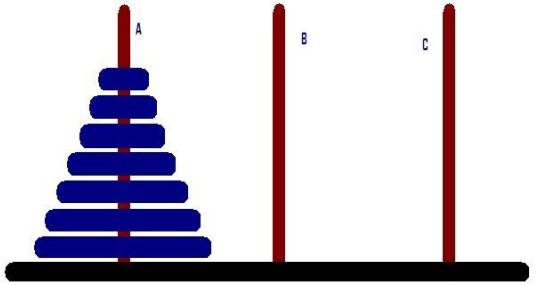
3. Problem solving and search

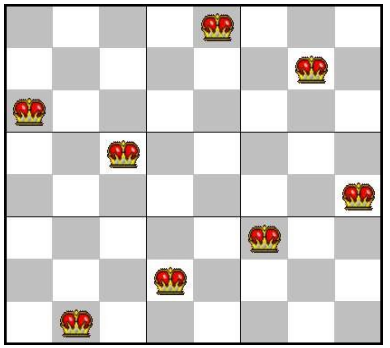
3.1 Definitions

- Problem solving –accom plish a complex, often unexpected task
  - Find/recognize problem→ formulate → solve
  - Not to open a door with a key, but rather what to do when there is no key.
  - “If you can't solve a problem, then there is an easier problem you can solve: find it.”(George Polya)
- Planning — find a sequence of actions that allows a system to accomplish a specific task
- Decision making –choose between alternative options
  - Buying a new car
  - Betting on a sports team
  - Choosing a partner
- All three are related, but traditionally approached differently.

3.2 Problem domains in AI

- Toy problems
  - Simple
  - Well-defined
  - A solution exists and it can be found





- Real-world problems
  - Complex
  - Open-ended and/or ill-defined
  - Examples: scheduling and routing
- How well do toy problems scale up to real-world domains?

3.3 Defining the problem

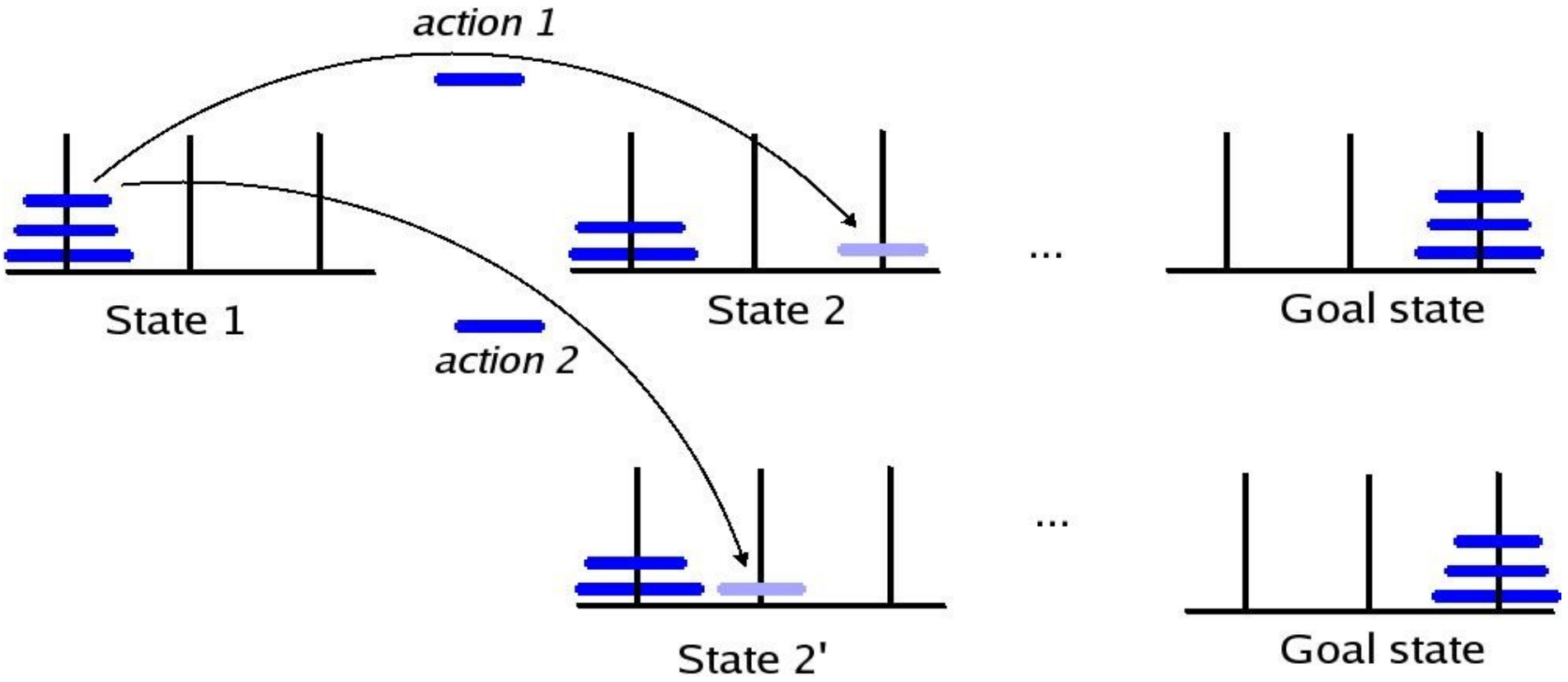
- Problem is defined as a *state space* consisting of partial problem solutions
- State space often represented as a graph.
- Problem components:
  - *States* **S** = (s_1, ..., s_N) are nodes in the graph.
  - *Initial state* s_I is the state the problem solver is when starting.
  - Available *actions* **A** = (a_1, ..., a_M) the solver makes to move deterministically from state to state. Actions are represented as arcs or edges in the graph.
  - *Goal* — Either a single state or a set of states in **S**, or a more abstract entity.
  - *Test* to check if the goal has been reached.
  - *Cost function* C applied to actions.

