

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	👑	13	16	13	16
👑	14	17	15	👑	14	16	16
17	👑	16	18	15	👑	15	👑
18	14	👑	15	15	14	👑	16
14	14	13	17	12	14	12	18

Local Search

Steepest Ascent Hill Climbing, Stochastic Hill Climbing, Simulated Annealing

Local Search Algorithm and Optimization Problem

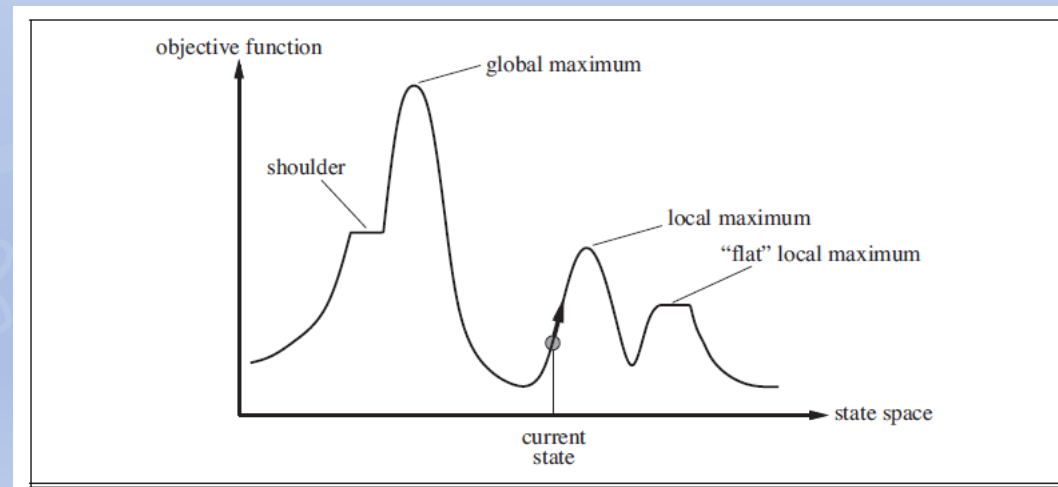
- Uninformed and informed search are designed to explore search spaces systematically.
- This systematicity is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each point along the path.
- When a goal is found, the path to that goal also constitutes a solution to the problem.
- In many problems the path to the goal is irrelevant.
- 8-queens problem, integrated-circuit design, factory-floor layout, job-shop scheduling, automatic programming, telecommunications network optimization, vehicle routing, and portfolio management.

Local Search Algorithm and Optimization Problem

- Local search algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbors of that node.
- The paths followed by the search are not retained.
- Local search algorithms are not systematic, they have two key advantages: (1) they use very little memory—usually a constant amount; and (2) they can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.
- local search algorithms are useful for solving pure optimization problems, in which the aim is to find the best state according to an objective function.

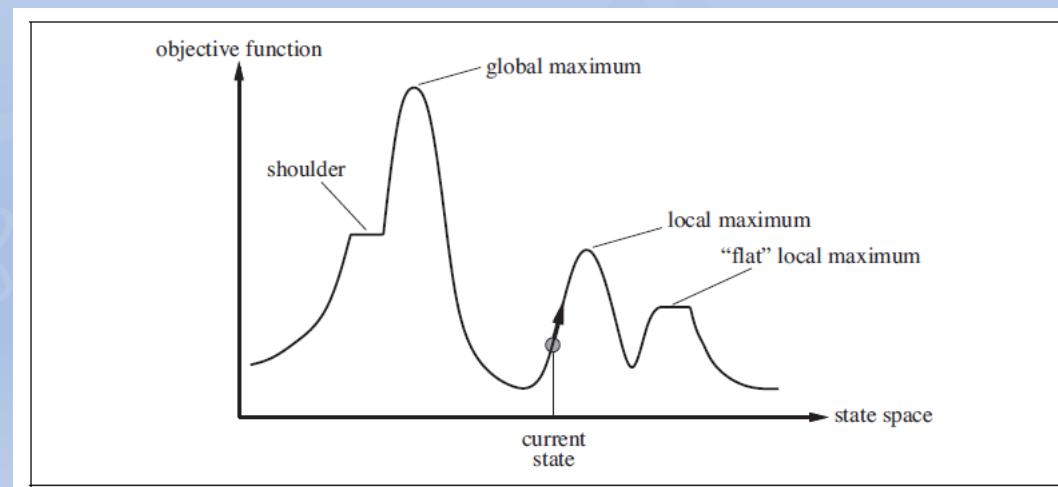
Local Search Algorithm and Optimization Problem

- To understand local search, it is useful to consider the state-space landscape.
- A landscape has both “location” (defined by the state) and “elevation” (defined by the value of the heuristic cost function or objective function).
- If elevation corresponds to cost, then the aim is to find the lowest valley—a global minimum.



