Chapter 3: Classical search algorithms

DIT410/TIN173 Artificial Intelligence

Peter Ljunglöf

(inspired by slides by Poole & Mackworth, Russell & Norvig, et al)

23 March, 2016

Outline

1. *Introduction (Russell & Norvig 3.1–3.3)*
    - Graphs and searching
    - Examples
    - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
    - Depth-first search
    - Breadth-first search
    - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
    - Greedy best-first search
    - A^* search
    - Admissible and consistent heuristics

Outline

1. *Introduction (Russell & Norvig 3.1–3.3)*
   - Graphs and searching
   - Examples
   - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
   - Depth-first search
   - Breadth-first search
   - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
   - Greedy best-first search
   - A^* search
   - Admissible and consistent heuristics

Outline

1. *Introduction (Russell & Norvig 3.1–3.3)*
   - Graphs and searching
   - Examples
   - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
   - Depth-first search
   - Breadth-first search
   - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
   - Greedy best-first search
   - A^* search
   - Admissible and consistent heuristics

Searching

- Often we are not given an algorithm to solve a problem, but only a specification of a solution — we have to search for it.
- A typical problem is when the agent is in one state, it has a set of deterministic actions it can carry out, and wants to get to a goal state.
- Many AI problems can be abstracted into the problem of finding a path in a directed graph.
- Often there is more than one way to represent a problem as a graph.

State-space search: Complexity dimensions

Observable?	fully
Deterministic?	deterministic
Episodic?	episodic
Static?	static
Discrete?	discrete
Number of agents	single

State-space search: Complexity dimensions

Observable?	fully
Deterministic?	deterministic
Episodic?	episodic
Static?	static
Discrete?	discrete
Number of agents	single

More complex problems (partly ob., stochastic, sequential, . . .) usualy have components using state-space search.

Directed Graphs

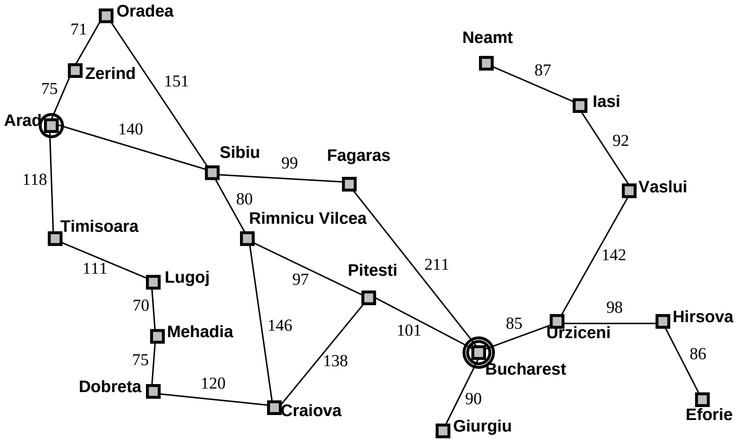
- A graph consists of a set N of nodes and a set A of ordered pairs of nodes, called arcs.
- Node n_2 is a neighbor of n_1 if there is an arc from n_1 to n_2. That is, if $\langle n_1, n_2 \rangle \in A$.
- A path is a sequence of nodes $\langle n_0, n_1, \ldots, n_k \rangle$ such that $\langle n_{i-1}, n_i \rangle \in A$.
- The length of path $\langle n_0, n_1, \ldots, n_k \rangle$ is k.
- A solution is a path from a start node to a goal node, given a set of start nodes and goal nodes.
- (Russel & Norvig sometimes call the graph nodes states).

Outline

1. *Introduction (Russell & Norvig 3.1–3.3)*
   - Graphs and searching
   - Examples
   - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
   - Depth-first search
   - Breadth-first search
   - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
   - Greedy best-first search
   - A^* search
   - Admissible and consistent heuristics

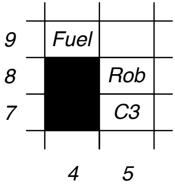
Example problem: Travel in Romania

We want to drive from Arad to Bucharest in Romania.



Example problem: Grid game

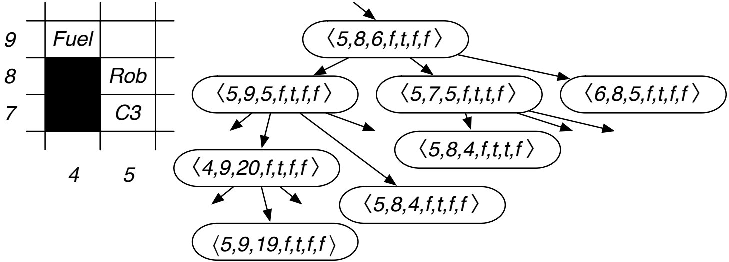
Grid game: Rob needs to collect coins C_1, C_2, C_3, C_4, without running out of fuel, and end up at location $(1, 1)$:



What is a good representation of the search states and the goal?