3.4 Solving the problem

- Problem is solved by searching the state space
  - ***Data driven***: start from the initial state and move forward applying the allowed actions → forward chaining
  - ***Goal driven***: start from the goal and move towards the initial state applying moves that could have resulted in a goal → backward chaining
- Data driven search process builds up a *search tree*; the root of the tree is the initial state, and the final state is one of the leaves.

3.5 Example

TOWER OF HANOI

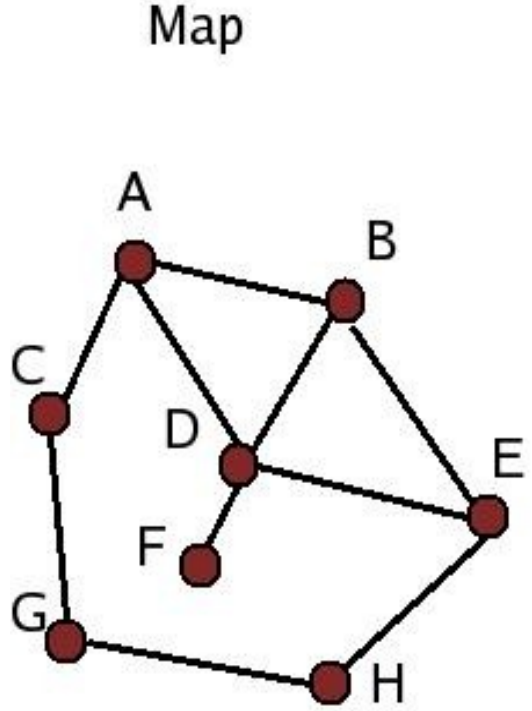


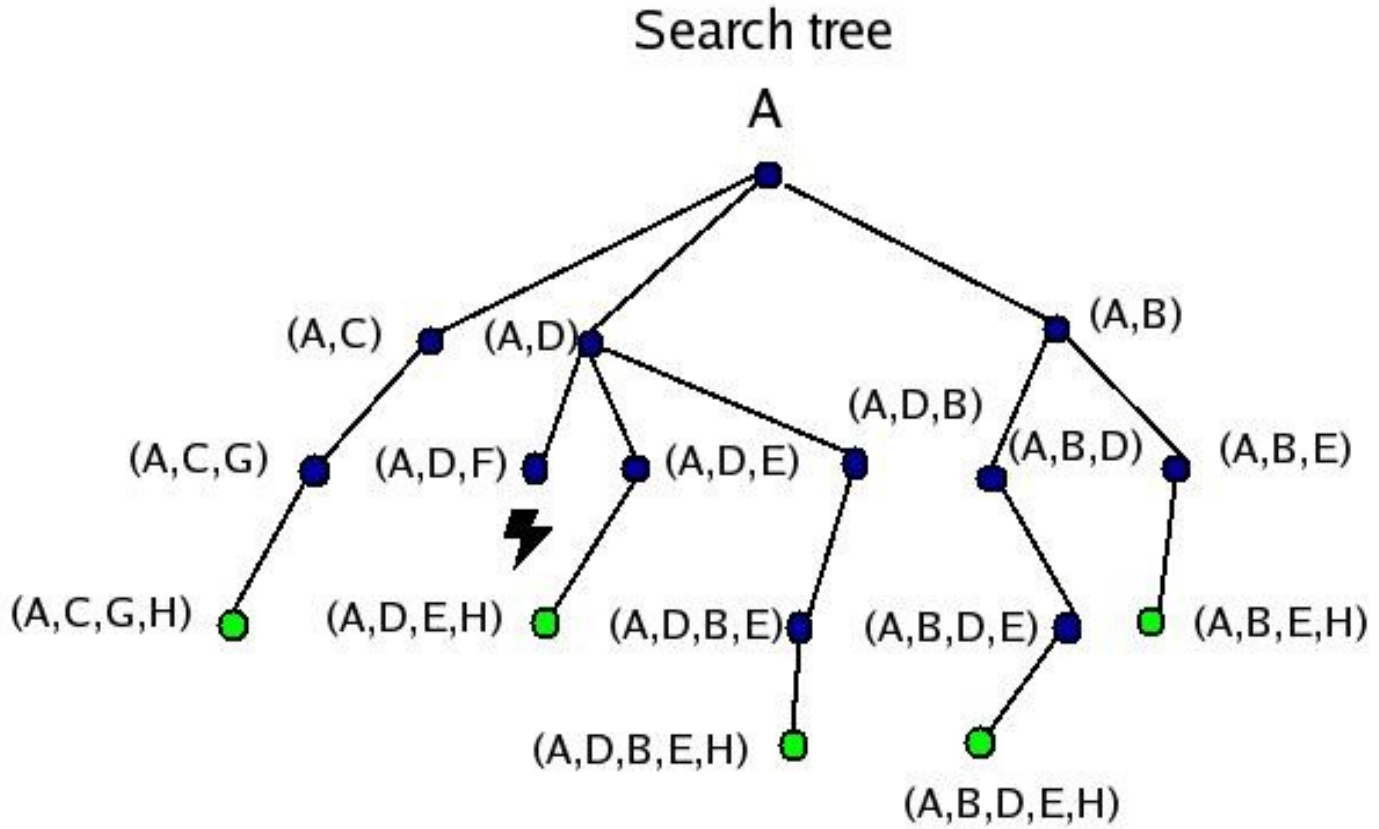
Goal is to move all disks from one peg to another, so that only one disk is move at the time, and a larger disk is never put on top of a smaller one.

3.6 Generate and test

- Simplest form of *brute-force* (exhaustive) search
- Generates every state in the state space and tests if that is a goal.
- Steps in forward chaining brute-force search:
  1. Check if the current state is the goal state.
  2. IF yes, terminate, ELSE generate successors of the current state (=expand the current state).
  3. Choose one of the successors, and make it the current state.
- Solution is the path from the initial state to the goal.

3.7 Example
ROUTING





Goal is to travel from A to H

Note the difference between the search space, which is limited, and the search tree, which is potentially infinite.

