CHALMERS
UNIVERSITY OF TECHNOLOGY

RULE-BASED ARTIFICIAL INTELLIGENCE

PART ONE

Bastiaan Bruinsma

October 17, 2022

Chalmers University of Technology

Topics

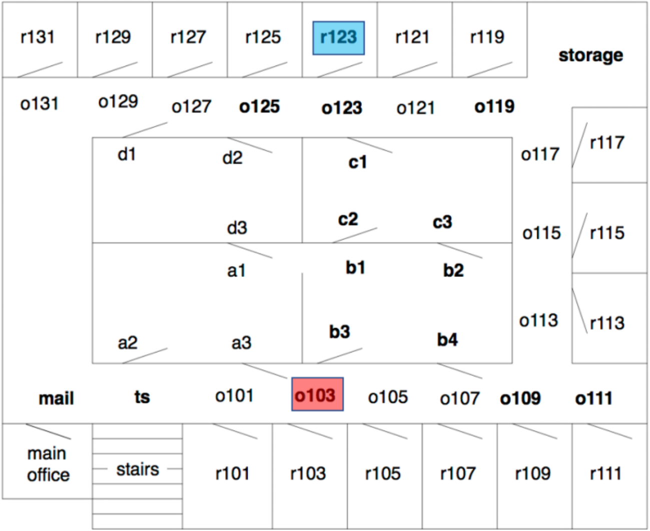
- Search problems
- Generic search
- Uninformed search
- Heuristics and Informed search

Acknowledgements:

- Poole and Mackworth (2017). *Artificial Intelligence*. Cambridge University Press
- Peter Ljunglöf and Moa Johansson for sharing material

Delivery Robot Problem

Suppose a delivery robot wants to go from start (red) to goal (blue):

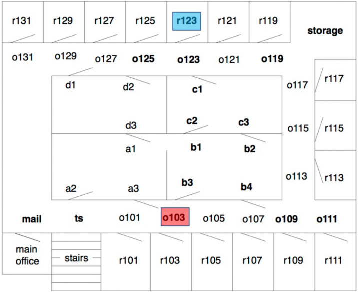


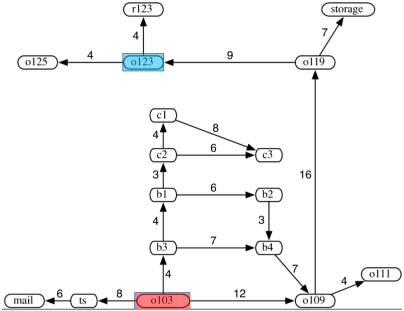
Graph Problem

- A graph problem (or path-finding problem) consists of:
  - a set of *nodes*
  - a set of *edges* between nodes (that may be labelled with actions and/or costs)
  - a node called *start node*
  - a set of nodes called *goal nodes*

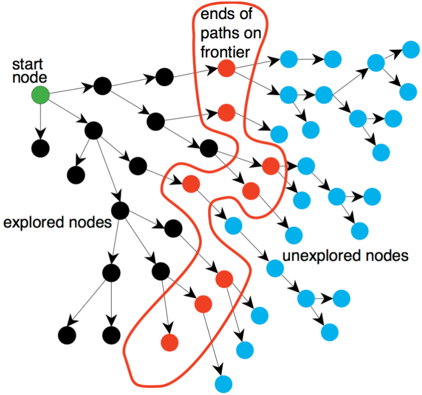
- A *solution* to a graph problem is a path (i.e., a sequence of states) leading from the start node to a goal node

