Technologies for Autonomous Vehicles
Assignment 2 - Report

1st Emanuele Giuseppe Siani
s330980@studenti.polito.it

Abstract—**This report presents a comparative analysis of two popular pathfinding algorithms, Dijkstra's and A*, applied to urban road networks in Turin, Italy, and Piedmont, California. The study focuses on evaluating their performance in terms of the number of explored nodes and the total travel time cost of the computed paths. Using a heuristic based on the Haversine distance converted to travel time, the A* algorithm significantly reduces the search space—up to 88.6% fewer iterations in Turin—while maintaining near-optimal path costs within 1%. Results demonstrate that A* outperforms Dijkstra, especially in large and complex urban environments, confirming its efficiency and effectiveness in real-world scenarios.**

I. Introduction

This report presents a detailed commentary and analysis of the code implementing a comparative study of two well-known pathfinding algorithms: **Dijkstra's algorithm** and the **A* algorithm**.

In particular, the report includes a high-level explanation of the **A* algorithm** implementation, accompanied by a discussion of the **experimental results**.

The comparison focuses on two main metrics: **performance** in terms of the number of iterations until computing the optimal path, and its **total weight (cost)**.

To ensure a **fair and meaningful evaluation**, the experiments were conducted using two distinct urban layouts: **Turin, Italy** and **Piedmont, California**. These choices highlight the relative advantages of the A* algorithm, especially in larger and more complex urban environments like Turin.

For both cities, the results represent the **average over 10 independent runs**. In each run, both Dijkstra and A* were executed with the same randomly selected **start and destination nodes** to guarantee comparability.

II. A* Algorithm

The *A* algorithm* is a well-known pathfinding and graph traversal algorithm, often preferred over **Dijkstra's algorithm** due to its improved performance in terms of **computational efficiency**. While Dijkstra's algorithm explores all possible paths to find the shortest one, A* enhances this process by using a **heuristic function** to estimate the cost to reach the goal from a given node. This estimation allows A* to discard non-promising paths early, significantly reducing the number of iterations required.

A. Heuristic

As a heuristic function, I implemented the *Haversine distance*, a well-known metric for estimating the great-circle distance between two points on the Earth's surface, taking into account the planet's curvature.

To ensure that the *A** algorithm remains effective, the heuristic must be consistent with the scale of the **edge weights**. If the heuristic values are too small relative to the actual path costs, their influence is negligible, and the algorithm's behavior degenerates to that of **Dijkstra's algorithm**, losing the benefits of *informed search.*

For this reason, both the edge weights and the heuristic are expressed in comparable units: **travel time in seconds**. Each edge weight is computed as the *length of the road segment in meters divided by the road's speed in meters per second.* Similarly, the heuristic—originally a distance in meters—is also divided by an **average speed** to convert it into a time estimate. I chose an average speed of $40\,\mathrm{km/h} = 11.11\,\mathrm{m/s}$, which is a reasonable assumption for **urban environments**.

The function `calculate_heuristic` performs this computation as follows:

$$\texttt{heuristic}(n, \texttt{goal}) = \frac{\texttt{haversine\_distance}(n, \texttt{goal})}{11.11}$$

In this way, the heuristic provides a **time-based estimate** of the remaining cost to the goal, which is consistent with the edge weights. This choice ensures a fair estimate, avoiding overestimation and thereby **preserving the algorithm's ability to find the optimal path**, while also promoting **efficiency**.

B. Code Comment

The function `a_star(G, orig, dest)` implements the A* search algorithm on a directed graph `G`, aiming to find the shortest path from a source node `orig` to a destination node `dest`.

Initialization: For each node in `G`, `astar_visited` is set to `False`. The `gScore` and `fScore` are initialized to infinity (∞). `astar_uses` is set to `None` to store predecessors. For the source node `orig`, the `gScore` is set to 0, while the `fScore` is set to the computed heuristic value.

Open Set and Path Map: `open_set` is a priority queue, using the node's `fScore` as weight and initialized with the starting node with a cost of 0. `came_from` is a dictionary to reconstruct the path after reaching `dest`.

Main Loop: While `open_set` is not empty:

- Pop the node `current` with the lowest `fScore`.
- If `current` equals `dest`, terminate and return the iteration count.
- Skip if `current` was already visited. If not:
- Mark `current` as visited.
- For each `neighbor` of `current`, compute tentative cost as `tentative_gScore` = `gScore[current]`+ weight(current, neighbor).
- If this cost is better than `gScore[neighbor]`, update `came_from[neighbor]` and `gScore[neighbor]`, compute heuristic h, update `fScore[neighbor] = gScore[neighbor] + h`, and push the neighbor (with its `fScore`) onto `open_set`.

Increment the iteration counter each loop.

Termination: If `dest` is reached, return the number of iterations. Otherwise, if `open_set` is empty, return the iteration count indicating failure to find a path.

III. Results and Discussion

To evaluate the performance of the **A* algorithm** compared to **Dijkstra's algorithm**, we conducted experiments on two different urban settings: **Turin, Italy**, and **Piedmont, California, USA**. These cities were selected due to their contrasting geographic and urban characteristics—Turin being a large European city with dense street networks, and Piedmont a smaller Californian town with a simpler structure.

The key metrics compared were:

- **Number of iterations** required by each algorithm to find the optimal path.
- **Total weight** of the computed path (in seconds), based on edge weights representing travel time.

