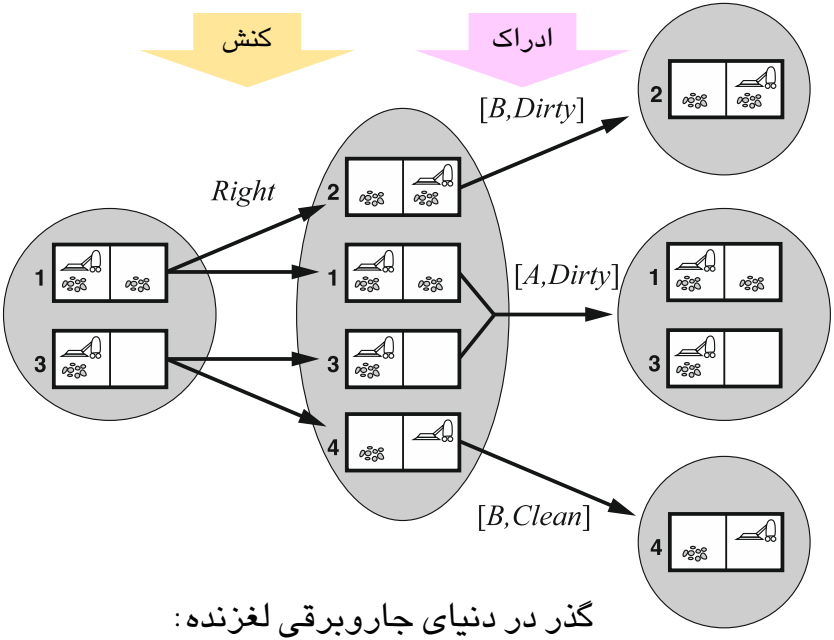
که منجر به یک حالت باور جدید با دو حالت فیزیکی ممکن وابسته به ادراکات زیر می‌شود.

$[B,\ dirty], \quad [B,\ Clean]$

* در حس کردن قطعی، حالت‌های باور برای ادراکات مختلف به‌طور جداگانه،
یک افراز از حالت باور پیش‌بینی شده‌ی اصلی را تشکیل می‌دهند.

جستجو با مشاهدات

گذر در فضای باور: مثال (گذر در حالت غیرقطعی)



گذر در دنیای جاروبرقی لغزنده:

اعمال کنش *Right* در حالت باور اولیه، چهار حالت فیزیکی می‌دهد
که منجر به سه حالت باور جدید وابسته به ادراکات زیر می‌شود:

$[B,\ dirty], \quad [A,\ dirty], \quad [B,\ Clean]$

* حالت باور به‌هنگام‌شده نمی‌تواند بزرگ‌تر از حالت باور پیش‌بینی‌شده باشد،
زیرا مشاهدات کمک می‌کنند عدم اطمینان در مقایسه با حالت بدون حسگر کاهش یابد.

جستجو با مشاهدات

حل مسائل مشاهده‌پذیر جزئی

حالت‌های باور ممکن حاصل از یک کنش و ادراک متوالی بعدی:
(با قرار دادن سه‌مرحله‌ی پیش‌بینی، پیش‌بینی مشاهده و به‌هنگام‌سازی در کنار هم)

$$\text{RESULTS}(b,a) = \{b_o : b_o = \text{UPDATE}(\text{PREDICT}(b,a),o) \text{ and } o \in \text{POSSIBLE-PERCEPTS}(\text{PREDICT}(b,a))\}$$

