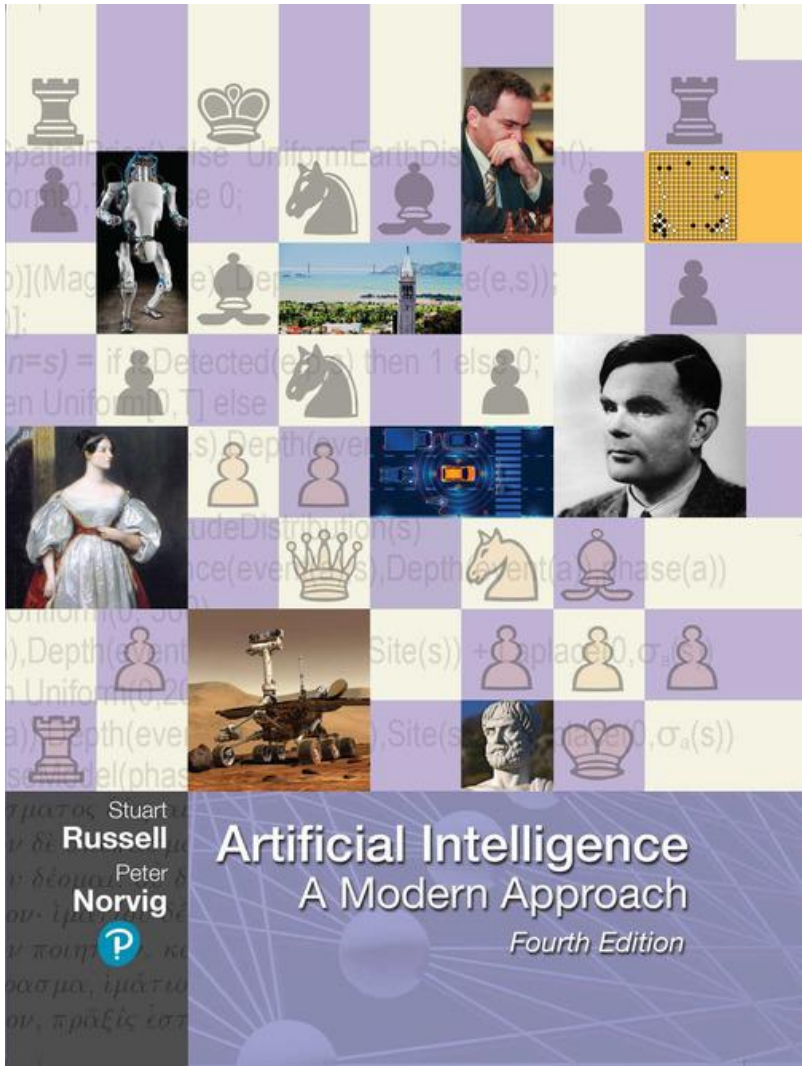
Artificial Intelligence Fundamentals

2023-2024



"The question of whether a computer can think is no more interesting than the question of whether a submarine can swim."

— Edsger W. Dijkstra

AIMA Chapter 4

Search in Complex Environments

Pearson

Copyright © 2022 Pearson Education, Inc. All Rights Reserved

Outline

- Local Search and Optimization Problems
  - Hill-climbing
  - Simulated annealing
  - Genetic algorithms
- Local search in continuous spaces
- Search with Nondeterministic Actions
- Search in Partially Observable Environments

Pearson

© 2022 Pearson Education Ltd.

Motivation

we relax the simplifying assumptions of the previous lecture, to get **closer to the real world**.

We begin with the problem of **finding a good state** without worrying about the path to get there, covering both discrete and continuous states.

Then we relax the assumptions of **determinism** and **observability.** In a non-deterministic world, the agent will need a conditional plan and carry out different actions depending on what it observes—for example, stopping if the light is red and going if it is green.

With partial observability, the agent will also **need to keep track of the possible states it might be in**.

© 2022 Pearson Education Ltd.

Local Search and Optimization Problems

In many optimization problems, path is irrelevant; the goal state itself is the solution

Then state space = set of "complete" configurations; find optimal configuration, e.g., Travelling Salesman Problem (TSP)
or, find configuration satisfying constraints, e.g., timetable

In such cases, one can use iterative improvement algorithms; keep a single "current" state, try to improve it

Local search algorithms operate by searching from a start state to neighboring states, **without keeping track of the paths, nor the set of states that have been reached**.

They are not systematic—they might never explore a portion of the search space where a solution actually resides.

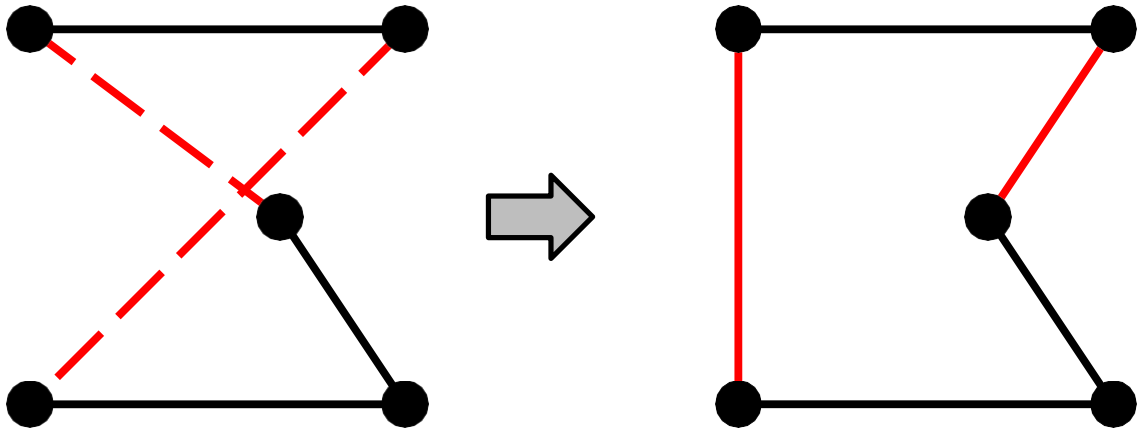
However, they have two key advantages: (1) they use **very little memory**; and (2) they can often find **reasonable solutions in large or infinite state spaces** for which systematic algorithms are unsuitable.

