Beyond Classical Search

"Artificial Intelligence: A Modern Approach", Chapter 4

Outline

- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms
- Searching in more complex environments
  - Non-deterministic environments
  - Partially observable environments
  - Unknown environments

Sample problems for local & systematic search

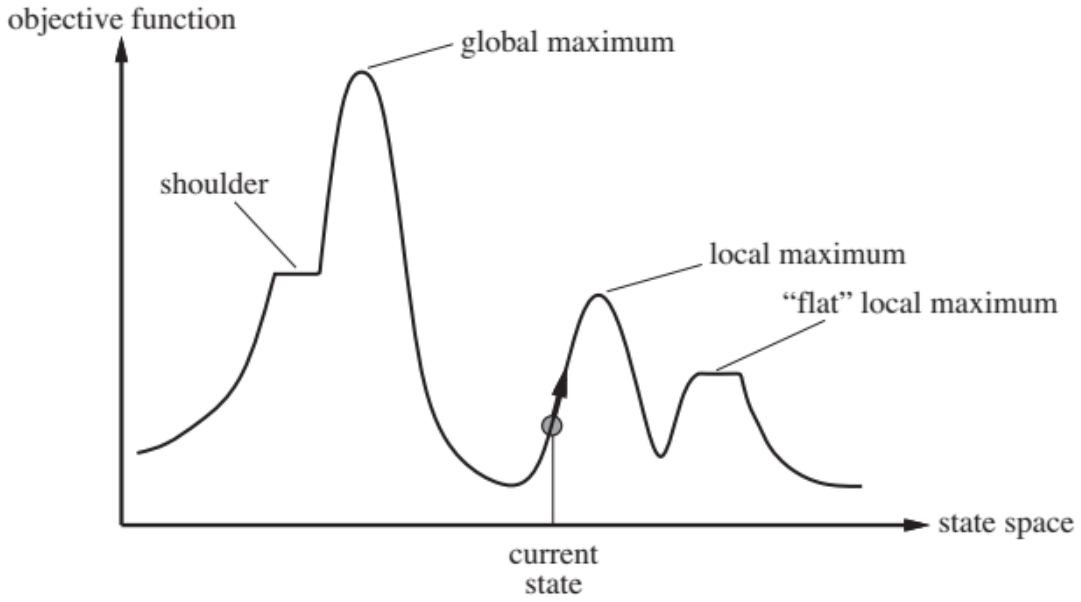
- Path to goal is important
  - Theorem proving
  - Route finding
  - 8-Puzzle
  - Chess
- Goal state itself is important
  - 8 Queens
  - TSP
  - VLSI Layout
  - Job-Shop Scheduling
  - Automatic program generation

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.
- they have two key advantages:
  - They use very little memory
    - usually a constant amount
  - They can often find reasonable solutions in large or infinite (continuous) state spaces
- local search algorithms are useful for solving pure optimization problems

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).
  - Local search algorithms explore the landscape
  - Solution: A state with the optimal value of the objective function

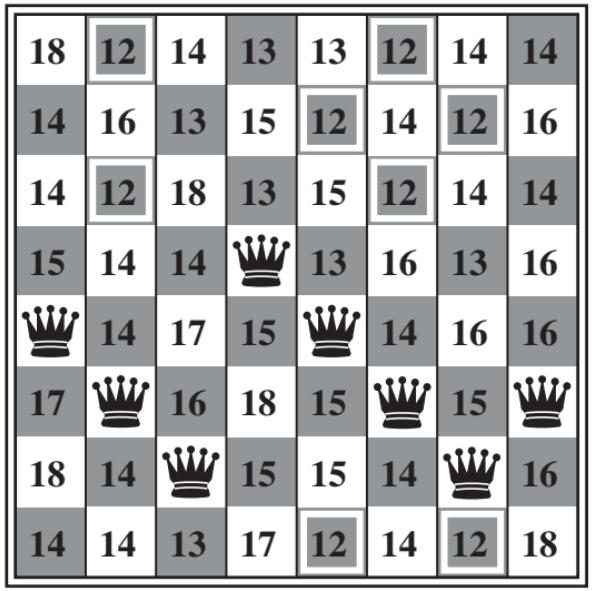


Local search

- A complete local search algorithm always finds a goal if one exists.
- An optimal algorithm always finds a global minimum/maximum.
- Example: n-queens:
  - Put n queens on an n × n board with no two queens on the same row, column, or diagonal
  - What is state-space?
  - What is objective function?

Local search: 8-queens problem

- States: 8 queens on the board, one per column ($8^8 \approx 17$ million)
- Successors(s): all states resulted from s by moving a single queen to another square of the same column (8 × 7 = 56)
- Cost function h(s): number of queen pairs that are attacking each other
- Global minimum: h(s) = 0



h = 17

Hill-climbing search

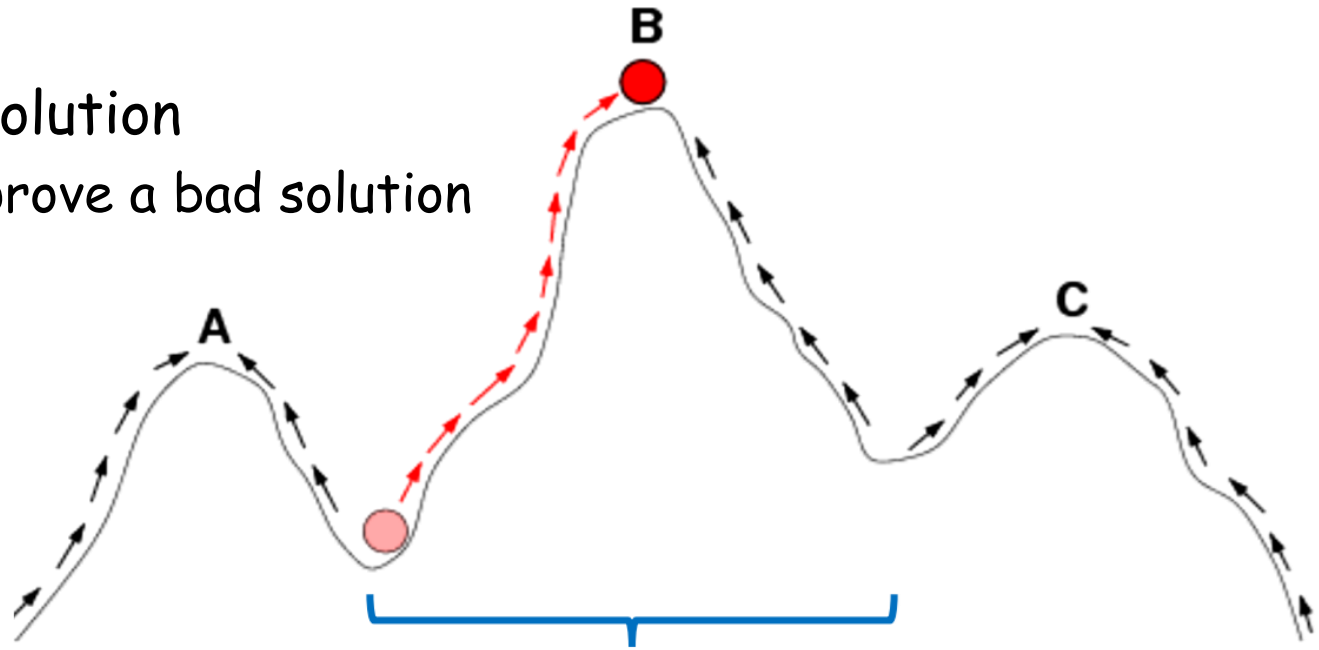
- Node only contains the state and the value of objective function in that state (not path)
- Search strategy
  - Steepest ascent (or descent) among immediate neighbors until reaching a peak

```
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
```

Current node is replaced by the best successor
(if it is better than current node)

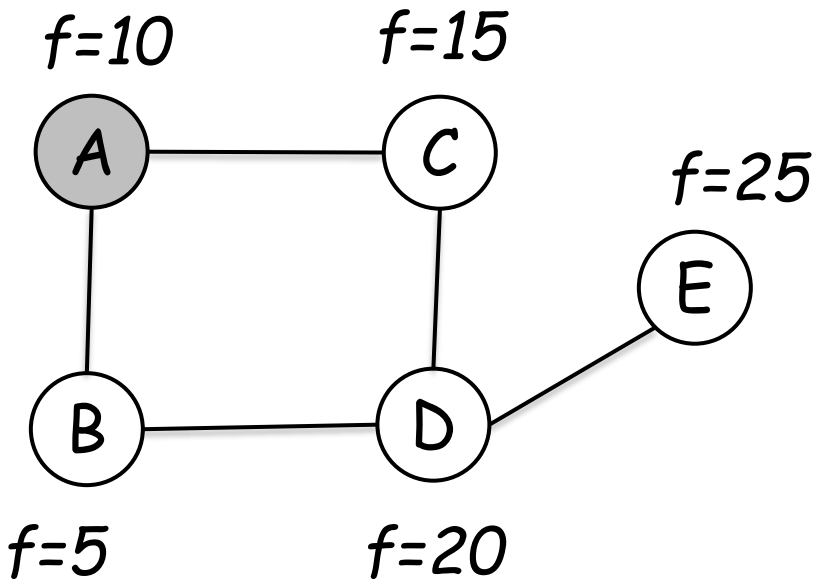
Hill-climbing search

- Hill-climbing search is greedy
- Greedy local search
  - Considering only one step ahead and select the best successor state (steepest ascent)
  - Rapid progress toward a solution
    - Usually quite easy to improve a bad solution



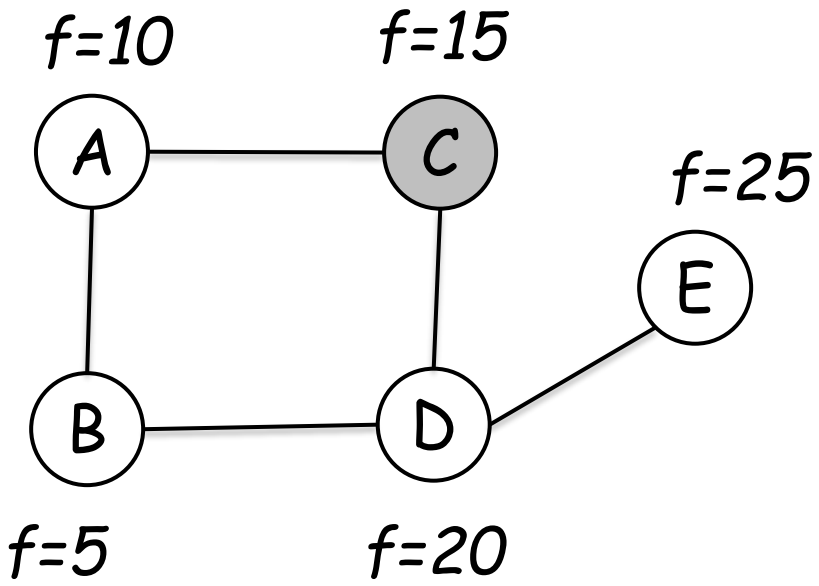
Optimal when starting in one of these states

