Local Search and Optimization Problems

Dr. Sandareka Wickramanayake

Outline

- Hill Climbing
- Simulated Annealing
- Local Beam Search
- Genetic Algorithms

Reference

- Artificial Intelligence - A Modern Approach – Chapter 4 – Section 1

Search Types

- Backtracking state-space search
- **Local Search and Optimization**
- Constraint satisfaction search
- Adversarial search

Local Search and Optimization

- Previous searches: keep paths in memory and remember alternatives so search can backtrack. The solution is a path to a goal.
- Path may be irrelevant if only the final configuration is needed
  - 8-queens
  - IC design
  - Network optimization
  - Factory floor layout
  - Job shop scheduling
  - Automatic programming
  - Cop planning etc.

Local Search

- Use a single current state and move only to neighbours.
  - No track of the paths
  - No track of the set of states that have been reached
- Not systematic ➔ Might never explore a portion of the search space where the solution exists.

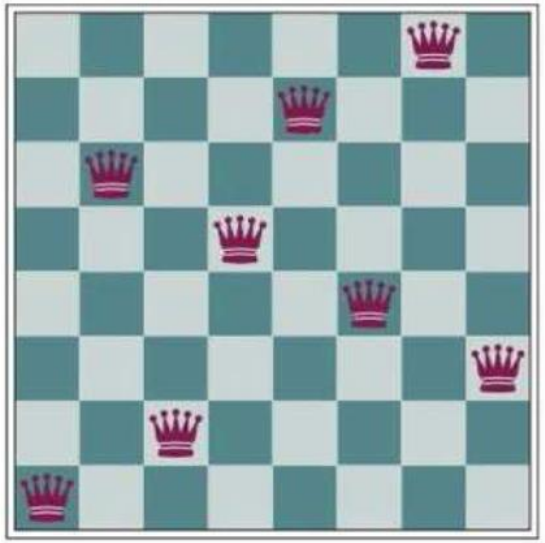
- Advantages:
  - Use little space
  - Can find reasonable solutions in large or infinite (continuous) state spaces for which the other algorithms are unsuitable.

Optimization

- Local search is often suitable for optimization problems. Search for the best state by optimizing an objective function.
- F(x) where often x is a vector of continuous or discrete values.
- Begin with a complete configuration.
- A successor of state S is S with a single element changed.
- Move from the current state to a successor state.
- Low memory requirements, because the search tree or graph is not maintained in memory (paths are not saved).

Examples – 8 Queens

- Find an arrangement of 8 queens on a chess board such that no two queens are attacking each other (A queen attacks any piece in the same row, column, or diagonal.).
- Start with some arrangement of the queens, one per column.
- X[j]: row of queen in column j
- Successors of a state: move one queen
- $F(x)$: # pairs attacking each other



Examples – Traveling Salesman Problem

- Visit each city exactly once.
- Start with some ordering of the cities.
- State representation – order of the cities visited.
- Successor state: a change to the current ordering
- $F(x)$: length of the route

Comparison to Tree Search Framework

- Chapter 3: start state is typically not a complete configuration.

Chapter 4: all states are

- Chapter 3: binary goal test;

Chapter 4: no binary goal test, unless one can define one in terms of the objective function for the problem (e.g., no attacking pairs in 8-queens).