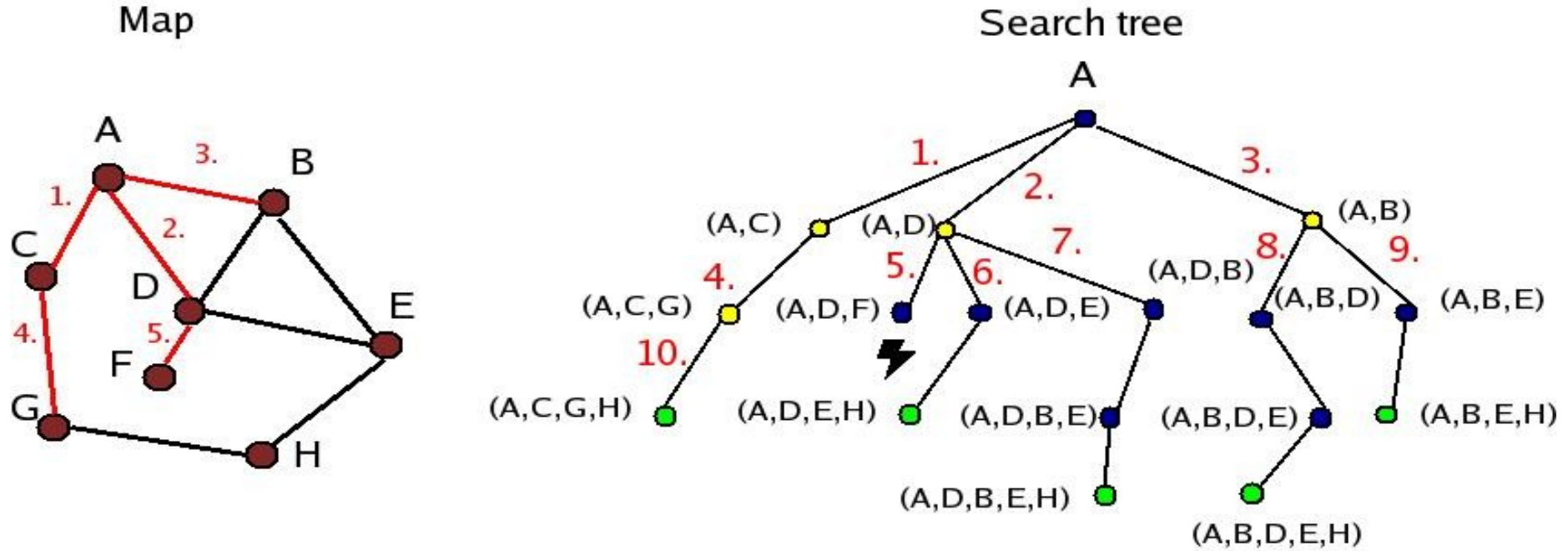
3.8 Search strategies

- Determine the order in which the state space is traversed, i.e., which node in the tree is expanded next
- *Uninformed* (blind) search strategies
  - Do not use any information about the state space, but can tell the difference between goal state(s) and non-goal states.
- *Informed* search strategies
  - Have information about the search space to guide the search.