Hill-climbing search: Example



current = A

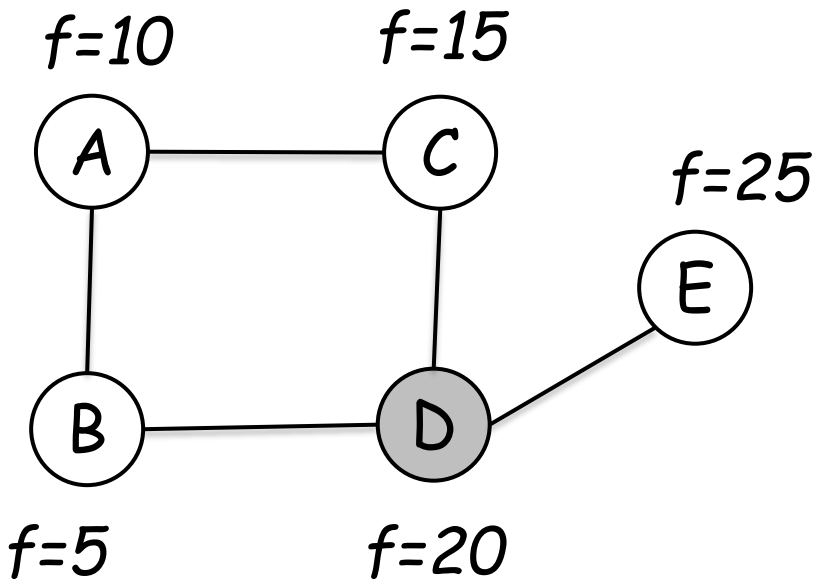
Hill-climbing search: Example



current = A

current = C

Hill-climbing search: Example

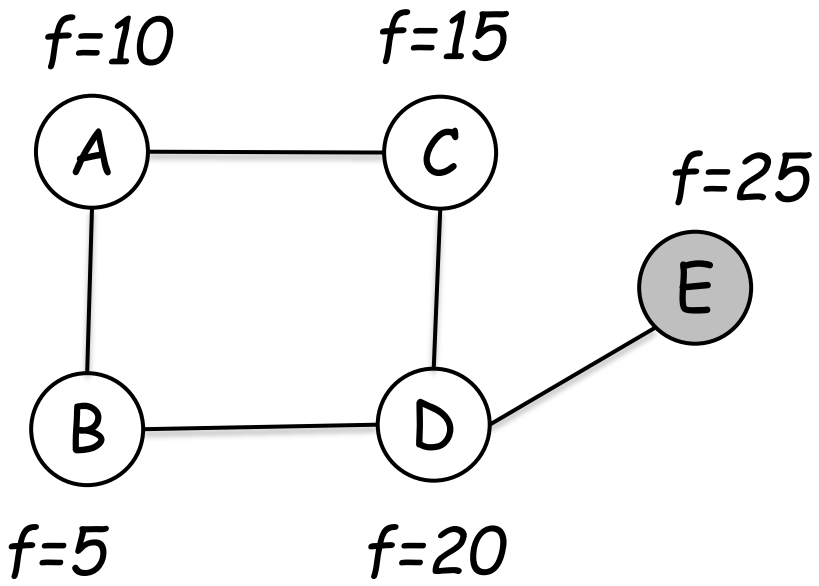


current = A

current = C

current = D

Hill-climbing search: Example



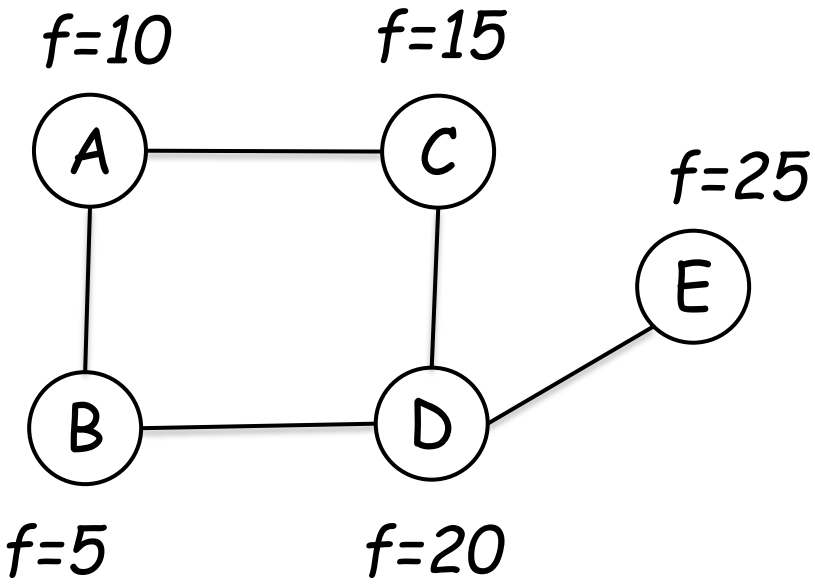
current = A

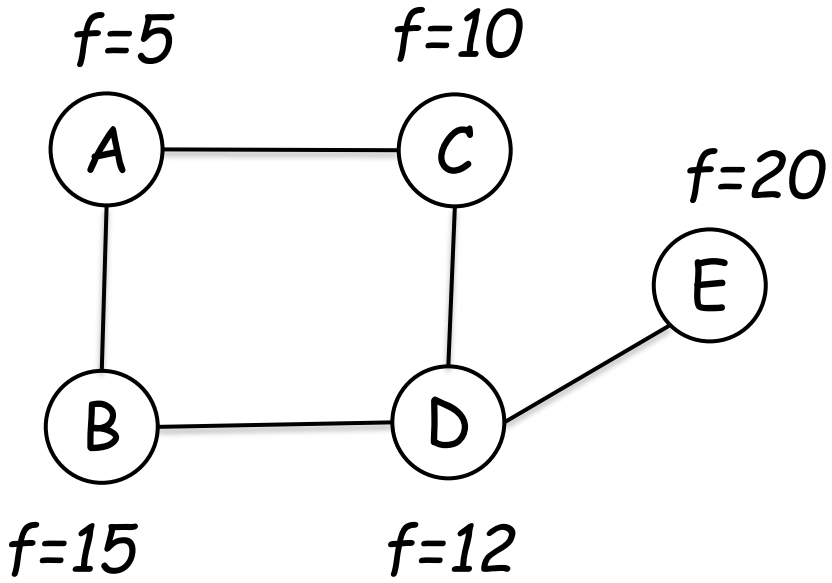
current = C

current = D

current = E

Hill-climbing search: Example





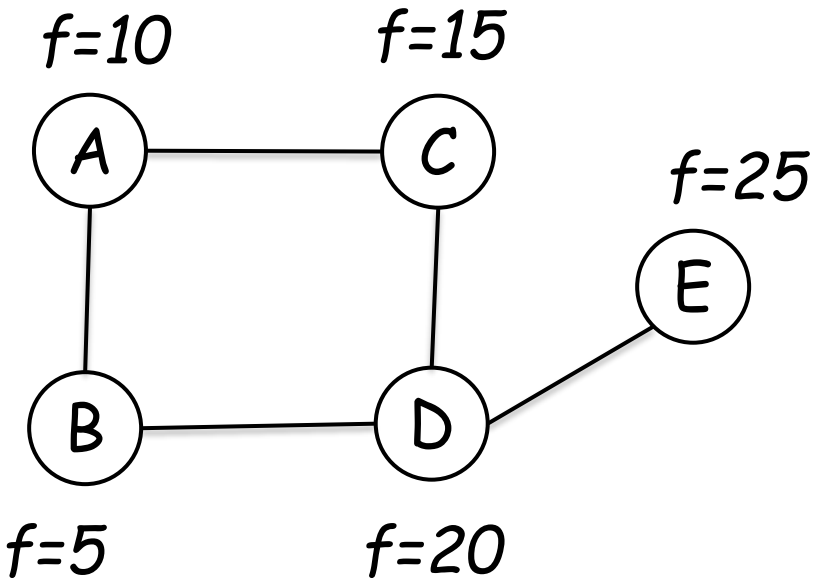
current = A

current = C

current = D

current = E

Hill-climbing search: Example

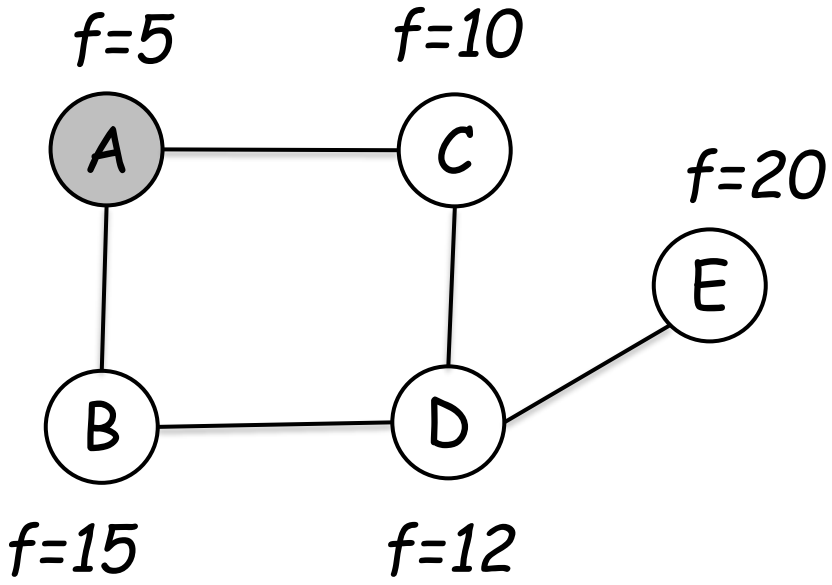


current = A

current = C

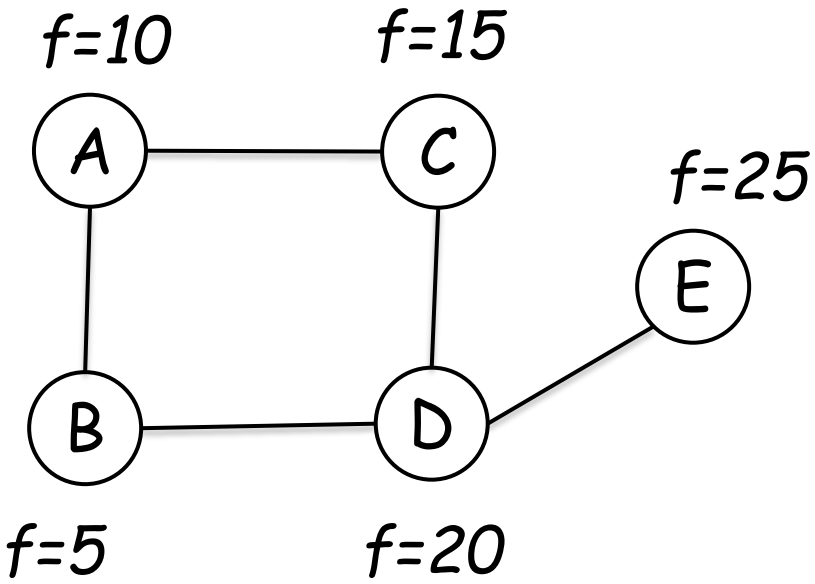
current = D

current = E



current = A

Hill-climbing search: Example

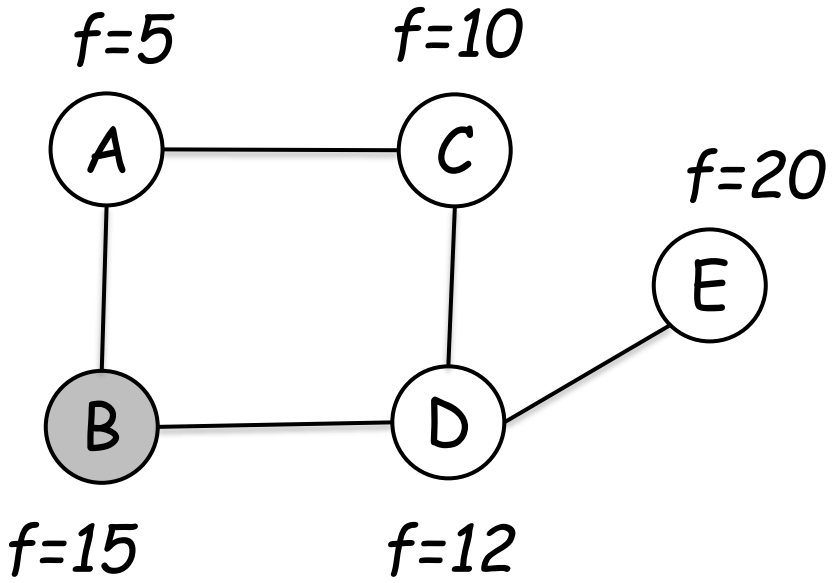


current = A

current = C

current = D

