Efficient path planning algorithms for multilayered traversability maps

Author: Domonkos Rózsay, Supervisor: Csaba Sidló

Dec 2024

Previous Work and Current Status

I continued my work from the previous semester with the ILAB research group at HUN-REN SZTAKI. My primary objective was to discover an efficient way to represent multilayered maps and develop algorithms capable of solving the pathfinding problem on these representations.

Previously, I implemented the A^* algorithm on a single-layer map represented as a 0-1 grid. While effective for basic maps, it proved insufficient for realistic, multilayered representations. This semester, I explored further algorithms and data structures to address the limitations. I implemented and tested A^*, Theta* (Θ^*) and the basics of visibility graph approaches. "Anytime Weighted A^*" (AWA*), the "Continuous Dijkstra" and some other ideas were tested to analyze their trade-offs in various scenarios.

1 Rasterized Map Representation and A^* Algorithm

We assume raster images as map layers, defining traversable and inaccessible areas. These are treated as weighted square grids, where cells represent nodes, and there are 4 or 8 dirrections of movement with costs scaling to the cell weight.

A^* extends Dijkstra's algorithm by incorporating a heuristic function $h(n)$, guiding the search towards the goal. It selects nodes based on $f(n) = g(n) + h(n)$, where $g(n)$ is the cost from the start to node n. To ensure optimality, $h(n)$ must be admissible:

$$h(n) \leq h^*(n) \quad \forall n,$$

where $h^*(n)$ is the true cost to the goal. For efficiency, $h(n)$ should also be consistent:

$$h(N) \leq c(N, P) + h(P), \quad \text{for all successors } P \text{ of } N.$$

Example rasterized layers of the original test data

Admissibility can be derived from consistency; however, the reverse does not hold true. For an admissible but not consistent heuristic, it can take more time than a normal Dijkstra to find the path.

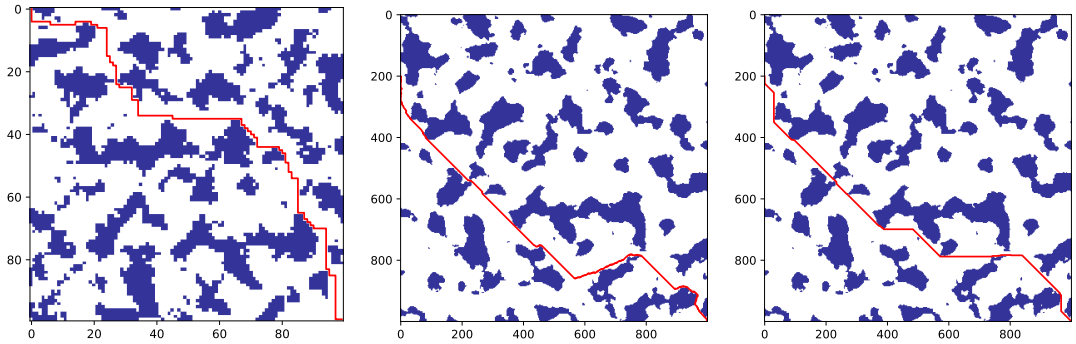

Figure 1: Path comparison: 4-way movement vs. diagonal movement with and without admissible heuristics.

During implementation, I introduced weight scaling for blocked cells. Assigning high weights (e.g., 1000) to inaccessible cells allowed the algorithm to consider paths through these cells if no better options existed. This approach provided flexibility, though at the cost of increased computational overhead.

1.1 Theta* Algorithm

Θ^* improves upon A^* by allowing straight-line paths between nodes, bypassing grid constraints. It modifies the cost function by checking line-of-sight between the current node and its ancestors. This often reduces the path length compared to A^*.