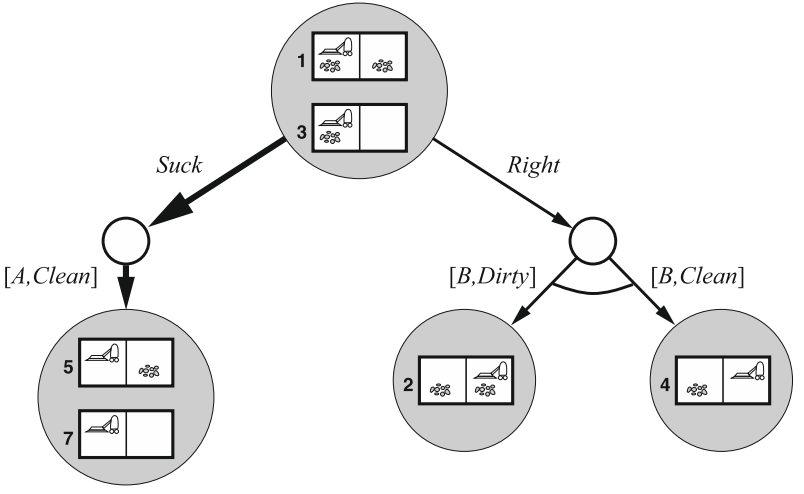
حل مسائل مشاهده‌پذیر جزئی

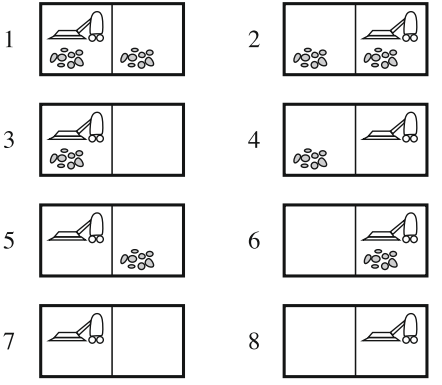
با داشتن این فرمول‌بندی، جستجوی AND-OR می‌تواند برای یافتن یک راه‌حل مستقیماً اعمال شود.
(برخورد با حالت‌های باور به صورت جعبه سیاه)

حل مسائل مشاهده‌پذیر جزئی

استفاده از جستجوی AND-OR: مثال

سطح اول یک درخت جستجوی AND-OR برای ادراک آغازین $[A,\ dirty]$
در دنیای جاروبرقی (قطعی) با حس کردن محلی





راه حل یک پلان شرطی است:

$$[Suck,\ Right,\ \textbf{if}\ BState = \{6\}\ \textbf{then}\ Suck\ \textbf{else}\ [\ ]]$$

یک عامل برای محیط‌های مشاهده‌پذیر جزئی

عامل حل مسئله برای محیط‌های مشاهده‌پذیر جزئی:

۱) فرمول‌بندی مسئله
۲) فراخوانی الگوریتم جستجو (مثل جستجوی گرافی AND-OR) و حل مسئله
۳) اجرای راه‌حل

تفاوت‌ها با عامل حل مسئله‌ی ساده

- راه‌حل یک طرح اقتضائی است (به‌جای یک دنباله)
- عامل نیاز دارد حالت باور خودش را (در حال اجرای کنش‌ها و دریافت ادراکات) نگه دارد.

نگهداری حالت باور، مشابه گام به‌هنگام‌سازی در مرحله‌ی جستجو است:

$$b' = \text{UPDATE}(\text{PREDICT}(b, a), o)$$

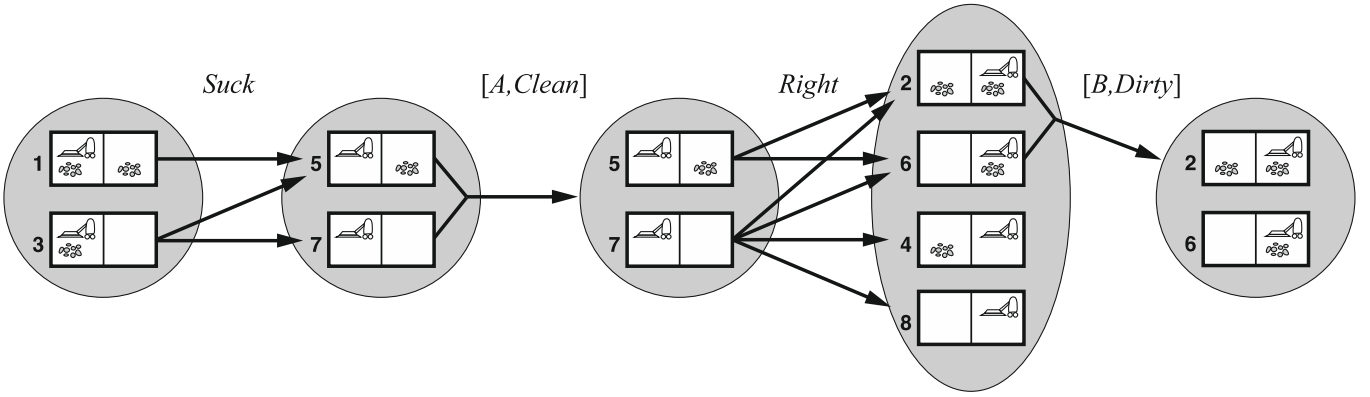
حالت باور جدید — حالت باور — کنش — مشاهده

یک عامل برای محیط‌های مشاهده‌پذیر جزئی

مثال: دنیای جاروبرقی کودکستان

KINDERGARTEN VACUUM WORLD

یک خانه می‌تواند مجدداً آلوده شود، مگر اینکه عامل به‌طور فعالانه در همان لحظه آن را تمیز کند.