current = E

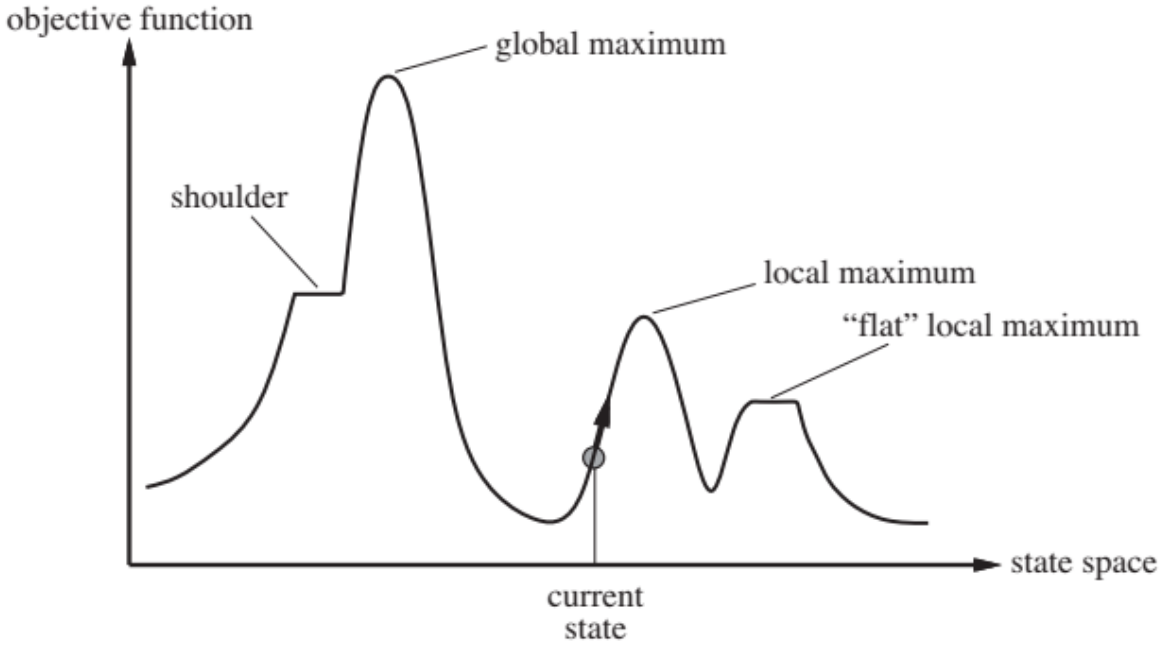


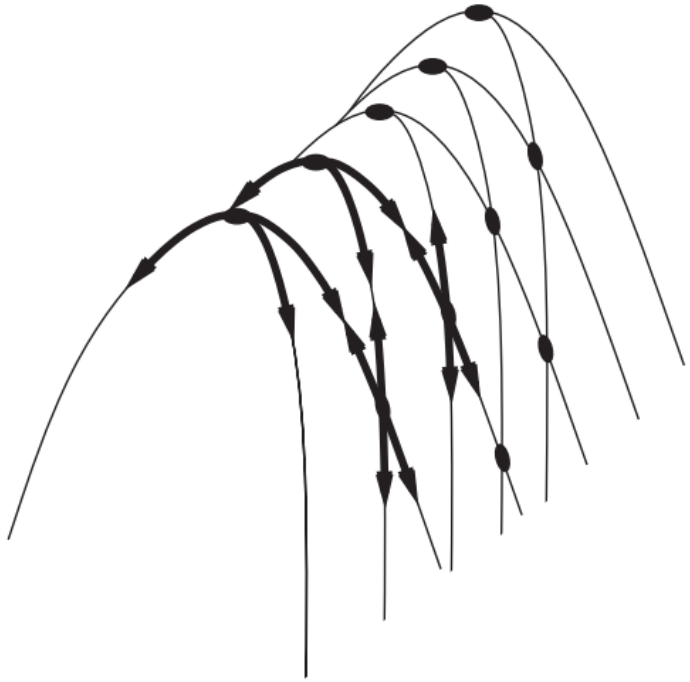
current = A

current = B

Hill-climbing search problems

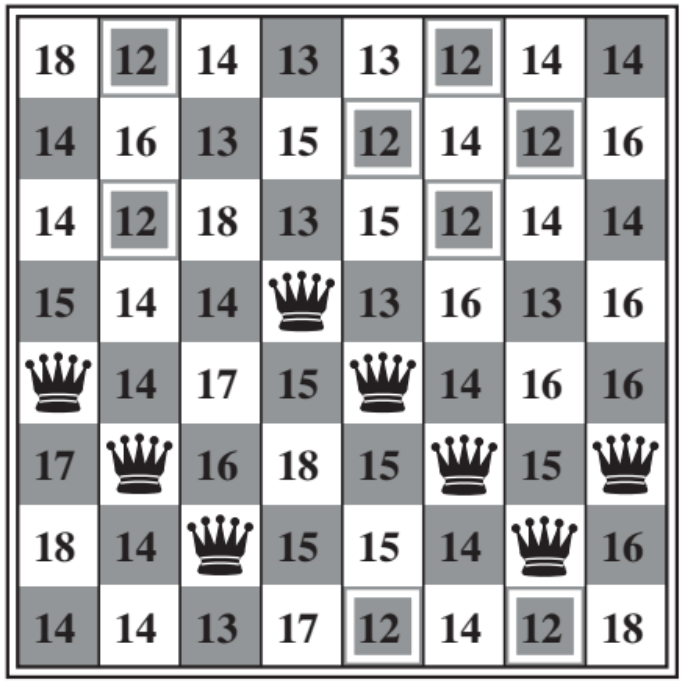
- Local maxima: a peak that is not global max
- Plateau: a flat area (flat local max, shoulder)
- Ridges: a sequence of local max that is very difficult for greedy algorithm to navigate





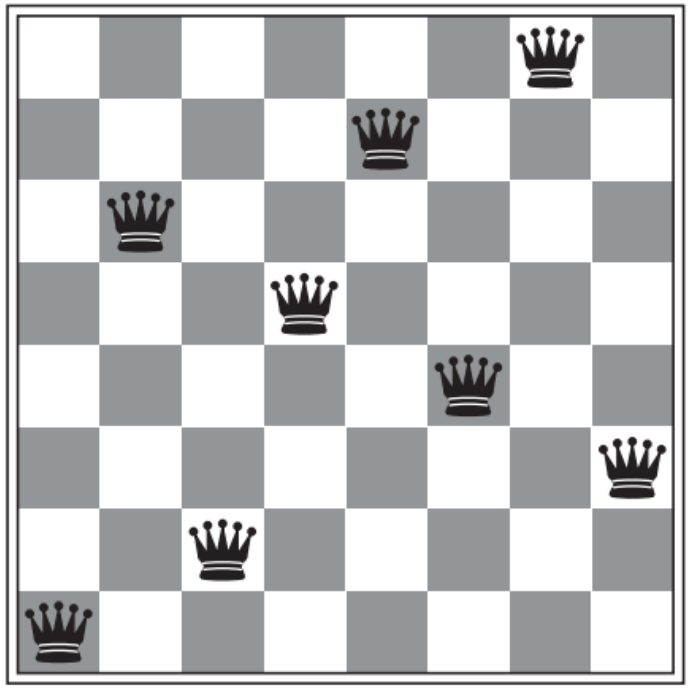
Hill-climbing search problem: 8-queens

- From random initial state, 86% of the time getting stuck
  - On average, 4 steps for succeeding and 3 steps for getting stuck



h(s)=17

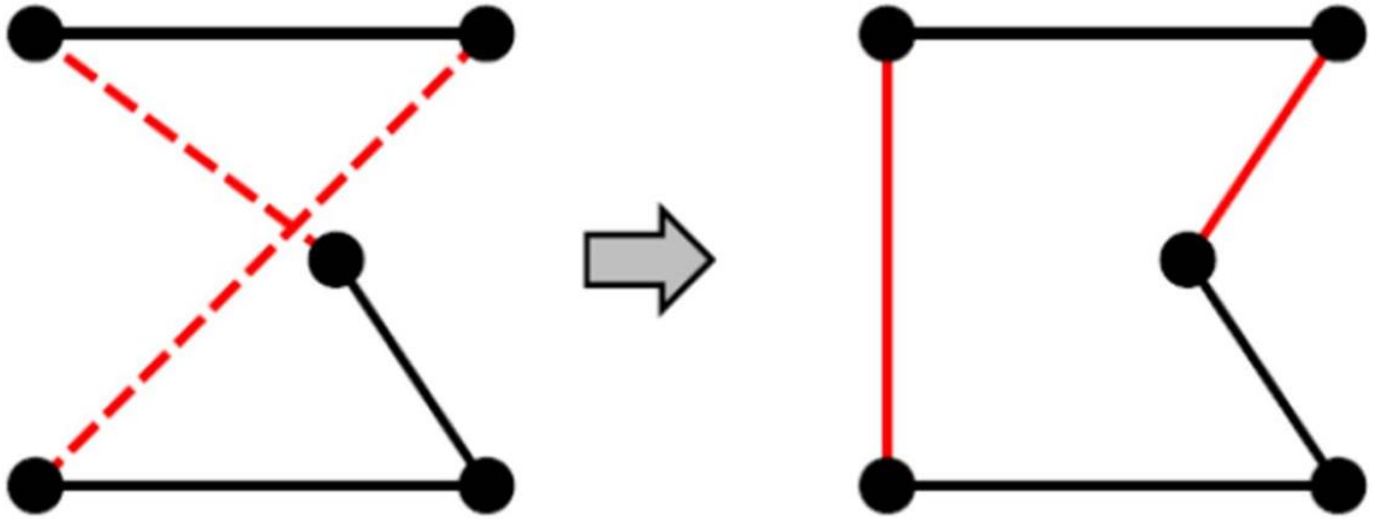
Five steps



h(s)=1

Hill-climbing search problem: TSP

- Start with any complete tour, perform pairwise exchanges
  - Variants of this approach get within 1% of optimal very quickly with thousands of cities

