Introduction to Artificial Intelligence

Problem definition, State Space, Uninformed Search

Ing. Tomas Borovicka

Department of Theoretical Computer Science (KTI), Faculty of Information Technology (FIT)
Czech Technical University in Prague (CVUT)

BIE-ZUM, LS 2014/15, 2. lecture

https://edux.fit.cvut.cz/courses/BIE-ZUM/

Problem Definition

- **Goal formulation**
  - Set of states that satisfies the agent (or maximizing some performance measure of the agent).
- **Problem formulation**
  - Process of deciding what actions and states to consider.

The Environment:

- Observable vs. partially observable vs. unobservable?
- Episodic vs. sequential?
- Discrete vs. continuous?
- Deterministic vs. stochastic?
- Static vs. dynamic?
- Known vs. unknown?

Problem Solving Agent

We assume that the environment is

- observable,
- discrete,
- deterministic,
- static.

- Under these assumptions, the solution to any problem can be expressed as a fixed **sequence of actions**.
- The process of looking for a sequence of actions that reaches the goal is called **search**.
- A search algorithm takes a problem as input and returns a solution in the form of an action sequence.
- Once a solution is found, the actions it recommends can be executed.

Definition of the State Space

- A problem can be defined by state space.
- The state-space is the configuration of the possible states and how they connect to each other e.g. the legal moves between states.

Definition (State Space)

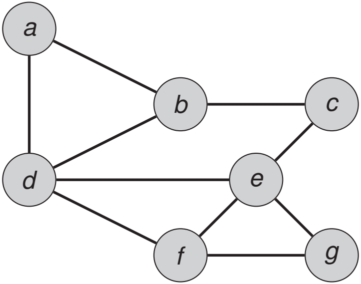
A state-space is a tuple $\langle S, A, \mathcal{I}, G \rangle$, such as (S, A) is a directed graph with a set of nodes S representing states and a set of edges A representing actions, $\mathcal{I}$ is then a set of initial states and G is a set of goal states.

- **State:** Some configuration of the problem.
- **Initial State:** The state in which the agent is at the beginning.
- **Actions:** Description of possible actions (moves) from the state s. Available actions might be a function of the state s $Actions(s) = \{a_1, \ldots\}$.
- **Goal State:** The state with certain properties (passes the a goal function).

A path in the state space is a sequence of states connected by a sequence of actions.

Graph Theory

- The graph theory is a branch of mathematics highly utilized in computer science. However the graph theory has a myriad of applications in many different domains.
- It is a study of graphs, where **graph** is a structure formed by **vertices** and **edges** connecting the vertices.
- Maps, websites, communication networks, electrical circuits, program structures, etc are practical examples of graphs.



Undirected Graphs

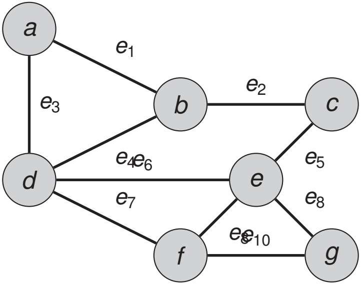
Definition (Undirected Graph)

Graph is a pair of sets (V, E) where V is the set of vertices and E is the set of edges, $E \subseteq \{\{u, v\} | u, v \in V\}$.

Example:

$V = \{a, b, c, d, e, f, g\}$

$E = \{\{a, b\}, \{b, c\}, \{a, d\}, \{b, d\}, \{c, e\}, \{d, e\}, \{d, f\}, \{e, f\}, \{e, g\}, \{f, g\}\}$

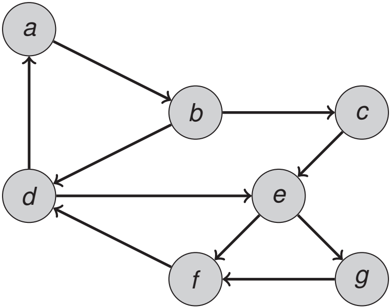


Directed Graphs

Definition (Directed Graph)

Directed graph is a pair of sets (V, E) where V is the set of vertices and E is the set of edges, such as $E \subseteq \{(u, v) | u, v \in V\}$, where (u, v) is an ordered pair, i.e. $(u, v) \neq (v, u)$.

- Each edge has a direction, i.e. (u, v) is an edge directed from node u to node v.