Grid game: State representation

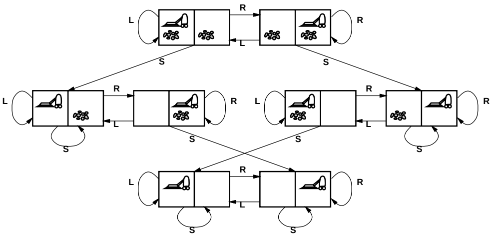
Grid game: Rob needs to collect coins C_1, C_2, C_3, C_4, without running out of fuel, and end up at location $(1, 1)$:



State:
⟨X-pos,Y-pos,Fuel,C1,C2,C3,C4⟩

Goal:
⟨1,1,?,t,t,t,t⟩

Example: A vacuum-cleaning agent



States:

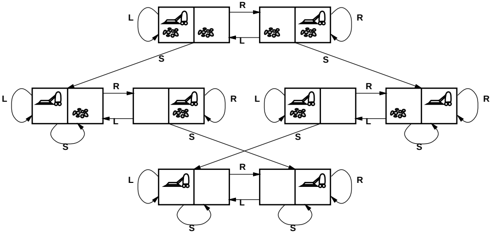
Initial state:

Actions:

Goal test:

Path cost:

Example: A vacuum-cleaning agent



States: tuple [*room A dirty?, room B dirty?, robot location*]

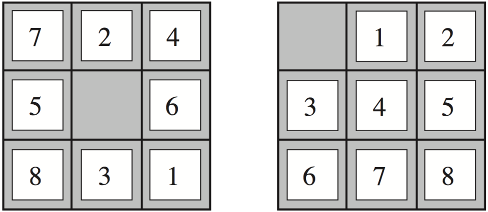
Initial state: any state

Actions: left, right, suck, do-nothing

Goal test: [*false*, *false*, –]

Path cost: 1 per action (0 for do-nothing)

Example: The 8-puzzle



Start State Goal State

States:

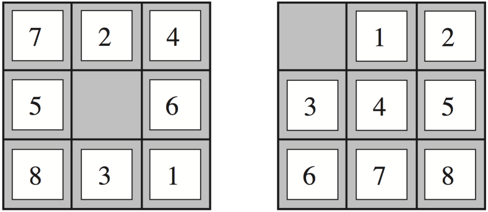
Initial state:

Actions:

Goal test:

Path cost:

Example: The 8-puzzle



Start State Goal State

States: a 3×3 matrix of integers $0 \leq n \leq 8$

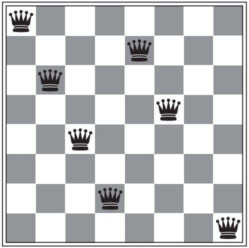
Initial state: any state

Actions: move the blank space: left, right, up, down

Goal test: equal to goal state (given above)

Path cost: 1 per move

Example: The 8-queens problem



States:

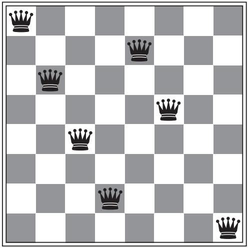
Initial state:

Actions:

Goal test:

Path cost:

Example: The 8-queens problem



States: any arrangement of 0 to 8 queens on the board

Initial state: no queens on the board

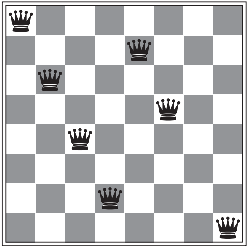
Actions: add a queen to any empty square

Goal test: 8 queens on the board, none attacked

Path cost: 1 per move

This gives us $64 \cdot 63 \cdot \cdots \cdot 57 \approx 1.8 \times 10^{14}$ possible sequences to explore!

Example: The 8-queens problem (alternative)



States:

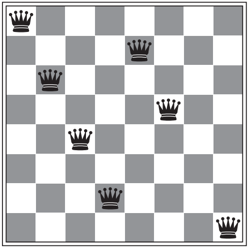
Initial state:

Actions:

Goal test:

Path cost:

Example: The 8-queens problem (alternative)



States: one queen per column in leftmost columns

Initial state: no queens on the board

Actions: add a queen to any square in the leftmost empty column, making sure that no queen is attacked

Goal test: 8 queens on the board, none attacked

Path cost: 1 per move

Using this formulation, we have only 2,057 sequences!

Example: Knuth's conjecture

Donald Knuth has conjectured that every positive integer can be obtained by beginning with the number 4 and applying some combination of the factorial, square root, and floor functions.