- Heuristic function: an estimate of the distance to the nearest goal

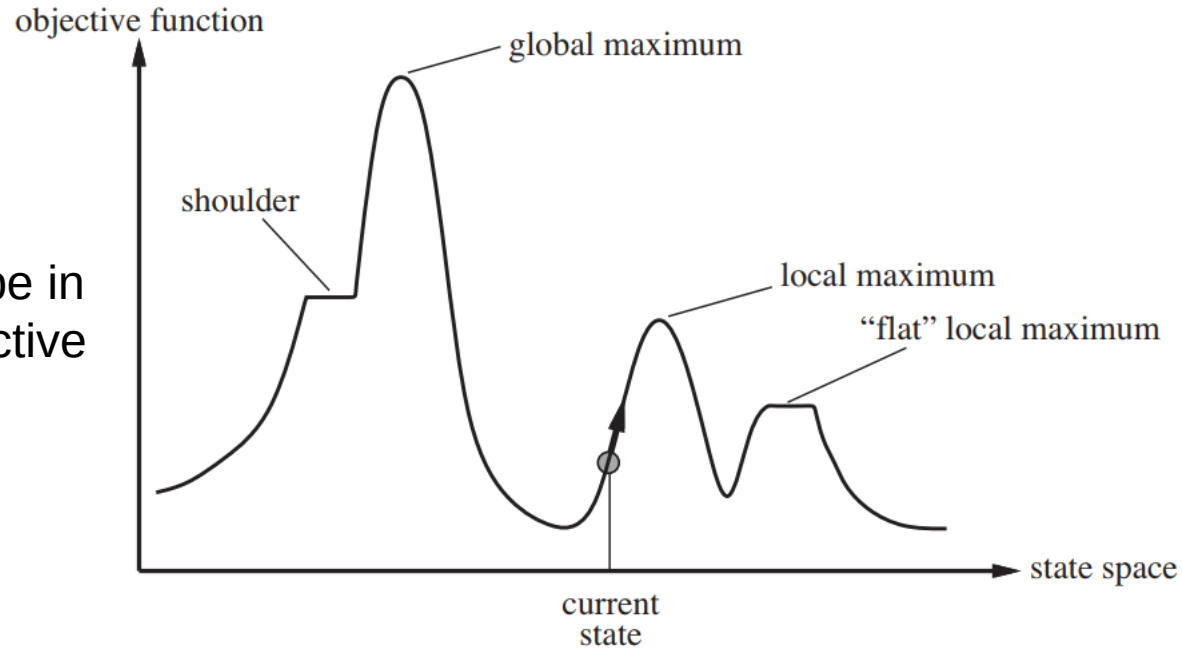
Objective function: preference/quality measure – how good is this state?

- Chapter 3: saving paths

Chapter 4: start with a complete configuration and make modifications to improve it

Visualization

- States are laid out in a landscape.
- Elevation corresponds to the objective function value.
- Move around the landscape to find the highest (or lowest) peak.
- Only keep track of the current states and immediate neighbors.



A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum.

Local Search Algorithms

- Strategies for choosing the state to visit next
  - Hill climbing
  - Simulated annealing
- Then, an extension to multiple current states:
  - Genetic algorithms

Hill Climbing (Greedy Local Search)

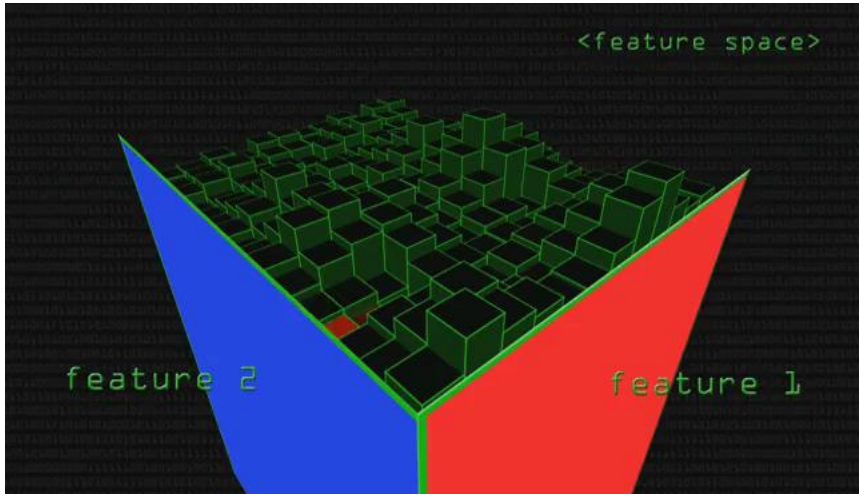
- Generate nearby successor states to the current state.
- Pick the best and replace the current state with that one.
- Loop