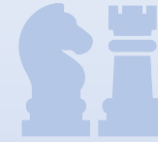
Local Search Algorithm and Optimization Problem

- If elevation corresponds to an objective function, then the aim is to find the highest peak—a global maximum.
- Local search algorithms explore this landscape.
- A complete local search algorithm always finds a goal if one exists; an optimal algorithm always finds a global minimum/maximum.



by Tumpa Banerjee

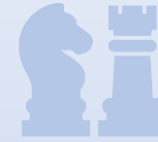
Hill-climbing search





- It is simply a loop that continually moves in the direction of increasing value—that is, uphill.
- It terminates when it reaches a “peak” where no neighbor has a higher value.
- The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function.
- Hill climbing does not look ahead beyond the immediate neighbors of the current state.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum  
  current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)  
  loop do  
    neighbor  $\leftarrow$  a highest-valued successor of current  
    if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE  
    current  $\leftarrow$  neighbor
```

Hill Climbing Search



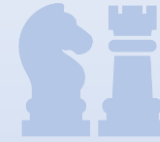
- Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next.
- hill climbing often gets stuck for the following reasons:
- Local maxima: a local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go.
- Ridges: Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.
- Plateaux: a plateau is a flat area of the state-space landscape. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible. A hill-climbing search might get lost on the plateau.



Hill Climbing Search

- Starting from a randomly generated 8-queens state, steepest-ascent 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—not bad for a state space with $88 \approx 17$ million states.
- The algorithm (Steepest Ascent Hill Climbing) halts if it reaches a plateau where the best successor has the same value as the current state.

Hill Climbing Search



- A sideways move may be allowed in the hope that the plateau is really a shoulder.
- How many sideways move? should have limit!
- For example, 100 consecutive sideways moves in the 8-queens problem.
- This raises the percentage of problem instances solved by hill climbing from 14% to 94%.
- Success comes at a cost: the algorithm averages roughly 21 steps for each successful instance and 64 for each failure.

Hill Climbing Search



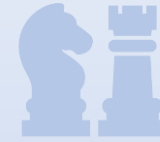
- **Stochastic hill climbing** chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move.
- This usually converges more slowly than steepest ascent, but in some state landscapes, it finds better solutions.
- **First-choice hill climbing** implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state.
- This is a good strategy when a state has many (e.g., thousands) of successors.

Hill Climbing Search



- **Random-restart hill climbing** adopts the well-known adage, “If at first you don’t succeed, try, try again.” It conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.

Simulated Annealing



- A hill-climbing algorithm that never makes “downhill” moves toward states with lower value (or higher cost) is guaranteed to be 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 from the set of successors—is complete but extremely inefficient.
- Therefore, it seems reasonable to try to combine hill climbing with a random walk in some way that yields both efficiency and completeness.
- Simulated annealing is such an algorithm.



Simulated Annealing Algorithm

```

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

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

Local Beam Search



- Keeping just one node in memory might seem to be an extreme reaction to the problem of memory limitations.
- The local beam search algorithm keeps track of k states rather than just one. It begins with k randomly generated states.
- At each step, all the successors of all k states are generated. If anyone is a goal, the algorithm halts. Otherwise, it selects the k best successors from the complete list and repeats.





Local Beam Search

- local beam search can suffer from a lack of diversity among the k states—they can quickly become concentrated in a small region of the state space, making the search little more than an expensive version of hill climbing.
- A variant called stochastic beam search, analogous to stochastic hill climbing, helps alleviate this problem.
- Instead of choosing the best k from the pool of candidate successors, stochastic beam search chooses k successors at random, with the probability of choosing a given successor being an increasing function of its value.
- Stochastic beam search bears some resemblance to the process of natural selection, whereby the “successors” (offspring) of a “state” (organism) populate the next generation according to its “value” (fitness).



