A* search

Outline

In this topic, we will look at the A* search algorithm:

- It solves the single-source shortest path problem
- Restricted to *physical* environments
- First described in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael
- Similar to Dijkstra's algorithm
- Uses a hypothetical shortest distance to weight the paths

Background

Assume we have a heuristic lower bound for the length of a path between any two vertices

E.g., a graph embedded in a plane

- the shortest distance is the Euclidean distance

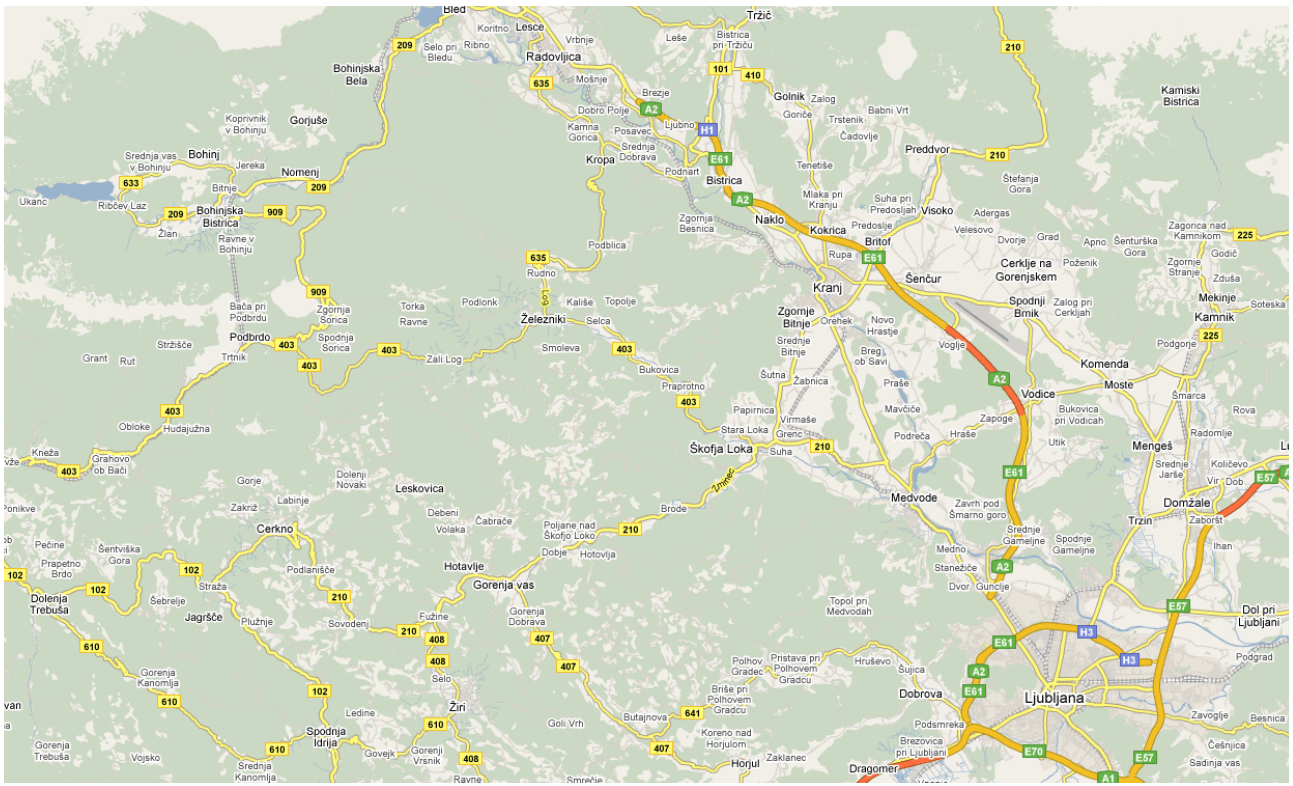
  "as the crow flies"

- use this to guide our search for a path

  "as the fox runs"