Pearson

© 2022 Pearson Education Ltd.

Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges



Variants of this approach get within 1% of optimal very quickly with thousands of cities

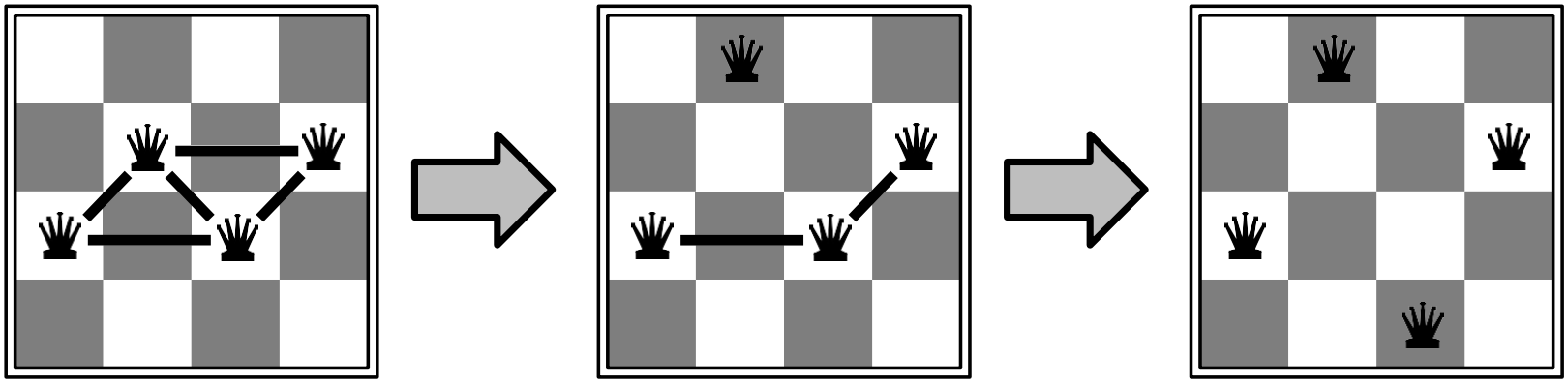
Pearson

© 2022 Pearson Education Ltd.

Example: n-queens

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Move a queen to reduce number of conflicts



h = 5 h = 2 h = 0

Almost always solves n-queens problems almost instantaneously for very large n, e.g., $n = 1$ *million*

Pearson

© 2022 Pearson Education Ltd.

Hill-climbing (or gradient ascent/descent)

"Like climbing Everest in **thick fog** with **amnesia**"

```
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
    end
```

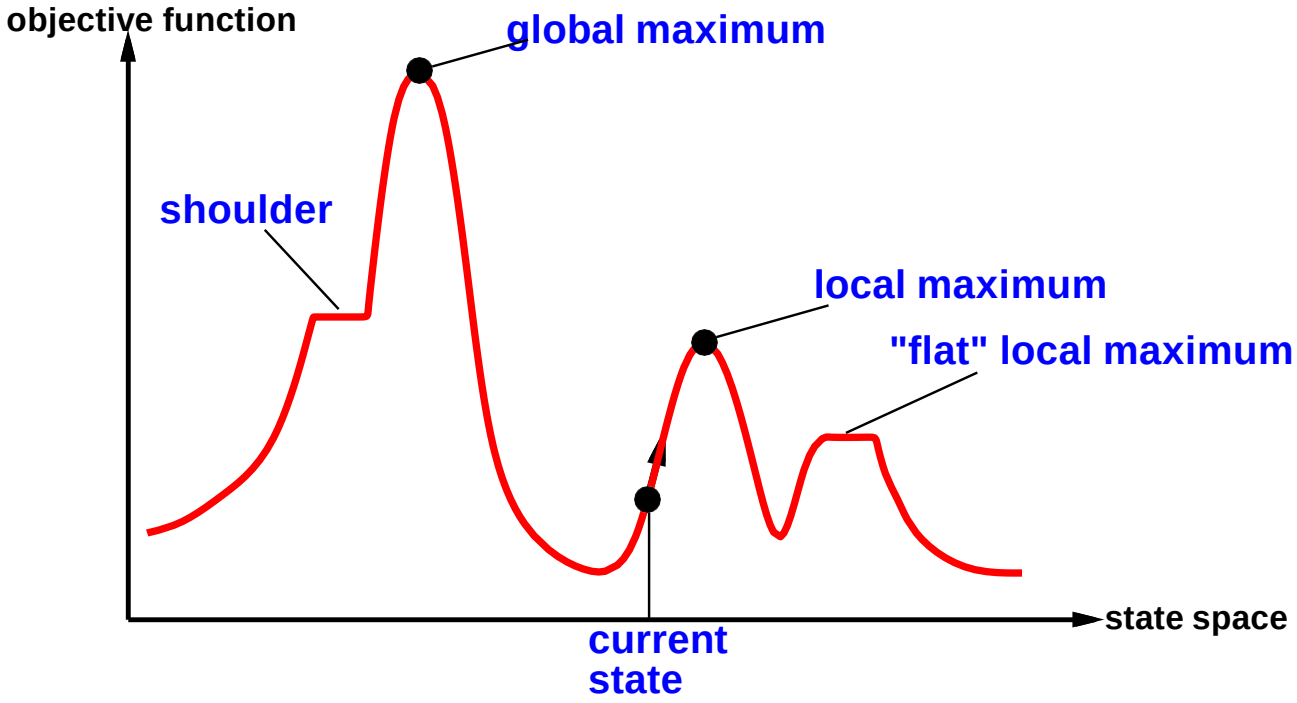
It keeps track of **one current state** and **on each iteration moves to the neighboring state with highest value**—that is, it heads in the direction that provides the steepest ascent (better solution).