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*

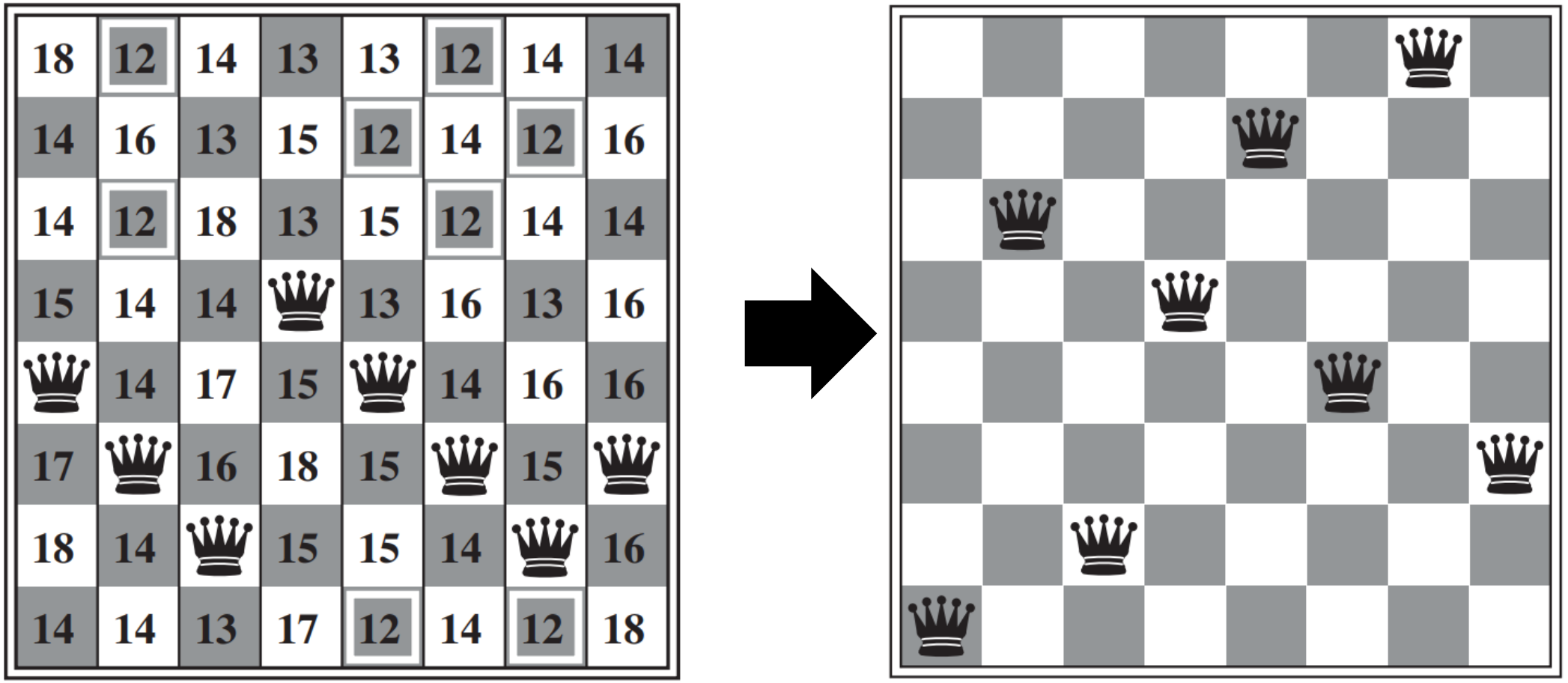
Hill Climbing (Greedy Local Search)

- Generate nearby successor states to the current state.
- Pick the best and replace the current state with that one.
- Loop



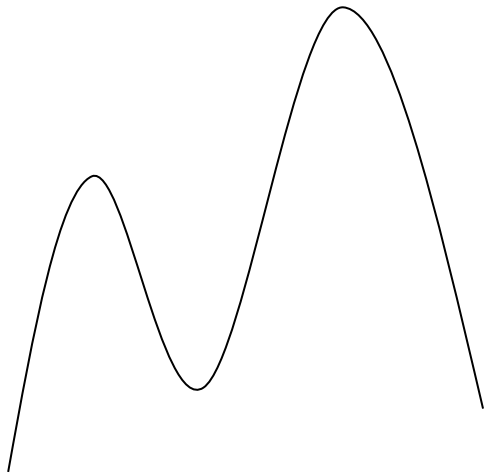
https://thumbs.gfycat.com/AlienatedImmaculateFlyingsquirrel-mobile.mp4