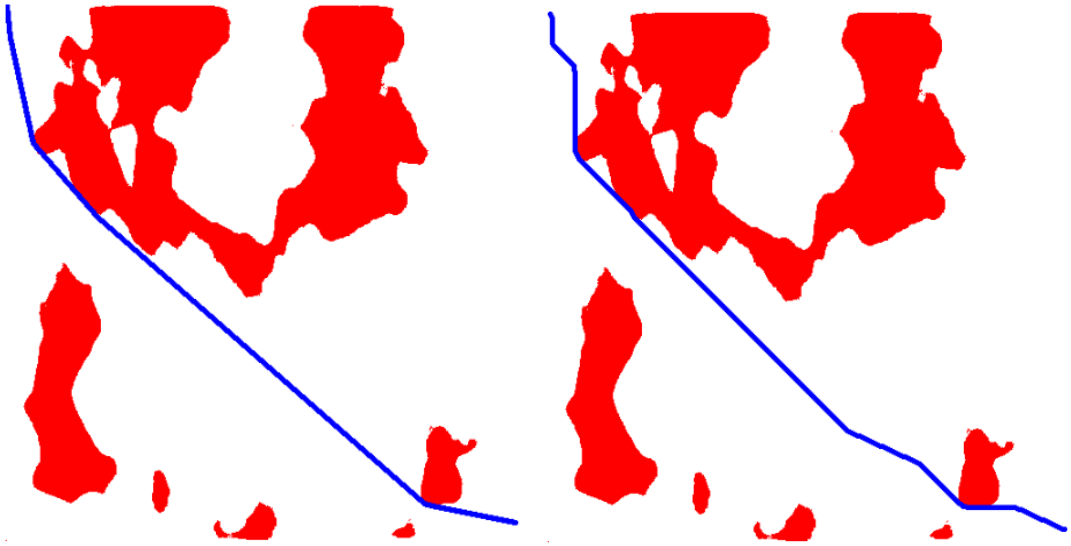

Figure 2: Smoothed path and normal path.

2 Anytime Weighted A^*

With admissible heuristic we get an optimal path, but what if we need a fast calculation instead?

AWA* balances path quality and computational effort by prioritizing suboptimal solutions early and refining them over time. It modifies A^* by scaling the heuristic:

$$f(n) = g(n) + \varepsilon \cdot h(n), \quad \varepsilon > 1.$$

Smaller ε values yield higher-quality solutions but require more computation. This approach is suitable for scenarios demanding fast, approximate results, such as real-time navigation. Implementing AWA* is currently a work-in-progress. However this should allow rapid pathfinding, with subsequent refinements yielding near-optimal paths as time allows.

3 Geometric Approach: Visibility Graphs

For vector-based maps, I explored visibility graphs. These represent nodes as polygon vertices and edges as unobstructed lines of sight. Paths are computed using standard graph algorithms like the previously written A^*. Constructing the visibility graph is computationally expensive, particularly for large datasets.

3.1 Optimization Attempts

My initial implementation was the most basic one I could think of, with $O(n^3)$ time complexity, where n is the number of vertices. This was improved to $O(n^2)$ using

Algorithm 1 Θ^* Algorithm (simplified)

```
Initialize open list with start node
while open list is not empty do
    Pop node with smallest f(n)
    if goal is reached then
        Return path
    end if
    for all successors of the current node do
        if successor is visible from parent then
            Update cost and parent
        else
            Use standard A* cost update
        end if
    end for
end while
```

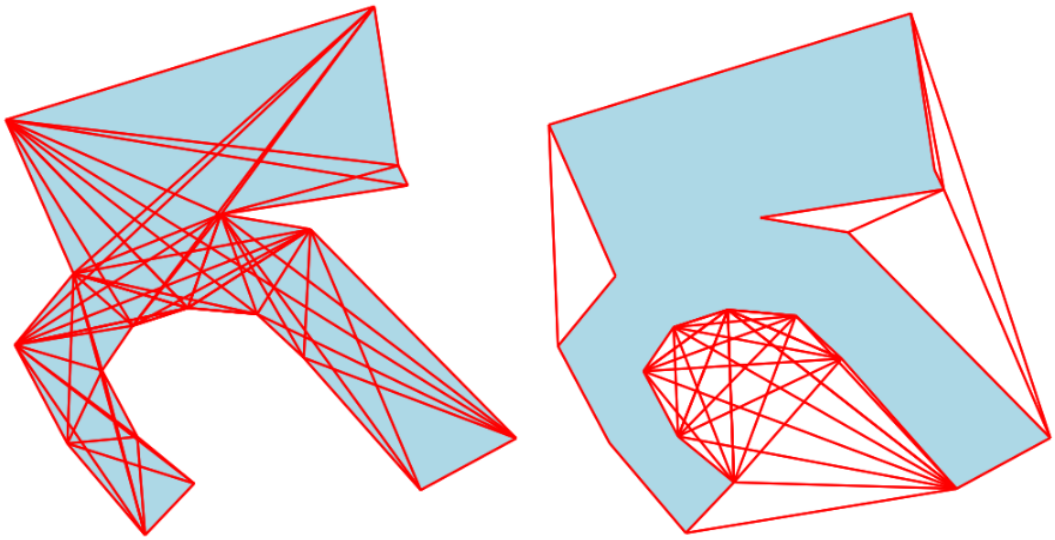

Figure 3: Visibility graphs of one poligon. Inner bisibility on the right and outer on the left.