Genetic algorithms

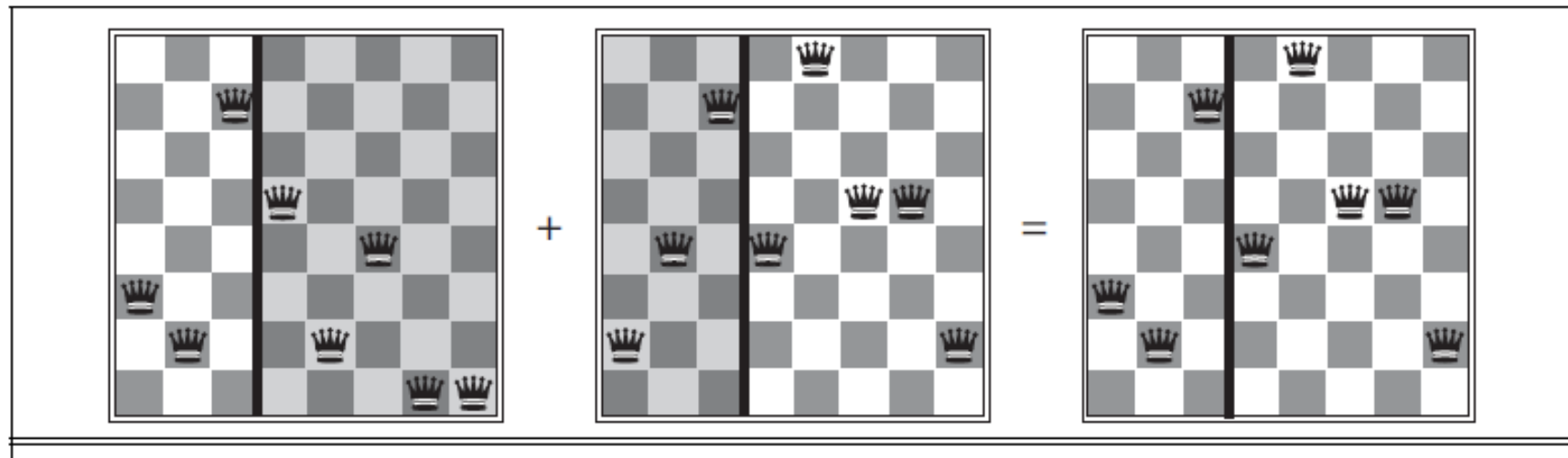
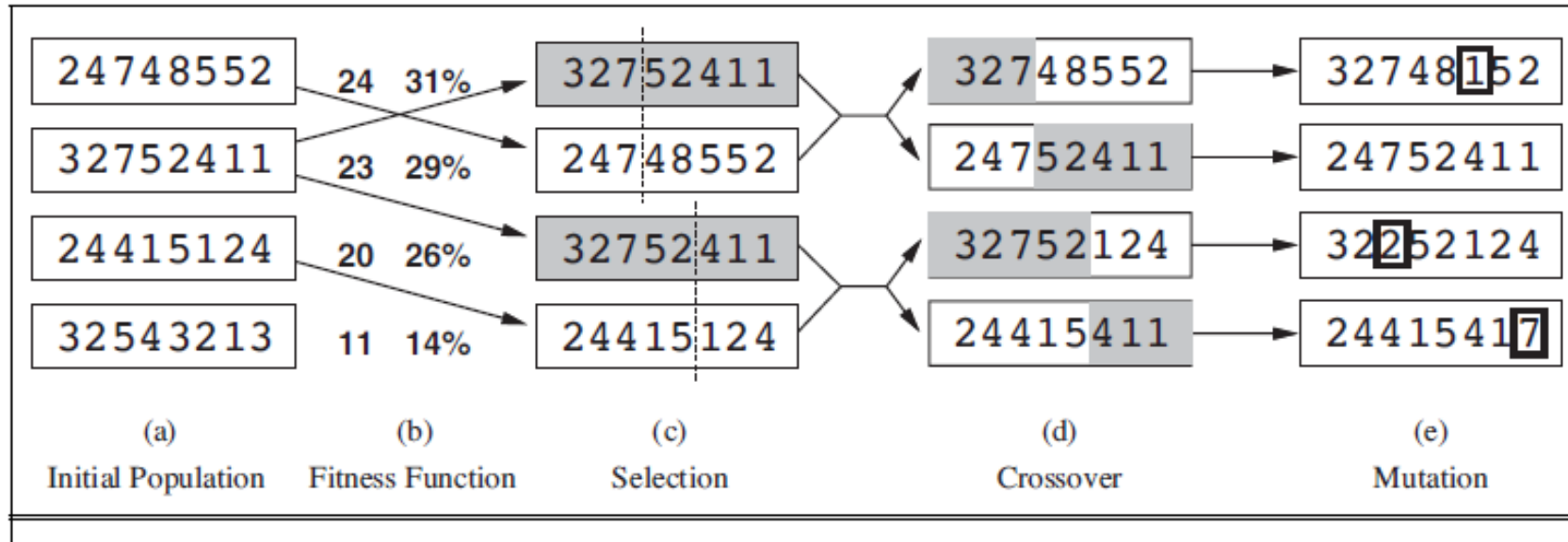
- A genetic algorithm (or GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state.
- The analogy to natural selection is the same as in stochastic beam search, except that now we are dealing with sexual rather than asexual reproduction.
- Like beam searches, GAs begin with a set of k randomly generated states, called the population.

Genetic Algorithm



- Each state, or individual, is represented as a string over a finite alphabet—most commonly, a string of 0s and 1s.
- each state is rated by the objective function, or (in GA terminology) the fitness function.
- A fitness function should return higher values for better states, so, for the 8-queens problem we use the number of nonattacking pairs of queens, which has a value of 28 for a solution.





Genetic Algorithm



- For each pair to be mated, a crossover point is chosen randomly from the positions in the string.
- As the population is 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.
- Each location is subject to random mutation with a small independent probability.

