Artificial Intelligence

Task 1 - Search for Routing

By *Özgun Mirtchev*

Professor *Franziska Klügl*

Örebro University

April 20, 2017

In this lab two different search algorithms was going to be tested and compared. The underlying data structure is a navigation graph with connections between nodes, over the different areas of the city of Örebro. The task work includes constructing the search algorithms and to compare them to each other and how many nodes that are tested for each algorithm. Through the use of well designed objects and some provided pseudo code, it was trivial to write the code by using the specific class members of each class and following the written code.

0.0.1 Scenario Preparation

The first task was to implement a system where data could be read from .csv-files into data structures. Since each node has a position, name and connected neighbours, it was only natural to make a Node class and a Connection class, which stores the connections from a specific Node to another Node, which is also added to the Node class.

0.0.2 Iterative Deepening Search

To implement this search algorithm, the first step was to create a depth-limited search. This algorithm is similar to a depth-first search however it includes a depth parameter, which prevents the algorithm from searching further than a specified node depth. It may or may not provide the most optimal route but it will compute fast. To run this algorithm, a depth restriction was provided to be iterated through with a while-loop with the depth-limited search, trying to find a route with the least depth. The depth-limited search was constructed by using the example provided in the lecture but with a list of visited nodes to make it a bit more optimized. Below is the iterative deepening loop iterating through the search algorithm to find the path with the least depth.

```
def iterative_deepening(start, end):
    maxDepth = 50
    depth = 0
    while maxDepth > depth:
        result = depth_limited_search(start, end, depth)
        if result:
            return result
        depth = depth + 1
```

0.0.3 A* Search

One of the most used pathfinding algorithms in the industry is A*. It is an informed search algorithms which always finds the most optimal route by using a heuristic search system. The heurisitc function for this algorithm was implemented with Euclidean-distance since it's the best choice for node-based networks. Manhattan-distance is best suited for finding the distance where only vertical and horisontal moves are allowed. While the A* algorithm is known to be a slow algorithm, dependant on which heuristic and how the big the map is, however in small maps, the time it takes to compute is minuscule and can be disregarded. To implement this algorithm the hints from the lectures were used. The source code has comments to explain what the algorithm does.

0.0.4 Reconstruct path

The reconstruct path function was used for both algorithms. It iteratively goes through a dictionary with the key as target node and the node it came from as the value. The way the function works is by starting from the goal node and working its way to the start node. By using a while-loop, it iteratively adds the node it came from to a separate list, which gets reversed before it's returned since the nodes are added backwards. The function is by itself pretty self-explanatory with the named variables.

```
def reconstruct_path(cameFrom, start, end):
    current = end
    path = [current]
    while current != start:
        current = cameFrom[current]
        path.append(current)
    path.reverse()
    return path
```

0.0.5 Tests

On the next page you will find some tests that were run to measure the performance of each algorithm. Between the two algorithms there are clearly differences in which path they generate and how many nodes are visited to find the

optimal path. The results show that the Iterative-Deepening search visits more nodes to find a path but generally has the same amount of path nodes, but taking longer paths. See for example, the first test between Vivalla and Norra Bro, where it goes through Ladugardsangen from Central, instead of taking the best path which is through Sorby from Central. Obviously this is because of the heuristic for the A*, which always calculates the least costly path.

```
----------Iterative-Deepening-Search----------
-----------------Test Results-----------------

---------Route: Vivalla to Norra Bro---------
Created path:
   {Vivalla}---->{Baronbackarna}---->{Hjorsta}---->{Central}---->
   {Ladugardsangen}---->{Sorbyangen}---->{Brickebacken}---->{Norra
      Bro}

Reached depth: 7
Total path nodes: 8
Total tested nodes: 78

---------Route: Markbacken to Tybble---------
Created path:
   {Markbacken}---->{Central}---->{Sorby}---->{Tybble}

Reached depth: 3
Total path nodes: 4
Total tested nodes: 65

---------Route: Solhaga to Hagaby---------
Created path:
   {Solhaga}---->{Alvtomta}---->{Rosta}---->{Central}---->{Hagaby}

Reached depth: 4
Total path nodes: 5
Total tested nodes: 42

---------Route: Grenadjarstaden to Solhaga---------
Created path:
   {Grenadjarstaden}---->{Norr}---->{Central}---->
   {Rosta}---->{Vasthaga}---->{Solhaga}

Reached depth: 5
Total path nodes: 6
Total tested nodes: 78
```

```
----------A*-Search----------
--------Test Results---------

---------Route: Vivalla to Norra Bro---------
Created path:
   {Vivalla}--142-->{Baronbackarna}--257-->{Holmen}--387-->
   {Central}--564-->{Sorby}--704-->{Tybble}--846-->{Brickebacken}--1054-->{Norra
      Bro}

Total path nodes: 8
Total tested nodes: 74

Press enter to continue...

---------Route: Markbacken to Tybble---------
Created path:
   {Markbacken}--297-->{Central}--474-->{Sorby}--614-->{Tybble}

Total path nodes: 4
Total tested nodes: 54

---------Route: Solhaga to Hagaby---------
Created path:
   {Solhaga}--110-->{Vasthaga}--295-->{Rosta}--567-->
   {Central}--754-->{Hagaby}

Total path nodes: 5
Total tested nodes: 37

---------Route: Grenadjarstaden to Solhaga---------
Created path:
   {Grenadjarstaden}--85-->{Norr}--218-->{Central}--490-->
   {Rosta}--675-->{Vasthaga}--785-->{Solhaga}

Total path nodes: 6
Total tested nodes: 73
```