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

States: positive numbers $(1, 2, 2.5, 3, \pi, \ldots)$

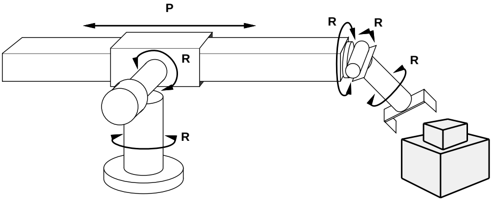
Initial state: 4

Actions: apply factorial, square root, or floor operation

Goal test: any positive integer (e.g., 5)

Path cost: 1 per move

Example: robotic assembly



States: real-valued coordinates of robot joint angles
parts of the object to be assembled

Actions: continuous motions of robot joints

Goal test: complete assembly of the object

Path cost: time to execute

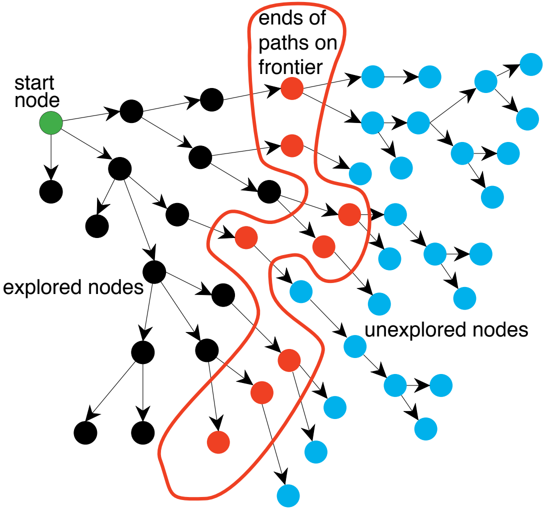
Outline

1. *Introduction (Russell & Norvig 3.1–3.3)*
   - Graphs and searching
   - Examples
   - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
   - Depth-first search
   - Breadth-first search
   - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
   - Greedy best-first search
   - A^* search
   - Admissible and consistent heuristics

How do we search in a graph?

- Generic search algorithm: given a graph, start nodes, and a goal description, incrementally explore paths from the start nodes.
- Maintain a *frontier* of nodes that have been explored.
- As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.
- The way in which the frontier is expanded defines the *search strategy*.

Illustration of searching in a graph



A generic tree search algorithm

```
procedure Search(graph, initial-state, goal-state):
    initialise frontier using the initial-state

    while frontier is not empty:
        select and remove node from frontier
        if node.state is a goal-state then return node

        for each child in ExpandChildNodes(node, graph):
            add child to frontier

    return failure
```

Turning tree search into graph search

procedure Search(*graph*, *initial-state*, *goal-state*):
    initialise *frontier* using the *initial-state*
    initialise *explored-set* to the empty set
    while *frontier* is not empty:
        select and **remove** *node* from *frontier*
        if *node.state* is a *goal-state* **then return** *node*
        add *node* **to** *explored-set*
        for each *child* **in** ExpandChildNodes(*node*, *graph*):
            add *child* **to** *frontier*
                if *child* **not in** *frontier* **or** *explored-set*
    return *failure*

Graph nodes vs. search nodes

The nodes used while searching are not the same as the graph nodes – search nodes contain more information:

- the corresponding graph node (called *state* in R&N)
- the total path cost from the start node
- the parent node – for building the final path
- the action that was used to get here from the parent node

procedure ExpandChildNodes(*parent*, *graph*):
    return { new Node(state = *result*,
                cost = $parent.cost + stepcost$,
                parent = *parent*,
                action = *action*)
        for each (*action*, *result*, *stepcost*)
                $\in graph.successors(parent.state)$
        }

Outline

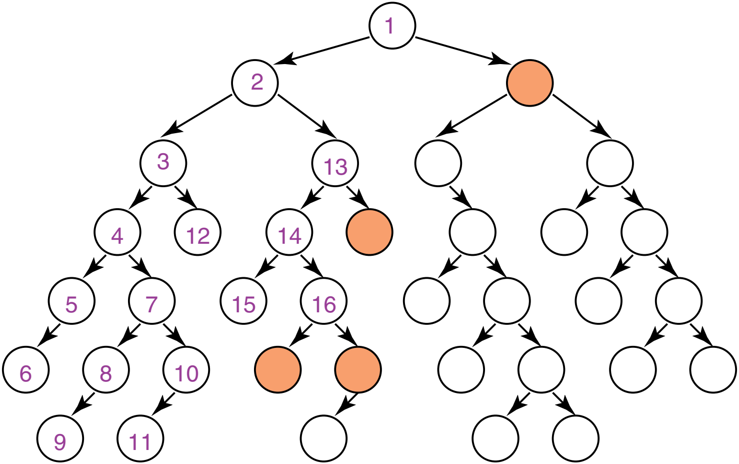
1. *Introduction (Russell & Norvig 3.1–3.3)*
   - Graphs and searching
   - Examples
   - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
   - Depth-first search
   - Breadth-first search
   - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
   - Greedy best-first search
   - A^* search
   - Admissible and consistent heuristics