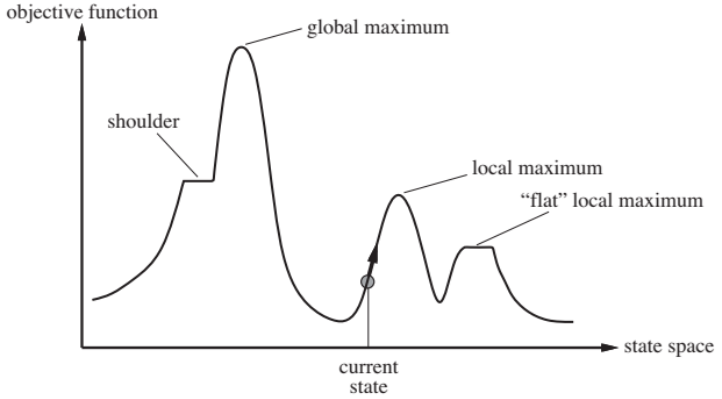


Variants of hill-climbing

- Trying to solve problems of hill-climbing search
  - Sideways moves
  - Stochastic hill climbing
    - First-choice hill climbing
  - Random-restart hill climbing

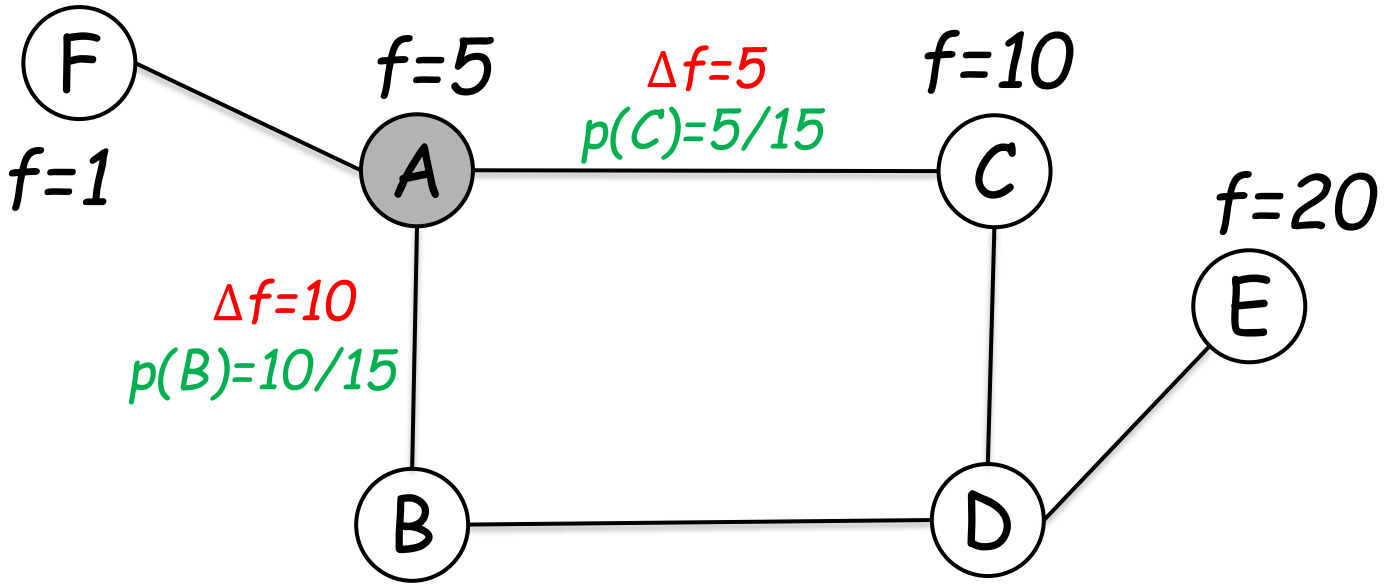
Sideways move

- Sideways move: plateau may be a shoulder so keep going sideways moves when there is no uphill move
  - Problem: infinite loop where flat local max
    - Solution: upper bound on the number of consecutive sideways moves
- Result on 8-queens:
  - Limit = 100 for consecutive sideways moves
    - 94% success instead of 14% success
      - on average, 21 steps when succeeding and 64 steps when failing



Stochastic hill climbing

- Randomly chooses among the available uphill moves according to the steepness of these moves



First-choice hill climbing

- Generating successors randomly until one better than the current state is found
- Good when number of successors is high

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
- Idea
  - If at first you don't succeed, try, try again

While state ≠ goal **do**
  run hill-climbing search from a random initial state

- It is trivially complete with probability approaching 1

Random-restart hill climbing

- p: probability of success in each hill-climbing search
  - Expected number of restarts = 1/p
  - Expected number of steps:

$$\left(\frac{1}{\mathrm{p}} - 1\right) \times \mathrm{n_f} + \mathrm{n_s}$$

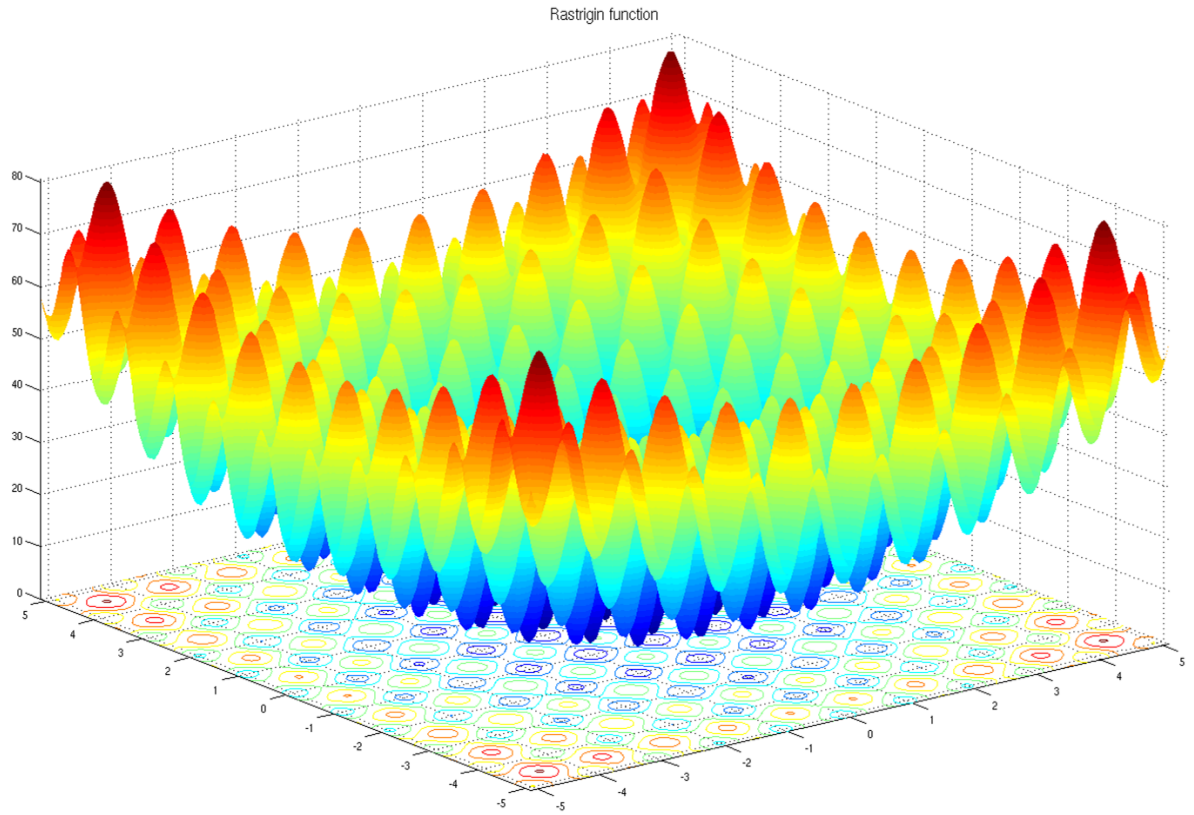
    - n_f: average number of steps in a failure hill-climbing search
    - n_s: average number of steps for the success

Result on 8-queens

- p=0.14 ⇒ 1/p ≈ 7 iterations
- n_s = 4
- n_f = 3
- Expected number of steps:
  - (7 – 1) × 3 + 4 ≈ 22 steps
  - For 3×10^6 queens needs less than 1 minute
- Using also sideways moves:
  - p = 0.94 ⇒ 1/p ≈ 1.06 iterations
  - (1.06 – 1) × 64 + 21 ≈ 26 steps

Effect of land-scape shape on hill climbing

- Shape of state-space land-scape is important
  - Few local max and platea: random-restart is quick
  - Real problems land-scape is usually unknown a priori
  - NP-Hard problems typically have an exponential number of local maxima
    - Reasonable solution can be obtained after a small no of restarts



Simulated Annealing (SA) Search

- Hill climbing: move to a better state
  - Efficient, but incomplete (can stuck in local maxima)
- Random walk: move to a random successor
  - Asymptotically complete, but extremely inefficient
- Idea: Escape local maxima by allowing some "bad" moves but gradually decrease their frequency.
  - More exploration at start and gradually hill-climbing become more frequently selected strategy

Simulated Annealing (SA) Search

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 ← schedule(t)

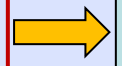
if T = 0 **then return** current

next ← a randomly selected successor of current

ΔE ← next.VALUE – current.VALUE

if ΔE > 0 **then** current ← next

else current ← next only with probability $e^{\Delta E / T}$



Pick a random successor of the current state

- If it is better than the current state go to it
- Otherwise, accept the transition with a probability