حالت‌های باور حاصل برای ۲ چرخه‌ی پیش‌بینی–به‌هنگام‌سازی
برای یک عامل در دنیای جاروبرقی کودکستان

یک عامل برای محیط‌های مشاهده‌پذیر جزئی

نگهداری حالت باور

MAINTAINING BELIEF STATE

نگهداری حالت باور، یک کارکرد مرکزی برای همه‌ی سیستم‌های هوشمند در یک محیط مشاهده‌پذیر جزئی است.

(شامل اکثر مسائل دنیای واقعی)

نام‌های دیگر

نظارت *Monitoring*	فیلتر کردن *Filtering*	تخمین حالت *Estimation*

گام **پیش‌بینی–به‌هنگام‌سازی** باید به‌اندازه‌ی ورود ادراک‌ها **سریع** باشد؛
در غیر این صورت عامل عقب می‌افتد.

* در محیط‌های پیچیده‌تر محاسبه‌ی دقیق امکان‌ناپذیر می‌شود و تقریب‌ها باید استفاده شوند.

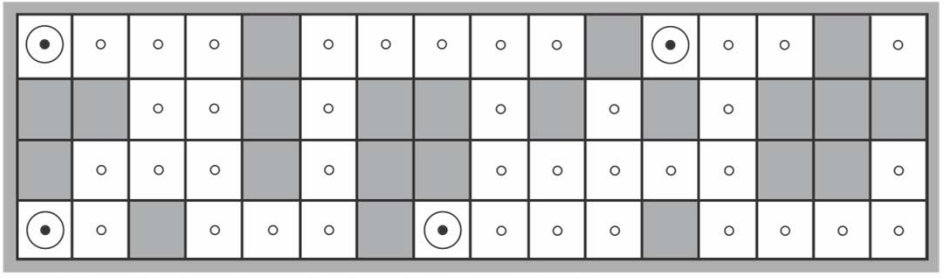
یک عامل برای محیط‌های مشاهده‌پذیر جزئی

نگهداری حالت باور: مثال (مکان‌یابی)

LOCALIZATION

مکان‌یابی، یک مسئله با مشاهدات جزئی است.

مثلاً: تعیین موقعیت یک ربات خطادار در یک راهرو با ۴ حسگر بیان‌کننده‌ی حالت سلول‌ها در ۴ همسایه‌ی آن (آزاد/دیوار)

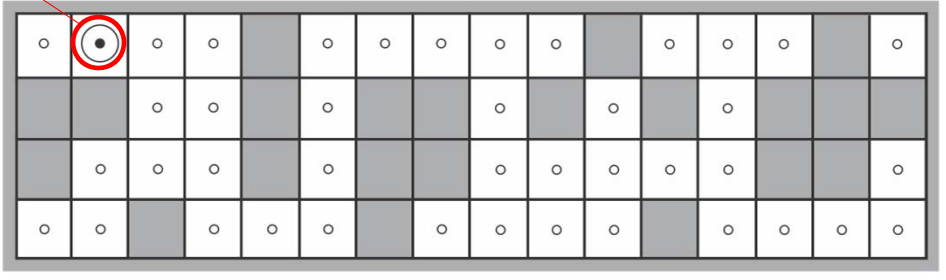


$E_1 =$ W (N, S)

(a) Possible locations of robot after $E_1 =$ NSW

Move
(nondeterministic)

ربات باید اینجا باشد پس از E_1, Move, E_2

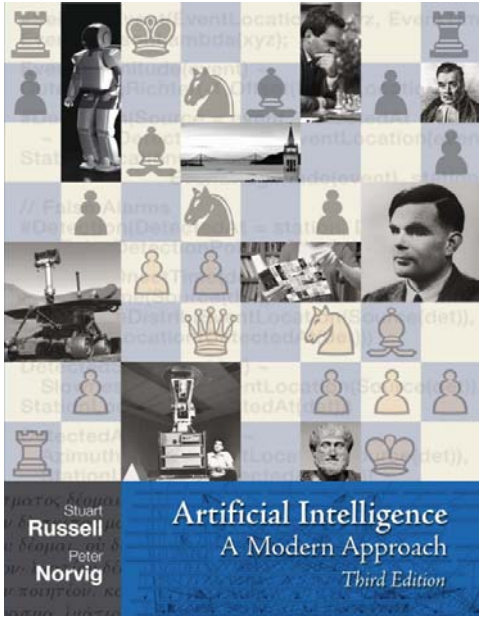


$E_2 =$ (N, S)

