Machine Intelligence

2. Background for Search-based Methods

A Reminder of Some Graph-Related Terminology and Basic Algorithms

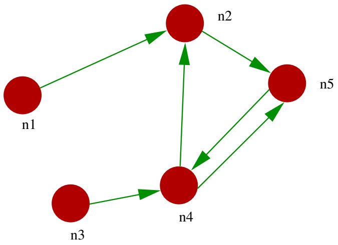
Álvaro Torralba

AALBORG UNIVERSITET

Fall 2022

Thanks to Thomas D. Nielsen and Jörg Hoffmann for slide sources

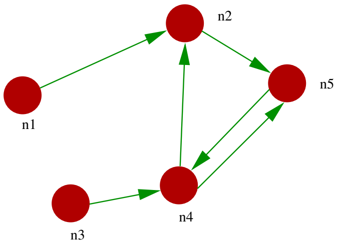
Graphs



A **directed graph** consists of

- a set of **nodes** (also called vertices)
- a set of **arcs** (ordered pairs of nodes) (also called edges)

Graphs



A **directed graph** consists of

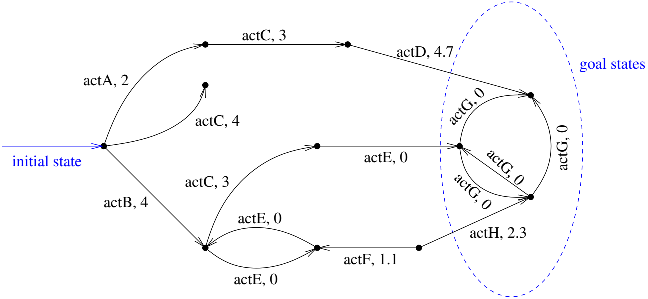
- a set of **nodes** (also called vertices)
- a set of **arcs** (ordered pairs of nodes) (also called edges)

Further terminology:

- n_2 is a **neighbor/successor** of n_4 (not the other way round!).
- n_3, n_4, n_2, n_5 is a **path** from n_3 to n_5.
- n_2, n_5, n_4, n_2 is a path that is a **cycle**.
- a graph is **acyclic** if it has no cycles.

Directed Labeled and Weighted Graphs

- Directed labeled graphs: each edge has a label (string) associated to it
- Directed weighted graphs: each edge has a weight (number) associated to it



Graph Search Terminology

Shortest Path on Graph: Given a graph (V, E), a vertex s^I, and a set of vertices $S^G \subseteq V$, what is the shortest path from s^I to any vertex in S^G?

Some commonly used terms:

- s' successor of s if $s \rightarrow s'$; s predecessor of s' if $s \rightarrow s'$.

Graph Search Terminology

Shortest Path on Graph: Given a graph (V, E), a vertex s^I, and a set of vertices $S^G \subseteq V$, what is the shortest path from s^I to any vertex in S^G?

Some commonly used terms:

- s' successor of s if $s \to s'$; s predecessor of s' if $s \to s'$.
- s' reachable from s if there exists a sequence of transitions:

$$s = s_0 \xrightarrow{a_1} s_1, \ldots, s_{n-1} \xrightarrow{a_n} s_n = s'$$

  - $n = 0$ possible; then $s = s'$.
  - $a_1, \ldots, a_n$ is called path from s to s'.
  - $s_0, \ldots, s_n$ is also called path from s to s'.
  - The cost of that path is $\sum_{i=1}^{n} c(a_i)$.

Graph Search Terminology

Shortest Path on Graph: Given a graph (V, E), a vertex s^I, and a set of vertices $S^G \subseteq V$, what is the shortest path from s^I to any vertex in S^G?

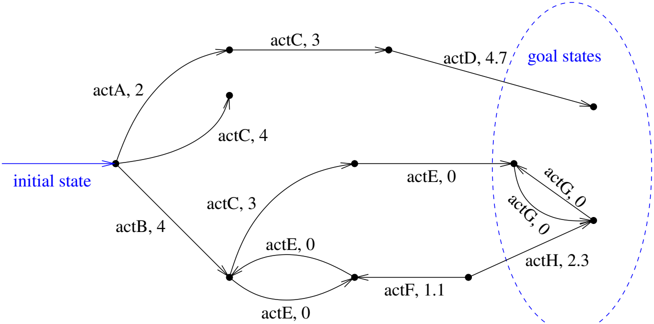
Some commonly used terms:

- s' successor of s if $s \to s'$; s predecessor of s' if $s \to s'$.
- s' reachable from s if there exists a sequence of transitions:

$$s = s_0 \xrightarrow{a_1} s_1, \ldots, s_{n-1} \xrightarrow{a_n} s_n = s'$$

  - $n = 0$ possible; then $s = s'$.
  - $a_1, \ldots, a_n$ is called path from s to s'.
  - $s_0, \ldots, s_n$ is also called path from s to s'.
  - The cost of that path is $\sum_{i=1}^{n} c(a_i)$.
- s' reachable (without reference state) means reachable from s^I.
- s is solvable if some $s' \in S^G$ is reachable from s; else, s is a dead end.

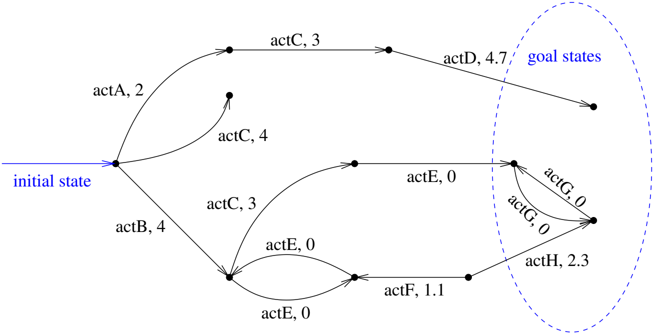
Directed labeled graphs + mark-up for initial state and goal states:



- Are all states in Θ reachable?

Examples

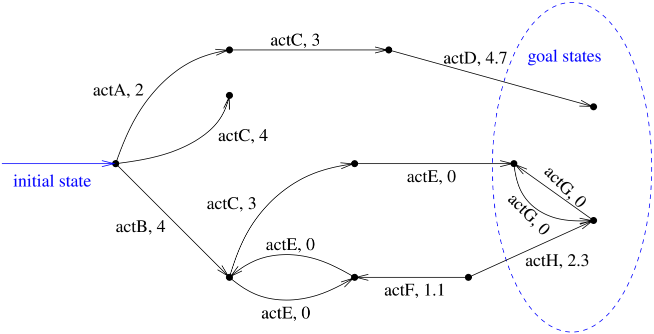
Directed labeled graphs + mark-up for initial state and goal states:



- Are all states in Θ reachable? No: state at bottom, 2nd from right.
- Are all states in Θ solvable?

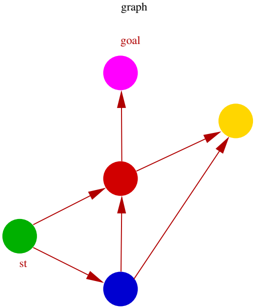
Examples

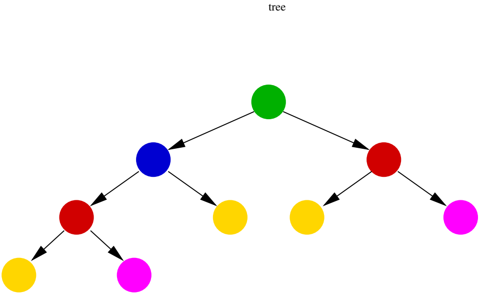
Directed labeled graphs + mark-up for initial state and goal states:



- Are all states in Θ reachable? No: state at bottom, 2nd from right.
- Are all states in Θ solvable? No: state near top, 2nd from left.

From Graph to Search Tree





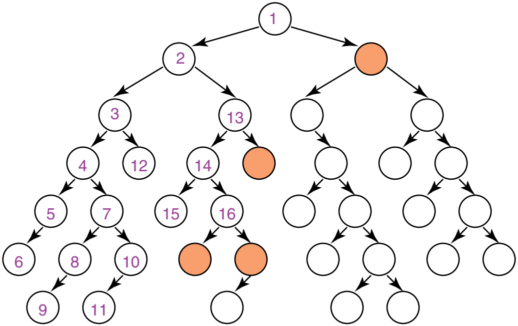
- Tree: special graph with
  - exactly one node that has no incoming arc (the **root**)
  - all other nodes have exactly one incoming arc (may have 0,1,2,3,...outgoing arcs)
- Nodes in the search tree correspond to paths in the graph beginning in the start state
- Nodes in the search tree also correspond to a graph vertex: the last vertex of the path
- Search strategy: In which order our algorithm explores the search tree?