Hill Climbing Search Problems



Hill Climbing Search Problems

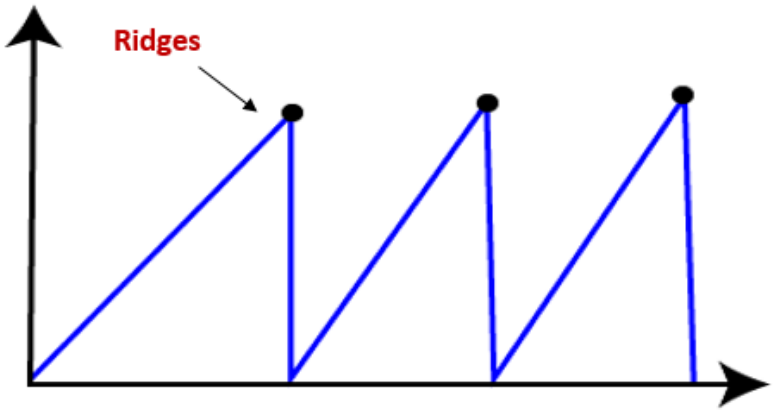
- Here, we assume maximization rather than minimization.
- Local maximum: A local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum.

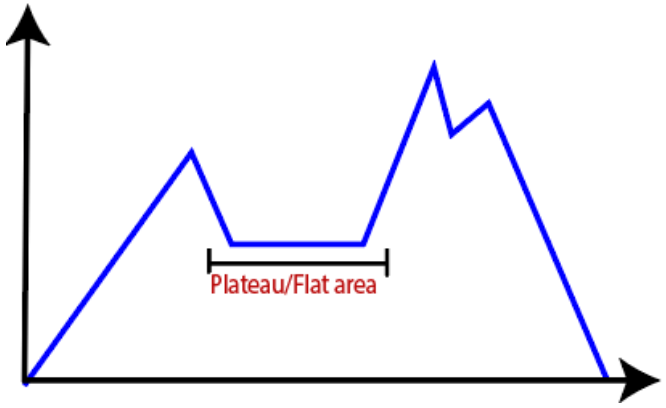


Local maximum

Hill Climbing Search Problems

- Ridges: It is a region that is higher than its neighbors but itself has a slope.
  - It is a special kind of local maximum.
  - Results in a sequence of local maxima that is very difficult for greedy algorithms to navigate.
- Plateau: the evaluation function is flat, resulting in a random walk.





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.

Variants of Hill Climbing

- **Random-restart hill climbing**
  - Start different hill-climbing searches from random starting positions stopping when a goal is found.
  - Save the best result from any search so far.
  - If all states have an equal probability of being generated, it is complete with a probability approaching 1 (a goal state will eventually be generated).
  - Finding an optimal solution becomes the question of a sufficient number of restarts.
  - Surprisingly effective, if there aren’t too many local maxima or plateau.

Simulated Annealing

- A hill-climbing algorithm that never makes “downhill” moves is always vulnerable to getting stuck in a local maximum.
- A purely random walk that moves to a successor state without concern for the value will eventually discover the global maximum but will be extremely inefficient.
- Combine the ideas: add some randomness to hill-climbing to allow the possibility of escape from a local optimum.

- Simulated annealing wanders around during the early parts of the search, hopefully toward a good general region of the state space
- Toward the end, the algorithm does a more focused search, making a few bad moves.