  - Use heuristics to choose the next state to be expanded.

3.9 Success criteria

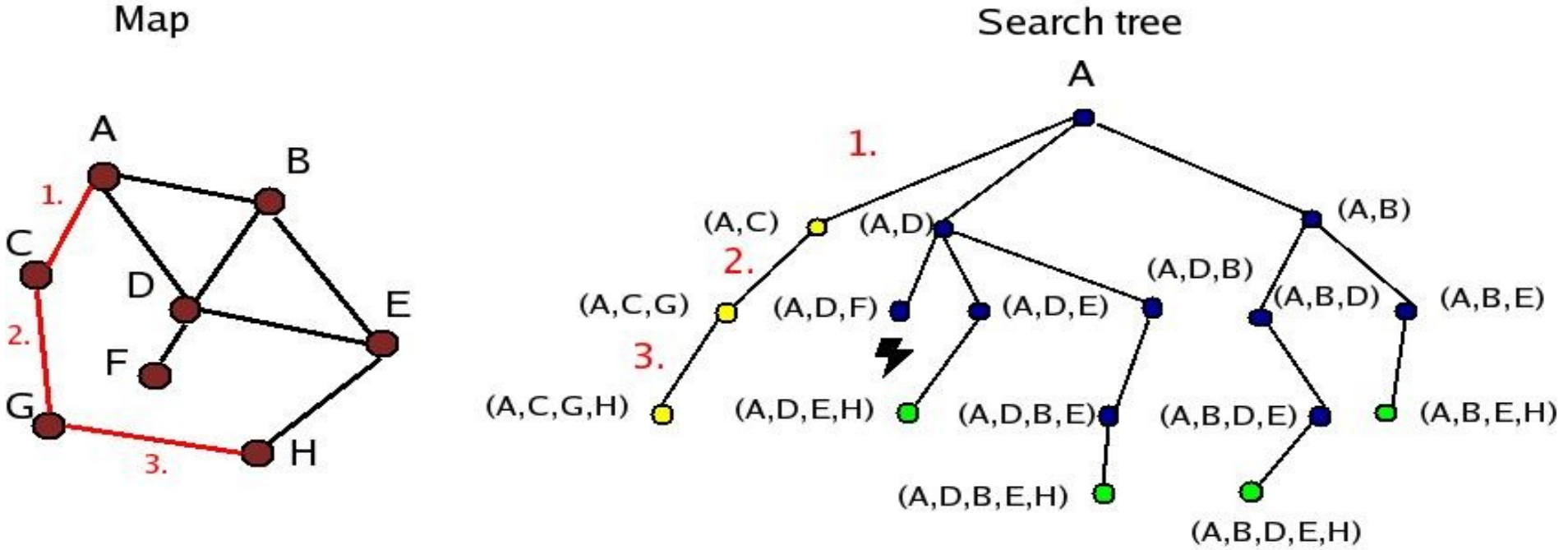
- Completeness — Is the search method guaranteed to find a solution if it exists?
- Complexity – What are the time and space requirements?
- Optimality – Does the search method find the best existing solution?
- Admissibility – Does the method find the solution in least amount of time?

