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

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.

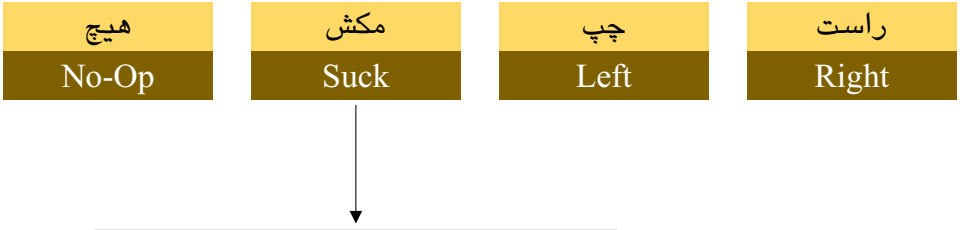
راه‌حل مسئله یک دنباله از کنش‌ها نیست، بلکه یک طرح اقتضائی (استراتژی) [درخت] است:

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



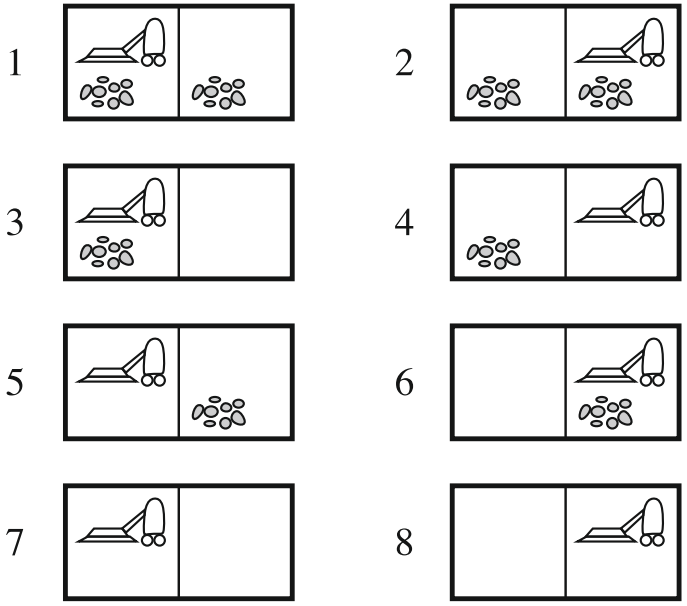
روی خانه‌ی آلوده:
- آن خانه تمیز می‌شود.
- گاهی خانه‌ی مجاور نیز تمیز می‌شود!

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*

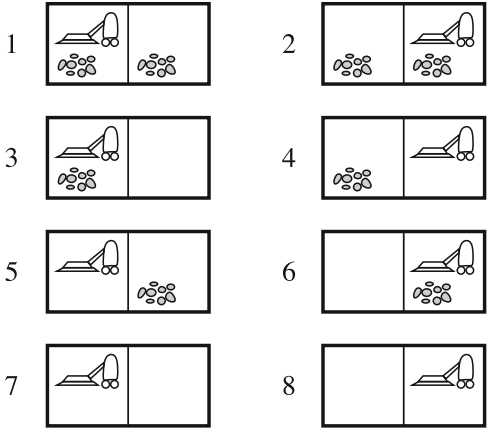
مجموعه‌ی حالت‌های دسترس‌پذیر از حالت آغازین با حداقل یک دنباله از کنش‌ها