Delivery robot problem as a graph problem



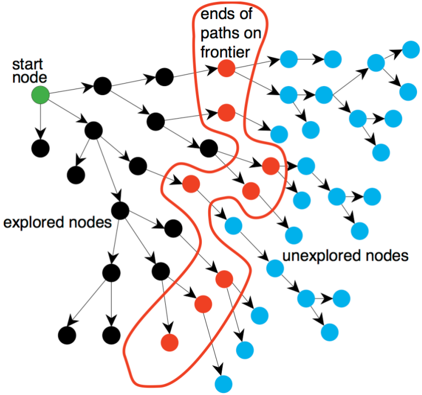


Generic Search: Frontier



- We will define a generic algorithm for solving graph problems
- The algorithm maintains a set of paths called the Frontier (or Fringe)
- Any solution must begin with a path that belongs to the Frontier

Generic Search Algorithm

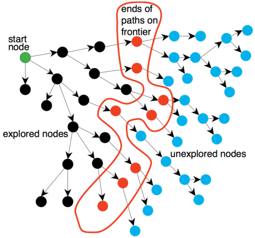


procedure *Search*(G, S, goal)
  Inputs
    G: graph with nodes N and arcs A
    s: start node
    goal: Boolean function of nodes
  Output
    path from s to a node for which goal is true
    or $\perp$ if there are no solution paths
  Local
    Frontier: set of paths
  Frontier $:= \{\langle s \rangle\}$
  while Frontier $\neq \{\}$ **do**
    select and **remove** $\langle n_0, \ldots, n_k \rangle$ from Frontier
    if goal (n_k) **then**
      return $\langle n_0, \ldots, n_k \rangle$
    Frontier $:=$ Frontier $\cup \{\langle n_0, \ldots, n_k, n \rangle : \langle n_k, n \rangle \in A\}$
  return $\perp$

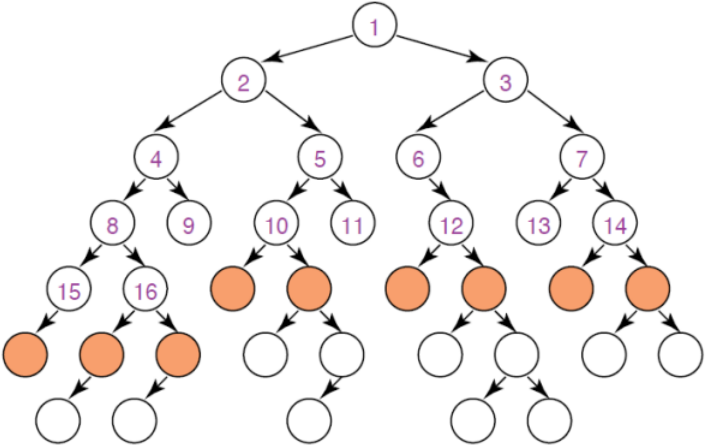
Instances

We can implement GA by letting Frontier be a list and always select the first path, ordered as:

- *Breadth-first search*: In this case new paths are inserted at the end of the list. Then the Frontier is an ordinary queue, i.e. a FIFO (First-in First-Out) queue.
- *Depth-first search*: In this case new paths are inserted at the front of the list. Then the Frontier is a stack, i.e. a LIFO (Last-in First-Out) queue.
- *Best-first search*: In this case new paths are inserted into the list based on their **heuristic score**. The Frontier is always sorted by score.



Uninformed Search: Breadth-first search (BFS)



Iterative deepening depth-first search proceeds as follows:

- First do a depth-first search down to depth 1. This way only paths of max length 1 are put into the Frontier
- If that does not lead to a solution, do a depth-first search down to depth 2
- If that does not lead to a solution, do a depth-first search down to depth 3
- . . . and so on until a solution is found