3.10 Breadth-first search



- Proceeds a level by level: one level in the search tree is examined before going to the next level.
- Data structure: queue
- Optimal and complete
- Not practical if
  - All paths to goal equally long
  - Branching factor is high

3.11 Depth-first search

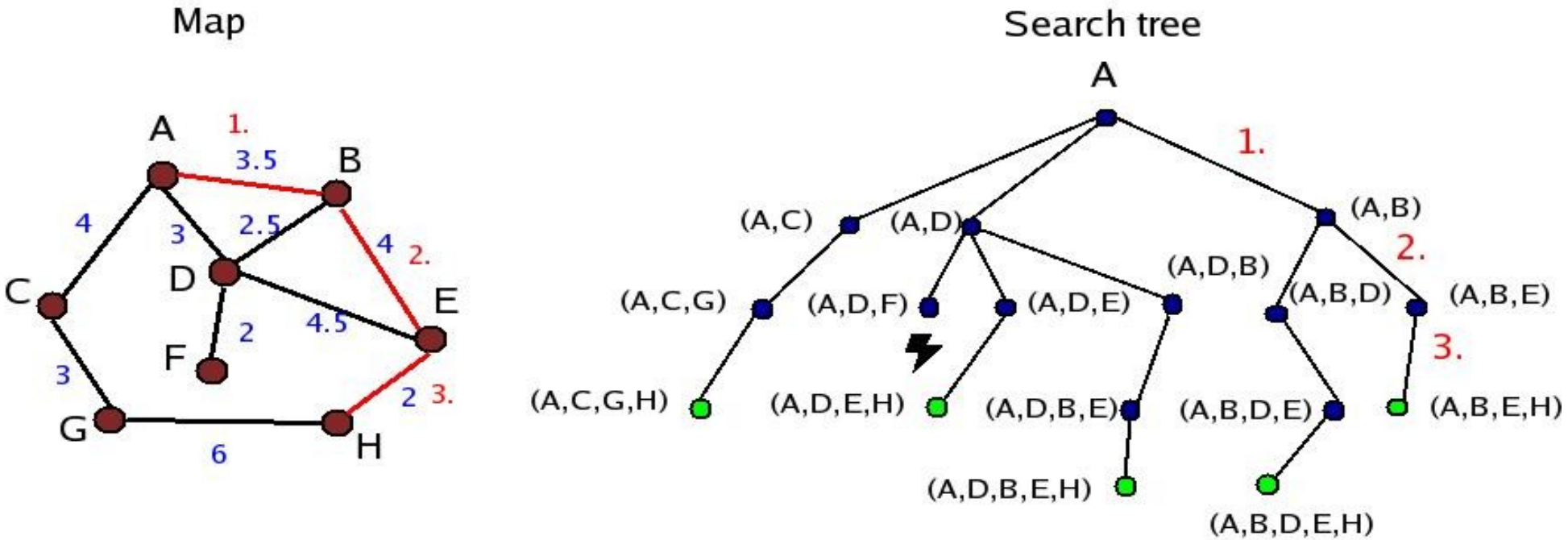


- Proceeds down one path until a solution or dead-end is found.
- Data structure: stack
- Neither optimal not complete
- Not practical if non-goal path is very long or potentially infinite.
- Variation: iterative deepening
  - carry depth-first search to certain depth: first to depth one, then to depth two, and so on, until goal found.

3.12 Informed search strategies

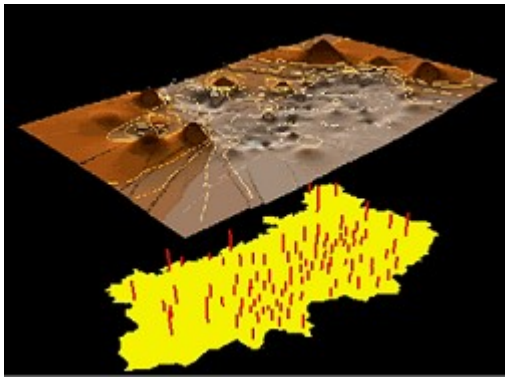
- Heuristic = "the study of the methods and rules of discovery and invention" (Polya 1945)
- Comes from Greek word *eurisco* = 'I discover.'
- Design of heuristic algorithms is core of artificial intelligence.
- Why heuristics?
  - No exact solution exists.
  - Computational cost of finding the exact solution is overwhelming.
- Heuristics are fallible
  - An informed guess
  - Based on intuition or experience
- What is a good heuristic?
  - Not too costly to compute
  - Minimizes the number of states that need to be examined to find a solution