Pearson

© 2022 Pearson Education Ltd.

Hill-climbing contd.

Useful to consider state space landscape



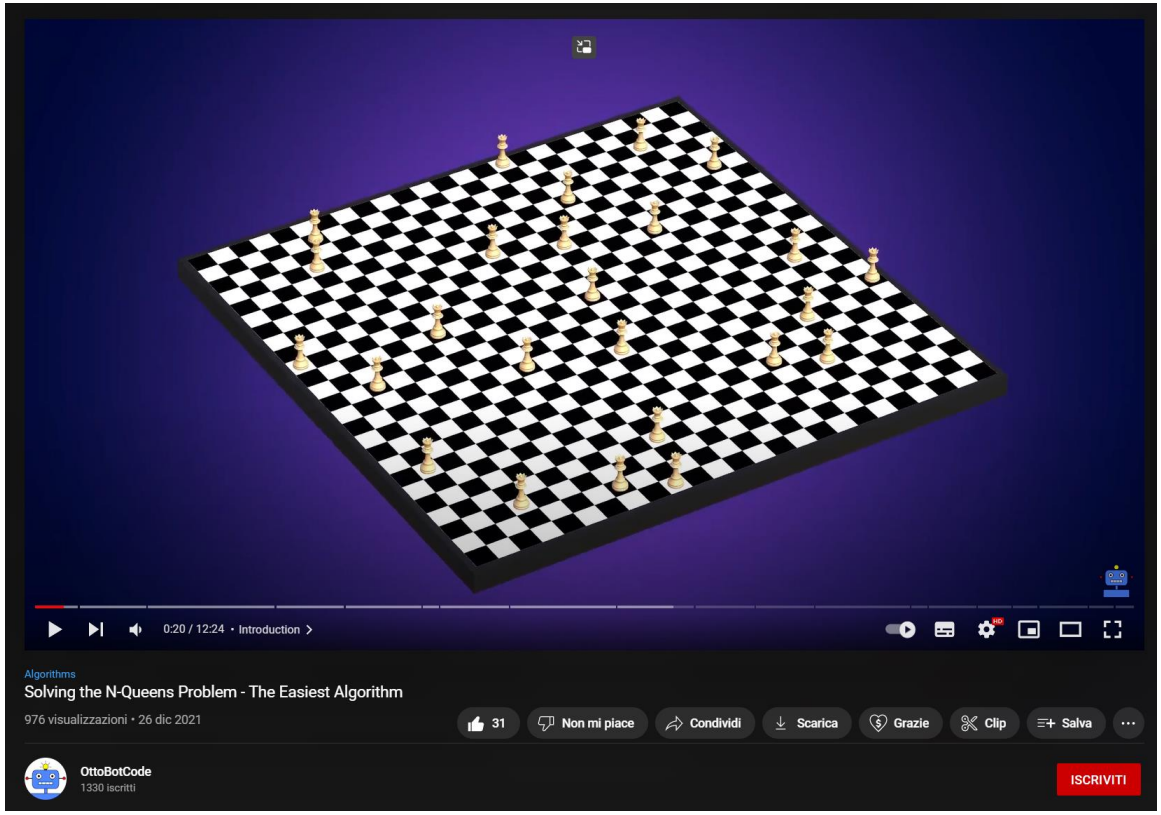
Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves escape from shoulders loop on flat maxima

Pearson

© 2022 Pearson Education Ltd.

Hill-climbing contd.



Animation + implementation tricks:
https://www.youtube.com/watch?v=7fjmGWkv-sY

Homework: *try to implement it yourself at home with Python!*

© 2022 Pearson Education Ltd.

Simulated annealing

Idea: escape local maxima by **allowing some "bad" moves**
but gradually decrease their size and frequency

A hill-climbing algorithm that never makes "downhill"

it seems reasonable to try to combine **hill climbing** with a **random walk** in a way that yields both efficiency and completeness.

In metallurgy, **"annealing"** is the process used to temper or harden metals and glass by heating them to a high temperature and then *gradually cooling them*, thus allowing the material to reach a low-energy crystalline state.

Pearson

© 2022 Pearson Education Ltd.