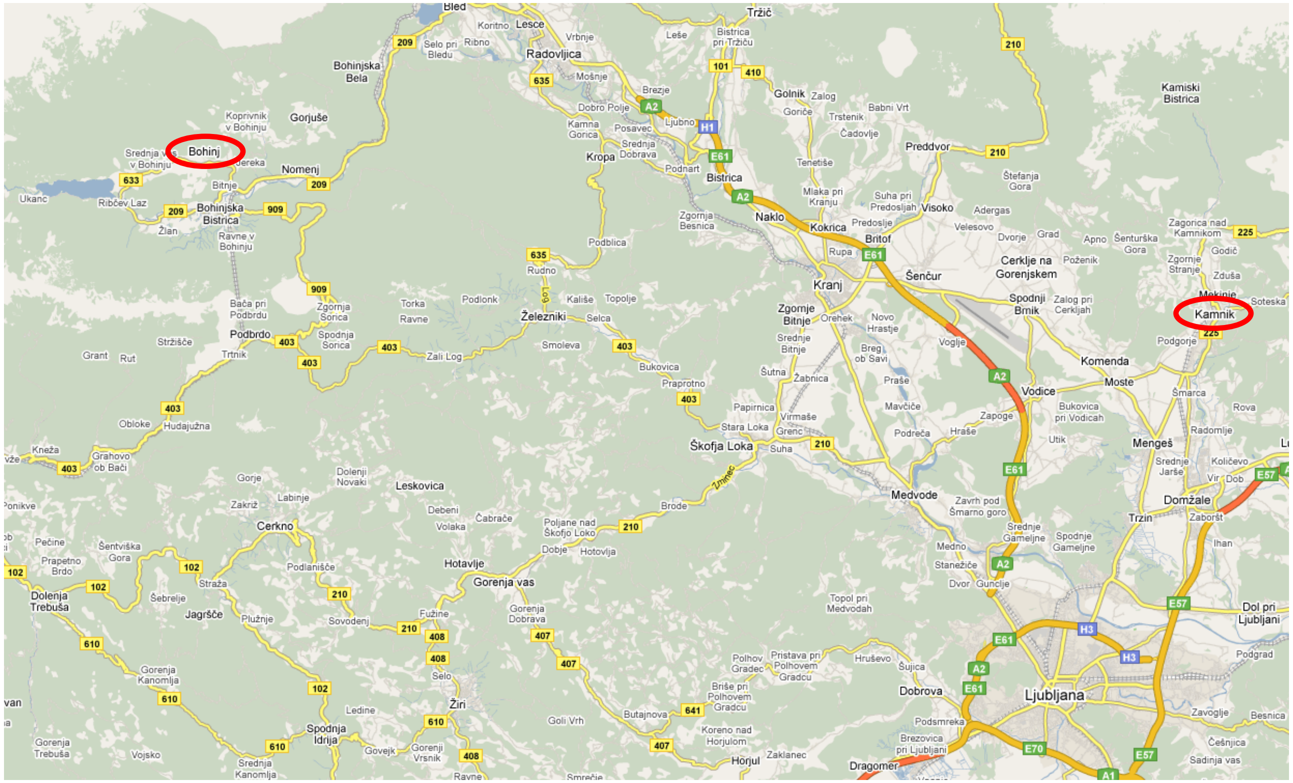
Idea

Consider this map of Slovenia



Idea

Suppose we want to go from Kamnik to Bohinj



Idea

A lower bound for the length of the shortest path to Bohinj is
h(Kamnik, Bohinj) = 53 km

