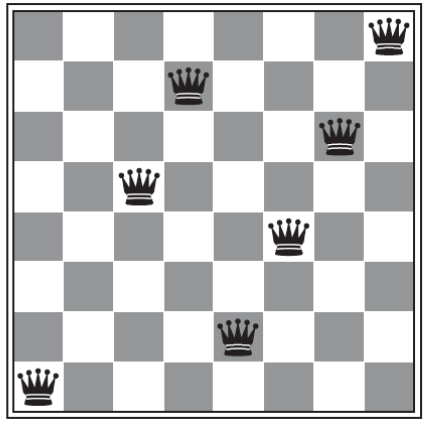
Chapter4: Local Search

Local Search

- Local search and optimization problems
- Local search algorithms
  - Hill climbing
  - Simulated annealing
  - Local beam search
  - Genetic algorithms

Local Search

- The search algorithms studied so far keep one or more paths in memory and record 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.
- However, for many problems all that matters is the goal state itself, not the path to reach it.
- For example, for the n-queen problem what matters is the final configuration of queens, not the order in which they are added.



Local Search

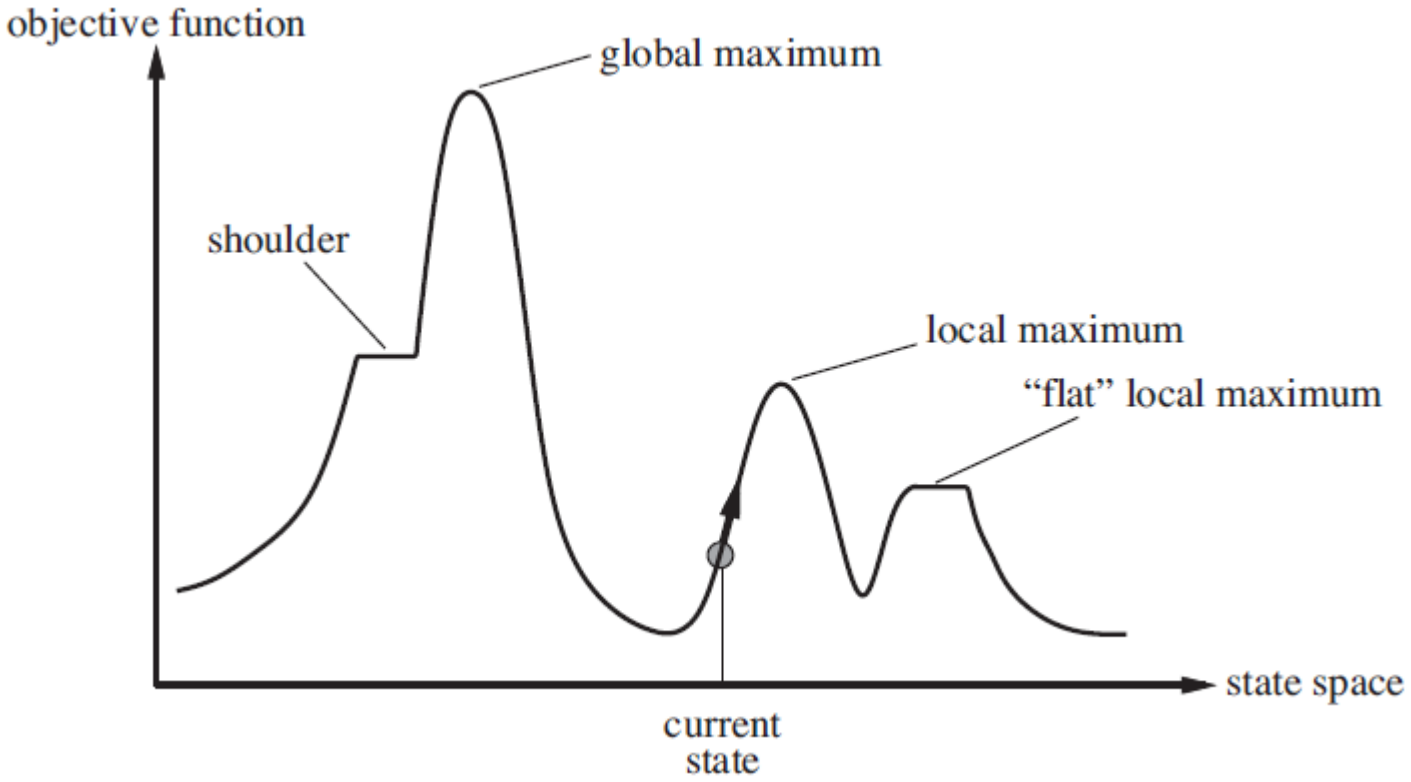
- Local search algorithms are suitable for problems where the path to the goal does not matter.
- **Local search** algorithms operate using a single **current node** (rather than multiple paths) and generally move only to the neighbors of that node.
- The paths followed by the search are not retained.

