Local Search Algorithms

- HILL CLIMBING
- SIMULATED ANNEALING SEARCH
- LOCAL BEAM SEARCH
- GENETIC ALGORITHM

CHAPTER FOUR FROM BOOK

Local Search Algorithms

- When a goal is found, the path to that goal also constitutes a solution to the problem.
- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution.
- 8-queens problem 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, network optimization and so on.
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
- Keep a single "current" state, try to improve it.

Local search algorithms

- Local search algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbors of that node.
- Paths followed by the search are not retained.
- Are not systematic,
- 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

- To understand local search, we find it 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**;
- if elevation corresponds to an objective function, then the aim is to find the highest peak—a **global maximum**.

- A complete local search algorithm always finds a goal if one exists; an optimal algorithm always finds a global minimum/maximum.

State Space Landscape

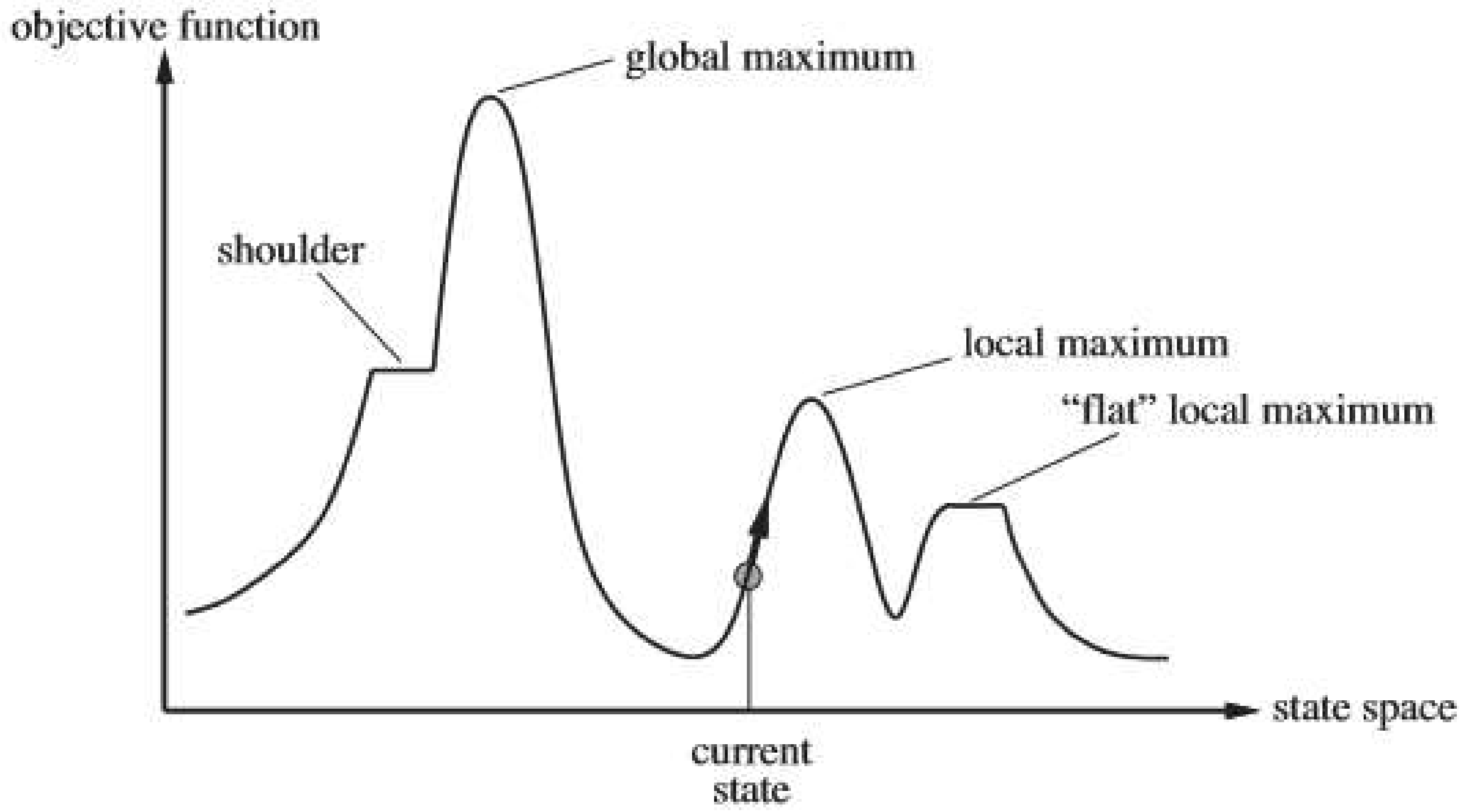


Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

Hill climbing search- Steepest-ascent version

- The hill-climbing search algorithm 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.
- No search tree, records the state and the value of the objective function.
- Does not look ahead beyond the immediate neighbors of the current state.
- This resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia

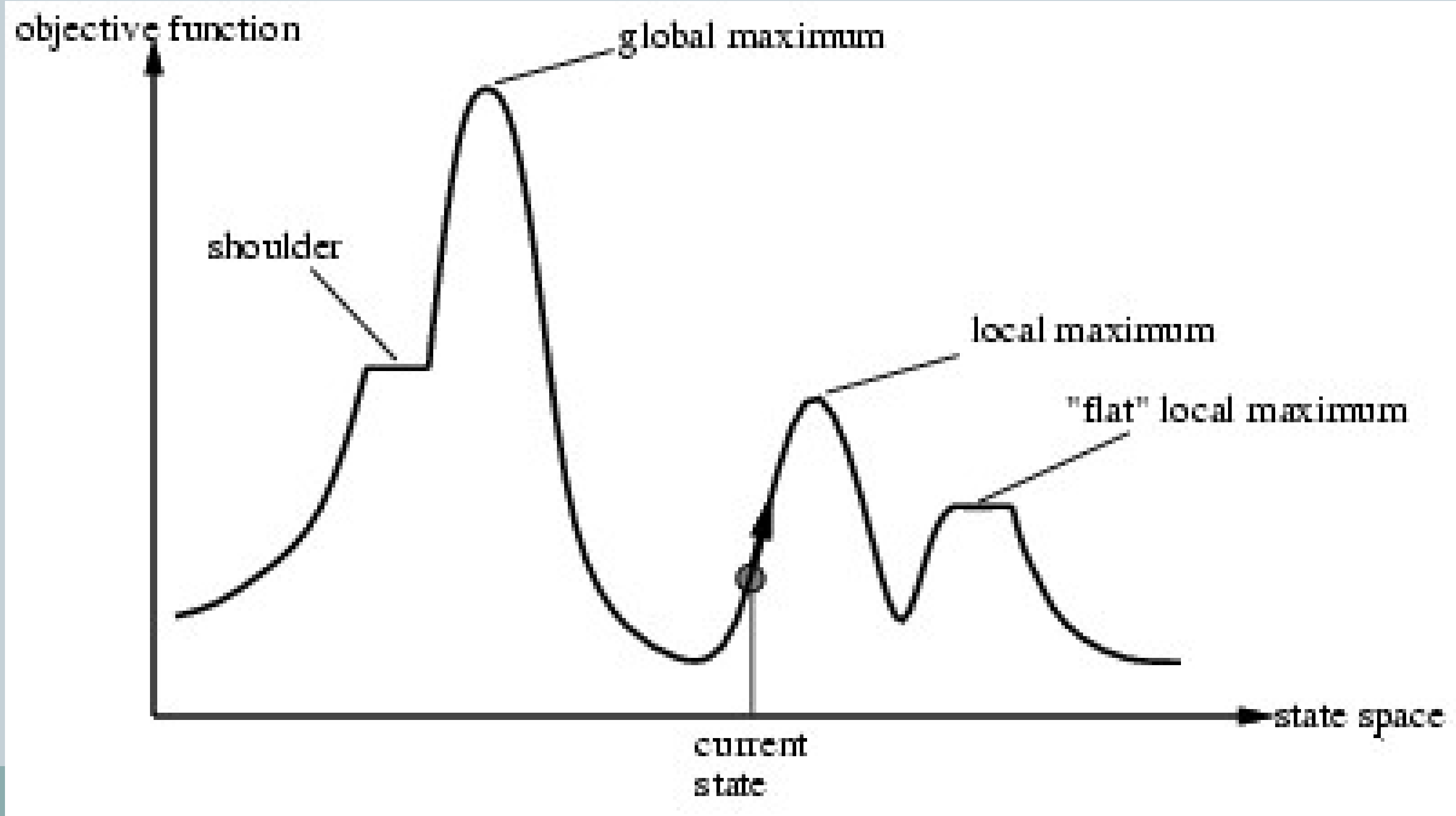
Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