3.13 Best-first strategies



- In search terms: **Heuristic evaluation function** gives an **estimate** of the distance from the current node to the goal.
  - **Branch and bound**: choose the node *n* that minimizes the cost so far.
  - **Greedy**: Choose the node *n* that appears to have the lowest cost to goal.
  - **A***: choose the next node *n* so that *C(n) = f(n) + g(n)* is minimized. *f(n)* is the cost (e.g., distance) traveled so far, and *g(n)* is an estimate of the remaining cost to goal, if traveled through the node *n.*

3.14 Hill-climbing

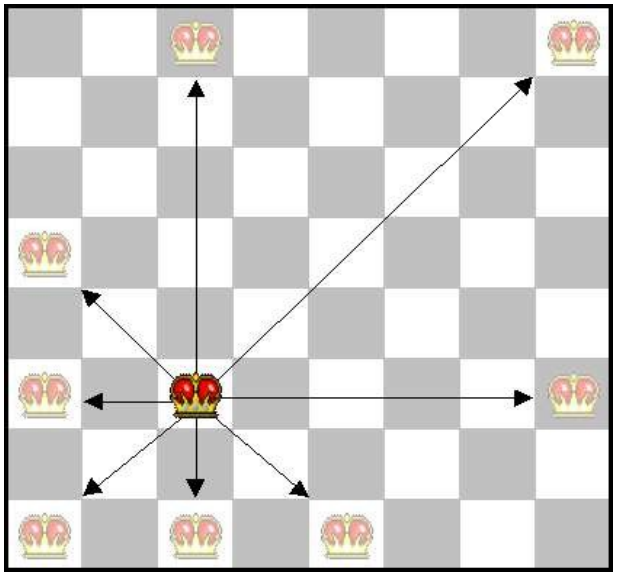


- The search space represented as a 3-dimensional landscape rather than a tree.
  - x- and y-axes represent variables
  - z-axis represents the outcome or goodness of a particular solution
- Goal is to maximize the outcome.
- Chooses the first state that is higher than the current state.
- Steepest ascent HC: check all the directions, and choose one that is highest.
- Plateaus, ridges and foothills (local optima)

3.15 Constraint satisfaction problems (CPS)

- Let us have variables $X=x_1, ..., x_n$, so that each x_i has a value range d_i.
- We also have constraints $C=c_1,..., c_m$.
  - Discrete
  - Continuous
  - Numerical or non-numeric
- A state is a value assignment to all or some of the variables ($x_1=v_1, x_2=v_2,...x_n=v_n$).
- A (complete and consistent) solution is a assignment of values to all variables that meets the constraints.

3.16 Example
EIGHT QUEENS



- Place eight queens on the chess board so that none of them attacks another:
  - Variables: coordinate pairs
  - Range: 1 ... 8 and a ... h
  - Constraints: no two queens can be on the same row, the same column or the same diagonal.
- Simplest strategy: analyze all possible configurations.
- Start placing pieces one by one. If no more pieces can be placed without violating the constrains,
  - *Backtrack* to safe state
  - *Forward checking*: current choice restrict the future choices.
  - *Most-constrained* variable heuristic

3.17 Another example

GENERALIZED SUDOKU PROBLEM

- Place *s* symbols on a *s x s* matrix, so that each s^2 cells is filled with one of the symbols *s* and no symbol is repeated in any row or column or block-region.
- Block-regions are predetermined non-overlapping subsets of *s* cells → there are *s* block-regions.
- Most common is the number Sudoku:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

3.18 Combinatorial optimization

- Find an assignment to a number of variables so that an objective function is either minimized (cost) or maximized (payoff).
- Several real-world allocation, routing and scheduling problems belong to this class, for instance traveling salesman problem.
- Optimal solution hard to find analytically, and brute-force search infeasible, since the number of possible solutions is huge.
- Metaheuristics:
  - Iterated local search
  - Tabu search
  - Ant colony optimization
  - Simulated annealing
  - Genetic algorithms

3.19 Genetic algorithms (GA)

- **Inspired by natural evolution;** fit individuals survive and reproduce
- **Population of chromosomes**
  - Potential solutions to the problem
  - Consist of genes (often represented as bits)
- **Genetic operations**
  - *Selection* — two chromosomes are selected for reproduction
  - *Crossover* — selected chromosomes (parents) are combined to form a new chromosome (offspring)
  - *Mutation* — a small random change applied with low probability to a single chromosome.
- **Fitness**
  - Objective measure for the quality of the chromosome
  - Probability to survive to the next generation
  - Likelihood of reproduction