Overmars and Welzl's approach [OW88], which cleverly sorts the edges in the visibility graph. However, the large number of vertices still posed challenges. I reduced the graph size by considering convex hull vertices only, and pruning redundant points (e.g., those obstructed by the polygon itself). With this optimization I can prune the points, but even then I ran into some issues.

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

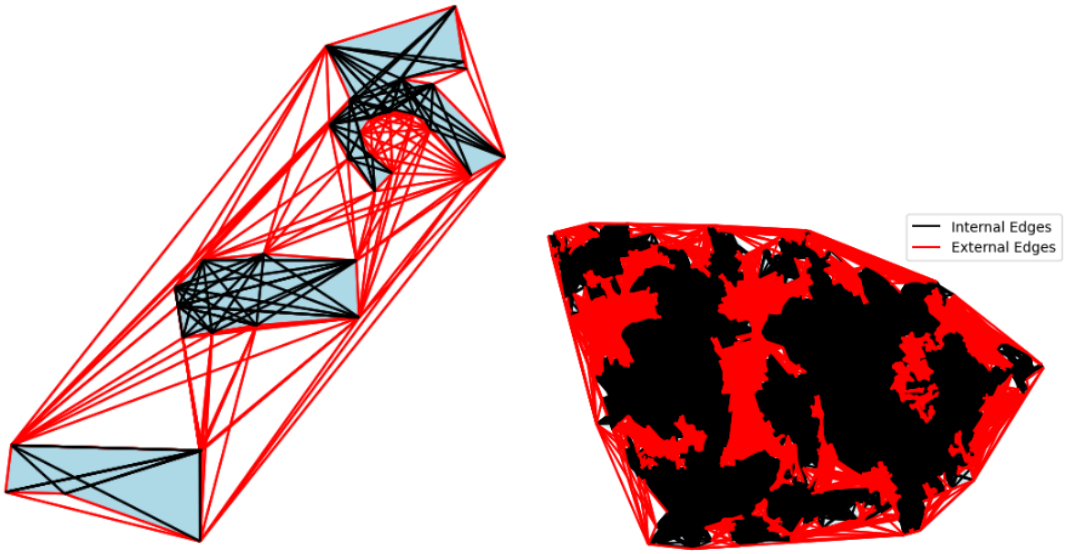

Figure 4: Visibility graphs of multiple polygons. On left is more managabel on right runs in approximately 100 hours and is dense without pruning.

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. □

With this approach, we were able to reduce the visibility graph to a much smaller size. However, one problem remained unresolved despite considering several stop-gap solutions.

In the main problem, these polygons would have an associated weight per meters crossed. We can consider inside edges as well but then the previous theorem losses its purpouse.

Another problem arises with a long but thin rectangle: in such cases, it is more advantageous to cross it in the middle rather than moving to one of its vertices.

4 Continuous Dijkstra

The algorithm that I refer to as "Continuous dijkstra"[Wan21] seemed to solve all my probles last semester.

Continuous Dijkstra extends the traditional algorithm to continuous spaces, which is ideal for vector maps. Instead of expanding grid cells, it propagates a wavefront through the map. Obstacles are treated as barriers redirecting the wavefront. This method has

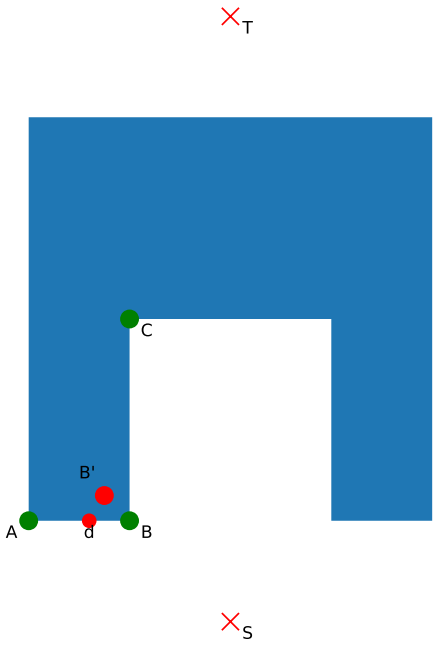

Figure 5: Convex hull and visibility pruning: Vertices B and C are excluded from consideration.

an additional advantage: it can enumerate not only the shortest path but also the second, third, and higher-order topologically distinct paths. These alternative paths can be particularly useful for redundancy in path planning (or multi-robot systems).