(b) Possible locations of robot After $E_1 =$ NSW, $E_2 =$ NS

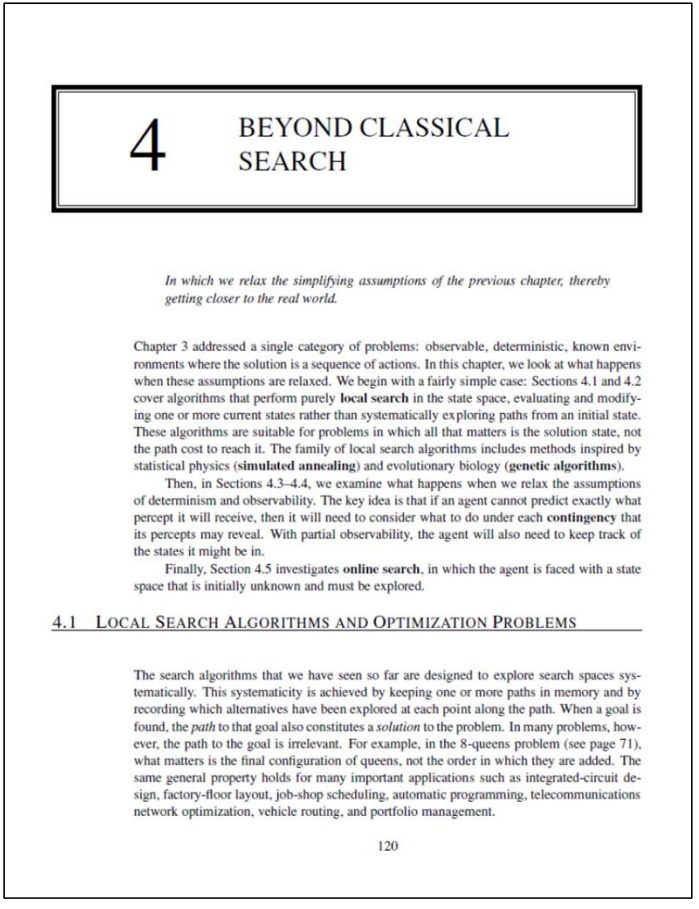
UPDATE(PREDICT(UPDATE(b,NSW),*Move*),NS)

منبع اصلی



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

Chapter 4 (4.4)



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

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

هوش مصنوعی پیشرفته

فراسوی جستجوی کلاسیک

۵

عامل‌های جستجوی برخط و محیط‌های ناشناخته

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:

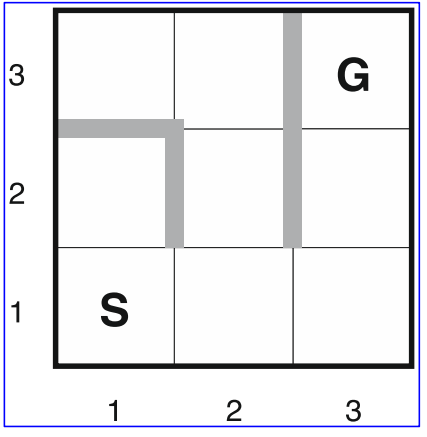
- عامل می‌تواند حالت‌های قبلاً ملاقات‌شده را بازشناسی کند
- کنش‌ها قطعی هستند
- عامل به یک تابع هیوریستیک $h(s)$ دسترسی دارد که فاصله از حالت جاری تا یک حالت هدف را تخمین می‌زند (مانند: فاصله شهری)

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):

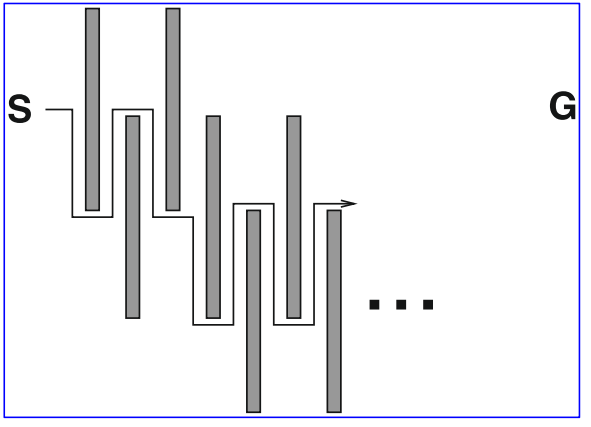
فرض می‌کنیم در حالی که عامل فضای حالت را کشف می‌کند،
یک رقیب وجود دارد که فضای حالت را می‌سازد.
مطابق شکل، حالت‌های S و A ملاقات شده‌اند؛
اما در دو گام دیگر در یکی از فضاهای حالت به بن‌بست برخورد می‌کند.