Outline

Depth-first search

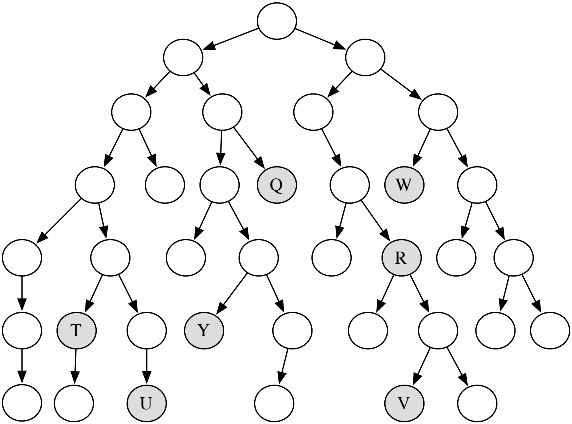
- Depth-first search treats the frontier as a stack.
- It always selects one of the last elements added to the frontier.
- If the list of nodes on the frontier is $[p_1, p_2, p_3, \ldots]$
  - p_1 is selected. Nodes that extend p_1 are added to the front of the stack (in front of p_2).
  - p_2 is only selected when all nodes from p_1 have been explored.

Illustrative graph: Depth-first search



Question time: Depth-first search

Which shaded goal will a depth-first search find first?



Complexity of depth-first search

- Does DFS guarantee to find the path with fewest arcs?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of the path length?
- What is the space complexity as a function of the path length?
- How does the goal affect the search?

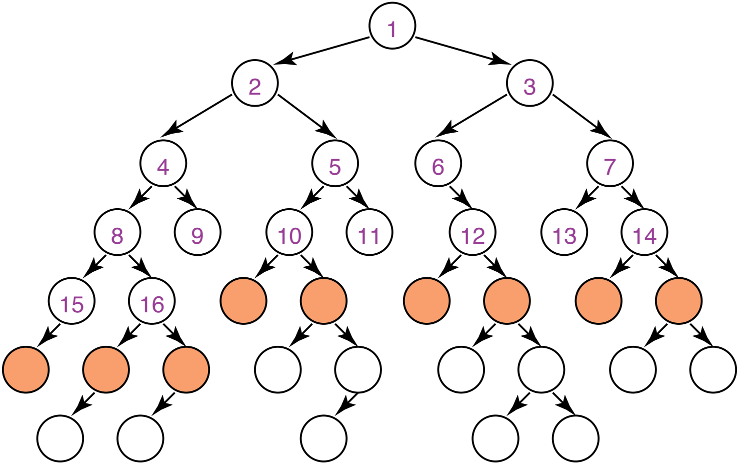
Outline

1. *Introduction (Russell & Norvig 3.1–3.3)*
   - Graphs and searching
   - Examples
   - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
   - Depth-first search
   - Breadth-first search
   - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
   - Greedy best-first search
   - A^* search
   - Admissible and consistent heuristics

Breadth-first search

- Breadth-first search treats the frontier as a queue.
- It always selects one of the earliest elements added to the frontier.
- If the list of paths on the frontier is $[p_1, p_2, \ldots, p_r]$:
  - p_1 is selected. Its neighbors are added to the end of the queue, after p_r.
  - p_2 is selected next.

Illustrative graph: breadth-first search

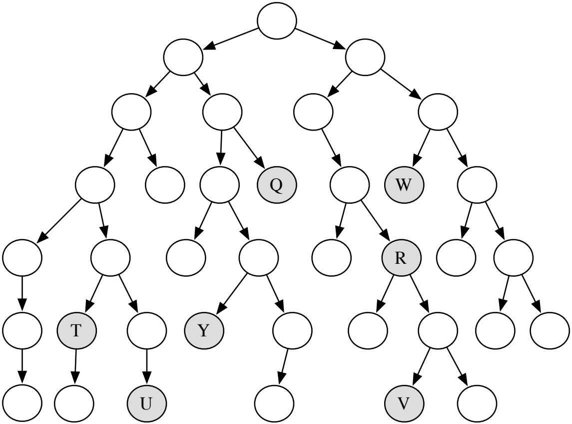


Complexity of breadth-first search

- Does BFS guarantee to find the path with fewest arcs?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of the path length?
- What is the space complexity as a function of the path length?
- How does the goal affect the search?

Question time: Breadth-first search

Which shaded goal will a breadth-first search find first?



Outline

1. *Introduction (Russell & Norvig 3.1–3.3)*
   - Graphs and searching
   - Examples
   - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
   - Depth-first search
   - Breadth-first search
   - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
   - Greedy best-first search
   - A^* search
   - Admissible and consistent heuristics

Uniform-cost search

- Sometimes there are costs associated with arcs. The cost of a path is the sum of the costs of its arcs.

$$cost(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} |\langle n_{i-1}, n_i \rangle|$$

  An optimal solution is one with minimum cost.
- At each stage, uniform-cost search selects a path on the frontier with lowest cost.
- The frontier is a priority queue ordered by path cost.
- It finds a least-cost path to a goal node.
  - i.e., uniform-cost search is optimal
- When arc costs are equal $\Longrightarrow$ breadth-first search.