T(t) = schedule[t] is a decreasing series

E(s): objective function

Probability of state transition

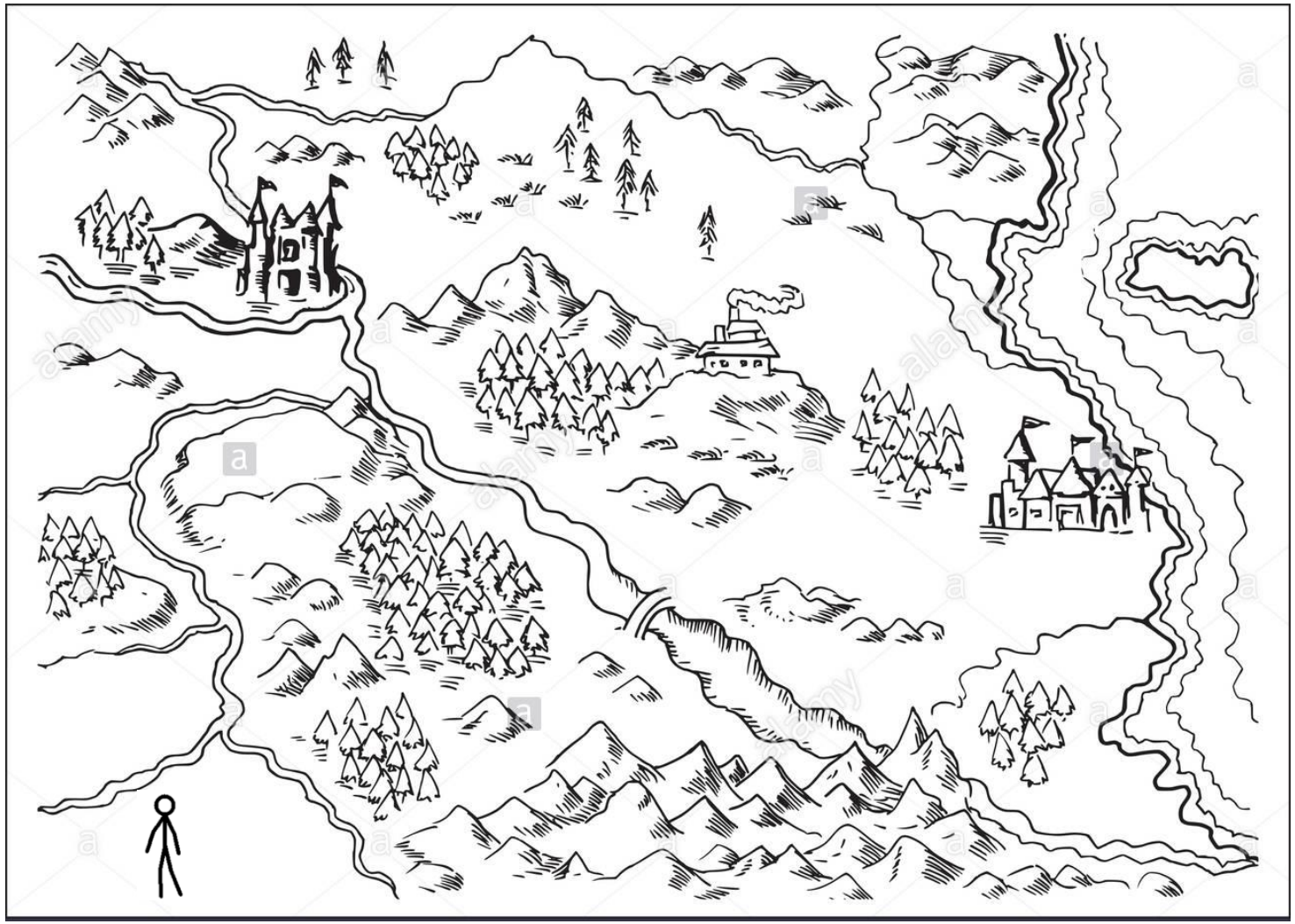
$$p(s, s', t) = \alpha \times \begin{cases} 1 & if\ E(s') > E(s) \\ e^{\frac{E(s')-E(s)}{T(t)}} & o.w. \end{cases}$$

- Probability of "un-optimizing" ($\Delta E < 0$) random movements depends on badness of move and temperature
  - Badness of movement: worse movements get less probability
  - Temperature
    - High temperature at start
      - higher probability for bad random moves
    - Gradually reducing temperature
      - random bad movements become more unlikely and thus hill-climbing moves increase

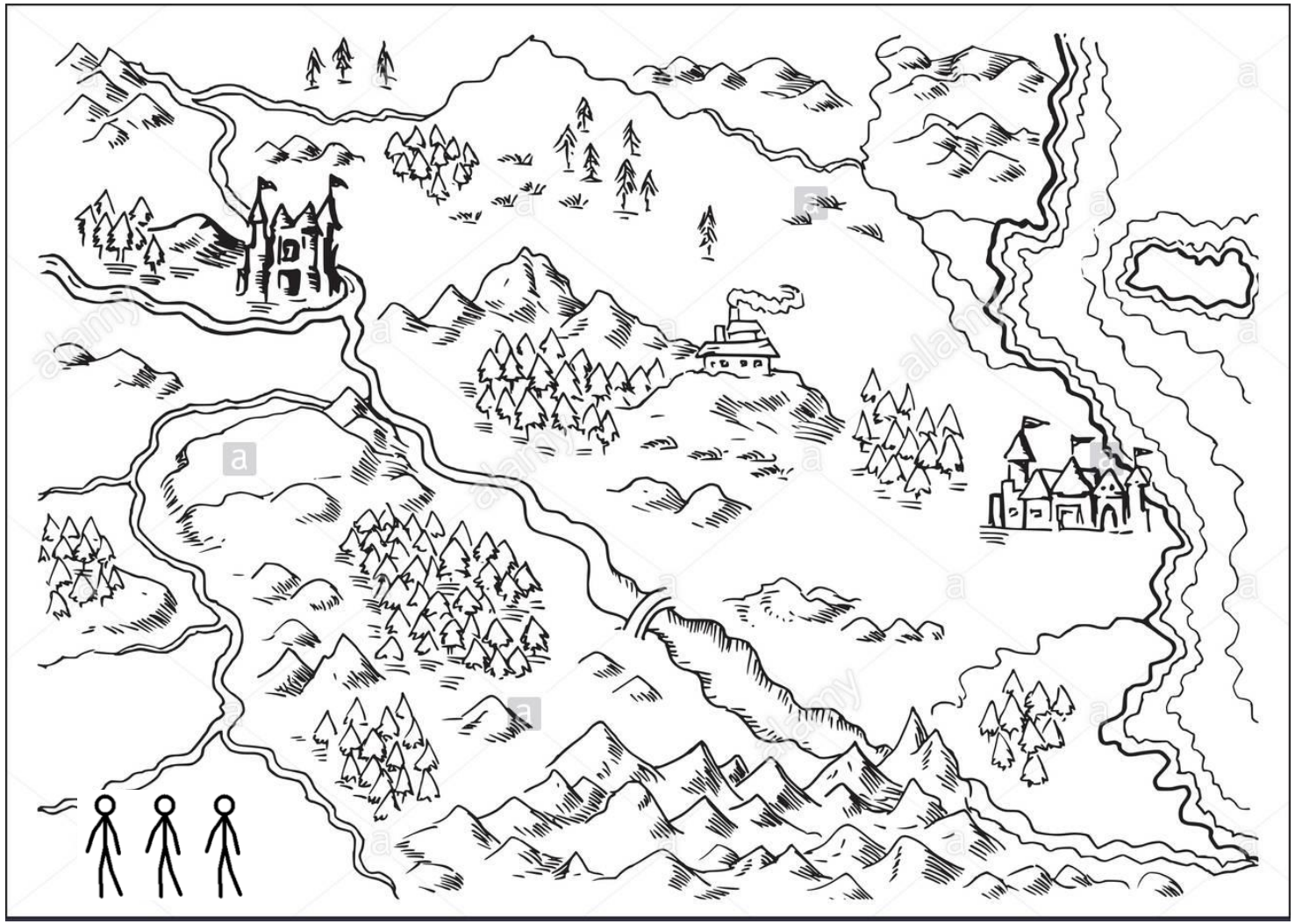
SA as a global optimization method

- Theoretical guarantee
  - If decreases slowly enough, simulated annealing search will converge to a global optimum (with probability approaching 1)
- Practical?
  - Time required to ensure a significant probability of success will usually exceed the time of a complete search