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)

عامل‌های جستجوی عمق-اول برخط

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
```

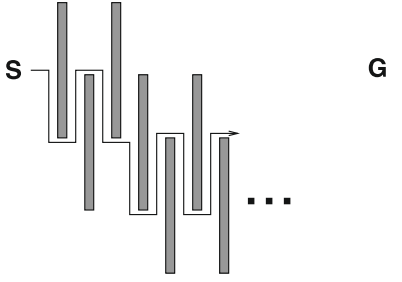
عامل‌های جستجوی عمق-اول برخط

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)

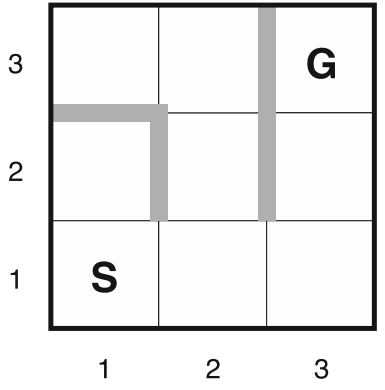


جستجوی عمق-اول برخط تنها زمانی کار می‌کند که کنش‌ها وارون‌پذیر باشند.

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

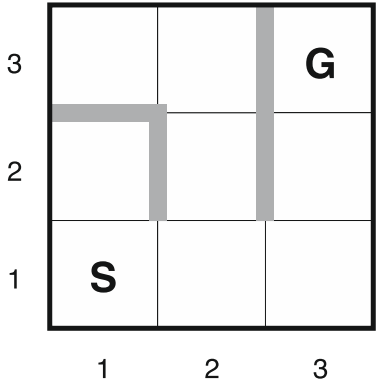
Result, $untried$ (UX), $unbacktracked$ (UB), ... are empty

s ,a are also empty

عامل‌های جستجوی عمق-اول برخط

مثال (۲ از ۷)

ONLINE DFS AGENTS