Simulated annealing

Imagine the task of getting a ping-pong ball into the deepest crevice in a very bumpy surface.

If we just let the ball roll, it will come to rest at a local minimum. If we shake the surface, we can bounce the ball out of the local minimum—perhaps into a deeper local minimum, where it will spend more time.

The trick is to **shake just hard enough to bounce the ball out of local minima** but not hard enough to dislodge it from the global minimum.

© 2022 Pearson Education Ltd.

Simulated annealing

Idea: escape local maxima by **allowing some "bad" moves**
but gradually decrease their size and frequency

```
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^(ΔE/T)
```

© 2022 Pearson Education Ltd.

Properties of simulated annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = ae^{\frac{E(x)}{kT}}$$

T decreased slowly enough ⇒ always reach best state x^*

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in Very large-scale integration (VLSI, for integrated circuit) layout, airline scheduling, etc.

Pearson

© 2022 Pearson Education Ltd.

Local beam search

Keeping just one node in memory might seem to be an extreme reaction to the problem of memory limitations.

Idea: **keep k states instead of 1**; choose top k of all their successors

Not the same as k searches run in parallel!
-> Searches that find good states recruit other searches to join them

Problem: quite often, all k states end up on same local hill

Idea: **choose k successors randomly**, biased towards good ones

Observe the close analogy to natural selection!

Pearson

© 2022 Pearson Education Ltd.

Genetic algorithms

Evolutionary algorithms can be seen as variants of stochastic beam search that are explicitly **motivated by the metaphor of natural selection in biology**:

there is a population of individuals (**states**), in which the fittest (**highest value**) individuals produce offspring (**successor states**) that populate the next generation, a process called *recombination*.

Algorithms differ for:

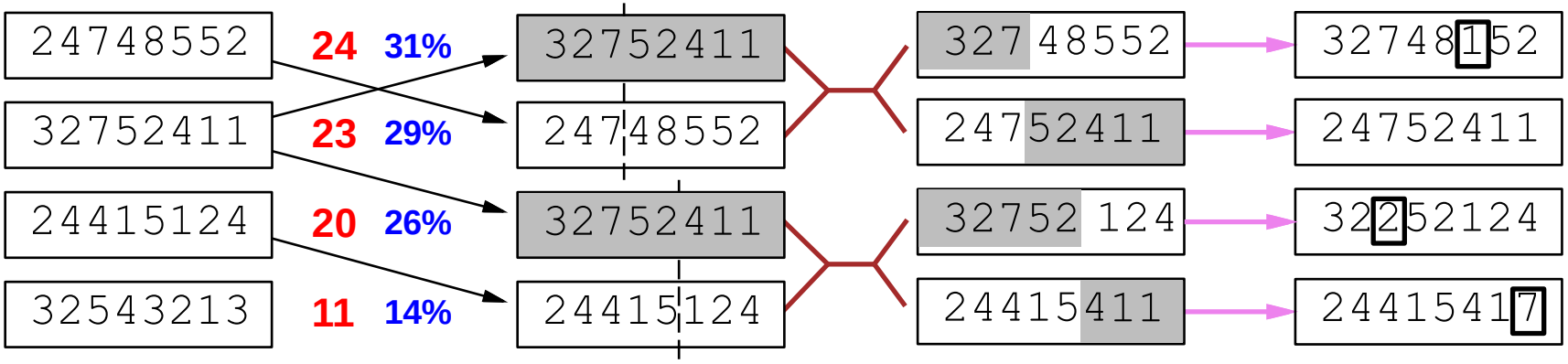
- The size of the population.
- The representation of each individual (DNA)
- The mixing number (how many parents)
- The selection process (fitness function + next gen. makeup)
- The recombination procedure (cross-over point, etc.)
- Mutation rate

Pearson

© 2022 Pearson Education Ltd.

Genetic algorithms

= stochastic local beam search + generate successors from pairs of states



DNA: digit strings representing **8-queens** states.
Fitness: *nonattacking pairs of queens* -> normalized by percentage

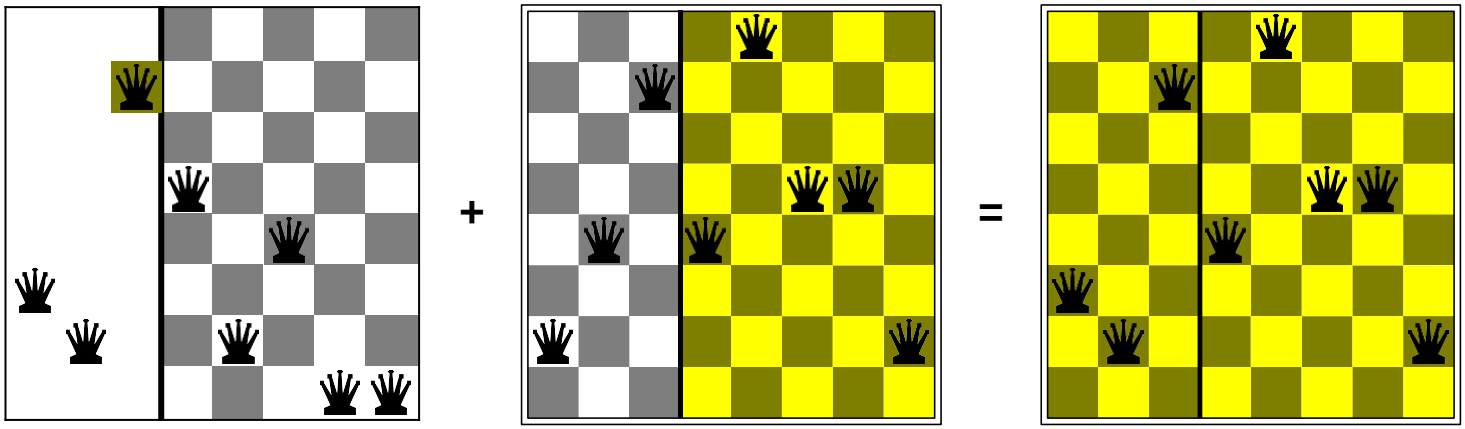
Genetic algorithms are similar to stochastic beam search, but with the addition of the crossover operation