Local beam search



Local beam search



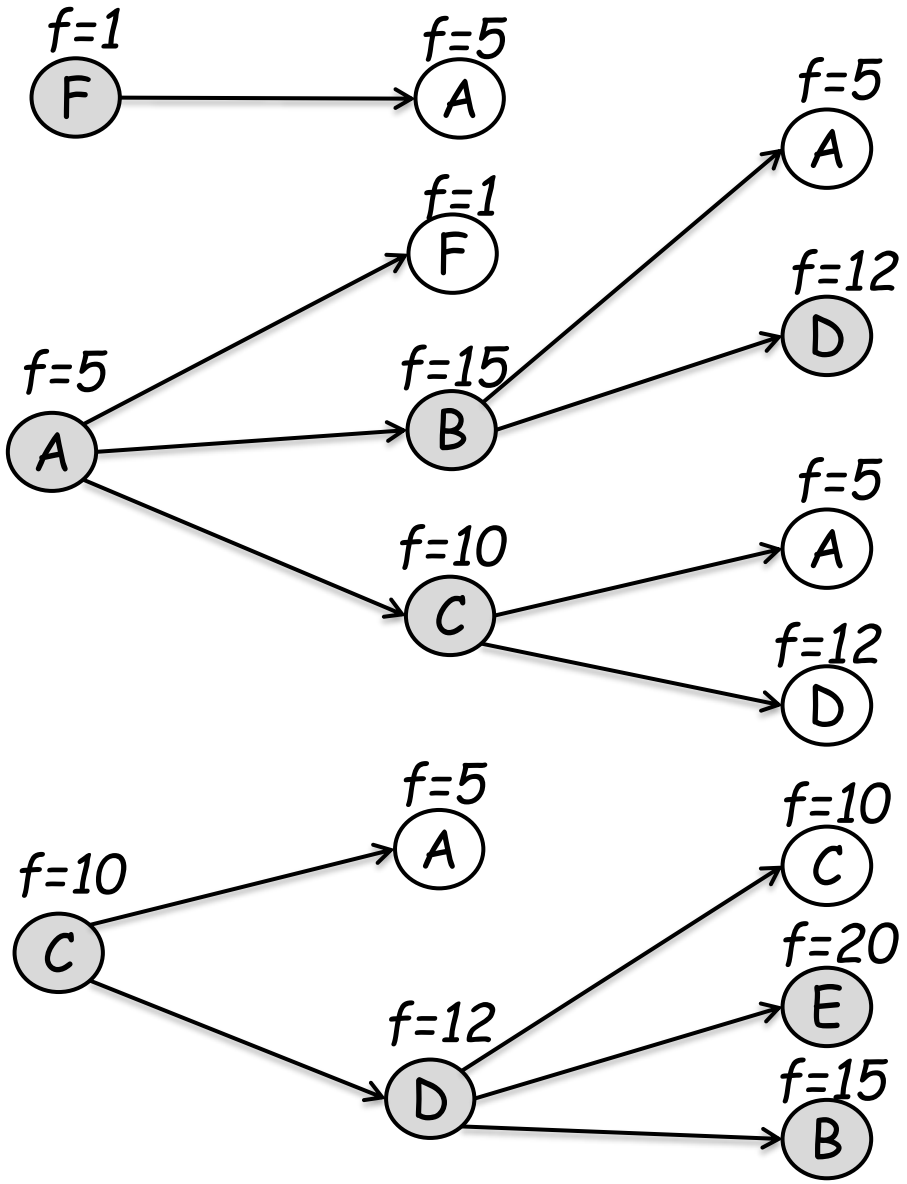
Local beam search

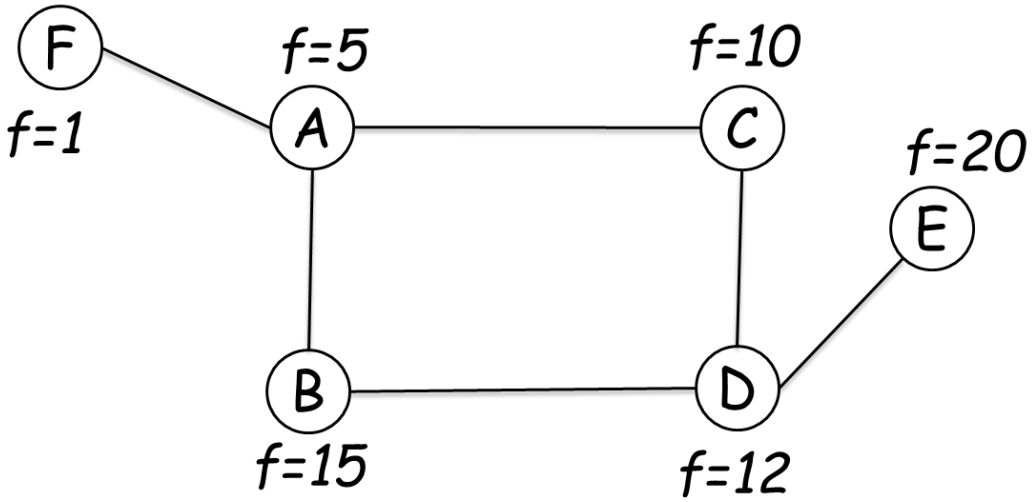
- Keep track of k states
  - Instead of just one in hill-climbing and simulated annealing

```
Start with k randomly generated states
loop:
    All the successors of all k states are generated
    If any one is a goal state then stop
    else select the k best successors from the complete list of successors and repeat.
```

- Is it different from running hill-climbing with random restarts in parallel instead of in sequence?
  - Passing information among parallel search threads

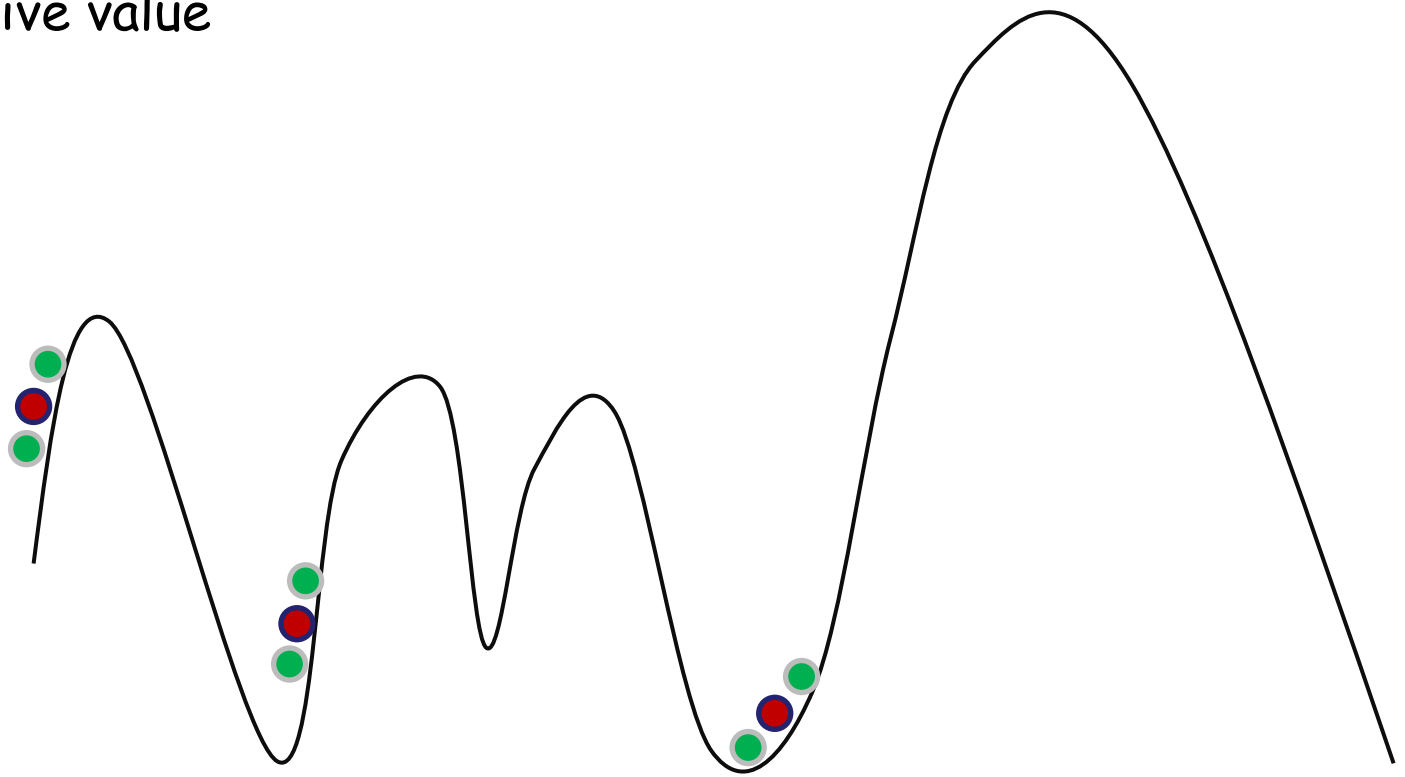
Local beam search: Example





Local beam search:

- Problem: Concentration in a small region after some iterations
  - Solution: Stochastic beam search
    - Choose k successors at random with probability that is an increasing function of their objective value

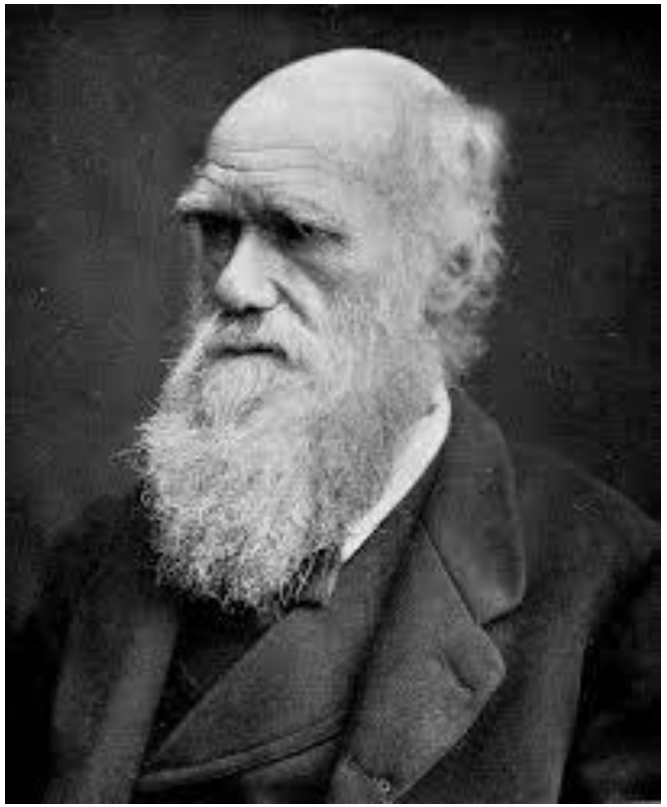


Genetic Algorithms

- A variant of stochastic beam search
  - Successors can be generated by combining two parent states rather than modifying a single state

Natural Selection

- There is competition among living things
- Reproduction occurs with variation
- Selection determines which individuals enter the adult breeding population
  - This selection is done by the environment
  - Those which are best suited reproduce
  - They pass these well suited characteristics on to their young
- Survival of the fittest
  - Those who have the most offspring that reproduce

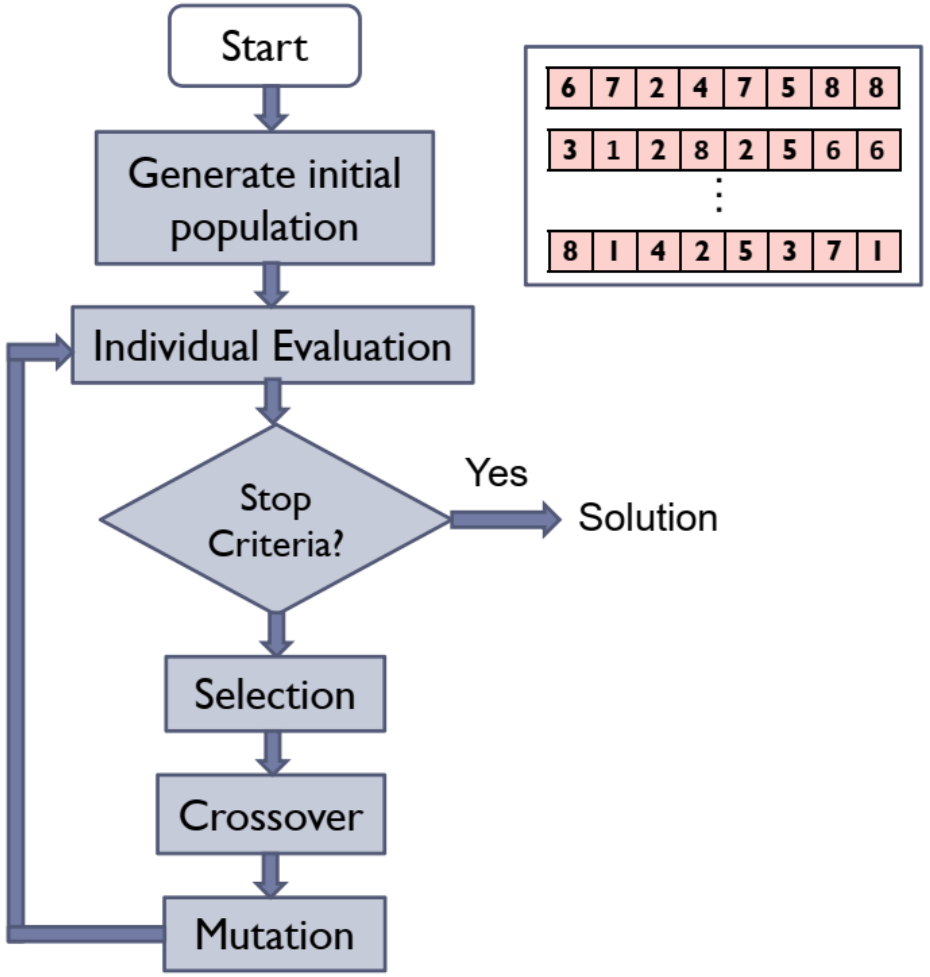


Genetic Algorithm

- The Algorithm
  - Encoding the objectives or optimization functions
  - Defining a fitness function or selection criterion
  - Initializing a population of individuals
  - Evaluating the fitness of all the individuals in the population
  - Creating a new population by performing crossover and mutation.
  - Evolving the population until certain stopping criteria are met
  - Decoding the results to obtain the solution to the problem