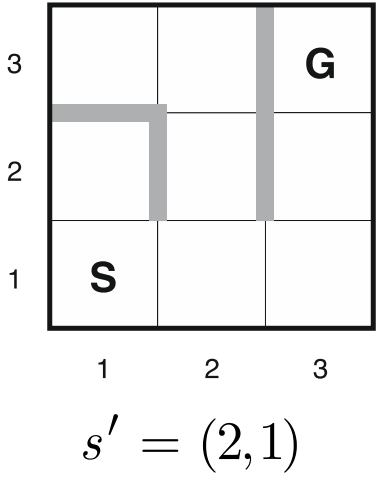
$s' = (1,1)$

- 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



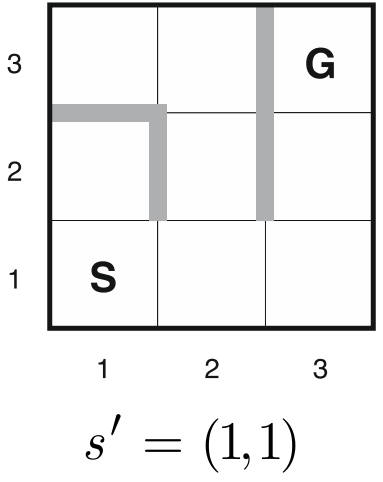
$s' = (2,1)$

- 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



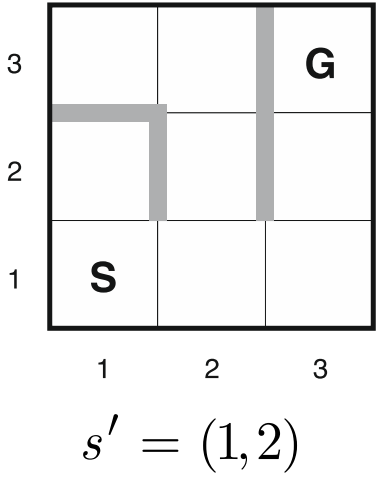
$s' = (1,1)$

- 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



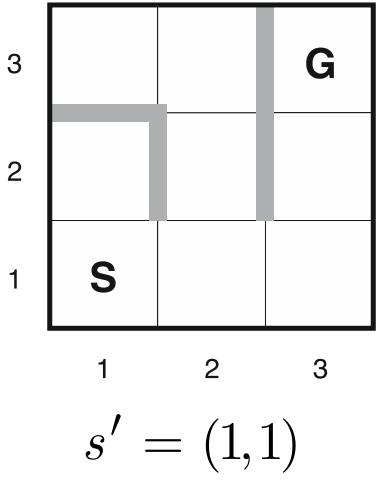
$s' = (1,2)$

- 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



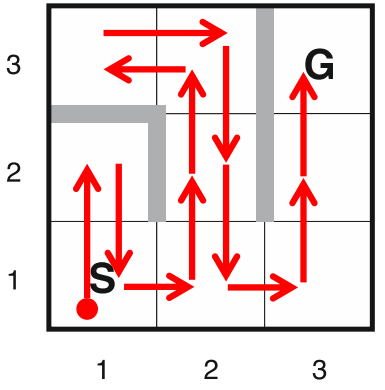
$s' = (1,1)$

- 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)

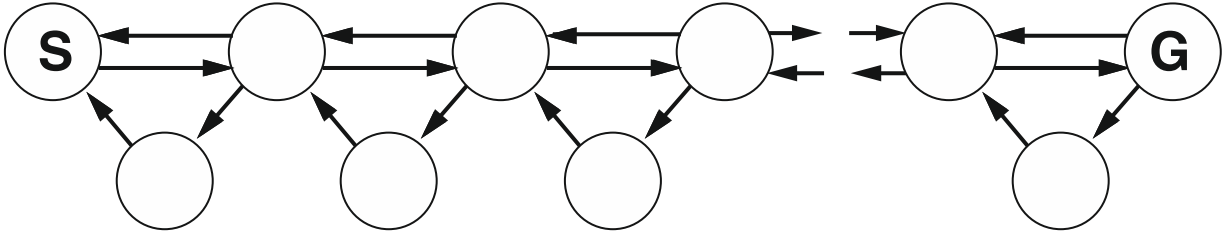
ثابت می‌شود که گام‌برداری تصادفی سرانجام یک هدف را می‌یابد اگر فضای حالت متناهی باشد.