Graph Terminology 1

- A **walk** in the graph $G = (V, E)$ is a finite sequence $v_0, e_1, v_1, \ldots, v_k$ of vertices v_i and edges e_i such that for $1 <= i <= k$, the edge e_i has endpoints $v_{(i-1)}$ and v_i.
  - if $v_0 \neq v_k$ it is open walk, otherwise it is closed.
- A **trail** is a walk $v_0, e_1, v_1, \ldots, v_k$ with no repeated edge.
- A trail is a **path** if any vertex is visited at most once except possibly the initial and terminal vertices.
  - A **Hamiltonian path** is a path that includes all vertices of G.
- A **cycle** (or circuit) of a graph G is a subset of the edge set of E that forms a path such that the first node of the path corresponds to the last i.e. $v_1 = v_k$.
  - A cycle that uses each vertex of a graph exactly once is called a **Hamiltonian cycle**.

Similarly for directed graphs...

Graph Terminology 2

- Having directed graph $G = (V, E)$, if there is an edge from vertex u to vertex v, then u is a (direct) **predecessor** of v and v is a (direct) **successor** of u. We denote $\Gamma(u)$ as a **set of successors** of vertex u.
  $\Gamma(u) = \{v \in V | (u, v) \in E\}$.
- Connected acyclic graph is called a **tree** (can be directed or undirected).
- A tree with one special singled out vertex labeled "root" is called a **rooted tree**.
- A vertex in a directed tree such that $\Gamma(u) = \{ \}$ is called a **leaf**.

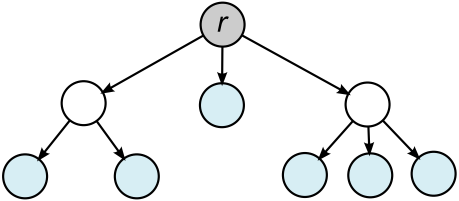


Figure: The Rooted Tree (with blue colored leafs)

Route Finding

The goal is to find a path from city *A* to city *B*.

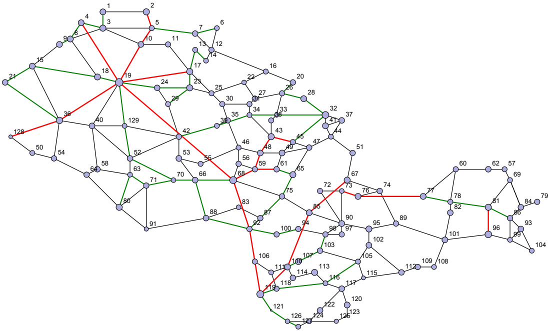
States: The city we are currently in.

Initial State: The city in which we start.

Actions: Moving from current city to adjacent cities.

Goal: City we want to reach.

- Intuitive State Space is then the road map:



Sliding-block Puzzles

Game consists of $N \times N$ board with $N - 1$ numbered tiles and a blank space. A tile adjacent to the blank space can slide into it. The goal is to reach specified order.

States: Description of the location of each tile and the blank square.

Initial State: Initial configuration of the puzzle.

Actions: Moving the blank left, right, up, or down.

Goal: Predefined configuration of the puzzle (ordering).

Example for $N = 4$:

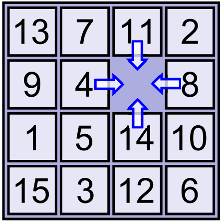


Figure: The Initial State

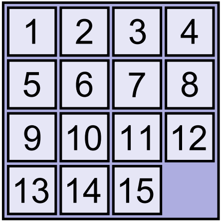
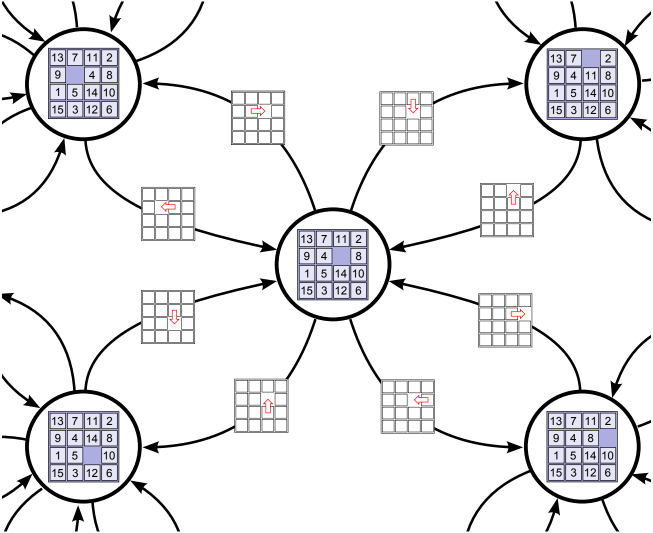
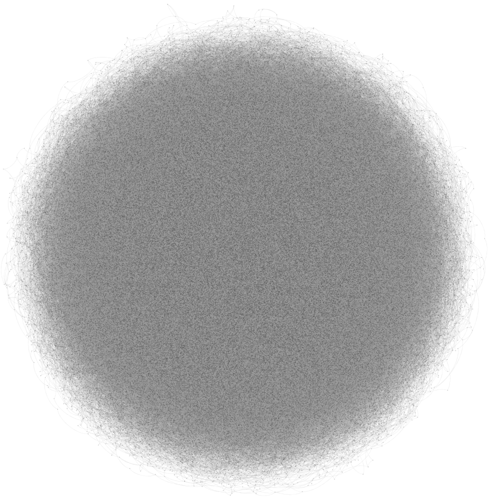


Figure: The Goal State

Sliding-block Puzzles: State Space $N = 4$



Sliding-block Puzzles: Complete State Space $N = 3$



State space for $N = 4$ is $5 \cdot 10^7$ times bigger!

8-Queens Problem

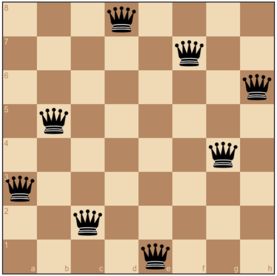
The goal is to place eight queens on a chessboard such that no queen attacks any other.

States: Any arrangement of 0 to 8 queens on the board.

Initial State: No queens on the board.

Actions: Add a queen to any square.

Goal: 8 queens are on the board, none attacked.



8-Queens Problem: Revision

Is the problem well formulated?

Any queen anywhere

$\approx 2.8 \cdot 10^{14}$ states $(n^2)^n$

Queens on different squares

$\approx 1.8 \cdot 10^{14}$ states $n^2!/(n^2 - n)!$

Queens in separate columns

$\approx 1.7 \cdot 10^{7}$ states n^n

Queens in separate cols, rows

40320 states $n!$

Searching the State Space

We have formulated some problems, how to solve them?

- A solution is a sequence of actions.
  - We are interested in a path to the goal state or the goal state itself.
- Search algorithm is searching for solution by applying each legal action to the current state. This is called **expansion**.
- How to choose which state to expand next is called **search strategy**.
- All possible action sequences form a **search tree** with the root corresponding to initial state.

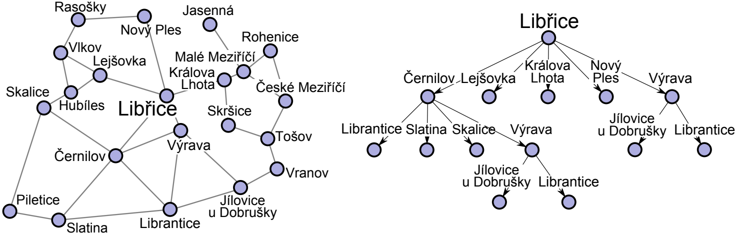
Search Tree

- Initial state at the root node.
- Edges are labeled with actions.
- Each node in the search tree corresponds to some path in the state graph.

Definition (The Search Tree)

Having state space $X = \langle S, A, \mathcal{I}, G \rangle$ with one initial state s_0, the search tree is a directed rooted tree with root s_0 such that each path in the tree exists in the graph (S, A).

Example of state graph and search tree for route finding problem:



Loops: Closed List

Algorithms that forget their history are doomed to repeat it!

We prevent exploring redundant paths by remembering already expanded nodes in **closed list**.

Each node can be in one of the following states:

- **FRESH** – not explored node,
- **OPEN** – explored, but not expanded node,
- **CLOSED** – expanded node.

Intuitively, already opened or closed nodes can be discarded from further exploration.