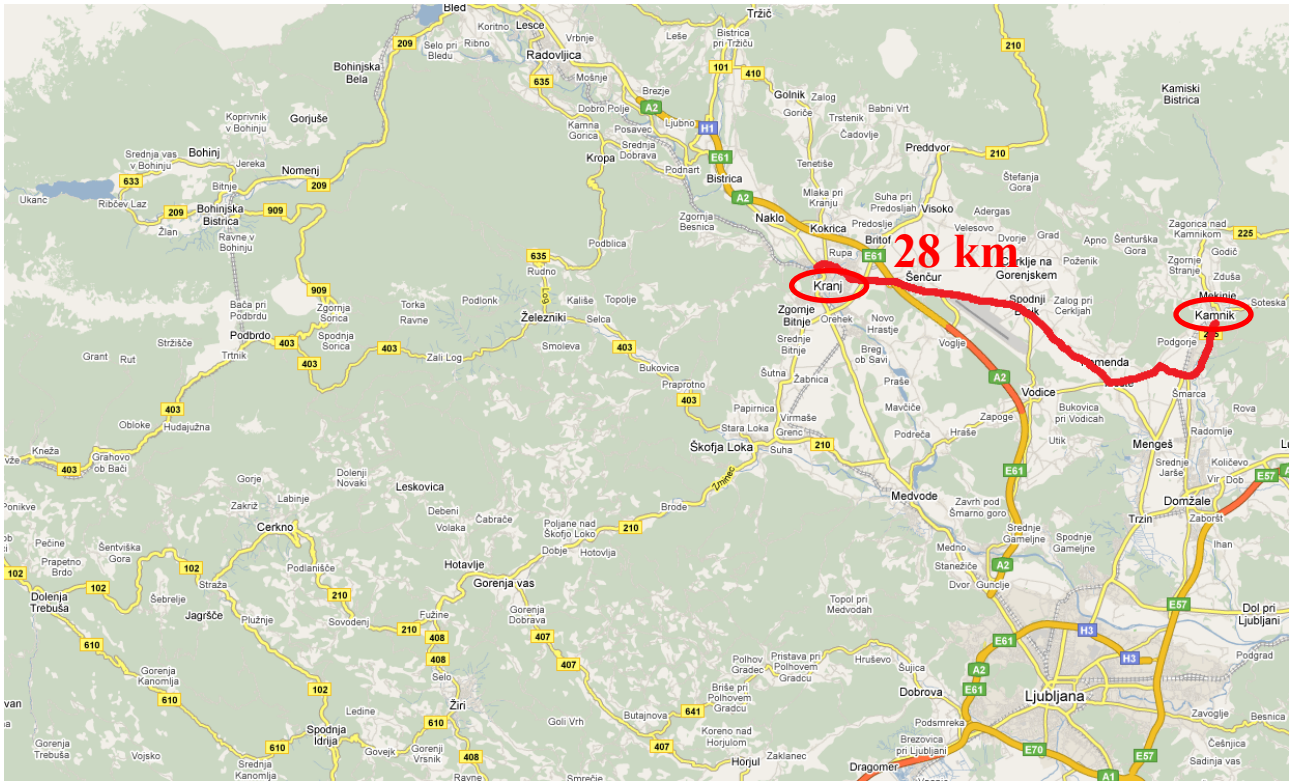
Idea

Any actual path must be at least as long as 53 km

Idea

Suppose we have a 28 km shortest path from Kamnik to Kranj:

d(Kamnik, Kranj) = 28 km



Idea

A lower bound on the shortest distance from Kranj to the destination is now h(Kranj, Bohinj) = 32 km

Idea

Thus, we weight of the path up to Kranj is

$$w(\text{Kranj}) = d(\text{Kamnik, Kranj}) + h(\text{Kranj, Bohinj}) = 60 \text{ km}$$

Idea

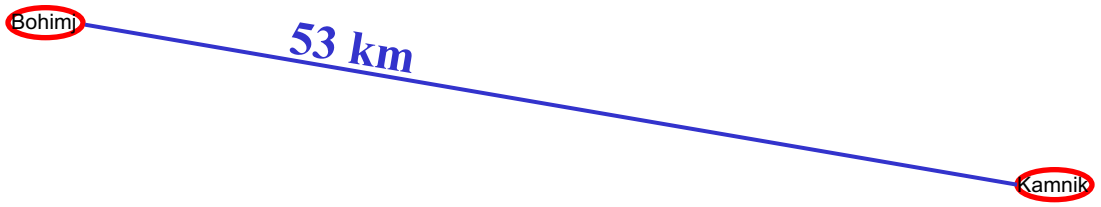
Any path extending this given path to Bohimj must be at least 60 km

Idea

The value w(Kranj) represents the shortest possible distance from Kamnik to Bohinj given that we follow the path to Kranj