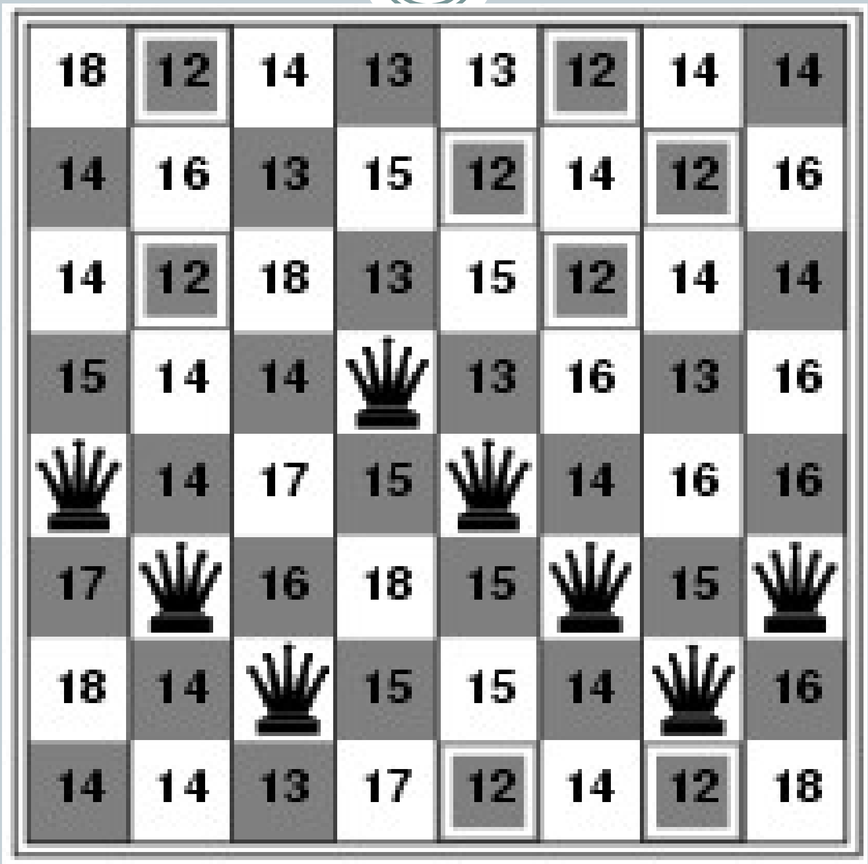
```
function HILL-CLIMBING(problem) returns a state that is a local maximum

  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

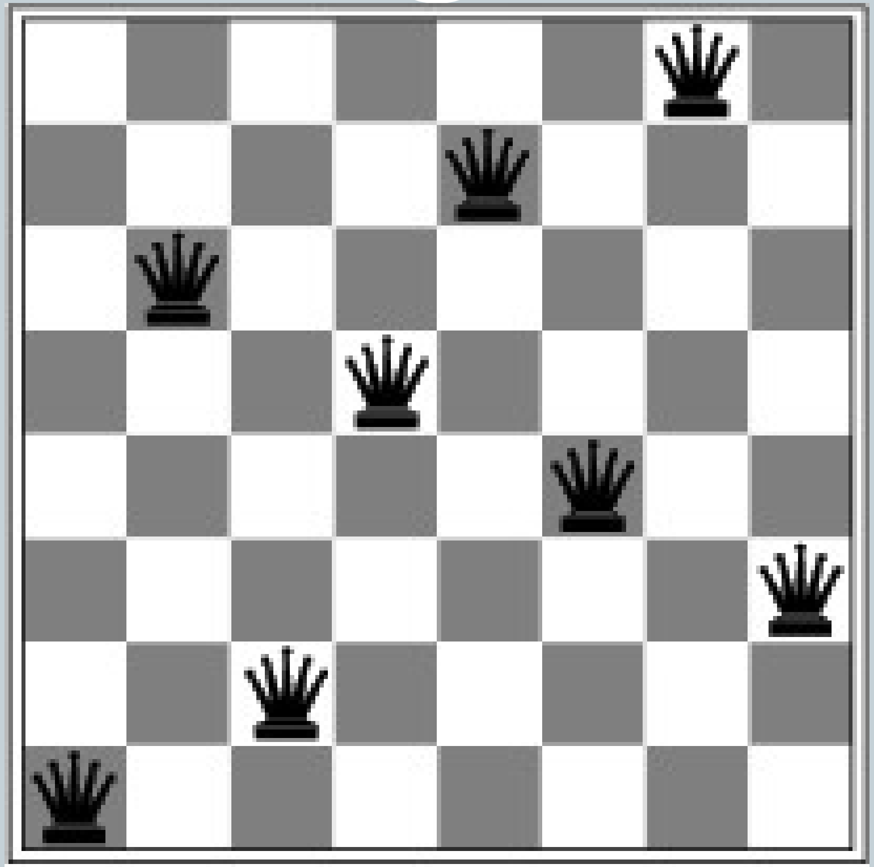
- Problem: depending on initial state, can get stuck in local maxima
- 

Hill-climbing search: 8-queens problem



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h* = *17* for the above state

Hill-climbing search: 8-queens problem



- A local minimum with $h = 1$

Hill climbing

- **Hill climbing** is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next.
- Greedy algorithms often perform quite well.
- Hill climbing often makes rapid progress toward a solution because itis usually quite easy to improve a bad state.
- For example, from the state (h= 17) it takes just five steps to reach the state (h=1) and is very nearly a solution.
- Unfortunately, hill climbing often gets stuck.

Problems of Hill climbing

- Local maxima is a peak that is higher than each of its neighboring states but lower than the global max.
- 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.
- Eg. every move of a single queen makes the situation worse.
- **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.

Sideways move – on plateau

- Might it not be a good idea to keep going—to allow a **sideways move** in the hope that the plateau is really a shoulder.
- The answer is usually yes, but we must take care.
- If we always allow sideways moves when there are no uphill moves, an infinite loop will occur whenever the algorithm reaches a flat local maximum that is not a shoulder.
- One common solution is to put a limit on the number of consecutive sideways moves allowed.

Variants of hill climbing

- **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.

Random-restart hill climbing

- The hill-climbing algorithms described so far are incomplete—they often fail to find a goal when one exists because they can get stuck on local maxima.
- **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,1 until a goal is found. It is trivially complete with probability approaching 1, because it will eventually generate a goal state as the initial state.

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their 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^(Δ E/T)
```