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*)

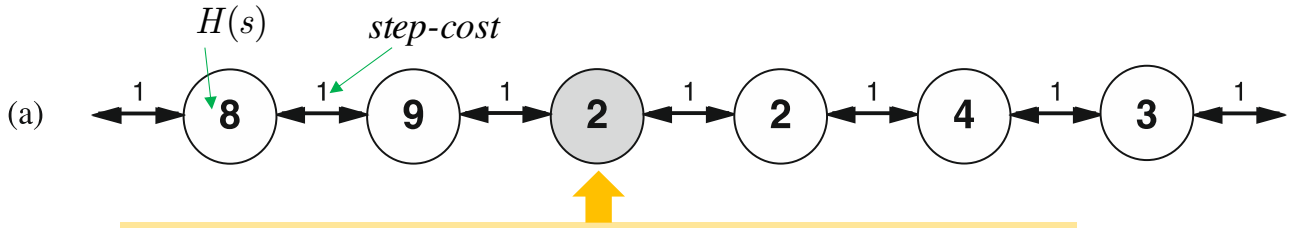
یک «بهترین تخمین فعلی» $H(s)$ از هزینه‌ی رسیدن به هدف
با شروع از هر یک از حالات ملاقات شده، ذخیره می‌شود.

- $H(s)$ 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)

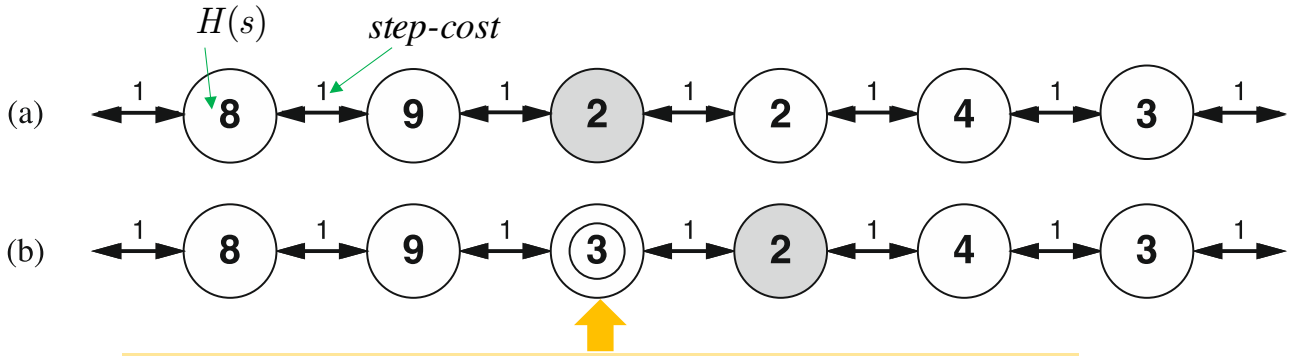
$\Leftarrow$

$H(s) = 2$ 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*)

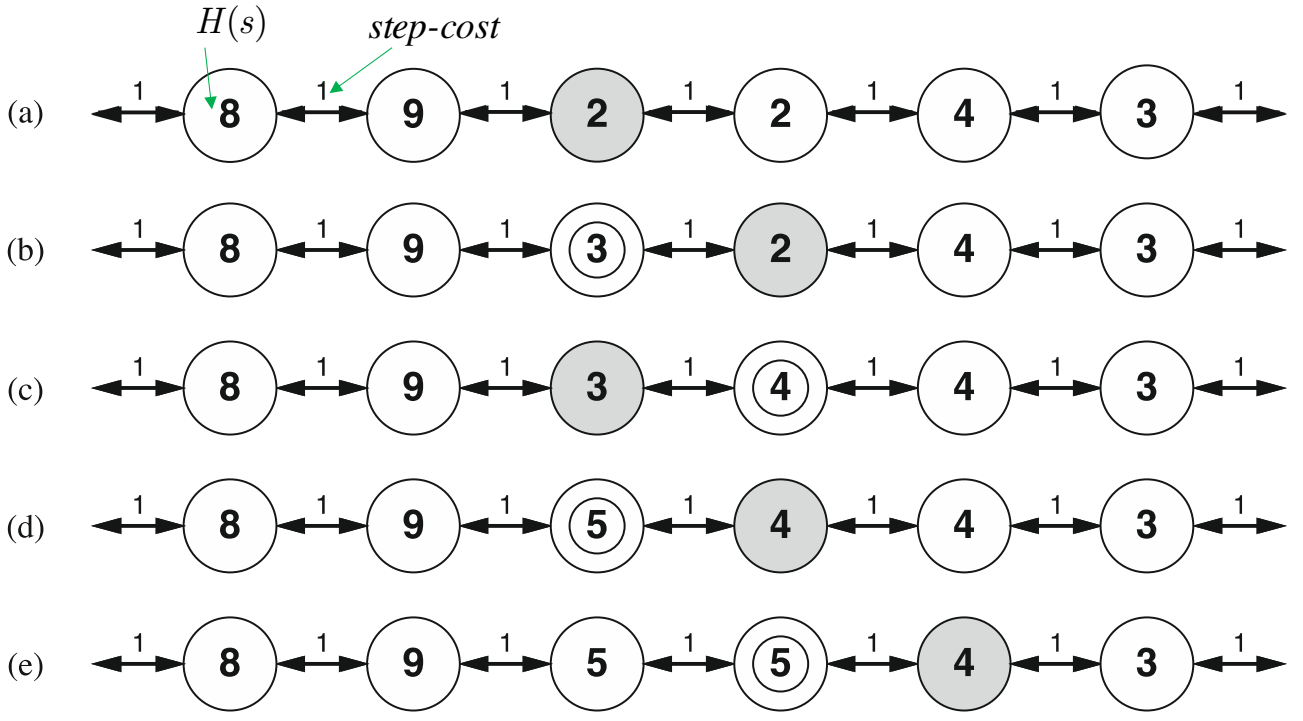


$H(s) = 2$ بیش از حد خوش‌بینانه بود و باید به $H(s) = 3$ به‌هنگام شود.

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)

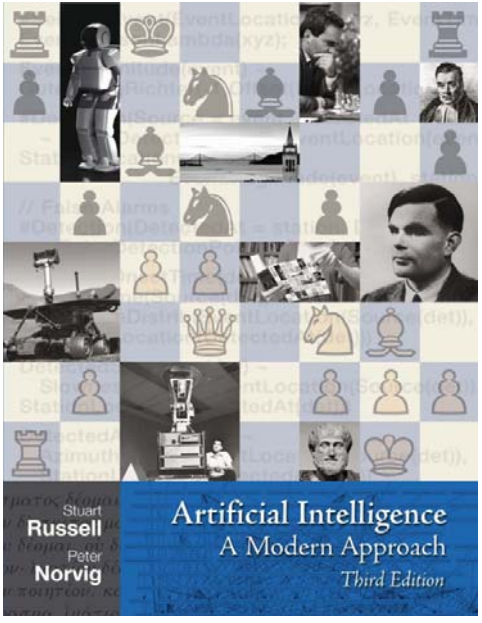