Outline

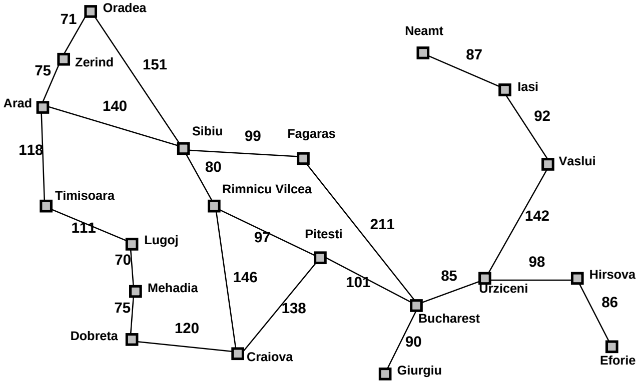
Heuristic search

- Idea: don't ignore the goal when selecting paths.
- Often there is extra knowledge that can be used to guide the search: heuristics.
- $h(n)$ is an estimate of the cost of the shortest path from node n to a goal node.
- $h(n)$ needs to be efficient to compute.
- $h(n)$ is an underestimate if there is no path from n to a goal with cost less than $h(n)$.
- An admissible heuristic is a nonnegative heuristic function that is an underestimate of the actual cost of a path to a goal.

Example heuristic functions

- If the nodes are points on a Euclidean plane and the cost is the distance, $h(n)$ can be the straight-line distance (SLD) from n to the closest goal.
- If the nodes are locations and cost is time, we can use the distance to a goal divided by the maximum speed, $h(n) = d(n)/v_{\mathrm{max}}$.
- If the goal is to collect all of the coins and not run out of fuel, we can use an estimate of how many steps it will take to collect the rest of the coins and return to goal position, without caring about the fuel consumption.
- A heuristic function can be found by solving a simpler (less constrained) version of the problem.

Example heuristic: Romania



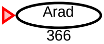
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Outline

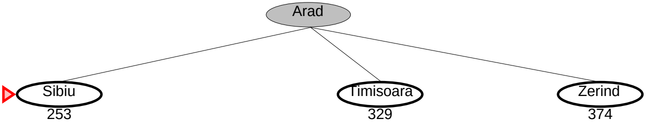
Greedy best-first search

- Idea: select the path whose end is closest to a goal according to the heuristic function.
- Best-first search selects a path on the frontier with minimal h-value.
- It treats the frontier as a priority queue ordered by h.

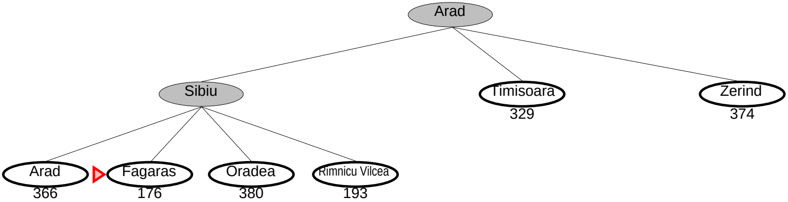
Example: Romania



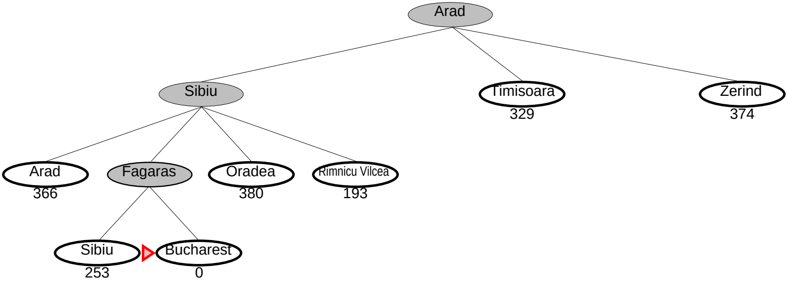
Example: Romania



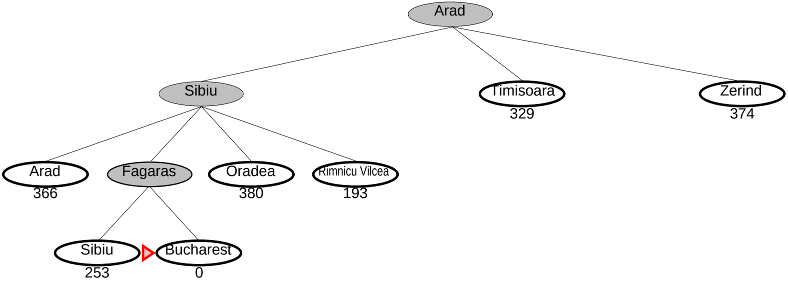
Example: Romania



Example: Romania

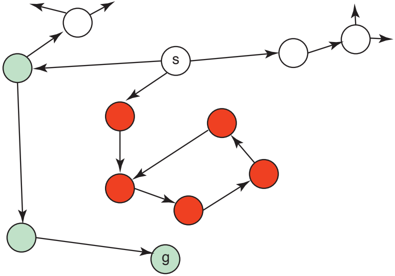


Example: Romania



Note: This is not the shortest path!