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

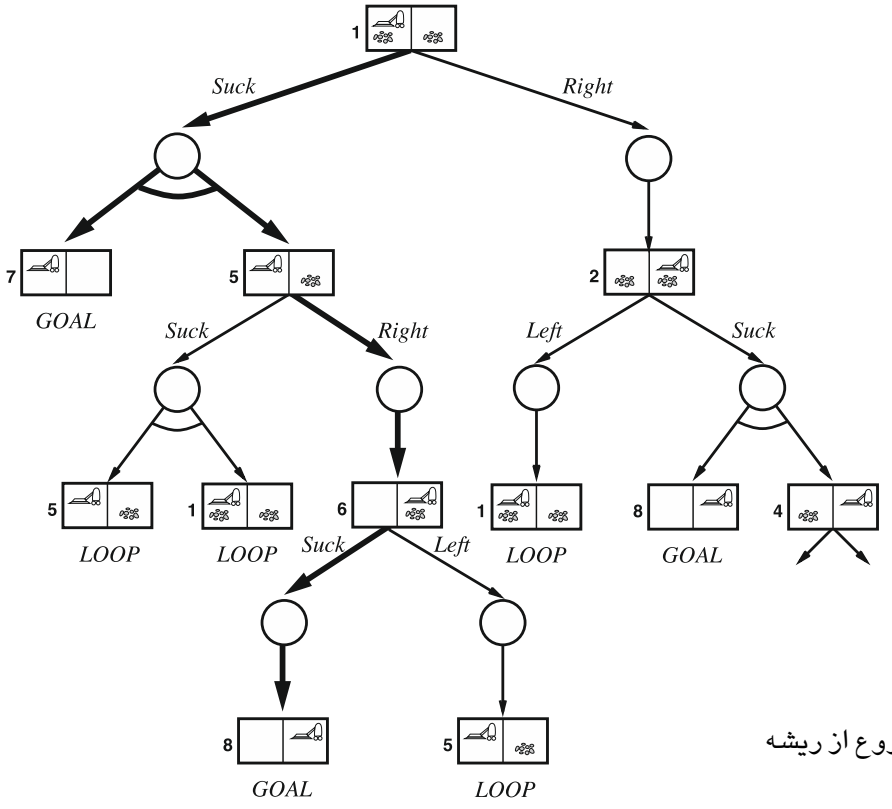
AND-OR Search Tree

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

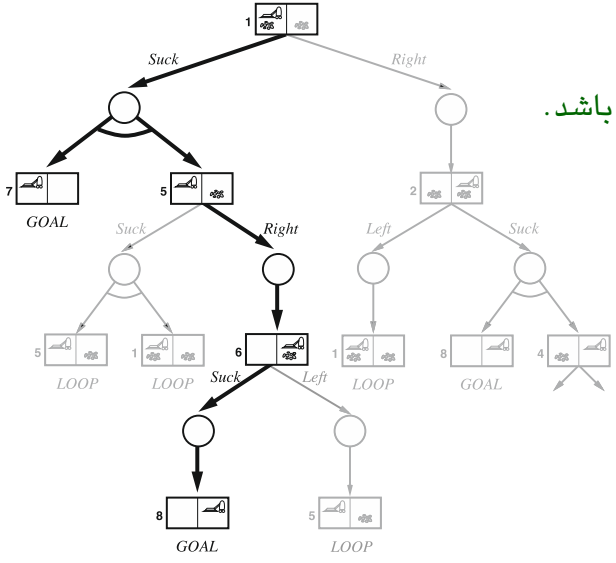
راه‌حل (برای یک مسئله‌ی جستجوی AND-OR)

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

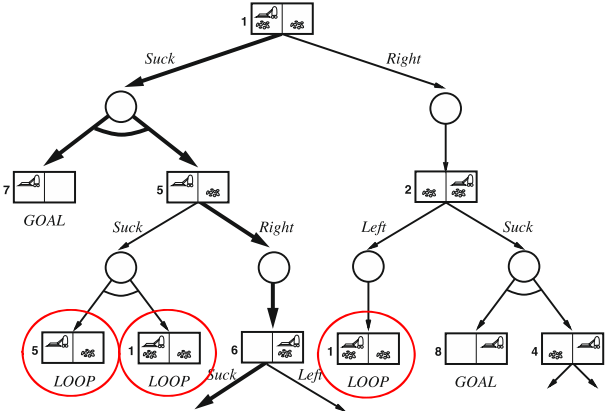
```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
  OR-SEARCH(problem.INITIAL-STATE, problem, [])

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]
```

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.