- Local search algorithms are useful for solving pure **optimization problems** (find the best state according to an **objective function)**.
- On the other hand, maximizing/minimizing an objective function cannot fit using Chapter 3 search algorithms as there is no goal test.

Advantages of Local Search Algorithms

- They use very little memory—usually a constant amount
- They can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.

State Space Landscape



Hill-climbing Search

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h.

Hill-climbing Search

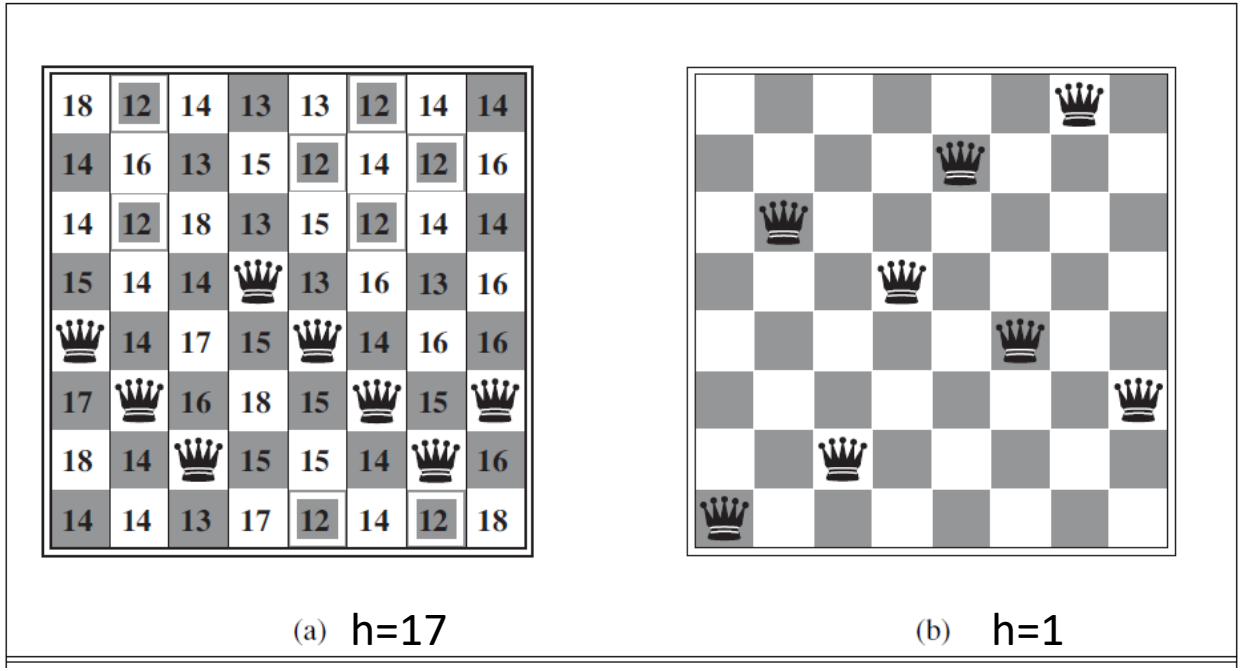
- It continually moves in the direction of **increasing value** until it reaches a “peak” where no **neighbor** has a higher value.

- The algorithm does not maintain a search tree, for the current node it only saves the state and the value of the objective function.