Properties of simulated annealing search

- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching.
- Widely used in VLSI layout, airline scheduling, etc
-

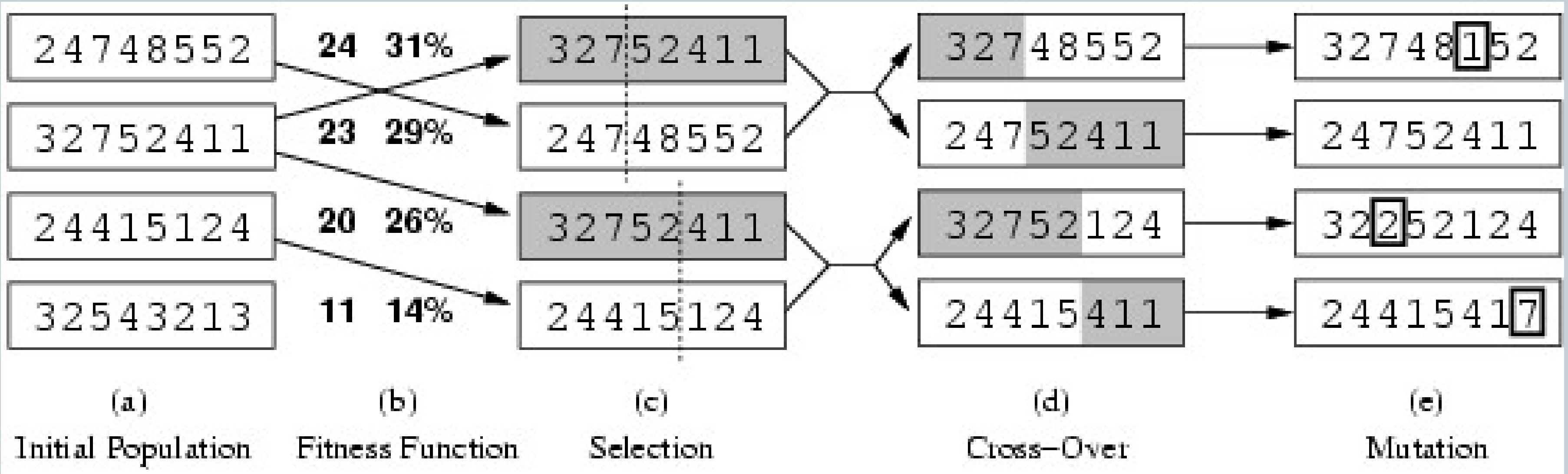
Local beam search

- 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.

Genetic algorithms

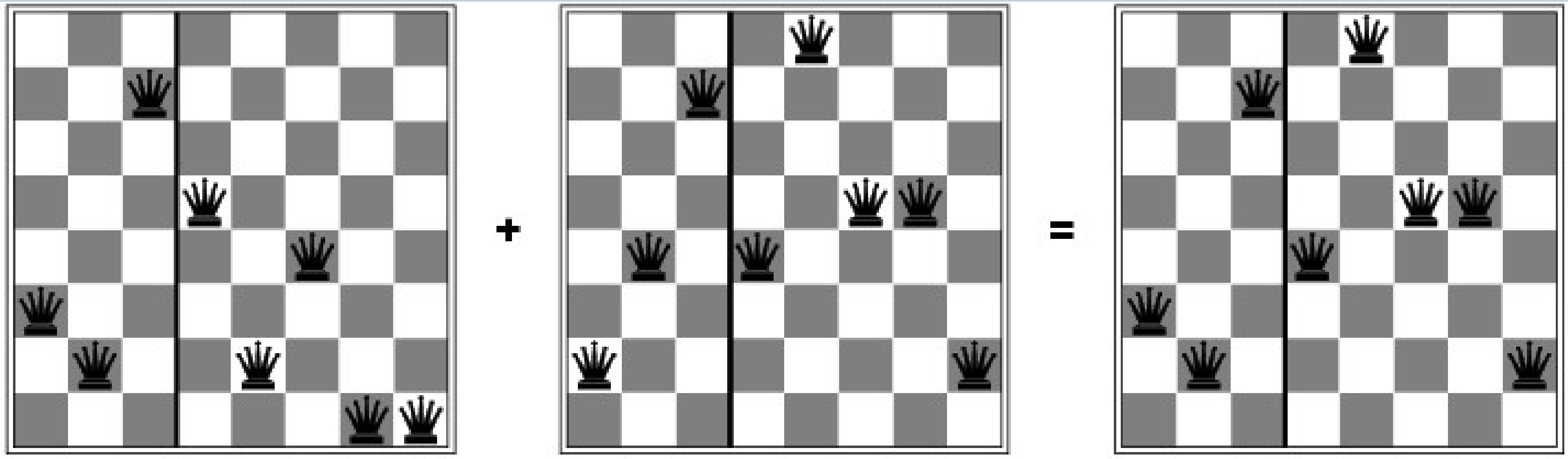
- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation.

Genetic algorithms



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithms



Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.

- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- Othello: human champions refuse to compete against computers, who are too good.
- Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.