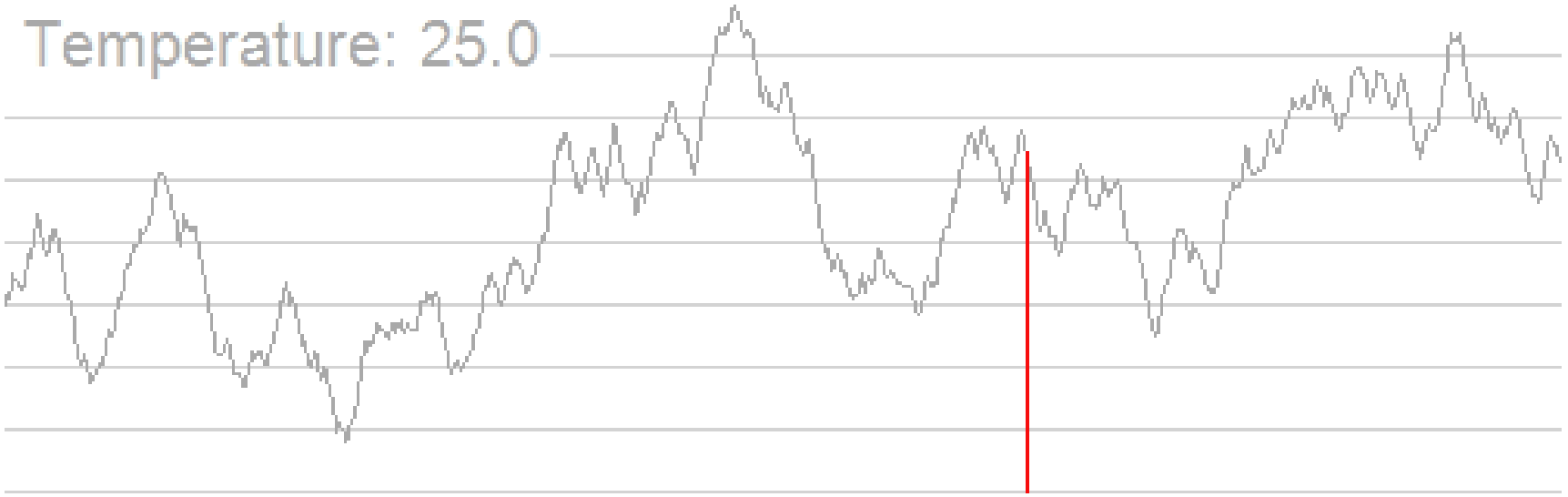
Simulated Annealing

- Based on a metallurgical metaphor.
- Annealing: harden metals and glass by heating them to a high temperature and then gradually cooling them.
- Start with a temperature set very high and slowly reduce it.

- Run hill-climbing with the twist that you can occasionally replace the current state with a worse state based on the current temperature and how much worse the new state is.
- At the start, make lots of moves and then gradually slow down.

Simulated Annealing

- Generate a random new neighbor from the current state.
- If it's better take it.
- If it's worse, then take it with some probability proportional to the temperature and the delta between the new and old states.



https://upload.wikimedia.org/wikipedia/commons/d/d5/Hill_Climbing_with_Simulated_Annealing.gif

Simulated Annealing

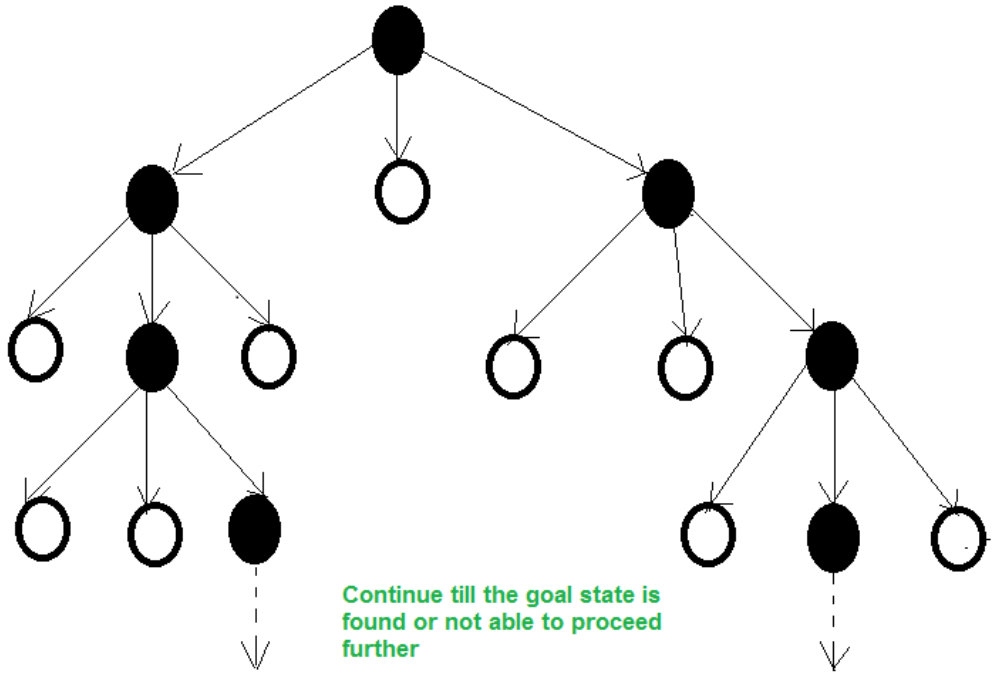
```
function Simulated-Annealing(start, schedule)
     current ← start
     for t ← 1 to ∞ do
          T ← schedule[t]
          if T=0 then return current
          next ← a randomly selected successor of the current
          ΔE ← Value[next] – Value[current]
          if  ΔE > 0 then current ← next
          else current  ← next only with probability e^(ΔE/T)
```

Simulated Annealing

- Probability of a move decreases with the amount ΔE by which the evaluation is worsened.
- A second parameter T is also used to determine the probability: high T allows worse moves, T close to zero results in few or no bad moves.
- Schedule input determines the value of T as a function of the completed cycles.