Iterative Deepening DFS

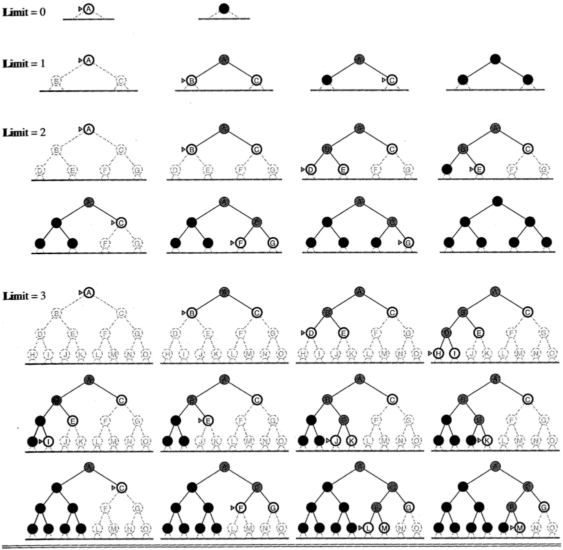


Figure 3.19 Four iterations of iterative deepening search on a binary tree.

Iterative Deepening DFS: Analysis

Advantages:

- . . . it always finds a solution if there is one (like BFS)
- . . . it always finds the shortest solution (like BFS)
- . . . it is memory efficient (like DFS)

Disadvantages:

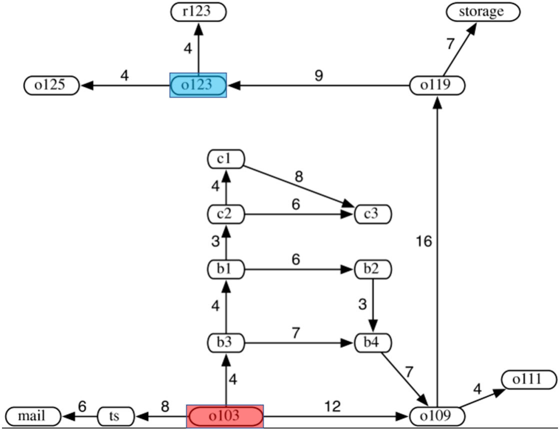
- . . . some nodes are revisited many times

Adding Arc Costs

Sometimes it is useful to put costs on arcs. For example, the costs might represent *travel time* or *travel distance*:

- We write the cost of arc (n_i, n_j) as $cost(n_i, n_j)$
- Given a path $p = (n_0, n_1, \ldots, n_k)$, the cost of p, $cost(p)$ is defined as the *sum of the costs of the arcs appearing in p*
- Given a cost function, we may *look for an optimal solution to a graph problem*, e.g., *the shortest path* or *the fastest path*.

Arc Costs



Example: The path $\{o103, o109, o123\}$ has cost: $12 + 16 + 9 = 37$

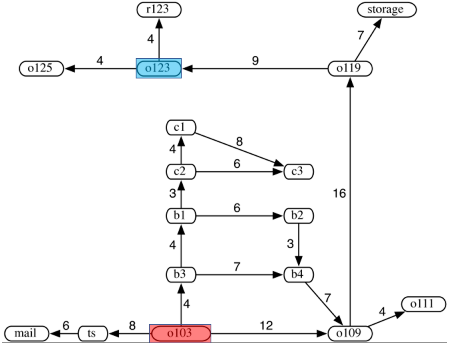
Dijkstra's Algorithm (Lowest-cost-first search)

- Applies when the arcs are labeled with costs
- A version of the Generic Search Algorithm
- *Lowest-cost-first search*: Let the Frontier be a list sorted by path cost (with the path with the lowest cost first)
- When arc costs are all equal, it coincides with BFS
- It always finds the cheapest solution, so it is optimal
- But it has limited scalability (like BFS)

Finding an Optimal Path

Find a path from the red node to the blue node with minimum cost

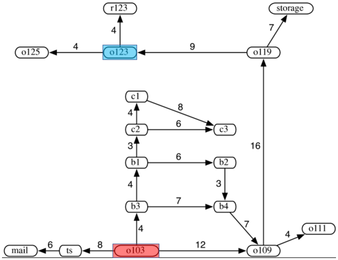
Example of problem that navigators need to solve



Finding an Optimal Path

Step: Frontier

1. $[o103_0]$
2. $[b3_4, ts_8, o109_{12}]$
3. $[b1_8, ts_8, b4_{11}, o109_{12}]$
4. $[ts_8, c2_{11}, b4_{11}, o109_{12}, b2_{14}]$
5. $[c2_{11}, b4_{11}, o109_{12}, \text{mail}_{14}, b2_{14}]$
6. ...