Best-first search and infinite loops



Best-first search might fall into an infinite loop!

Complexity of Best-first Search

- Does best-first search guarantee to find the path with fewest arcs?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of the path length?
- What is the space complexity as a function of the path length?
- How does the goal affect the search?

Outline

1. *Introduction (Russell & Norvig 3.1–3.3)*
   - Graphs and searching
   - Examples
   - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
   - Depth-first search
   - Breadth-first search
   - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
   - Greedy best-first search
   - A^* search
   - Admissible and consistent heuristics

A^* search

- A^* search uses both path cost and heuristic values.
- $cost(p)$ is the cost of path p.
- $h(p)$ estimates the cost from the end of p to a goal.
- $f(p) = cost(p) + h(p)$, estimates the total path cost of going from a start node to a goal via p:

$$\underbrace{\underbrace{start \xrightarrow{\text{path } p} n}_{cost(p)} \underbrace{\xrightarrow{\text{estimate}} goal}_{h(p)}}_{f(p)}$$

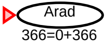
A^* search

- A^* is a mix of lowest-cost-first and best-first search.
- It treats the frontier as a priority queue ordered by $f(p)$.
- It always selects the node on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that node.

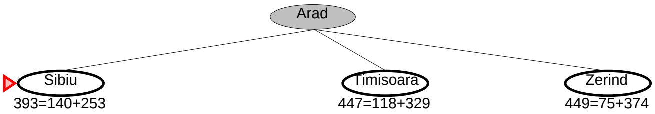
Complexity of A^* search

- Does A^* search guarantee to find the path with fewest arcs?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of the path length?
- What is the space complexity as a function of the path length?
- How does the goal affect the search?

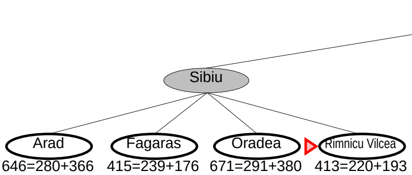
Example: Romania



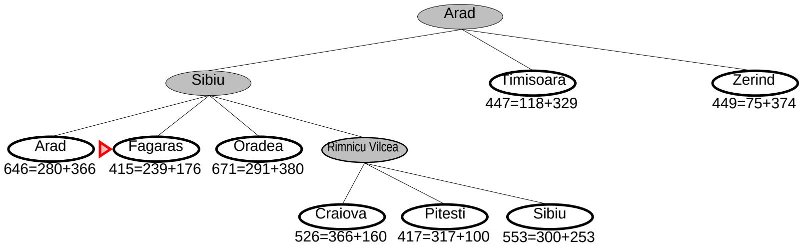
Example: Romania



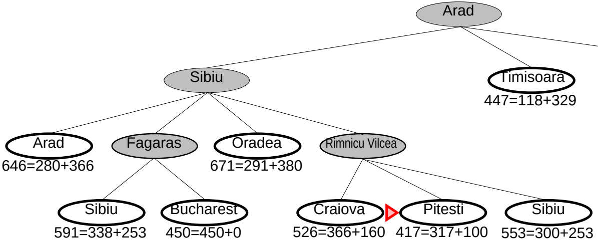
Example: Romania



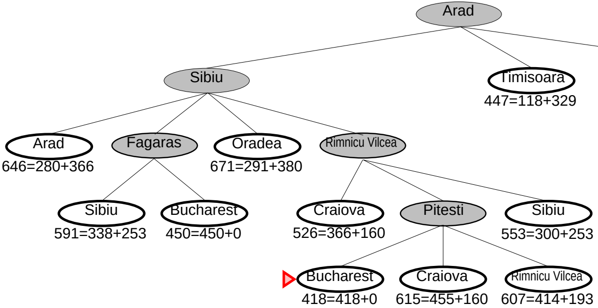
Example: Romania



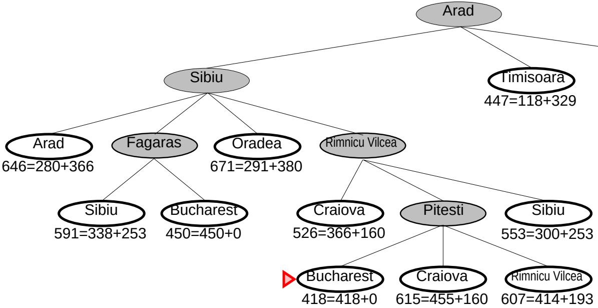
Example: Romania



Example: Romania



Example: Romania



Note: A^* guarantees that this is the shortest path!

Admissibility (optimality) of A^*

If there is a solution, A^* always finds an optimal one first, if:

- the branching factor is finite,
- arc costs are bounded above zero (there is some $\epsilon > 0$ such that all of the arc costs are greater than ϵ), and
- $h(n)$ is nonnegative and an underestimate of the cost of the shortest path from n to a goal node.