3.20 The algorithm

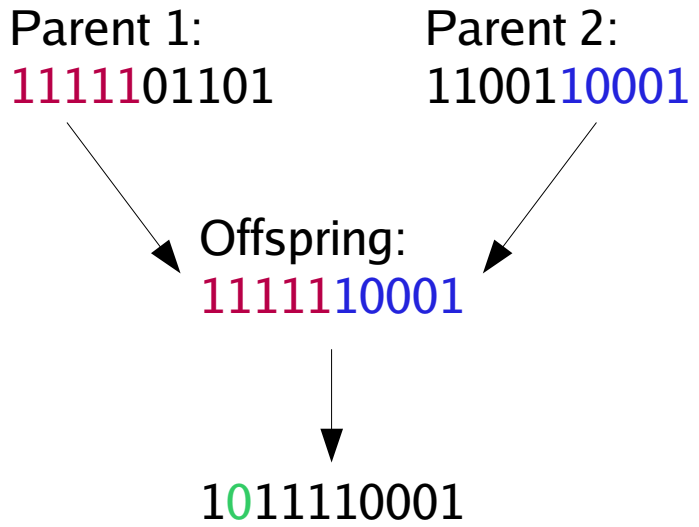
1. Generate a random initial population of chromosomes
2. Determine the fitness of each chromosome
3. Repeat until a new population is full:
    1. *Select* two parent chromosomes proportional to their fitness (stochastic step!)
    2. Apply *cross-over* to the parents to create offspring
    3. *Mutate* offspring
    4. Add offspring to new population
4. Exit IF
    1. Predetermined number of generations is reached, or
    2. Acceptable solution is found, or
    3. Population has converged;

   ELSE go to Step 2

3.21 A simple example

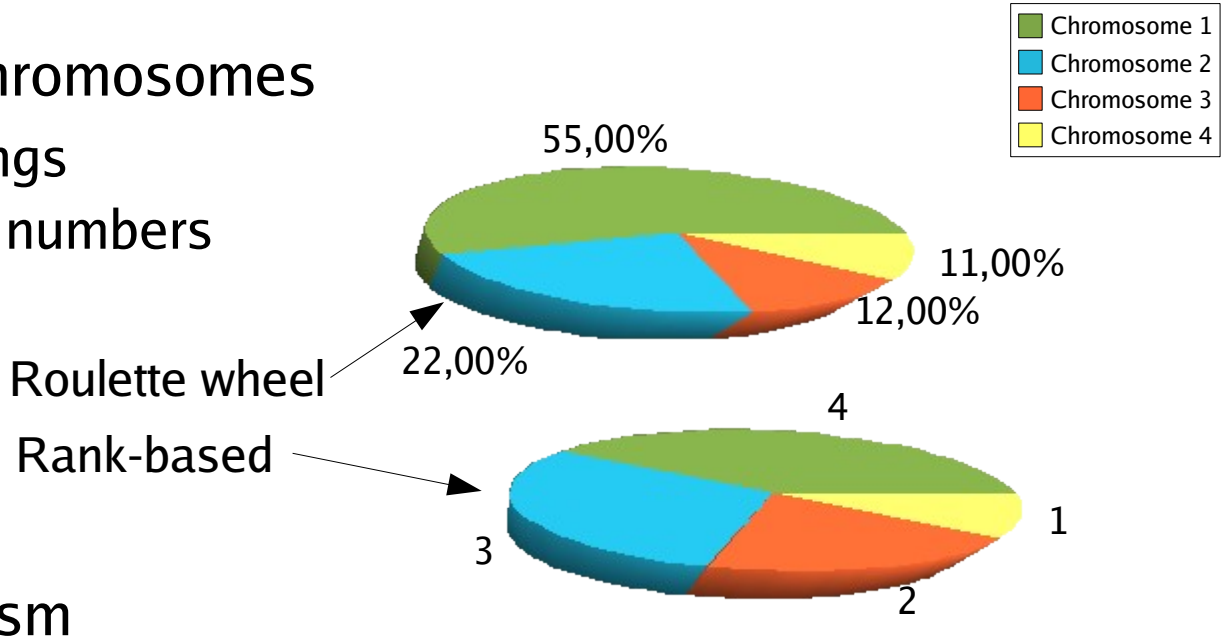
- Population of size 5
- Fitness (f) = number of 1's
- One-point cross-over:
- Mutation:
- Member of the new population

	f(n)
1100110001	5
0001000001	2
1010101010	5
1111101101	8
0000000001	1



3.22 Implementation

- Representation of chromosomes
  - Usually binary strings
  - Sequences of real numbers
  - Multi-dimensional
- Selection