© 2022 Pearson Education Ltd.

Genetic algorithms contd.

Crossover helps iff substrings are meaningful components

-> if there are blocks that perform useful functions (genes)



Pearson

© 2022 Pearson Education Ltd.

Genetic algorithms contd.

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Figure 4.8 A genetic algorithm. Within the function, *population* is an ordered list of individuals, *weights* is a list of corresponding fitness values for each individual, and *fitness* is a function to compute these values.

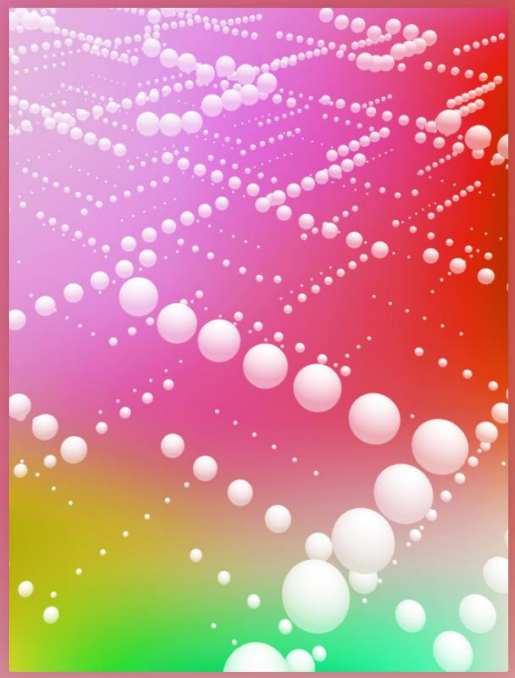
© 2022 Pearson Education Ltd.

Genetic algorithms contd.

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

We've discovered that *evolution strategies (ES)*, an optimization technique that's been known for decades, rivals the performance of standard *reinforcement learning (RL)* techniques on modern RL benchmarks (e.g. Atari/MuJoCo), while overcoming many of RL's inconveniences.

March 24, 2017
12 minute read



ES is simpler to implement (there is no need for backpropagation), it is easier to scale in a distributed setting, it does not suffer in settings with sparse rewards, and has fewer hyperparameters.

OpenAI Blog Post + Paper + GitHub -> https://openai.com/blog/evolution-strategies/

© 2022 Pearson Education Ltd.

Evolution vs Search

The theory of evolution was developed by Charles Darwin in ***"On the Origin of Species by Means of Natural Selection"*** (1859) and independently by Alfred Russel Wallace (1858) with **no knowledge** of how the traits of organisms can be inherited and modified (1953 -> DNA structure).

The central idea is simple: variations occur in reproduction and will be preserved in successive generations *approximately in proportion* to their effect on reproductive fitness.

The actual mechanisms of evolution are far richer than most genetic algorithms allow -> Most important is the fact that the **genes themselves encode the mechanisms whereby the genome is reproduced and translated into an organism**.

Valiant, Leslie in **"Probably Approximately Correct"**. Basic Books, New York. 2013, challenges Darwin theory -> **too inefficient**, there should be learning somewhere!

Further read: https://owlcation.com/social-sciences/What-is-the-Probably-Approximately-Correct-Learning-Theory

© 2022 Pearson Education Ltd.

Continuous state spaces

Suppose **we want to site** three airports in Romania:

- 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)
- objective function $f(x_1, y_2, x_2, y_2, x_3, y_3)$ =
  sum of squared distances from each city to nearest airport

Discretization methods turn continuous space into discrete space (map grid), e.g., empirical gradient considers $\pm\delta$ change in each coordinate

Gradient methods compute

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right]$$

to increase/reduce f, e.g., by $\mathrm{x} \leftarrow \mathrm{x} + a\nabla f(\mathrm{x})$

Sometimes can solve for $\nabla f(\mathrm{x}) = 0$ exactly (e.g., with one city).
Newton–Raphson (1664, 1690) iterates $\mathrm{x} \leftarrow \mathrm{x} - \mathrm{H}_\mathrm{f}^{-1}(\mathrm{x})\nabla f(\mathrm{x})$
to solve $\nabla f(\mathrm{x}) = 0$, where $\mathrm{H}_{\mathrm{ij}} = \partial^2 f / \partial x_\mathrm{i} \partial x_\mathrm{j}$

Pearson

© 2022 Pearson Education Ltd.

Continuous state spaces

For high-dimensional problems, however, computing the n^2 entries of the Hessian and inverting it **may be expensive**, so many approximate versions of the Newton–Raphson method have been developed.

Local search methods suffer from local *maxima*, *ridges*, and *plateaus* in continuous state spaces just as much as in discrete spaces. Random restarts and simulated annealing are often helpful.