However, I discovered that Continuous Dijkstra encounters significant challenges when applied to multilayered maps. When multiple layers with differing weights are combined, the wavefront propagation no longer correctly accounts for layer interactions. I attempted to think of ways to integrate layer weights into the wavefront's propagation rule or even combine $0-1$ layers but they all failed to preserve the method's correctness. If I somehow end up working on this problem in the future it must address how to aggregate weights effectively across layers without breaking continuity.

5 Multilayer Challenges and Weight Aggregation

Multilayered maps introduce additional complexity, as each layer represents different constraints or costs. For Continuous Dijkstra, the problem becomes more pronounced, as the method assumes uniformity within the domain of propagation. Existing algorithms like A^* and Θ^* can handle only single layers at a time but there is always a fix (maybe).

One approach I tried was to rasterize all layers and combine them into a single layer.

The weight of each pixel or grid cell in the combined layer can be computed using different functions of the layer weights:

- **Sum of weights:** Simple but often dominated by the highest-weighted layer and can give wierd results.
- **Maximum or minimum weight:** Captures extreme constraints but may ignore important nuances. I mostly neglected this approach.
- **Convex combinations:** Allows tuning between layers but requires careful calibration.

The best part about convex combination is that for even the heuristics it preserves admissibility.

Linear combinations of layer weights and heuristics do not conserve admissibility but with AWA* it can be thought of as a hidden parameter. As more complex functions proved to be challenging to think of a use for, such as non-linear aggregations, they require further investigation.

The final idea was to combine all layers into a unified representation, treating the aggregated weights as a new "super-layer". However, I have not determined an optimal function for weight aggregation yet.

As I work with the ILAB research group maybe exploring machine learning approaches to learn effective weighting functions from real-world data can be a potential direction.

6 Performance Comparison

During the work completed throughout the semester, I arrived at the following observations.

- **Rasterized Maps:** A^* and Θ^* performed well, with Θ^* offering shorter paths and found them usually much faster.
- **Vector Maps:** Visibility graphs proved tricky to work with, and they were also computationally expensive.
- **Hybrid Scenarios:** Anytime algorithms balance speed and accuracy, particularly in real-time contexts. Much improvement to be seen here.

7 Conclusion and Future Work

During this semester's project work, I focused on exploring diverse path-planning algorithms for multilayered maps and evaluating their strengths and limitations. Rasterized approaches like A^* and Θ^* were usually efficient and fast to calculate, while visibility

graphs have much higher computational demands that can be offloaded beforehand. However, scalability and computation still remain challenges.

Future work will focus on hybrid algorithms, combining the flexibility of geometric methods with the efficiency of heuristic search. Additional optimizations, such as dynamic visibility graphs or perhaps machine-learning-based heuristics may further enhance performance.

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.

[OW88] M. H. Overmars and E. Welzl. "New methods for computing visibility graphs". In: *Proceedings of the Fourth Annual Symposium on Computational Geometry*. SCG '88. Urbana-Champaign, Illinois, USA: Association for Computing Machinery, 1988, pp. 164–171. ISBN: 0897912705. DOI: 10.1145/73393.73410. URL: https://doi.org/10.1145/73393.73410.

[Gho97] Subir Kumar Ghosh. "On recognizing and characterizing visibility graphs of simple polygons". In: *Discrete & Computational Geometry* 17.2 (1997), pp. 143–162.

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

[UK15] Tansel Uras and Sven Koenig. "An empirical comparison of any-angle path-planning algorithms". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 6. 1. 2015, pp. 206–210.

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

[KS21] Tyler King and Michael Soltys. *Minimum Path Star Topology Algorithms for Weighted Regions and Obstacles*. 2021. arXiv: 2109.06944 [cs.DS]. URL: https://arxiv.org/abs/2109.06944.

[Wan21] Haitao Wang. *A New Algorithm for Euclidean Shortest Paths in the Plane*. 2021. arXiv: 2102.12589 [cs.CG]. URL: https://arxiv.org/abs/2102.12589.