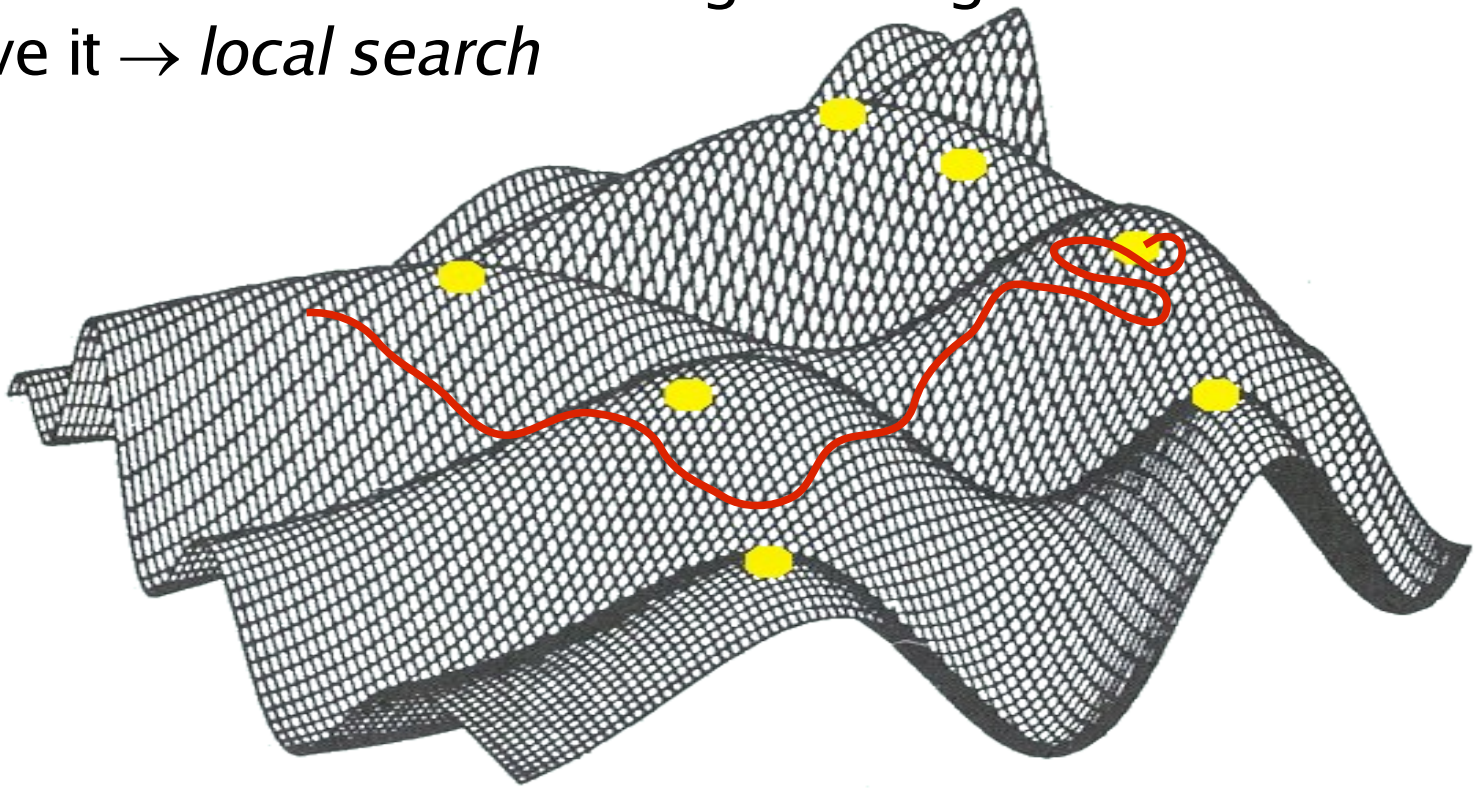
- Cross-over mechanism
  - One-point: 111000 + 000111 = 111111
  - Two-point: 111000 + 000111 = 110100
  - Uniform: 111000 + 000111 = 101101
- Cross-over and mutation rates
- Fitness function
  - Optimization of a mathematical function
  - Running a simulation and comparison to real data

The cross-over points can be selected randomly!

3.23 Exploration and exploitation

Exploration: Crossover — combine potentially good solutions to form even better ones → *global search*

Exploitation: Mutation — make small changes to a good solution to improve it → *local search*



Search space:
each point represents
a candidate solution

3.24 Pros and cons of GA

- Easy to implement, fun to explore
- May get stuck in a local optimum
- Slow
- Sensitive to
  - Problem representation
  - Implementation of genetic operations
  - Population size