Application of artificial intelligence methods for route planning on multilayered maps

Author: Domonkos Rózsay, Supervisor: Csaba Sidló

May 2024

Previous work and current status

Continuing from the previous semester, I joined ILAB research group working on the related project at SZTAKI. From my previous work, we had the (basic) ability to plot maps using map layer data provided in the project. Although this didn't exactly fit the my research goals, it was a good way to start and to see the details of a route planning problem in practice.

One of the project's goal is finding optimal paths between two points on a weighted geographic map, and also the topic of my project work. We have developed and successfully tested multiple versions of already exsiting and also novel route planning algorithms, with a lots of further ideas to be examined.

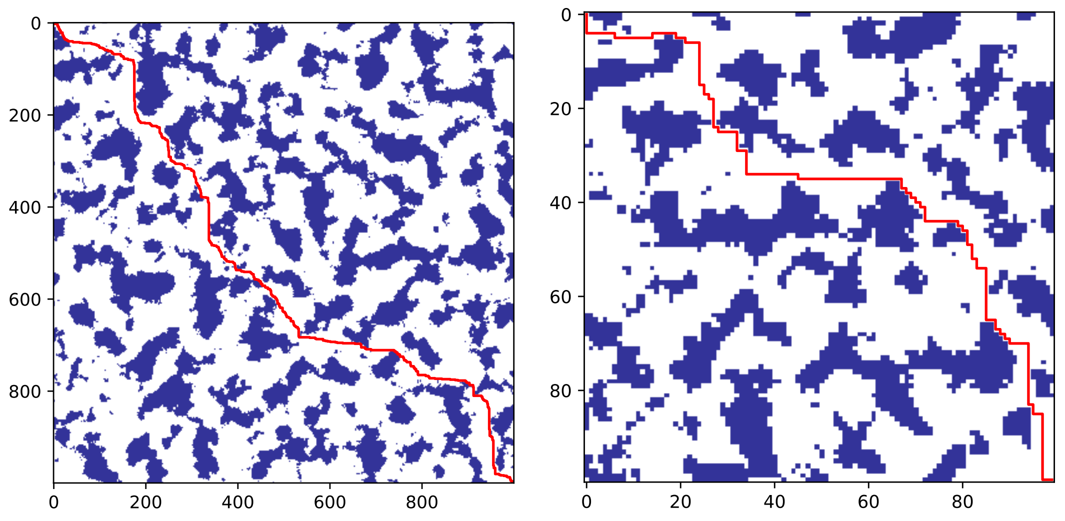
My main contribution this semester was devising and implementing pathfinding algorithms, while also collaborating with co-workers on other related tasks.

1 The A^* algorithm

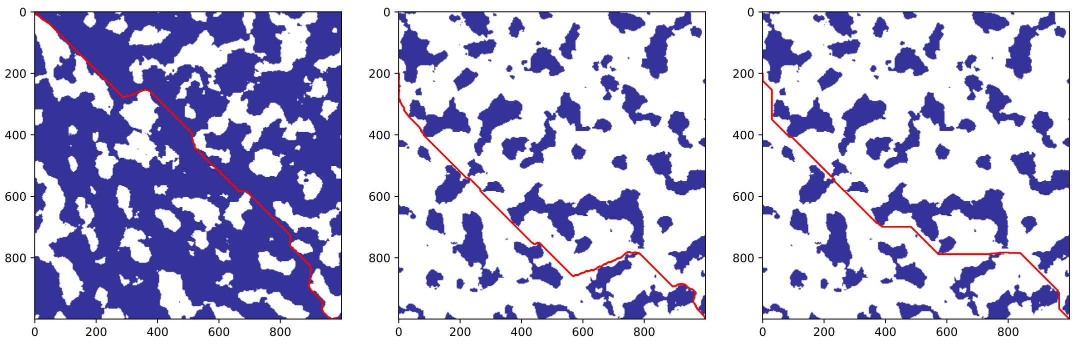
The map layers provided in the project are mostly raster images, saved as geo-referenced pictures. These layers define the traversable and the inaccessible areas. Here we had the idea to work on the grid of accessibility values: Imagining the grid as a regular weighted graph, having 4 edges or 8 edges per cell to the neighboring cells, I already knew of many pathfinding algorithms that may work. Because running time was a limiting factor, I could not apply a generic Dijkstra, Bellman-Ford and other basic algorithm. Considering the "weights" of each grid cells as strictly positive and only having to find an optimal path between two points I tried to implement the A^* algorithm instead.

While A^* is not the most efficient algorithm, it is easy to understood and implement. It adds a heuristic function that modifies the edge weights, and can be seen as an extension of Dijkstra's algorithm with edge weights $g(v) = f(v) + heu(v)$.