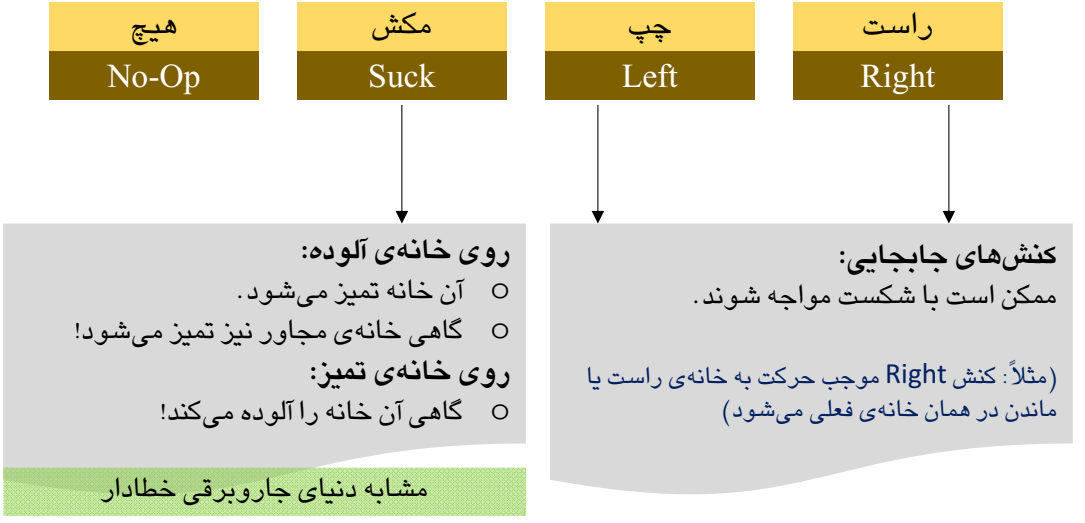
⇓

یک نسخه‌ی قابل مقایسه با جستجوی A* به‌دست می‌آید.

مثال: دنیای جاروبرقی لغزنده

کنش‌ها

THE SLIPPERY VACUUM WORLD



مثال: دنیای جاروبرقی لغزنده

راه‌حل

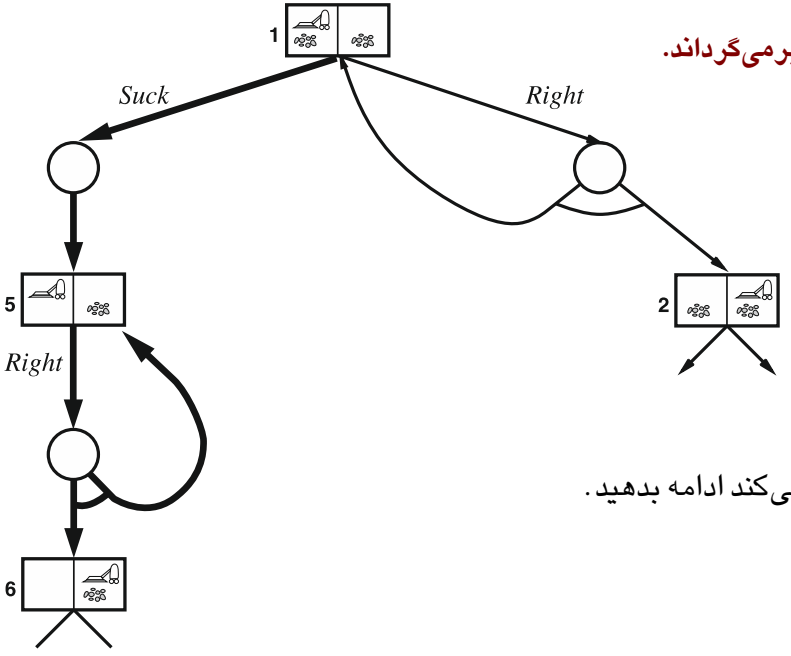
THE SLIPPERY VACUUM WORLD

راه‌حل «بدون دور»

هیچ راه‌حل بدون دوری با شروع از حالت 1 وجود ندارد

⇓

الگوریتم جستجوی گرافی AND-OR، شکست را برمی‌گرداند.