- Hill climbing is sometimes called **greedy local search** because it grabs a good neighbor state without thinking ahead about where to go next.

Example: n-queen problem

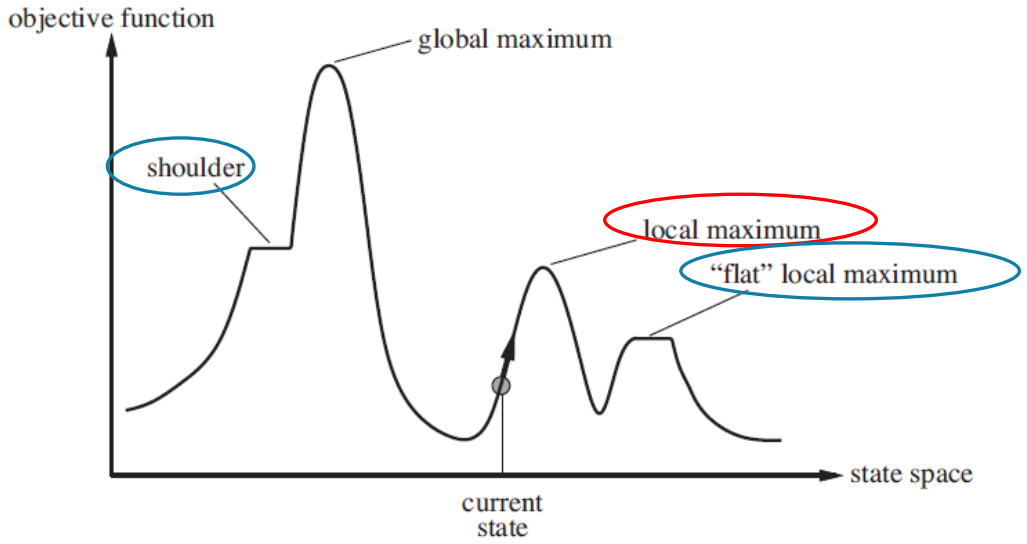
- The heuristic cost function h is the number of pairs of queens that are attacking each other, either directly or indirectly.



Hill-climbing Search

- Hill climbing gets stuck due to:
  - Local maxima/minima

A peak that is higher than each of its neighboring states, but lower than the global maximum.



  - Plateau (shoulder or flat local maxima/minima)

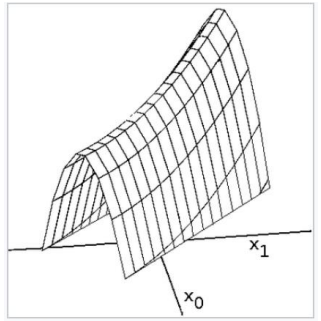
A plateau is a flat area of the state-space landscape.

Hill-climbing Search

Ridges

- Ridges result in a sequence of local maxima

that is very difficult for greedy algorithms to navigate.

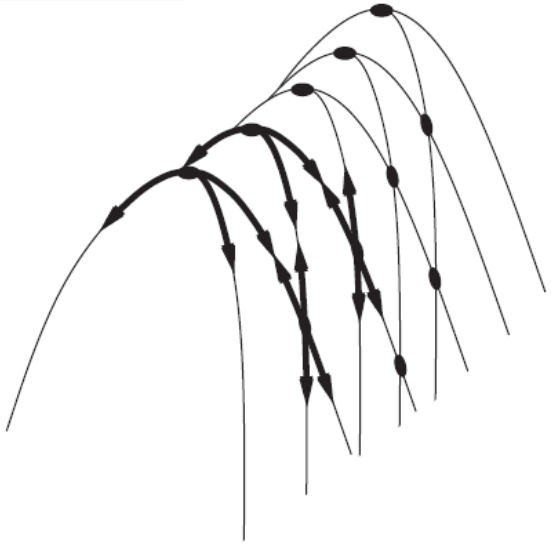


A ridge

- The grid of states is a sequence of local maxima that are not directly

connected to each other.

- From each local maximum, all the available actions point downhill.



Hill climbing search-example

- Starting from a randomly generated 8-queens state, hill climbing gets stuck 86% of the time, solving only 14% of problem instances.