A^* always finds a solution

A^* will always find a solution if there is one, because:

- The frontier always contains the initial part of a path to a goal, before that goal is selected.
- A^* halts, because the costs of the paths on the frontier keeps increasing, and will eventually exceed any finite number.

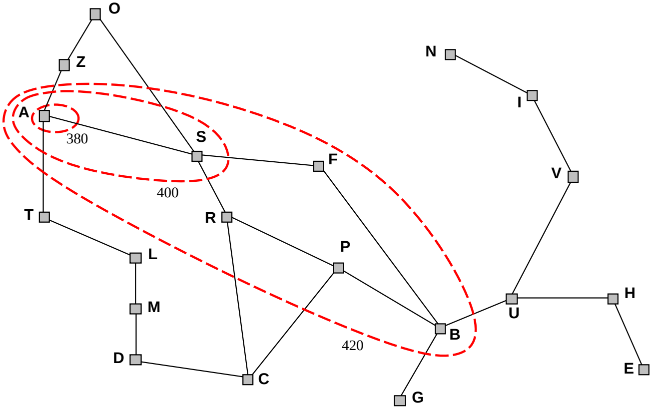
A^* finds an optimal solution first

The first path to a goal selected is an optimal path, because:

- The f-value for any node on an optimal solution path is less than or equal to the f-value of an optimal solution.
  - this is because h is an *underestimate* of the actual cost
- Thus, the f-value of a node on an optimal solution path is less than the f-value for any non-optimal solution.
- Thus, a non-optimal solution can never be chosen while a node exists on the frontier that leads to an optimal solution.
  - because an element with minimum f-value is chosen at each step
- So, before it can select a non-optimal solution, it will have to pick all of the nodes on an optimal path, including each of the optimal solutions.

Illustration: Why is A^* admissible?

A^* gradually adds "f-contours" of nodes (cf. BFS adds layers).
Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$.



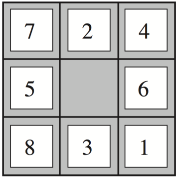
Outline

1. *Introduction (Russell & Norvig 3.1–3.3)*
   - Graphs and searching
   - Examples
   - A generic searching algorithm
2. *Uninformed search strategies (Russell & Norvig 3.4)*
   - Depth-first search
   - Breadth-first search
   - Uniform-cost search
3. *Heuristic search (Russell & Norvig 3.5–3.6)*
   - Greedy best-first search
   - A^* search
   - Admissible and consistent heuristics

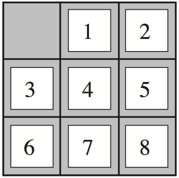
Example: Admissible heuristics

For the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total *Manhattan distance*
(i.e., no. of squares from desired location of each tile)



Start State



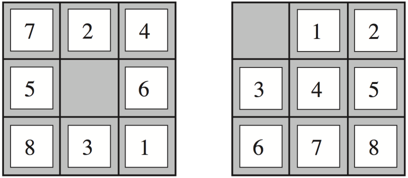
Goal State

$h_1(S) =$??
$h_2(S) =$??

Example: Admissible heuristics

For the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total *Manhattan distance*
(i.e., no. of squares from desired location of each tile)



Start State — Goal State

$h_1(S) = 8$
$h_2(S) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$

Dominating heuristics

If $h_2(n) \geq h_1(n)$ for all n (both admissible), then h_2 *dominates* h_1 and is better for search. Typical search costs (for 8-puzzle):

$depth = 14$	DFS $\approx$ 3,000,000 nodes
	$A^*(h_1) = 539$ nodes
	$A^*(h_2) = 113$ nodes
$depth = 24$	DFS $\approx$ 54,000,000,000 nodes
	$A^*(h_1) = 39{,}135$ nodes
	$A^*(h_2) = 1{,}641$ nodes

Given any admissible heuristics h_a, h_b,

$$h(n) \quad = \quad \max(h_a(n), h_b(n))$$

is also admissible and dominates h_a, h_b.

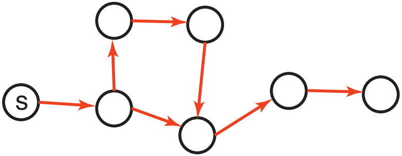
Heuristics from a relaxed problem

Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem:

- If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to *any adjacent square*, then $h_2(n)$ gives the shortest solution

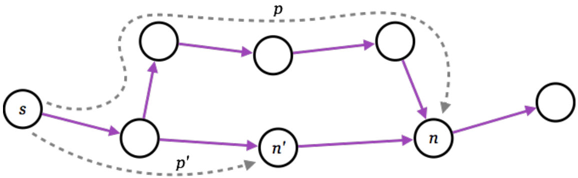
Key point: the optimal solution cost of a relaxed problem is never greater than the optimal solution cost of the real problem

Graph-search = Multiple-path pruning



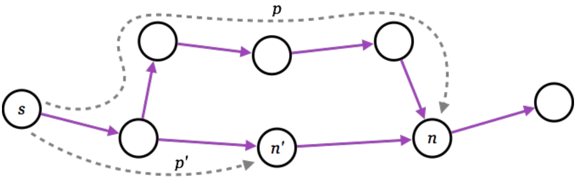
- Graph search keeps track of visited nodes, so that we don't visit the same node twice.
- Suppose the first time we visit a node is not via the most optimal path
  - then graph search will return a suboptimal path
- Under which circumstances can we guarantee that A^* graph search is optimal?