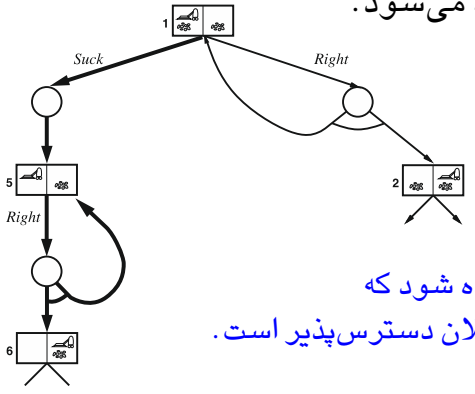
راه‌حل «دور دار»

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.

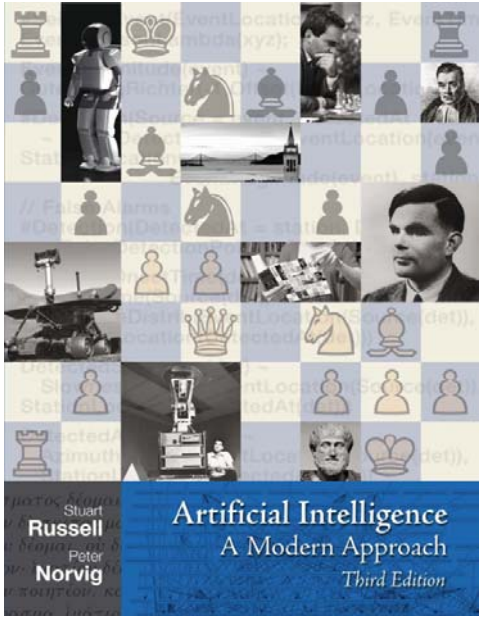


$$[Suck, \text{L1: } Right \textbf{ if } State = 5 \textbf{ then } \text{L1} \textbf{ else } Suck]$$

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.

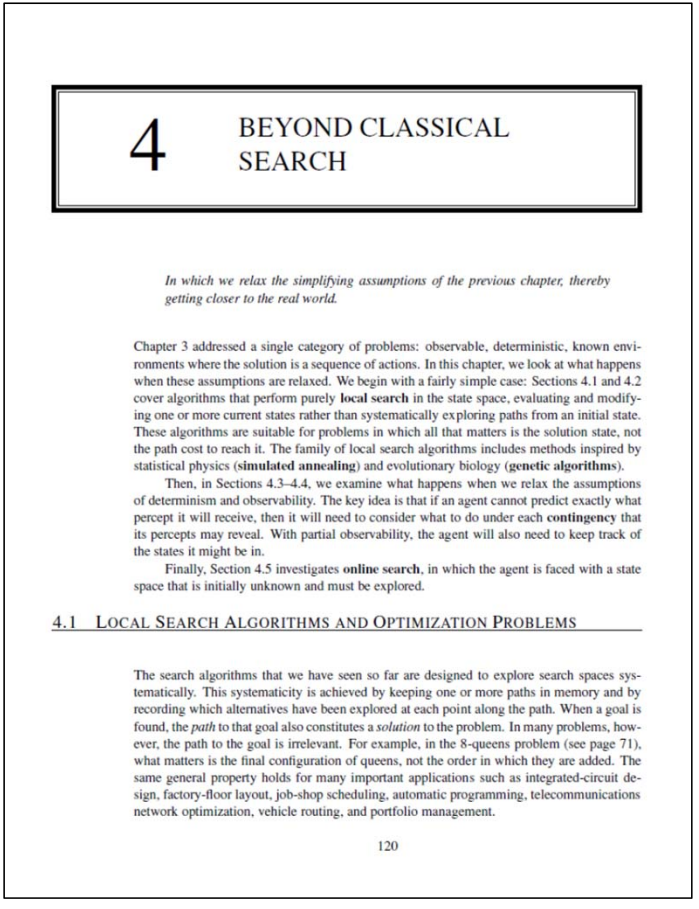
با داشتن تعریف یک راه‌حل دوردار، عاملی که این راه‌حل را اجرا می‌کند سرانجام به هدف می‌رسد، با احتمال یک،
تنها اگر عدم قطعیت ناشی از یک فرآیند اتفاقی باشد
و توسط یک خصوصیت پنهان محیط که از برخی گذرهای حالت جلوگیری می‌کند، ایجاد نشده باشد.

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