- It works quickly, taking just 4 steps on average when it succeeds and 3 when it gets stuck.

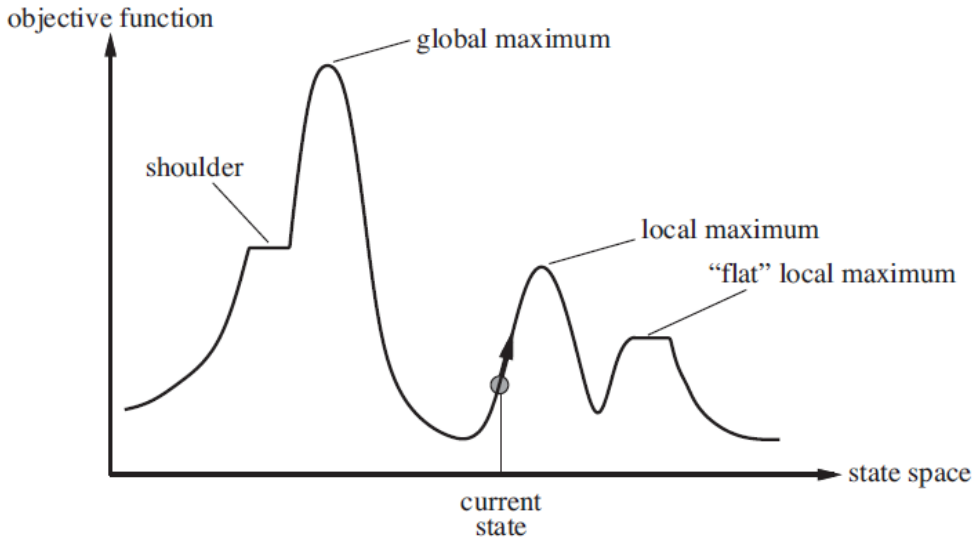
- These results are not bad for a state space with $8^8 \approx 17$ million states.

Hill Climbing Search

- May allow sideway move, this could help if plateau is really a shoulder.

- But what if there is no uphill movements?
  - Put a limit on the number of consecutive sideways moves allowed.

- This raises the percentage of n-queen problem instances solved by hill climbing from 14% to 94%.



- Success comes at a cost: After allowing sideway moves, the algorithm has on average roughly 21 steps for each successful instance and 64 for each failure.

Variants of Hill Climbing Search

- Stochastic hill climbing
  - chooses at random from among uphill moves
  - converges more slowly, but finds better solutions in some landscapes
- First-choice hill climbing
  - generate successors randomly until one is better than the current
  - good when a state has many successors
- Random-restart hill climbing
  - conducts a series of hill climbing searches from randomly generated initial states, stops when a goal is found
  - It's complete with probability approaching 1

Simulated annealing

- A hill-climbing algorithm that *never* makes "downhill" moves toward states with lower value (or higher cost) is incomplete, because it can get stuck on a local maximum.

- In contrast, a purely random walk that is, moving to a successor chosen uniformly at random is complete but extremely inefficient.

What about combining both completeness and efficiency?

Simulated annealing algorithm combines both.

Simulated annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow next.\text{VALUE} - current.\text{VALUE}$
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Picks a random move

Always accept the move if is better.

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of the temperature T as a function of time.

Simulated annealing

- **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.

- The idea of simulated annealing is to escape local maxima by allowing some “bad" moves but

gradually decrease their size and frequency.

- If the move improves the situation, it is always accepted. Otherwise, the algorithm accepts the move with
- probability $e^{\frac{\Delta E}{T}}$.
- The probability decreases exponentially with the “badness” of the move (ΔE).
- The probability also decreases as the “temperature” T goes down. Why?

  “Bad” moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases.

Simulated Annealing

- If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1.

- Widely used in VLSI layout, airline scheduling, etc.

Local beam search

- Keep track of k states rather than just one
  - Start with k randomly generated states.
  - At each iteration, all the successors of all k states are generated.
  - If any one is a goal state, stop; else select the k best successors from the complete list and repeat.

- Is it the same as running k random-restart searches?
  - No, in a random-restart search, each search process runs independently of the others.
  - In contrast, in local beam search, useful information is passed among the k parallel search threads.