- In everyday language, a *heuristics* is a rule of thumb that indicates where to search primarily
- The word has the same origin as the Greek “Eureka!” (“I found it!”) that Archimedes shouted in his bathtub
- Not always optimal - but it works!

Informed Search: Heuristics

Example of heuristic principles in everyday life:

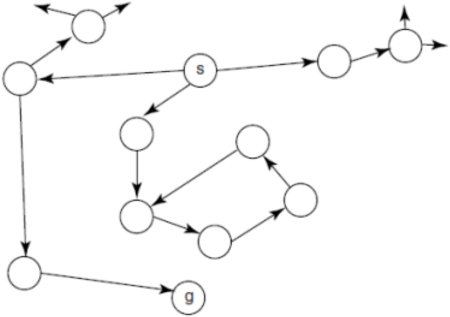
- Search for toys at low levels primarily
- Search for blueberries in forests primarily
- Search for translations that use common words primarily
- Search for solutions that are simple primarily

Definition

A *heuristic function* is a function h that assigns a non-negative real number $h(p)$ to each path p. Intuitively it is an *estimate* of the cost of the *cheapest path* from the end-node of p to a *goal node*

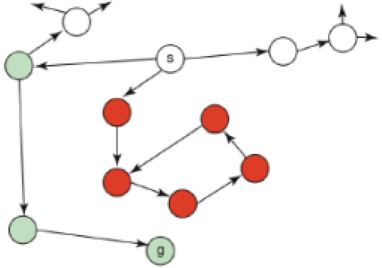
For calculating $h(p)$, the only relevant part of p is its end-node. Some texts define heuristic functions on nodes and costs on paths. Our choice here is to define both heuristic functions and costs on paths

Example of a heuristic function



- This is a graph problem with arc costs drawn to scale. The cost of each arc is its length. The aim is to find the shortest path from s to g
- A heuristic function for path p, $h(p)$ can be defined as the straight-line distance from the end node of p to g

Greedy best-first search (GBFS)



- *Greedy best-first search*: Keep the Frontier sorted by heuristic value $h(p)$ (with paths with low values first)
- In this example, the algorithm will get stuck in the red loop and never terminate!
- So greedy best-first search is not what we want. . .

The algorithm A*

- Very powerful search algorithm
- Pronounced “A star”
- Invented by Hart, Nilsson and Raphael in 1968
- A kind of best-first search: the Fringe is sorted by “grades.”
- A mix of lowest-cost first and best-first search

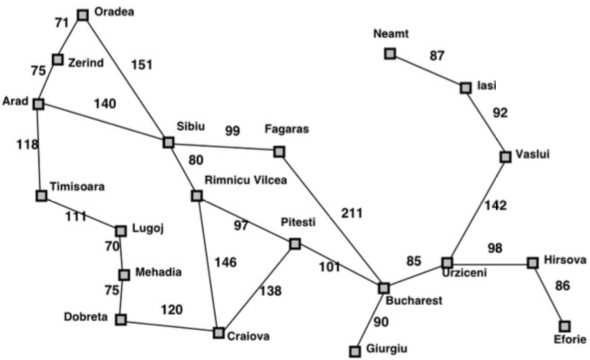
A* Search

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 node of p to a goal
- $f(p) = cost(p) + h(p)$, estimates the total path cost of going from the start node, via path p to a goal:

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

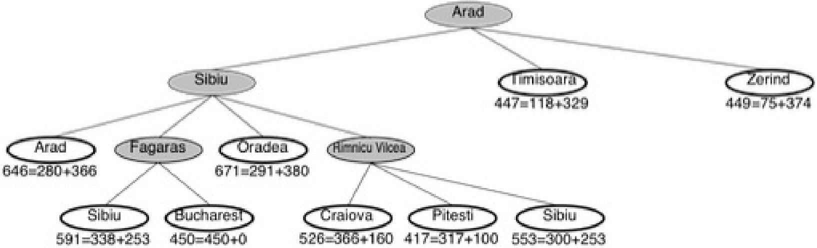
Running example: driving in Romania



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

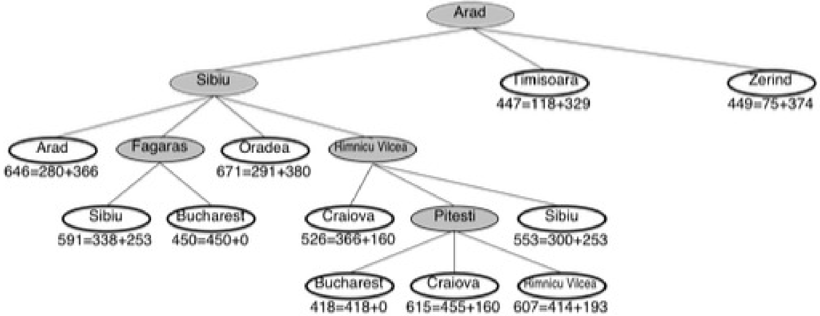
We want to find the shortest path from Arad to Bucharest using a map with road distances to neighbors (for computing $cost(p)$) and a table with straightline distances (for computing $h(p)$). This information is available to a navigator.

A* at Work



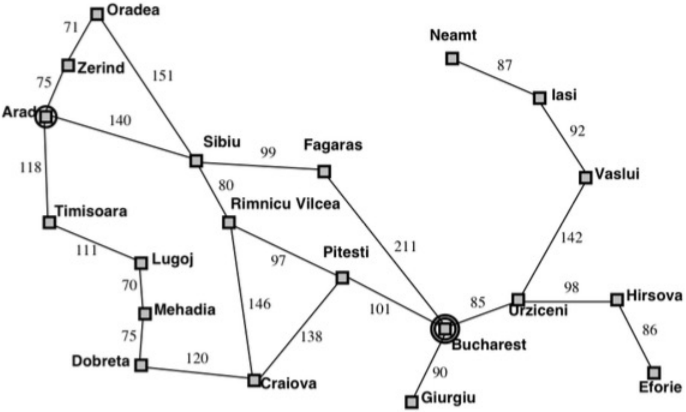
Since we follow the Generic Search Algorithm, we don't stop here just because we added Bucharest (a goal state) to the Frontier.

A* at Work



Since we follow the Generic Search Algorithm, we stop here. In fact we just removed (and returned) a path ending in a goal state (Bucharest) from the Frontier.

A* at Work



So A* found the path Arad-Sibiu-Rimnicu-Pitesti-Bucharest (418km). This is the shortest path. Actually, A* always finds the shortest path from any city to any city!

Admissible heuristics - A Definition

- The heuristic function h is *admissible* if:

  $$h(p) \leq \text{cost}(p')$$

- whenever p' starts at the end-node of p and ends at a goal node
- that is, an *admissible heuristic never overestimates the actual cost* of reaching a goal node
- In other words: The estimate of the remaining cost is *never higher than the actual cost*

Example

The straight-line distance heuristic is admissible, since the it is always smaller than or equal to the actual (road) distance