Optimal A^* graph search



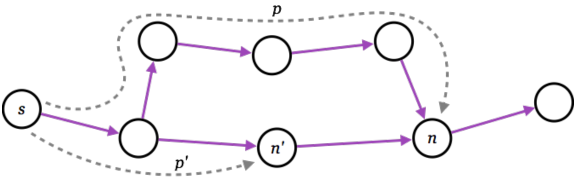
- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.

Optimal A graph search*



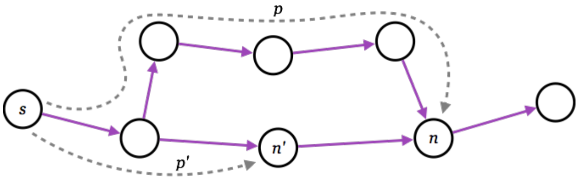
- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.
- p was selected before p', so: $\mathit{cost}(p) + h(n) \leq \mathit{cost}(p') + h(n')$.

Optimal A graph search*



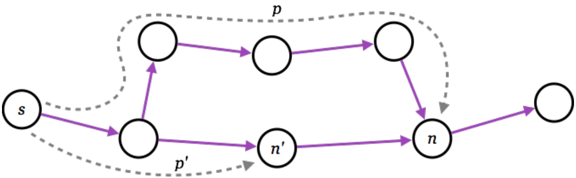
- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.
- p was selected before p', so: $cost(p) + h(n) \leq cost(p') + h(n')$.
- Suppose $cost(n', n)$ is the actual cost of a path from n' to n. The path to n via p' is shorter than p:
  $cost(p') + cost(n', n) < cost(p)$.

Optimal A^* graph search



- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.
- p was selected before p', so: $cost(p) + h(n) \leq cost(p') + h(n')$.
- Suppose $cost(n', n)$ is the actual cost of a path from n' to n. The path to n via p' is shorter than p: $cost(p') + cost(n', n) < cost(p)$.
- Combining the two: $cost(n', n) < cost(p) - cost(p') \leq h(n') - h(n)$

Optimal A graph search*



- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.
- p was selected before p', so: $\mathit{cost}(p) + h(n) \leq \mathit{cost}(p') + h(n')$.
- Suppose $\mathit{cost}(n', n)$ is the actual cost of a path from n' to n. The path to n via p' is shorter than p: $\mathit{cost}(p') + \mathit{cost}(n', n) < \mathit{cost}(p)$.
- Combining the two: $\mathit{cost}(n', n) < \mathit{cost}(p) - \mathit{cost}(p') \leq h(n') - h(n)$
- So, the problem won't occur if $|h(n') - h(n)| \leq \mathit{cost}(n', n)$.

Consistency, or monotonicity

- A heuristic function h is consistent (or monotone) if,

$$|h(m) - h(n)| \quad \leq \quad cost(m, n)$$

  for every arc $\langle m, n \rangle$.
  - (This is a form of triangle inequality)
- If h is consistent, then A^* graph search will always finds the shortest path to a goal.
- This is a strengthening of admissibility.

Summary of tree search strategies

Strategy	**Frontier selection**	**Halts if solution?**	**Halts if no solution?**	**Space**
Depth-first	Last node added			
Breadth-first	First node added			
Best-first	Global min $h(p)$			
Lowest-cost-first	Minimal $cost(p)$			
A^*	Minimal $f(p)$			

Halts if: If there is a path to a goal, it can find one, even on *infinite graphs*.

Halts if no: Even if there is no solution, it will halt on a *finite graph* (perhaps with cycles).

Space: Space complexity as a function of the length of the current path.

Summary of tree search strategies

Strategy	**Frontier selection**	**Halts if solution?**	**Halts if no solution?**	**Space**
Depth-first	Last node added	No	No	Linear
Breadth-first	First node added	Yes	No	Exp
Best-first	Global min $h(p)$	No	No	Exp
Lowest-cost-first	Minimal $cost(p)$	Yes	No	Exp
A^*	Minimal $f(p)$	Yes	No	Exp

Halts if: If there is a path to a goal, it can find one, even on *infinite graphs*.

Halts if no: Even if there is no solution, it will halt on a *finite graph* (perhaps with cycles).

Space: Space complexity as a function of the length of the current path.

Example demo

Here is an example demo of several different search algorithms, including A^*. Furthermore you can play with different heuristics:

`http://qiao.github.io/PathFinding.js/visual/`

Note that this demo is tailor-made for planar grids, which is a special case of all possible search graphs.