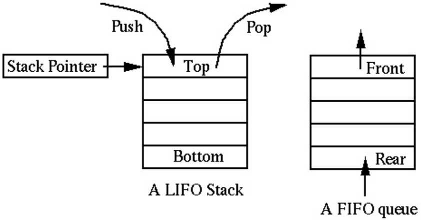
Implementation: Queues

Queue is a data structure to store elements (nodes). For queue is characterizing the order in which it stores the elements.

A FIFO queue , First-In First-Out.
Operations: enqueue, dequeue

A LIFO stack , Last-In First-Out.
Operations: push, pop, top

Priority queue pops the element of the queue according to some ordering function.



Search Strategy

Uninformed Search:

- no additional information about the states,
- goal-state or non-goal state.

Search strategy performance

- **Completeness:** If solutions exists does the strategy guarantee to find it?
- **Optimality:** Does the strategy find the optimal solution?
- **Time complexity:** How much time it takes?
- **Space complexity:** How much memory is needed?

For complexity estimation: **branching factor** b is the maximum number of successors of any node, d is the **depth** of the shallowest goal node, m is the maximum length of any path in the state space.

Informed (heuristic) search next lecture...

Random Search

- The simplest search algorithm.
- Algorithm chooses randomly the next node to be expanded.

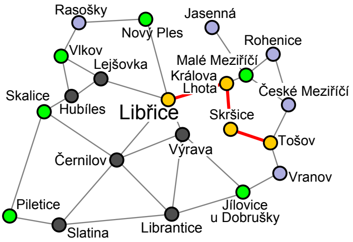
Complete: No

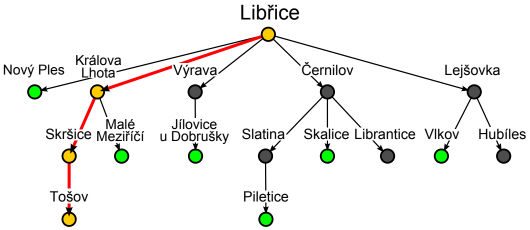
Optimal: No

Time: $O(b^d)$

Space: $O(b^d)$

Random search: Example





FRESH OPEN CLOSED INIT/GOAL EXPAND

Algorithm 1 Random search

```
open ← 𝓘;  closed ← {}
prev ← init_table()
while open ≠ {} do
    x ← random element of open
    if x ∈ G then
        return reconstruct_path(prev, x)
    end if
    for all y ∈ neighbors(x) do
        if y ∉ (open ∪ closed) then
            open ← open ∪ {y}
            prev[y] ← x
        end if
    end for
    open ← open \ {x};  closed ← closed ∪ {x}
end while
```

Breadth-First Search (BFS)

Simple strategy that expanding the search tree level by level. First, root node is expanded, then all the successors of root, then all their successors etc.

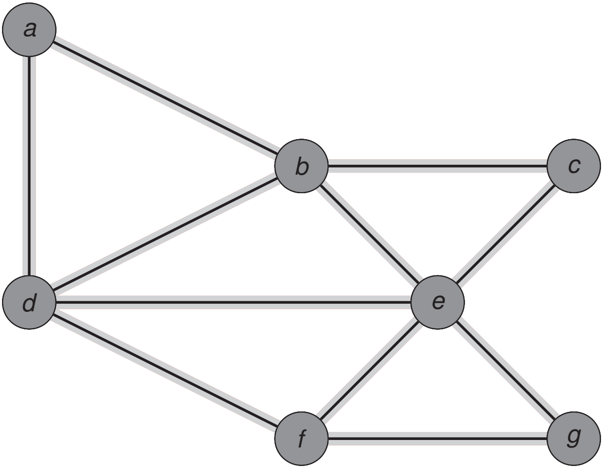
- Implemented using FIFO queue.