High-dimensional continuous spaces are, however, big places in which it is very easy to get lost…

© 2022 Pearson Education Ltd.

Search with Nondeterministic Actions

Agent doesn't know the state its transitioned to **after action**, the environment is **nondeterministic**.

Rather, it will know the possible states it will be in, which is called "**belief state**"

Many problems in the real, physical world are contingency problems, **because exact prediction of the future is impossible**.

For this reason, many people keep their eyes open while walking around.

Pearson

© 2022 Pearson Education Ltd.

The erratic vacuum world

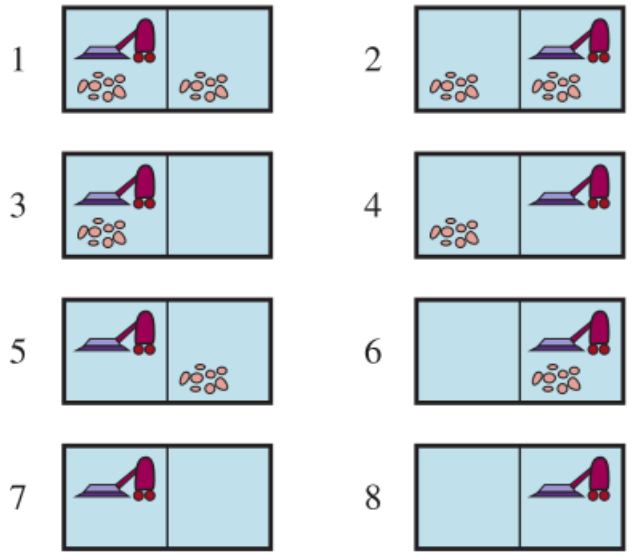


Figure 4.9 The eight possible states of the vacuum world; states 7 and 8 are goal states.

Now suppose that we introduce nondeterminism in the form of a powerful but erratic vacuum cleaner. In the erratic vacuum world, the **"Suck" action** works as follows:

- When applied to a dirty square the action cleans the square and sometimes cleans up dirt in an adjacent square, too.
- When applied to a clean square the action sometimes deposits dirt on the carpet.

© 2022 Pearson Education Ltd.

The erratic vacuum world

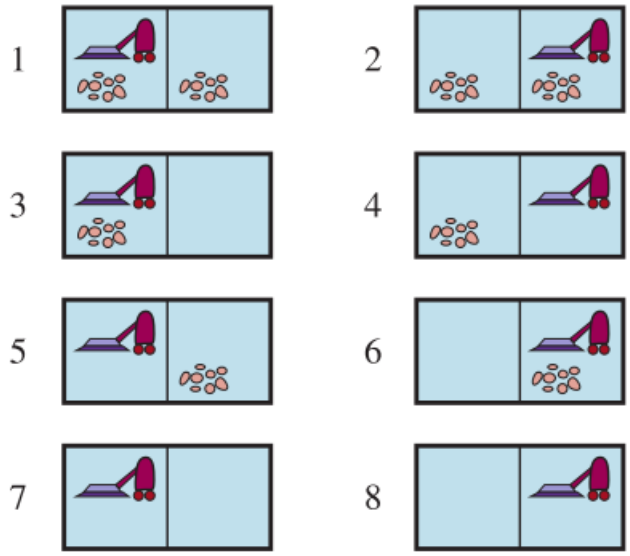


Figure 4.9 The eight possible states of the vacuum world; states 7 and 8 are goal states.

Instead of defining the transition model by a RESULT function that returns a single outcome state, we use a RESULTS function that returns a set of possible outcome states.

For example, in the erratic vacuum world, the Suck action in state 1 cleans up either just the current location, or both locations: **RESULTS(1, Suck) = {5,7}**

© 2022 Pearson Education Ltd.

The erratic vacuum world

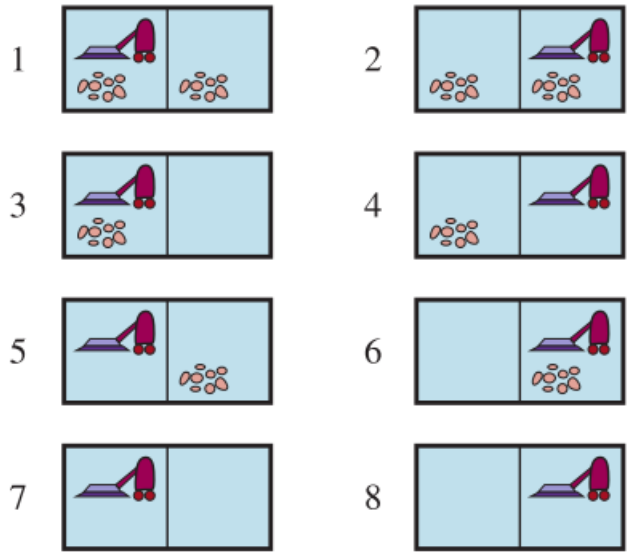


Figure 4.9 The eight possible states of the vacuum world; states 7 and 8 are goal states.

If we start in state 1, no single sequence of actions solves the problem, but the following conditional plan does: **[Suck, if State=5 then [Right, Suck] else [ ]]** .

Here we see that a conditional plan can contain if–then–else steps; this means that solutions are trees rather than sequences.