Local Beam Search

- Keep track of k states rather than just one, as in hill climbing
- Begins with k randomly generated states
- At each step, all successors of all k states are generated
- If anyone is a goal, algorithm halts
- Otherwise, selects the best k successors from the complete list, and repeats



Local Beam Search

- Successors can become concentrated in a small part of state space
- Stochastic beam search: choose k successors with the probability proportional to the successor's value.
  - Increase diversity

Genetic Algorithms

- Local beam search, but…
  - 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 = better.
- Produce the next generation of states by selection, crossover, and mutation.

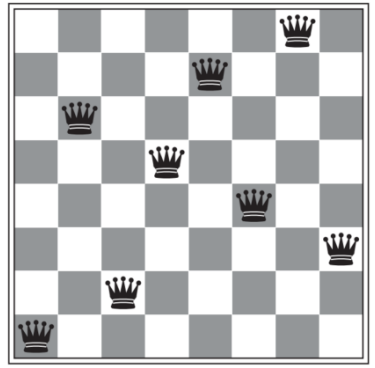
Genetic Algorithms

- Representation of individuals
  - Classic approach: an individual is a string over a finite alphabet, with each element in the string called a gene
  - Usually, binary instead of AGTC as in real DNA
- Mixing number
- Selection strategy
  - Random
  - Selection probability proportional to fitness
  - Selection is made with replacement to make a very fit individual reproduce several times

Genetic Algorithms

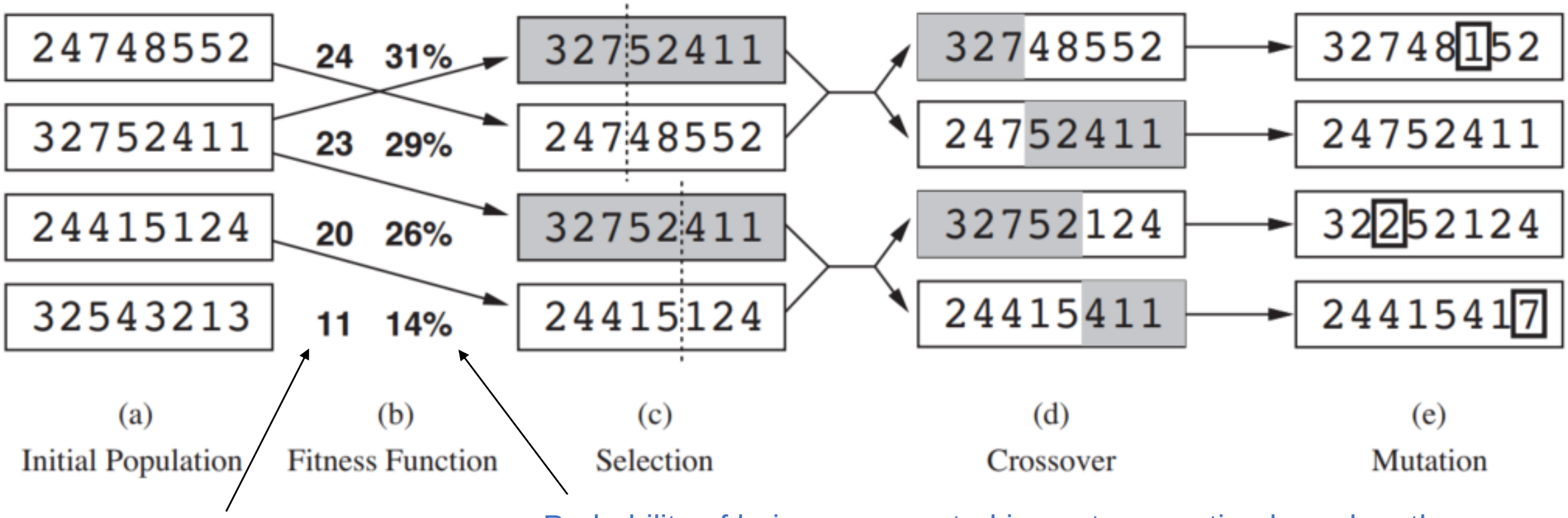
- Recombination procedure
  - Random pairing of selected individuals
  - Random selection of cross-over points
- Mutation rate – How often offspring have random mutations to their representation.
- The makeup of next generation
  - Forms with the newly generated offspring
  - Elitism - Include a few top-scoring parents from the previous generation in addition to newly formed offspring.
  - Culling – Discard individuals below a given threshold.