Complete: Yes

Optimal: Yes

Time: $O(b^d)$

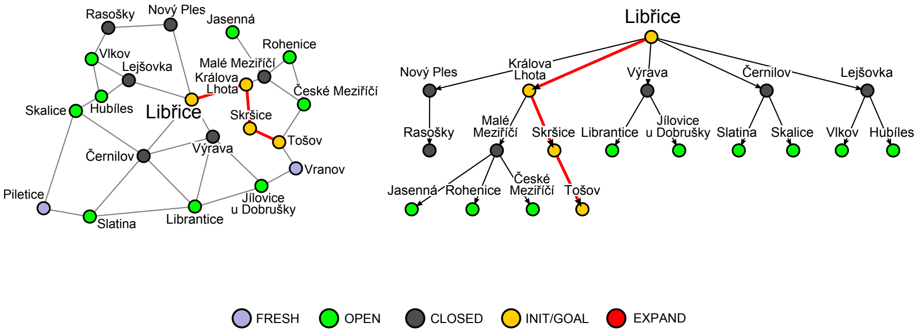
Space: $O(b^d)$


a
b
c
d
e
f
g

Algorithm 2 Breadth-First Search (BFS)

```
open ← init_queue(); closed ← {}
prev ← init_table()
for all s ∈ I do
    enqueue(open, s)
end for
while ¬empty(open) do
    x ← dequeue(open)
    if x ∈ G then
        return reconstruct_path(prev, x)
    end if
    for all y ∈ neighbors(x) do
        if y ∉ (open ∪ closed) then
            enqueue(open, y); prev[y] ← x
        end if
    end for
    closed ← closed ∪ {x}
```

Breadth-first search (BFS): Example



Depth-First Search

Idea of DFS is to expand in each step current node to travel as deep as possible.

- Implemented using LIFO stack.

Complete: No

Optimal: No

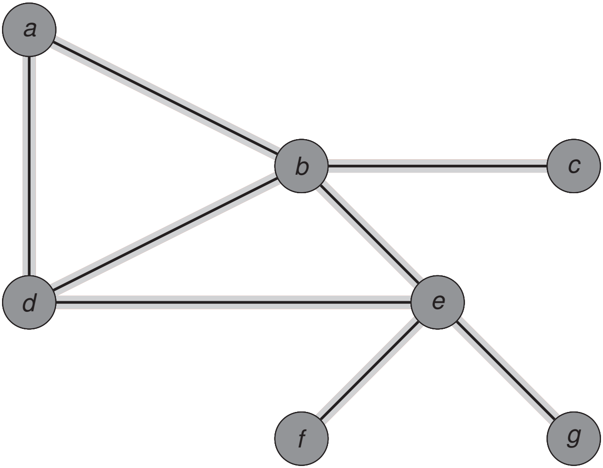
Time: $O(b^m)$

Space: $O(bm)$

Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored.

Extensions:

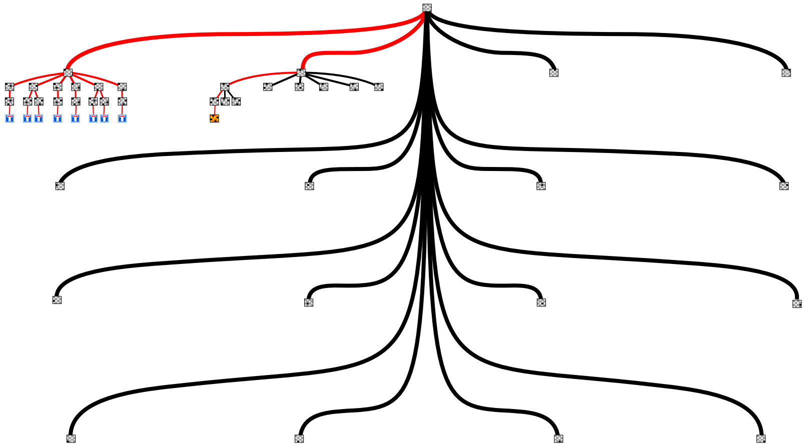
- Depth Limited Search
- Iterative Deepening Search (Complete, same time complexity as BFS, but space effective)


a
b
c
d
e
f
g

Algorithm 3 Depth-First Search (DFS)

```
open ← init_stack(); closed ← {}
prev ← init_table()
for all s ∈ I do
    push(open, s)
end for
while ¬empty(open) do
    x ← pop(open)
    if x ∈ G then
        return reconstruct_path(prev, x)
    end if
    for all y ∈ neighbors(x) do
        if y ∉ (open ∪ closed) then
            push(open, y); prev[y] ← x
        end if
    end for
    closed ← closed ∪ {x}
```

Depth-first search (DFS): 4 Queens problem



Path Reconstruction Algorithm

- Same for all the algorithms.
- Each node remembers its predecessor.

Algorithm 4 reconstruct_path(*prev*, *goal*)

```
x ← goal
path ← init_list()
while x ≠ NULL do
    append(path, x)
    x ← prev[x]
end while
return path
```