Depth-first search

- Select from the frontier that path that was most recently added to the frontier (frontier implemented as **stack**).

Example

- Explored nodes with order of exploration
- Frontier (colored)
- Unexplored nodes



Properties

- Space used is linear in the length of the current path.
- May not terminate if state-space graph has cycles
- With a forward branching factor bounded by b and depth n, the worst-case time complexity of a finite tree is b^n.

Guarantees:

- Optimality: No. After all, the algorithm just "chooses some direction and hopes for the best". (Depth-first search is a way of "hoping to get lucky".)
- Completeness: No, because search branches may be infinitely long: No check for cycles along a branch!

  $\rightarrow$ Depth-first search is complete in case the state space is acyclic. If we do add a cycle check, it becomes complete.

Complexity:

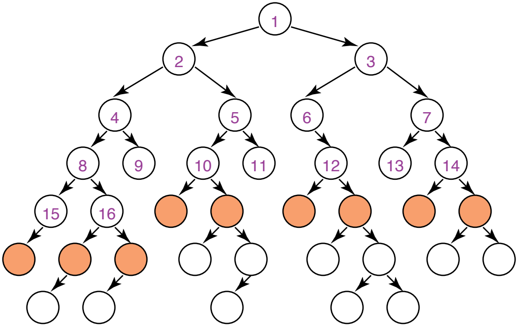
- Space: Stores nodes and applicable actions on the path to the current node. So if m is the maximal depth reached, the complexity is $O(b\,m)$.
- Time: If there are paths of length m in the state space, $O(b^m)$ nodes can be generated. Even if there are solutions of depth 1!

  $\rightarrow$ If we happen to choose "the right direction" then we can find a length-l solution in time $O(b\,l)$ regardless how big the state space is.

- Select from the frontier that path that was earliest added to the frontier (frontier implemented as **queue**).

Example

- Explored nodes with order of exploration
- Frontier (colored)
- Unexplored nodes



Properties

- Completeness: Yes, will always find a solution if one exists
- Optimality: Yes, for unit action costs (graphs where all edges have the same weight). Breadth-first search always finds the shortest path in terms of number of edges. If edges have weight, this is not necessarily optimal.
- Size of frontier always increases during search up to order of magnitude of total size of search tree.

Properties

- Completeness: Yes, will always find a solution if one exists
- Optimality: Yes, for unit action costs (graphs where all edges have the same weight). Breadth-first search always finds the shortest path in terms of number of edges. If edges have weight, this is not necessarily optimal.
- Size of frontier always increases during search up to order of magnitude of total size of search tree.
- Can be adapted to find a minimum cost path.

Time Complexity: Say that b is the maximal branching factor, and d is the goal depth (depth of shallowest goal state).

- Upper bound on the number of generated nodes: $b + b^2 + b^3 + \cdots + b^d$: In the worst case, the algorithm generates all nodes in the first d layers.
- So the time complexity is $O(b^d)$.
- And if we were to apply the goal test at node-expansion time, rather than node-generation time: $O(b^{d+1})$ because then we'd generate the first $d + 1$ layers in the worst case.

Space Complexity: Same as time complexity since all generated nodes are kept in memory.

Breadth-First Search: Example Data

Setting: $b = 10$; 10000 nodes/second; 1000 bytes/node.

Yields data: (inserting values into previous equations)

Depth	Nodes	Time		Memory	
2	110	.11	milliseconds	107	kilobytes
4	11110	11	milliseconds	10.6	megabytes
6	10^6	1.1	seconds	1	gigabyte
8	10^8	2	minutes	103	gigabytes
10	10^{10}	3	hours	10	terabytes
12	10^{12}	13	days	1	petabyte
14	10^{14}	3.5	years	99	petabytes

→ The critical resource here is memory. (In my own experience, breadth-first search typically exhausts RAM within a few minutes.)

Dijkstra's Algorithm

During the lecture, I will mention Dijkstra's algorithm, which should have been part of a previous lecture.

Please, check out the Algorithms book or the following references to refresh the basics on how the algorithm works:

- `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`
- `https://algorithms.discrete.ma.tum.de/graph-algorithms/spp-dijkstra/index_en.html`