A. Turin Results

TABLE I: Comparison of Dijkstra and A* on Turin Data (Averaged over 10 runs)

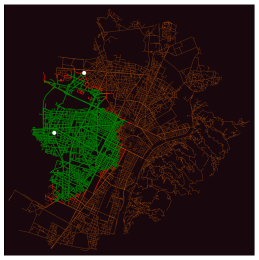
Metric	Dijkstra	A*	Difference (%)
Iterations	6755.0	767.3	**-88.6%**
Weight (s)	526.92	531.45	**+0.86%**

In Turin, A* shows a dramatic improvement in efficiency, reducing the number of iterations by **more than 88%** compared to Dijkstra. This confirms the effectiveness of the heuristic in guiding the search through a large, complex graph. The computed paths remain very similar in terms of total weight, with an average difference of only **0.86%**, indicating that A* not only finds a path faster but also maintains near-optimal quality.

To visually compare the performance of *A** and *Dijkstra*, Figures 1 and 2 show the results of one representative run for each algorithm, respectively.

In each figure, the first image highlights in green all the explored nodes, while the origin and destination nodes are marked in white. The two computed paths are also displayed in white.

It is evident that *A** explores significantly fewer nodes compared to *Dijkstra*, while the selected paths differ only slightly. This figure illustrates the minimal differences observed between the paths when they are not identical; in most cases, the chosen paths coincide, as confirmed by the weight difference reported in Table I and as depicted by the representative example in Figure 3 where the computed paths are identical.


(a) Explored paths by Dijkstra

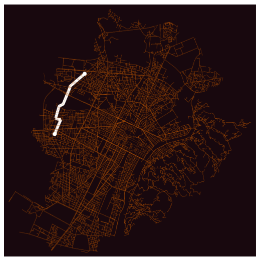
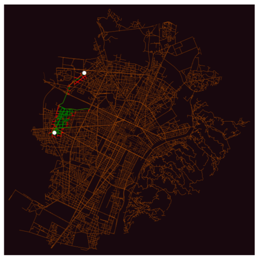

(b) Computed optimal path by Dijkstra

Fig. 1: Turin example 1: Dijkstra algorithm results


(a) Explored paths by A*

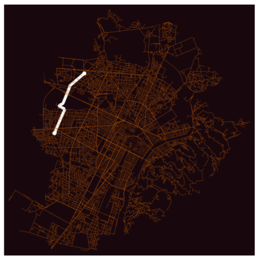
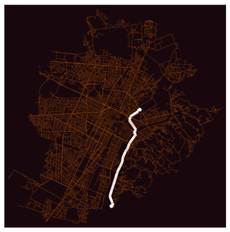

(b) Computed optimal path by A*

Fig. 2: Turin example 1: A* algorithm results


(a) Computed optimal path by Dijkstra

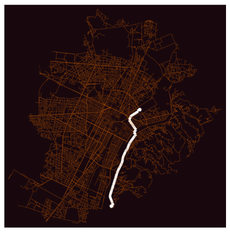

(b) Computed optimal path by A*

Fig. 3: Turin example 2: Dijkstra and A* computed optimal path

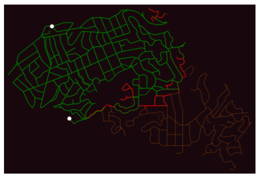
B. Piedmont, California Results

TABLE II: Comparison of Dijkstra and A* on Piedmont Data (Averaged over 10 runs)

Metric	**Dijkstra**	**A***	**Difference (%)**
Iterations	138.4	73.3	**-47.0%**
Weight (s)	168.87	168.87	**0.0%**

In Piedmont, the improvement in iteration count is still evident, with A* requiring about **47% fewer steps**. However, the benefit is less pronounced than in Turin due to the smaller and less complex graph. Interestingly, the path weights are **exactly the same**, demonstrating that the heuristic is consistent and admissible, preserving the optimality of the solution.

For Piedmont, again, we report a visual comparison in figures 4 an 5 which empower the conclusions discussed.


(a) Piedmont example: Explored paths by Dijkstra

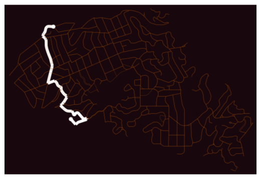
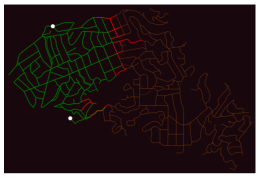

(b) Piedmont example: Computed optimal path by Dijkstra

Fig. 4: Piedmont example 1: Dijkstra algorithm results


(a) Piedmont example: Explored paths by A*

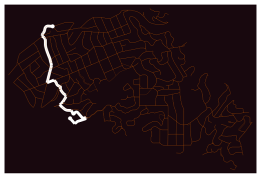

(b) Piedmont example: Computed optimal path by A*

Fig. 5: Piedmont example 1: A* algorithm results

C. Conclusion

While both algorithms produce paths with almost identical costs, slight differences can appear in larger graphs like Turin due to the possibility of multiple nearly-optimal paths. These differences—never exceeding 1% are **negligible** for most practical purposes, like the motion planning.

Overall, these results confirm the following:

- The **A* algorithm is significantly more efficient**, particularly in large urban networks, thanks to its informed search strategy.
- The **quality of the computed paths remains optimal or nearly optimal**.
- The use of a **time-based heuristic** (harvesine distance divided by average speed) ensures correctness and performance improvement over Dijkstra Algorithm.