As with Dijkstra's algorithm, we must start with the null path starting at Kamnik:

$$w(\text{Kamnik}) = d(\text{Kamnik, Kamnik}) + h(\text{Kamnik, Bohimj})$$
$$= 0 \text{ km} + 53 \text{ km}$$



Algorithm Description

Suppose we are finding the shortest path from vertex a to a vertex z

The A* search algorithm initially:

- Marks each vertex as unvisited
- Starts with a priority queue containing only the initial vertex
  - The priority of any vertex v in the queue is the weight $w(v)$ which assumes we have found the a shortest path to v
  - Shortest weights have highest priority

Algorithm Description

The algorithm then iterates:

- Pops the vertex u with highest priority
  - Mark the vertex u of the path as visited
- Ignore all visited adjacent vertices v
- For each remaining unenqueued adjacent vertex v:
  - Push v with $\mathrm{w}(v) = \mathrm{d}(a, u) + \mathrm{d}(u, v) + \mathrm{h}(v, z)$
- For each enqueued adjacent vertex v:
  - Determine if $\mathrm{w}(v) = \mathrm{d}(a, u) + \mathrm{d}(u, v) + \mathrm{h}(v, z)$ is less than the current weight/priority of v, and if so, update the path leading to v and its priority

Continue iterating until the item popped from the priority queue is the destination vertex z

Comparison of Priorities

Suppose we have a path with

$$w(Smarca) = 4\text{ km} + 52\text{ km} = 56\text{ km}$$

Comparison of Priorities

We can extend this path to Moste:

$$w(Moste) = 4\text{ km} + 5\text{ km} + 48\text{ km} = 57\text{ km}$$

Comparison of Priorities

We can also extend this path to Menges:

$$w(Menges) = 4\text{ km} + 4\text{ km} + 51\text{ km} = 59\text{ km}$$

Comparison of Priorities

The smaller weight path to Moste has priority—extend it first

Comparison with Dijkstra's Algorithm

This differs from Dijkstra's algorithm which gives weight only to the known path

- Dijkstra would chose Menges next:

$$d(\text{Kamnik}, \textit{Moste}) = 9 \text{ km} > 8 \text{ km} = d(\text{Kamnik}, \textit{Menges})$$

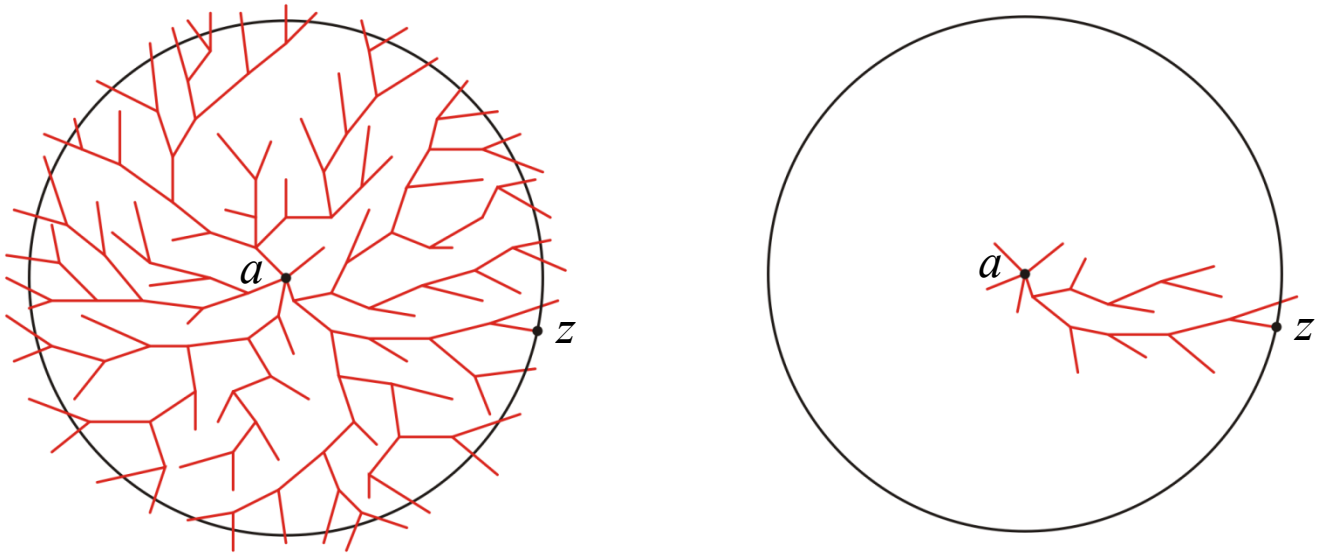
Difference:

- Dijkstra's algorithm radiates out from the initial vertex
- The A* search algorithm directs its search towards the destination

Comparison with Dijkstra's Algorithm

Graphically, we can suggest the behaviour of the two algorithms as follows:

– Suppose we are moving from a to z:



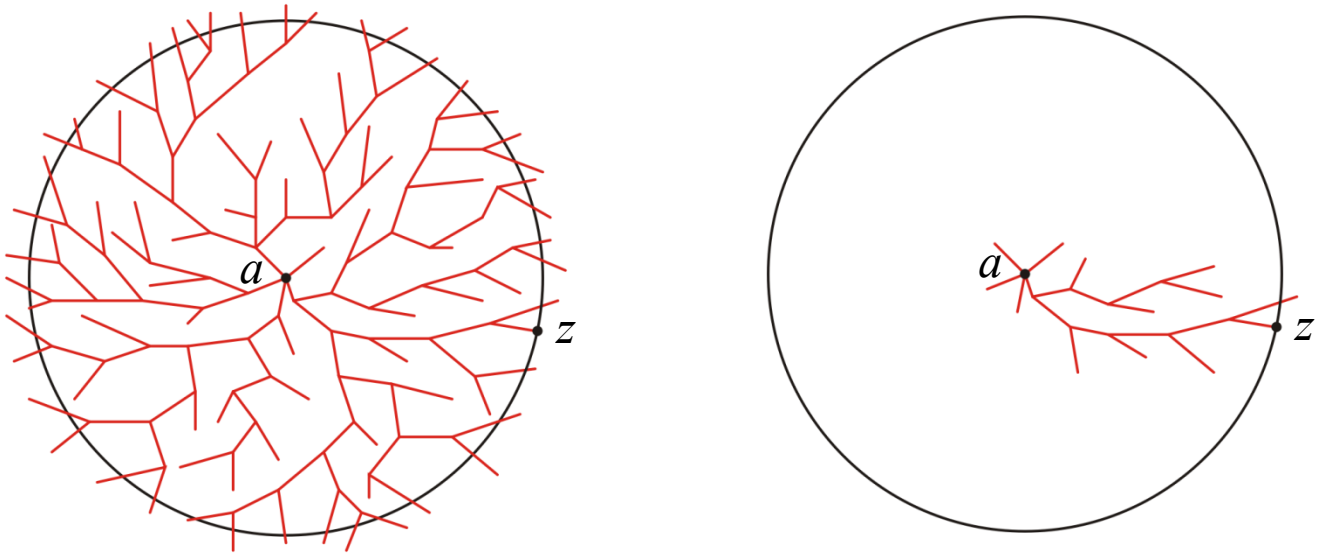
Representative search patterns for Dijkstra's and the A* search algorithms

Comparison with Dijkstra's Algorithm

Dijkstra's algorithm is the A* search algorithm when

using the discrete distance $\mathrm{h}(u,v) = \begin{cases} 0 & u = v \\ 1 & u \neq v \end{cases}$

– No vertex is better than any other vertex



Representative search patterns for Dijkstra's and the A* search algorithms

Summary

This topic has presented the A* search algorithm

- Assumes a hypothetical lower bound on the length of the path to the destination
- Requires an appropriate heuristic
  - Useful for Euclidean spaces (vector spaces with a norm)
- Faster than Dijkstra's algorithm in many cases
  - Directs searches towards the solution

References

Wikipedia, http://en.wikipedia.org/wiki/A*_search_algorithm

[1] Andrew Moore, Carnegie Mellon University, “A* Heuristic Search”, http://www.autonlab.org/tutorials/astar08.pdf

[2] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 2nd Ed., Prentice Hall, Chapters 4 and 5.