Example – N-Queens



[16257483]

- Put n queens on an n × n board with no two queens on the same row, column, or diagonal

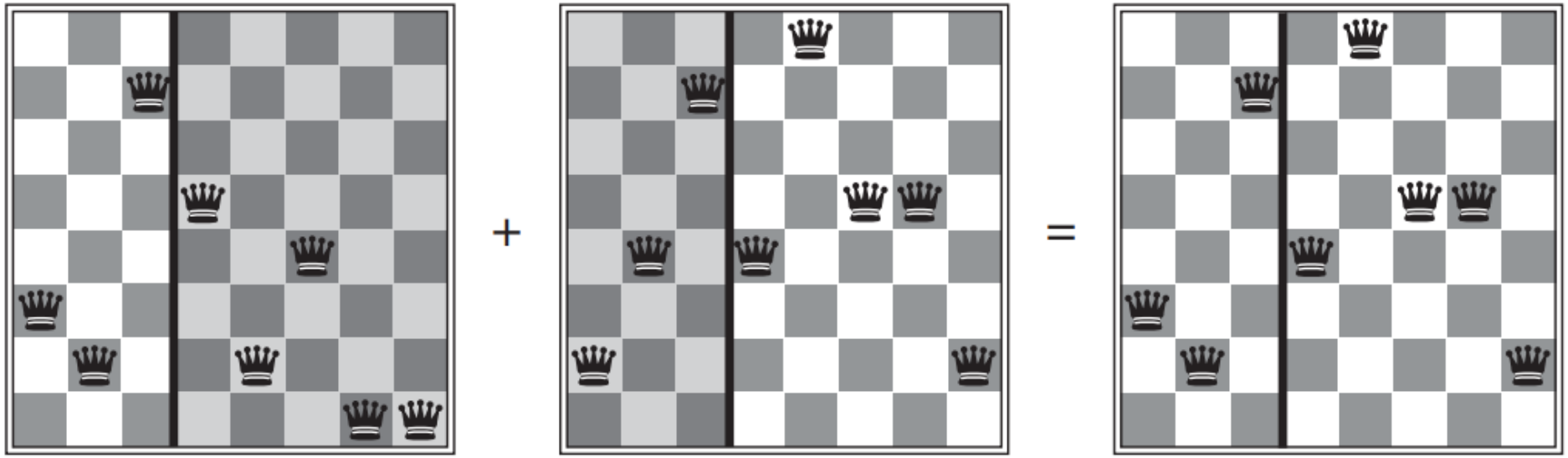


fitness: #non-attacking queens

Probability of being regenerated in next generation based on the fitness value.

Example – N-Queens

[32752411] + [24748552] = [32748552]



- Has the effect of “jumping” to a completely different new part of the search space (quite non-local)

Genetic Algorithms

```
function GA (pop, fitness-fn)
   Repeat
      new-pop = {}
      for i from 1 to size(pop):
         x = rand-sel(pop,fitness-fn)
         y = rand-sel(pop,fitness-fn)
         child = REPRODUCE(x,y)
         if (small rand prob): child ←
    mutate(child)
         add child to new-pop
      pop = new-pop
   until an indiv is fit enough, or out of
    time
   return best indiv in pop, according to
    fitness-fn
```

```
function REPRODUCE(x,y)
   n = len(x)
   c = random num from 1
to n
   return  APPEND
(substr(x,1,c),substr(y,
c+1,n)
```

Comments on Genetic Algorithms

- Genetic Algorithm is a variant of “stochastic beam search”.
- Positive points
  - Random exploration can find solutions that local search can’t (via crossover primarily).
  - Appealing connection to human evolution.
- Negative points
  - Many “tunable” parameters
  - Difficult to replicate performance from one problem to another
  - Lack of good empirical studies comparing to simpler methods
  - Useful on some (small?) problems, but no convincing evidence that GAs are generally better than hill-climbing w/random restarts.