Here the conditional in the if statement tests to see what the current state is; this is something the agent will be able to **observe at runtime**, not at planning time.

© 2022 Pearson Education Ltd.

AND–OR search trees

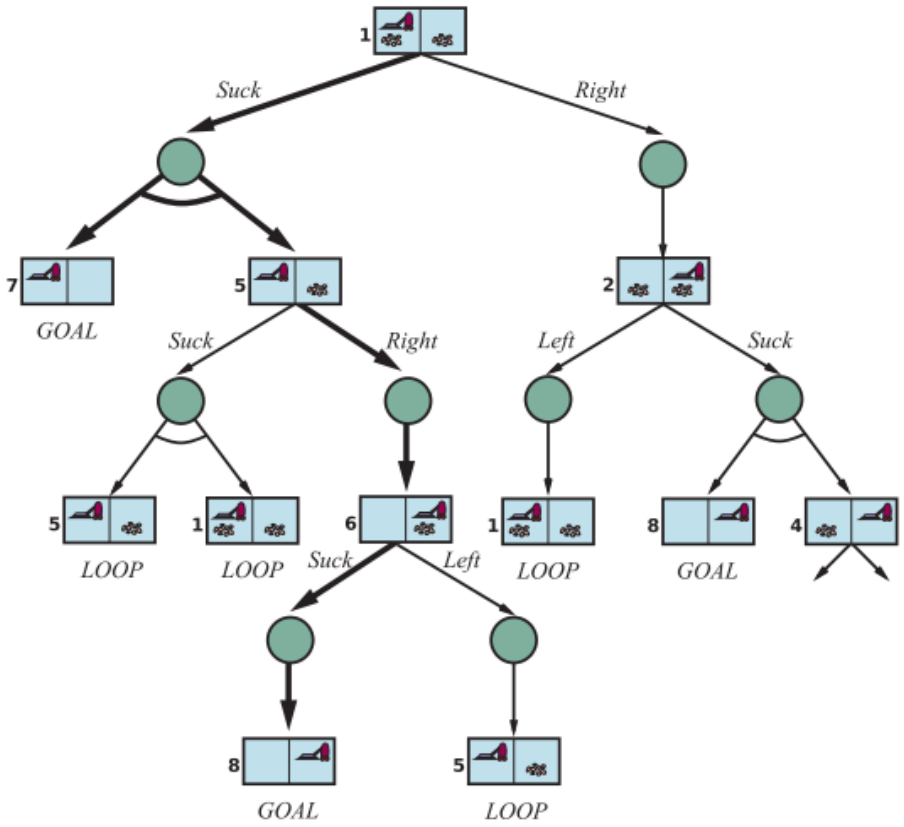


Figure 4.10 The first two levels of the search tree for the erratic vacuum world. State nodes are OR nodes where some action must be chosen. At the AND nodes, shown as circles, every outcome must be handled, as indicated by the arc linking the outgoing branches. The solution found is shown in bold lines.

AND–OR graphs can be explored either **breadth-first** or **best-first**.

The concept of a *heuristic function* must be modified to estimate the cost of a **contingent solution** rather than a sequence, but the notion of admissibility carries over and there is an analog of the **A∗** algorithm for finding optimal solutions.

© 2022 Pearson Education Ltd.

AND–OR search trees

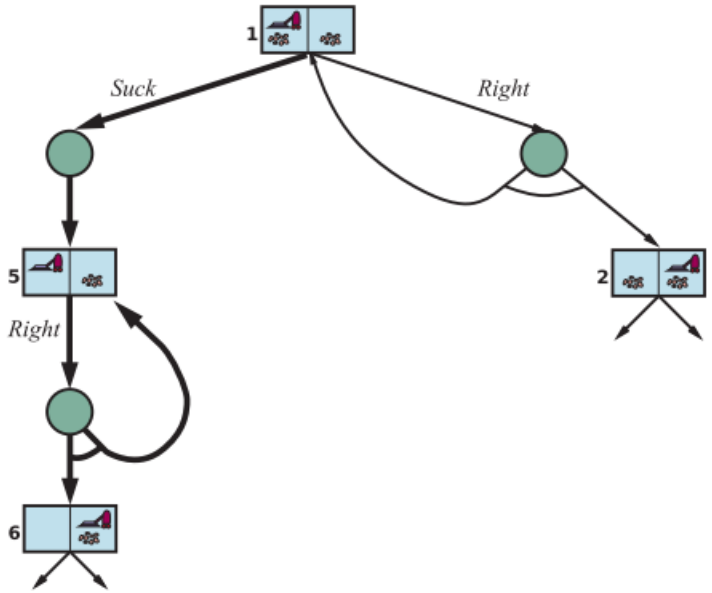


Figure 4.12 Part of the search graph for a slippery vacuum world, where we have shown (some) cycles explicitly. All solutions for this problem are cyclic plans because there is no way to move reliably.

Consider a **slippery vacuum world**, which is identical to the ordinary (non-erratic) vacuum world except that movement actions sometimes fail, leaving the agent in the same location.

When is a **cyclic plan a solution**? A minimum condition is that every leaf is a goal state and that a leaf is reachable from every point in the plan.

© 2022 Pearson Education Ltd.

Search in Partially Observable Environments

Problem of partial observability, where the agent's percepts are not enough to pin down the exact state.

Searching with no observation: Agent's percepts provide **no information at all -> *sensorless problem*** (or a conformant problem).