Optimality of A*

Theorem

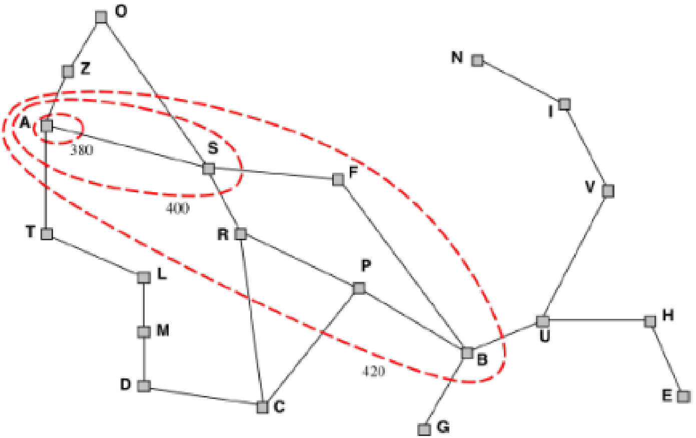
If there is a solution, then A* always returns an optimal solution, if:

1. The search graph is finite
2. The arc costs are uniformly bounded (i.e., there is an $\varepsilon > 0$ such that all of the arc costs are greater than ε)
3. The heuristic function h is admissible

Proof

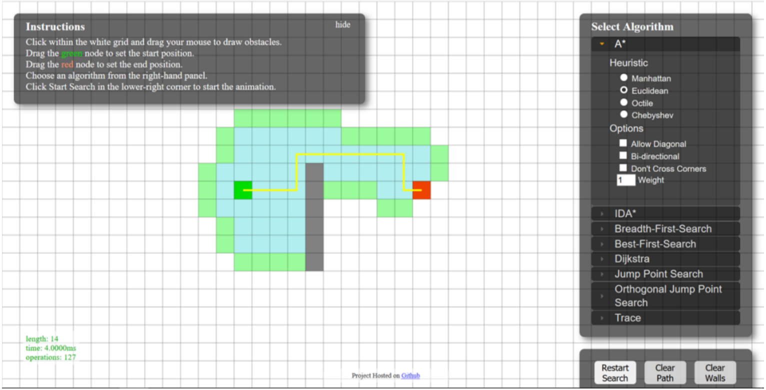
First, suppose there is only one optimal solution, p. Then the first two requirements ensure that p will eventually enter the Frontier. The last requirement ensures that p will be sorted before any other solution. Hence A* will eventually return p. The case when there are several optimal solutions is similar.

Why is A* optimal?



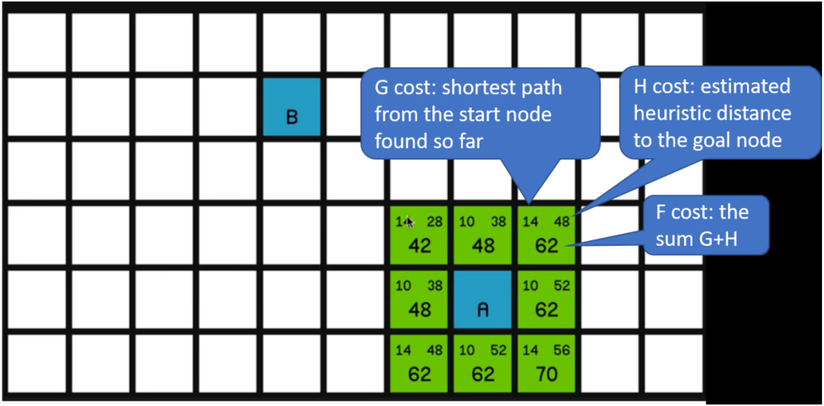
Paths with bigger and bigger *f*-values will be put on the Fringe.

Play with Search Algorithms



Animation - Green: state at the end of some generated path. Blue: state at the end of some selected path. Yellow: returned path.

Video about A*



Video - Explanation of the A* Algorithm

Summary

- Uninformed search:
  - Breadth first
  - Depth first
  - Dijkstra's Algorithm (lowest-cost-search)
- Informed search - taking heuristics into account
  - Best-first search
  - A*
- Thursday:
  - Applications of rule-based AI
  - Relationship between Search and Reinforcement Learning

RULE-BASED ARTIFICIAL INTELLIGENCE

PART ONE

Bastiaan Bruinsma

October 17, 2022

Chalmers University of Technology