A^* is a heavily researched algorithn, and with conditions on the heuristic function it is proven to be optimal. First I tried a basic approach that only used the up-down-left-



Route planning with no diagonal movement (blue: inaccessible areas).



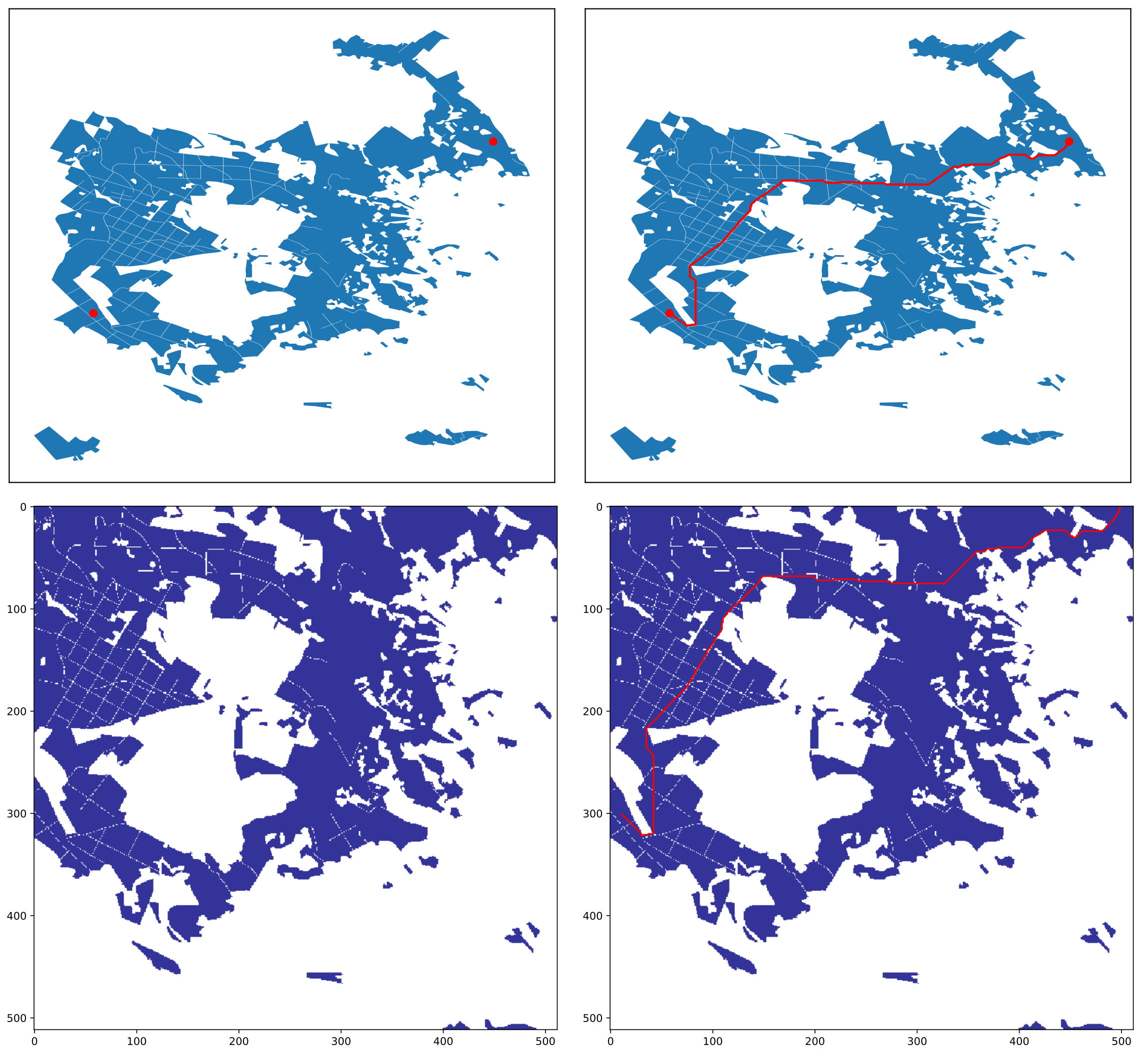
Comparing algorithms using diagonal movement without (1st and 2nd) and with good heuristic (3rd picture)(blue: inaccessible areas).

right movement. As the example shows on the figure, it already works quite well, but in smaller maps it is quite obvious that we can get a better result if we add diagonal movements as well.

Here we encounter the problem with A^*. It is only optimal if we choose the heuristic well. Up until this point the basic Manhattan distance was a good choice, but now we need another one. I opted for the Euclidean distance, and it proved to be appropriate.