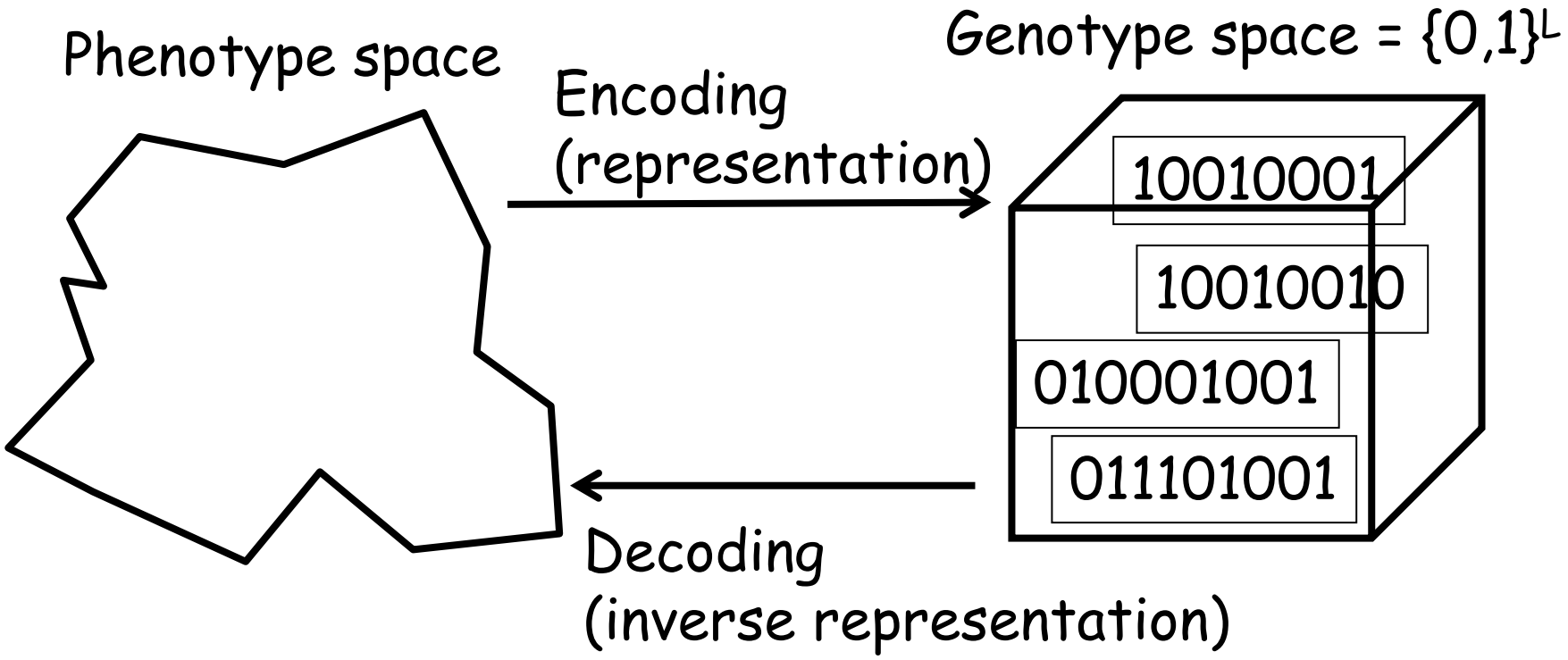
Genetic Algorithm

- The Algorithm



Representation

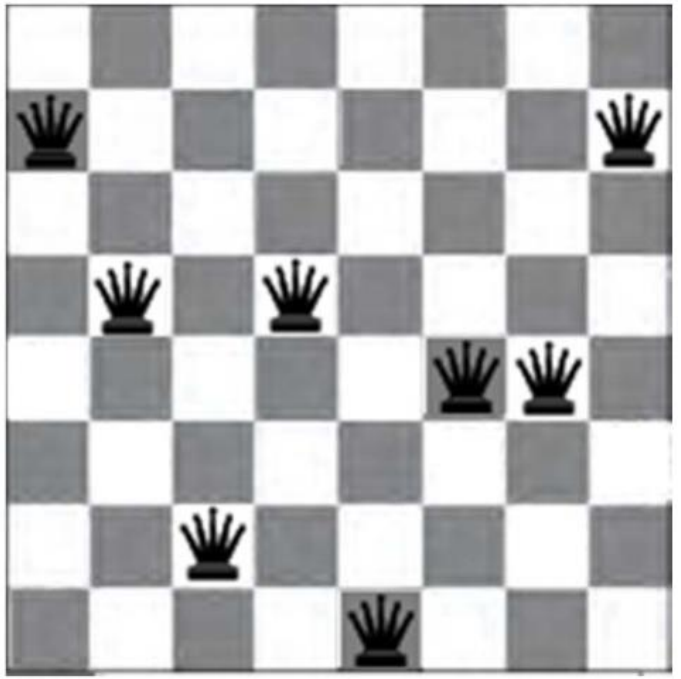
- Each state, or individual, is represented as a string over a finite alphabet



Fitness Function

- A fitness function should return higher values for better states

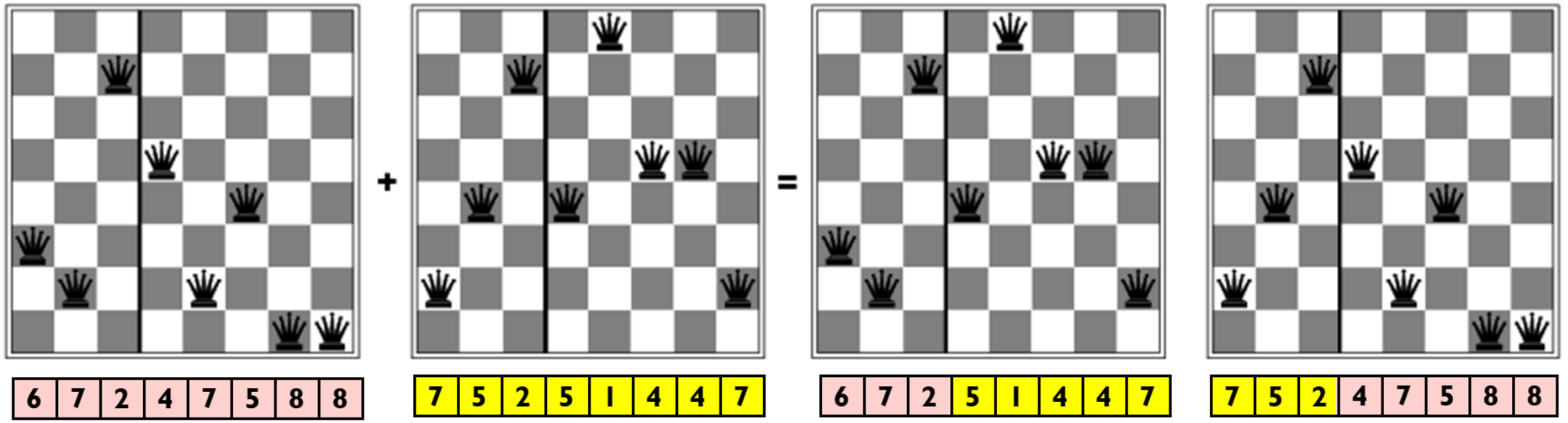
- Example: 8-queens
  - Representation (Chromosome)
    - Describe the individual (or state) as a string
  - Fitness Function
    - Number of non-attacking pairs of queens
      - 24 for this figure



2	4	7	4	8	5	5	2

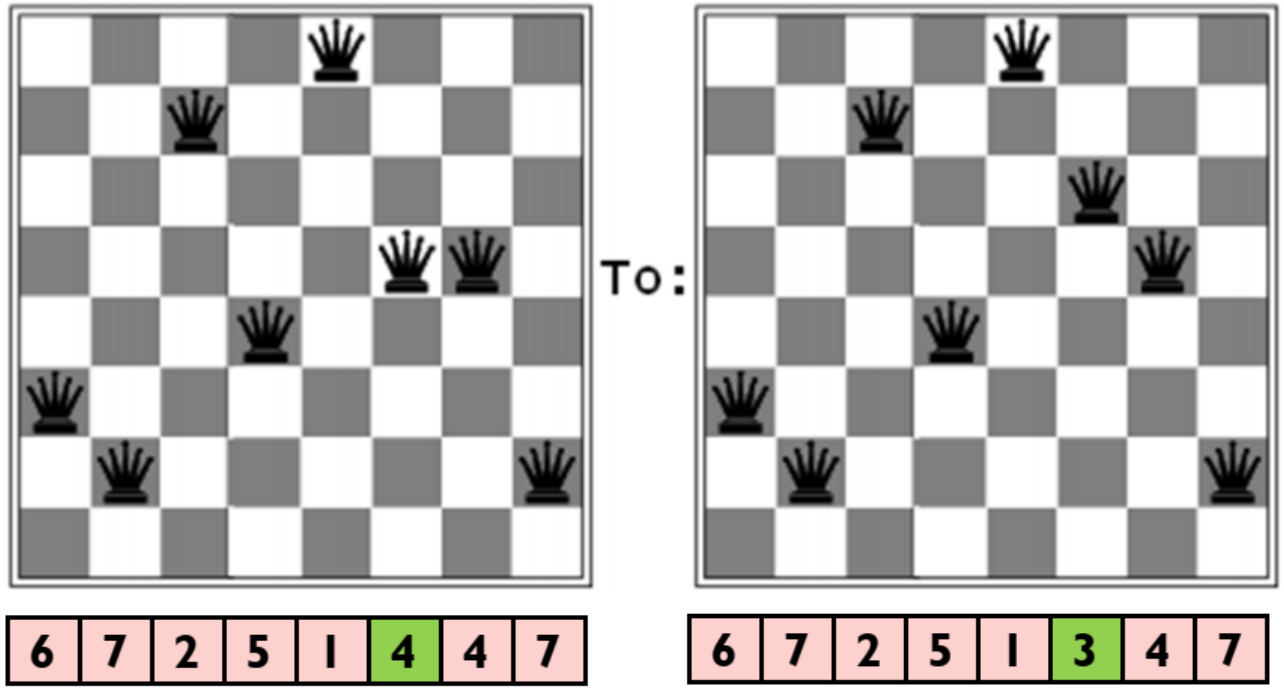
Genetic operators

- Cross-over
  - To select some part of the state from one parent and the rest from another