یادگیری تخمین مقدار هر حالت (تا فهمیدن مقدار دقیق آن)

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

ملزومات:

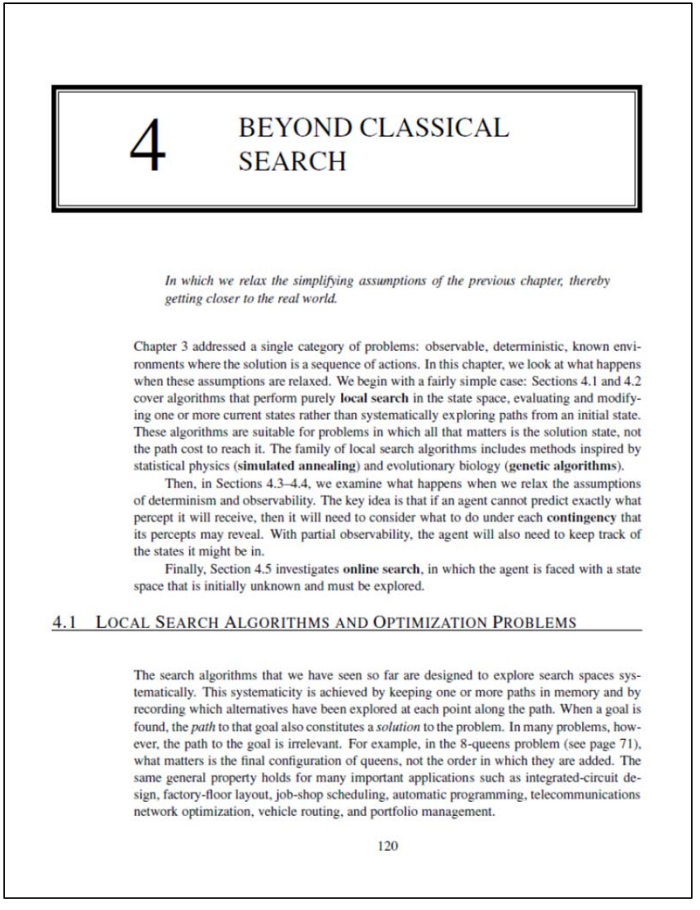
- یک بازنمایی صوری و صریح برای دستیابی به قواعد عمومی (بازنمایی اعلانی)
- الگوریتم‌های سازنده‌ی قواعد عمومی مناسب از روی مشاهدات خاص انجام شده توسط عامل (یادگیری قواعد)

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

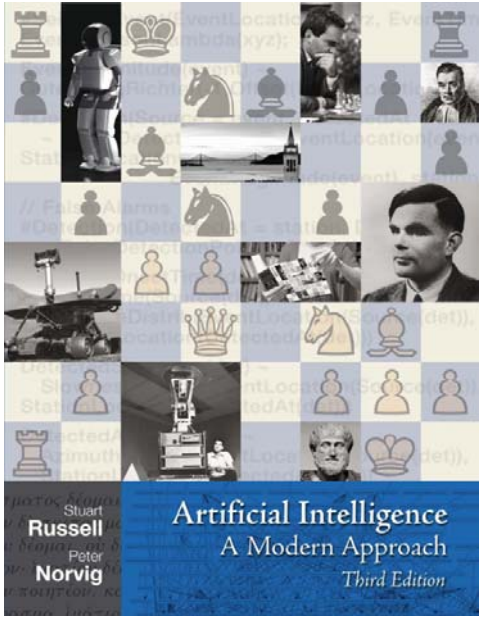
هوش مصنوعی پیشرفته

فراسوی جستجوی کلاسیک

۶

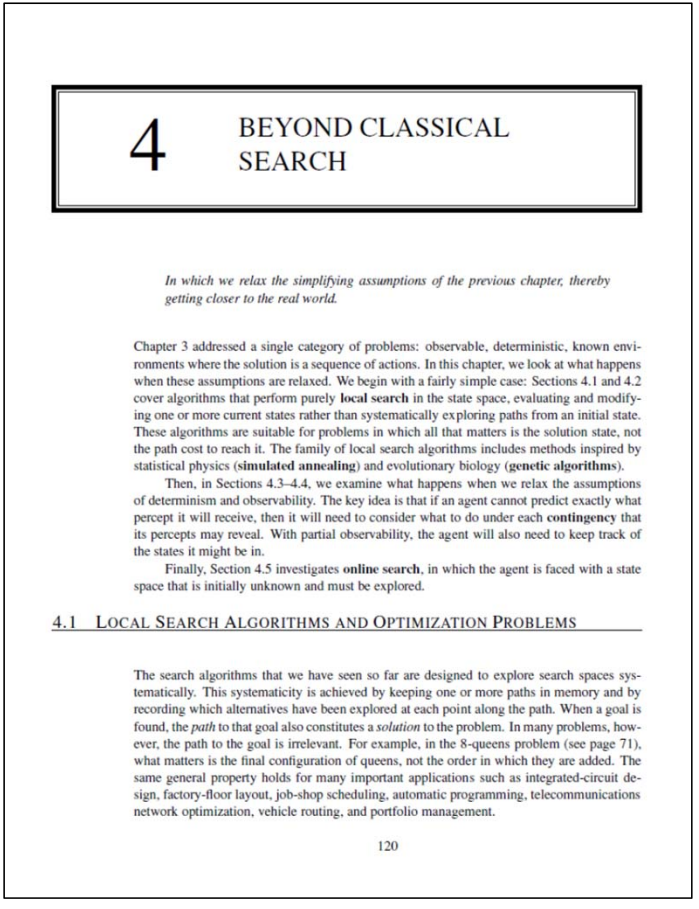
منابع

منبع اصلی



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

Chapter 4 (4.3 .. 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