I still had to speed up the method, as it was taking around 2-15 minutes to compute routes on a 1000 by 1000 grid - too slow for our application. I needed at least sub 1 minute running time on a grid of this scale, which I achieved successfully: current route planning times are around $5 - 30$ seconds.

As an extension, I added the possibility to go over blocking cells by assigning the cell a "big" weight, and defining an upper (and lower) bound that the algorithm considers not accessible. For example, giving the blocking cells an 1000 weight, and 1 to normal blocks, when we get a length 2737 of a path, then we can reasonably conclude that we



Map from the original data, and the rasterized version to the find paths.

had to cross 2 blocked cells. Do note that, because of how A^* works to cross one of these cells, we had to discover 1000 additional cells. This also gives us another parameter to play with if there is no optimal path, or we want to be "forgiving" with the blocking elements.

With these issues resolved, I concluded this algorithm as finished, and focused my attention on applying the results on the project data, and integrating in the project environment.

2 Geometrical approach

Along with implementing A^* we continuously improved upon a novel algorithm that does not need rasterized images, and can work directly on vector layers.

From the provided data we could get polygons and the vertices of those polygons. We had an idea for a geometrical algorithm that iteratively tries to find the best solution i.e. the shortest path between two points. The main idea is as follows:

1 Try the straight path between the points first.

2 If there is no blocking polygon on our current path, then we are finished.

3 If there is a blocking polygon, then we try to bypass it. We can do this in two ways (left and right).

4 Store the two paths, look for the shortest path, and iterate from step 2.

It is easy to see that this algorithm works, but it is challenging to implement. The main problems at first were how to choose which polygons to get around, and how to pass left and right.

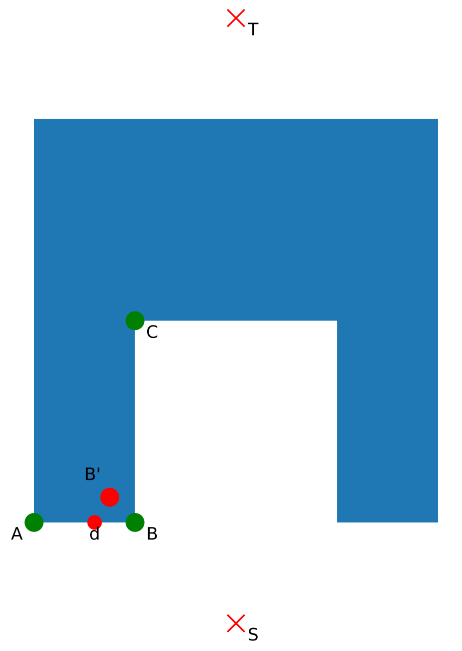
We wanted to store the paths in a format from which we could easily choose the shortest one. As a simple solution, we store each path as a collection of points with the lenght of the path (for example in a binary tree).

We considered several ideas who we can choose the next polygon to go around, but all of them were problematic.

If we try to get the closest (or the farthest), then we can easily get to situations when we get into infinite loops or getting suboptimal solutions. Same with grabbing a random one.

Our next idea was to only look at the convex hull of the polygons, rather the vertices of this hull. This was a step towards the right direction, but we should not exclude just any point inside.

To consider all the points would also be wasteful. As it can be seen from the Figure, we do not need to consider points B and C if we start from point S and want to go around to point T.



What defines these points? We can get a line segment from S that if extended, intersects the polygon. I proved the following.

Theorem. *For any optimal path P there is no vertex v that if extended from S by any $\varepsilon > 0$ it intersects the polygon.*

Proof. Suppose the point B is on an optimal path but it has an elongation B' from S that is inside the polygon. Then there is a $\delta > 0$ such that there is a point d that is δ distance from B and is on the same path but the length of Sd is smaller than the length of SB and Bd. This point d need not be on the polygon. Therefore in such case B cannot be on an optimal path. □

3 Further research

Now that we know which vertices to exclude we still have the problem of finding the order to go through them.

My conjecture is that we can take the points in similar way as in the A^*. We try to inflate a bubble (circle or ellipse) centered on the starting point S. This remains to be proven.

References

[DP85] Rina Dechter and Judea Pearl. "Generalized best-first search strategies and the optimality of A". In: *Journal of the ACM (JACM)* 32.3 (1985), pp. 505–536.

[Fel+11] Ariel Felner et al. “Inconsistent heuristics in theory and practice”. In: *Artificial Intelligence* 175.9-10 (2011), pp. 1570–1603.

[RN16] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.