- Solution: *sequence of actions, not a conditional plan*

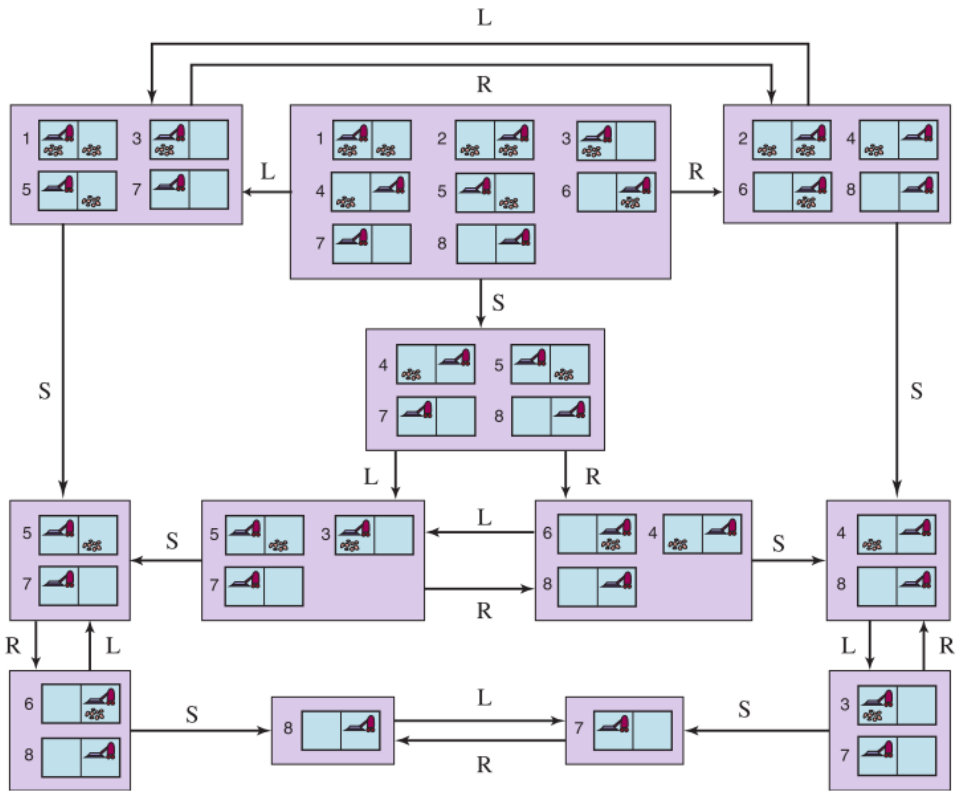


Figure 4.14 The reachable portion of the belief-state space for the deterministic, sensorless vacuum world. Each rectangular box corresponds to a single belief state. At any given point, the agent has a belief state but does not know which physical state it is in. The initial belief state (complete ignorance) is the top center box.

Pearson

© 2022 Pearson Education Ltd.

Search in Partially Observable Environments

Many problems cannot be solved without sensing. For example, the *sensorless 8-puzzle* is impossible. **Searching in partially observable environments** requires a function that **monitors** or **estimates** the environment to maintain the belief state.

Consider a **local-sensing vacuum world**, in which the agent has a position sensor that yields the percept L in the left square, and R in the right square, and a dirt sensor that yields Dirty when the current square is dirty and Clean when it is clean. Thus, the PERCEPT in state 1 is **[L, Dirty]**

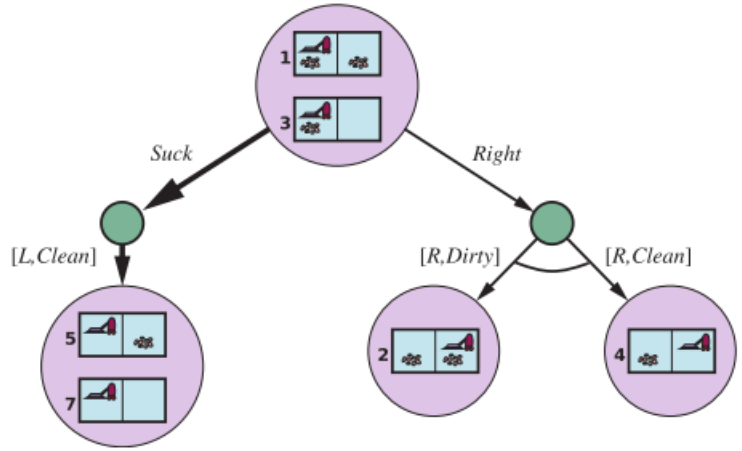


Figure 4.16 The first level of the AND–OR search tree for a problem in the local-sensing vacuum world; *Suck* is the first action in the solution.

© 2022 Pearson Education Ltd.

Summary

Local search methods keep only a **small number of states** in memory that are useful for optimization.

In **nondeterministic environments**, agents can apply **AND–OR search** to generate contingency plans that reach the goal regardless of which outcomes occur during execution.

Belief-state is the set of **possible states** that the agent is in for **partially observable environments**.

Standard search algorithms can be **applied directly to belief-state** space to solve **sensorless** problems.

Pearson

© 2022 Pearson Education Ltd.

In the next lecture...

- Defining *Constraint Satisfaction Problems* (CSP)
- CSP examples
- Backtracking search for CSPs
- Local search for CSPs
- Problem structure and problem decomposition

© 2022 Pearson Education Ltd.