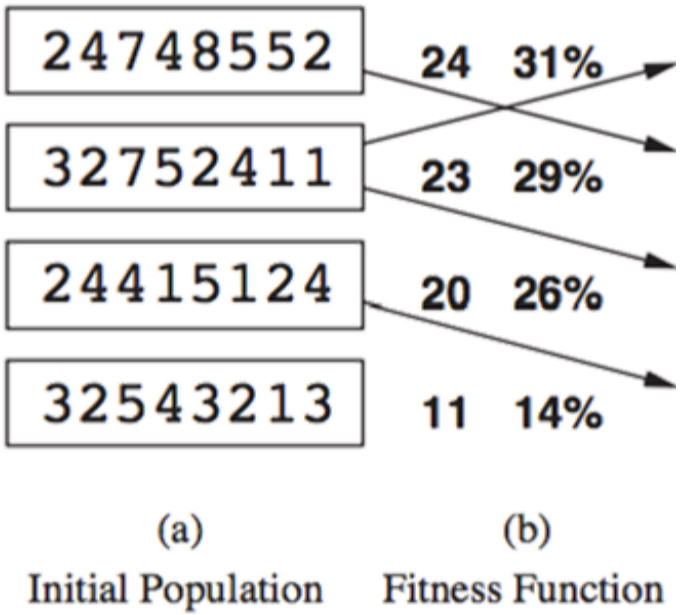


Genetic operators

- Mutation
  - To change a small part of one state with a small probability

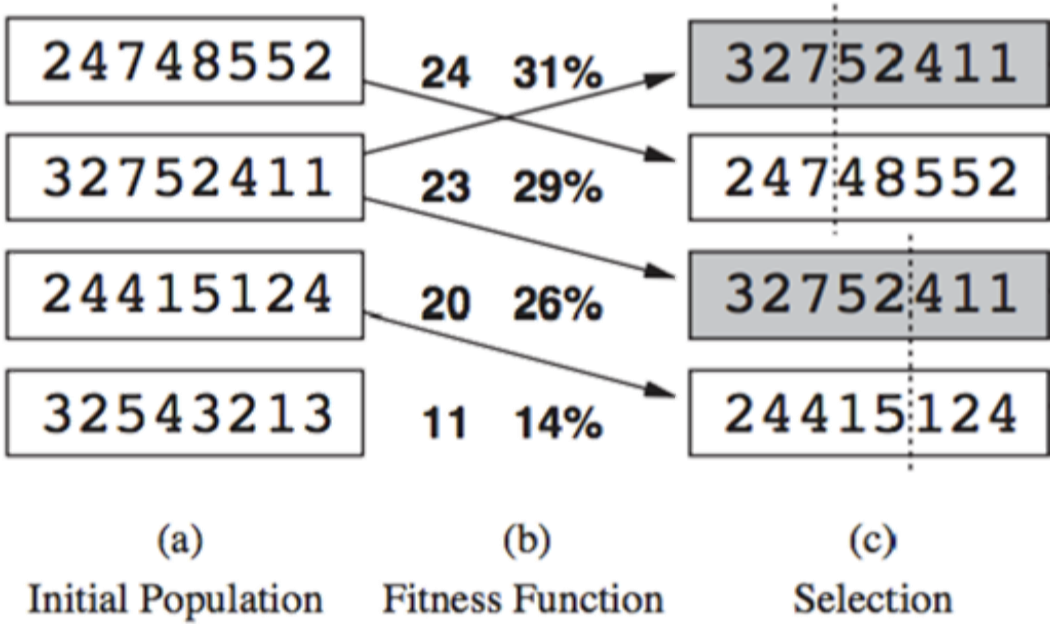


A variant of genetic algorithm: 8-queens



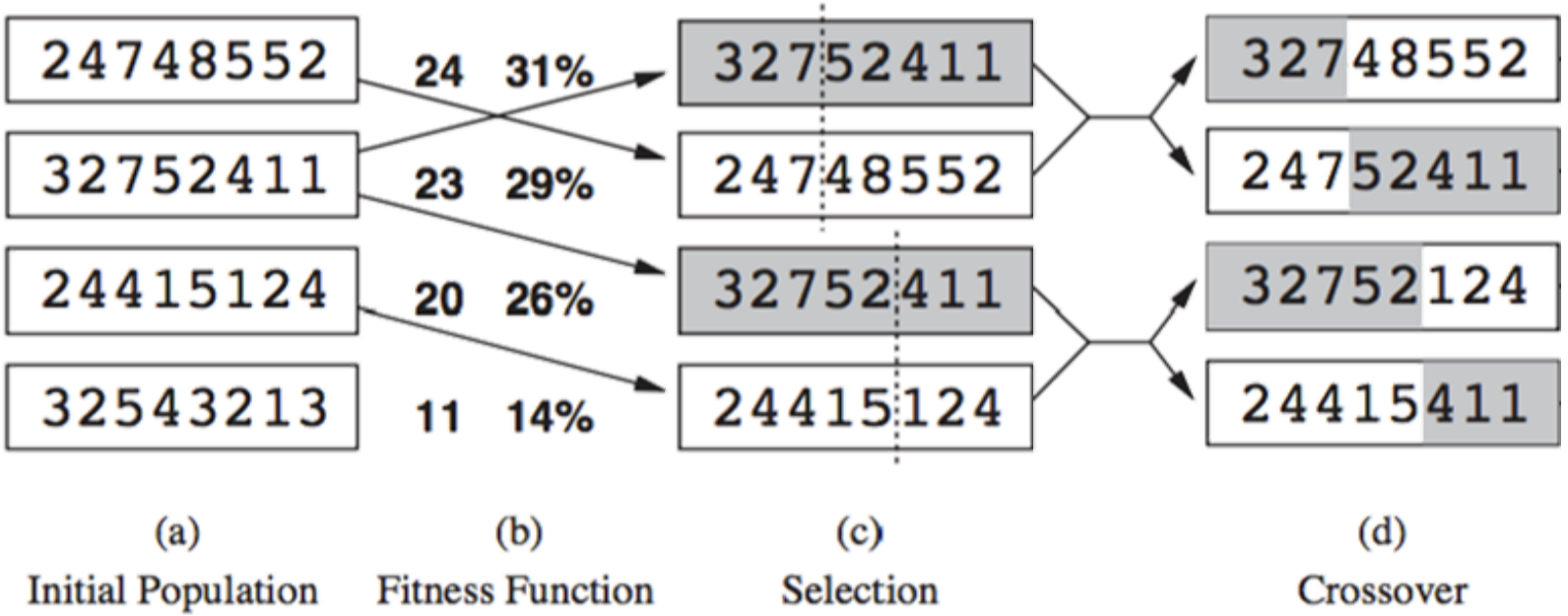
- Fitness function: number of non-attacking pairs of queens
  - Min=0, Max=(8×7)/2=28
  - $Reproduction - rate(i) = fitness(i) / \sum_{k=1}^{n} fitness(k)$
    - e.g., 24/(24+23+20+11) = 31%

A variant of genetic algorithm: 8-queens



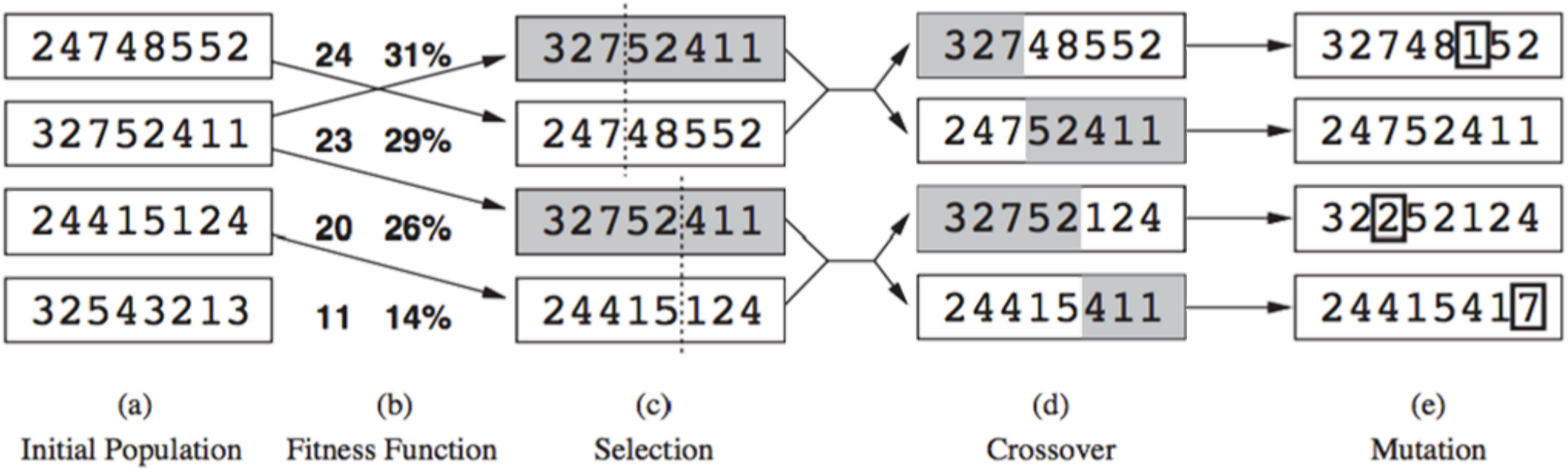
- Fitness function: number of non-attacking pairs of queens
  - Min=0, Max=(8×7)/2=28
  - $Reproduction - rate(i) = fitness(i) / \sum_{k=1}^{n} fitness(k)$
    - e.g., 24/(24+23+20+11) = 31%

A variant of genetic algorithm: 8-queens



- Fitness function: number of non-attacking pairs of queens
  - Min=0, Max=(8×7)/2=28
  - $Reproduction - rate(i) = fitness(i) / \sum_{k=1}^{n} fitness(k)$
    - e.g., 24/(24+23+20+11) = 31%

A variant of genetic algorithm: 8-queens



- Fitness function: number of non-attacking pairs of queens
  - Min=0, Max=(8×7)/2=28
  - $Reproduction - rate(i) = fitness(i) / \sum_{k=1}^{n} fitness(k)$
    - e.g., 24/(24+23+20+11) = 31%

Genetic Algorithm

```
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))
```

Genetic algorithm properties

- Genetic algorithm usually take large steps in earlier generations and smaller steps later
  - Initially, population individuals are diverse
    - Cross-over operation on different parent states can produce a state long a way from both parents
  - More similar individuals gradually appear in the population
- Cross-over as a distinction property of GA
  - Ability to combine large blocks of genes evolved independently
    - Representation has an important role in benefit of incorporating crossover operator in GA

Local search vs. systematic search

	Systematic search	Local search
Solution	Path from initial state to the goal	Solution state itself
Method	Systematically trying different paths from an initial state	Keeping a single or more "current" states and trying to improve them
State space	Usually incremental	Complete configuration
Memory	Usually very high	Usually very little (constant)
Time	Finding optimal solutions in small state spaces	Finding reasonable solutions in large or infinite (continuous) state spaces
Scope	Search	Search & optimization problems