Local beam search

- Local beam search can suffer from a lack of diversity among the k state as they can quickly become concentrated in a small region of the state space.

- **Stochastic beam search** chooses k successors at random, with the probability of choosing a given successor being an increasing function of its value.

Genetic algorithms

- A **genetic algorithm** (or **GA**) is a variant of stochastic beam search in which successor states are generated by combining *two* parent states rather than by modifying a single state.
- Like beam searches, GAs begin with a set of k randomly generated states, called the **population**.
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s or digits).
- Evaluation function (fitness function) has higher values for better states.
- The probability of being chosen for reproducing is directly proportional to the fitness score.
- Produce the next generation of states by selection, crossover, and mutation.

Genetic algorithms: n-queens example

Crossover points

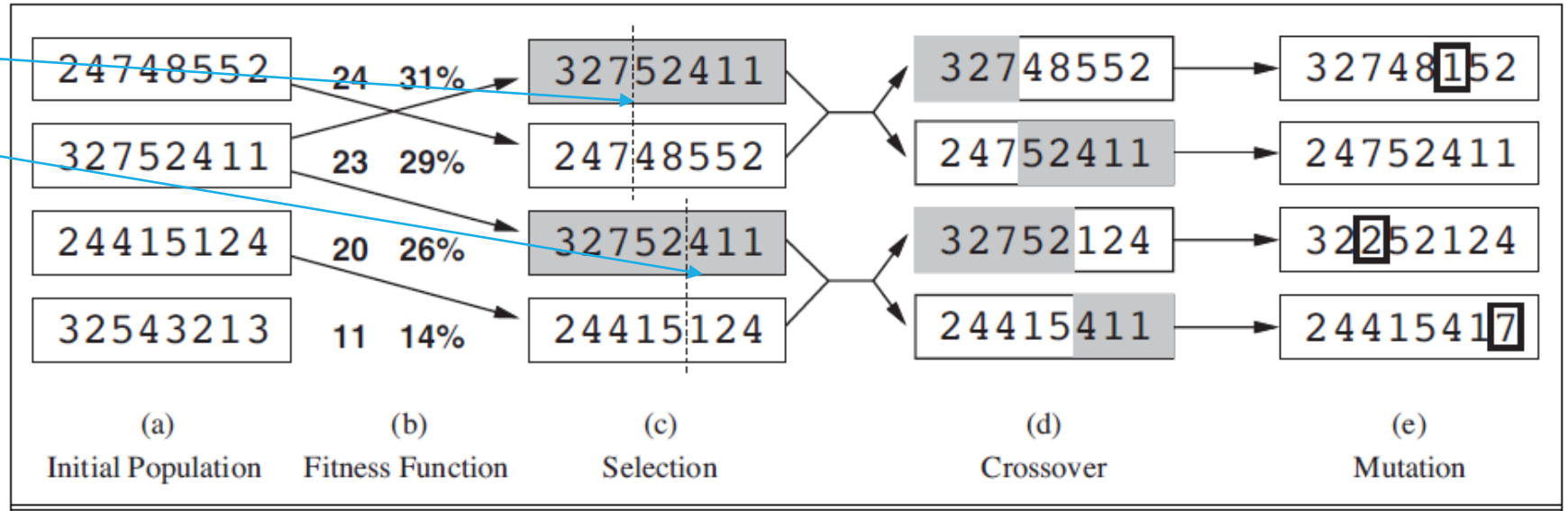


Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Genetic algorithms

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Figure 4.8 A genetic algorithm. The algorithm is the same as the one diagrammed in Figure 4.6, with one variation: in this more popular version, each mating of two parents produces only one offspring, not two.

Genetic Algorithms

- Like stochastic beam search, genetic algorithms combine an uphill tendency with random exploration.

- The population is generally quite diverse early on in the process, so crossover (like simulated annealing) frequently takes large steps in the state space early in the search process and smaller steps later on when most individuals are quite similar.

- Genetic algorithms are used in optimization problems such as circuit layout and job-shop scheduling.

Section 4.1 from Chapter 4 of the textbook.