How do A* and Dijkstra's algorithms compare in terms of speed, memory usage, and accuracy when pathfinding in unweighted node-graphs, weighted node-graphs, and mazes?

Rodrigo Yuste

Extended Essay in IB Computer Science

Session: May 2025

Abstract

This essay investigates two foundational pathfinding algorithms — A* (A-star) and Dijkstra's — comparing their performance across three graph environments: small-scale unweighted mazes, large-scale weighted node-graphs, and urban street networks. Performance is evaluated along three axes: speed (computational time and iteration count), memory usage (proxied by iteration count), and accuracy (path optimality). Drawing on two independent experiments — a controlled maze study by Permana et al. (2018) and a real-world urban simulation by Fiorino (2024) — the essay finds that A* consistently requires fewer iterations owing to its heuristic function $f(v) = g(v) + h(v)$, making it more memory-efficient. However, Dijkstra's algorithm, by exhaustively exploring all weighted edges, produces time-optimal paths in weighted graphs, outperforming A* on travel time by approximately 21% across seven simulations. Neither algorithm is universally superior; the optimal choice depends on whether the graph is weighted and whether distance or travel time is the primary optimisation target.

Keywords: A*, Dijkstra's algorithm, pathfinding, heuristics, weighted graphs, node-graphs, computational efficiency

Contents

1. Introduction

Pathfinding algorithms are a class of computational procedures that determine an optimal route between two or more points within a graph structure. Their applications span a wide range of domains: navigation systems such as Google Maps compute driving directions in real time; logistics companies optimise delivery routes across road networks; and game developers use them to control non-player character movement. The shift from pre-computed lookup tables of shortest paths to dynamic, on-the-fly route calculation was made possible by the efficiency of modern pathfinding algorithms [5].

An algorithm is formally defined as "any well-defined computational procedure" that maps a given input to a desired output [5]. Pathfinding algorithms are a specialised subset of this broader category, constrained to solving shortest-path problems on graphs [4].

The performance of a pathfinding algorithm is typically evaluated along three dimensions [13]: *speed* (the computational time required to find a path), *memory usage* (the computational resources consumed, estimated here by the number of iterations), and *accuracy* (the degree to which the returned path is truly the shortest or most optimal route).

This essay examines two widely used pathfinding algorithms — A* and Dijkstra's — across three graph environments: small-scale unweighted mazes, large-scale weighted node-graphs, and urban road networks. The central research question is:

How do A and Dijkstra's algorithms compare in terms of speed, memory usage, and accuracy when pathfinding in unweighted node-graphs, weighted node-graphs, and mazes?*

Two independent experiments, sourced from distinct academic contexts, are analysed to address this question rigorously.

2. Background: Algorithms and Parameters

2.1 A* (A-star) Algorithm

A* was originally developed in 1968 by researchers at SRI International to guide *Shakey the Robot* — the first autonomous mobile robot capable of reasoning about its own actions [14, 7]. It remains the most widely cited pathfinding algorithm in computer science and game development.

A* guarantees discovery of the shortest path between a source node and a destination node by incorporating a *heuristic function* that estimates the cost of reaching the goal from any given node [14]. A heuristic is a problem-solving method that uses available information to guide search towards a solution more efficiently than exhaustive exploration [16]. The A* evaluation function is:

$$f(v) = g(v) + h(v) \tag{1}$$

where $g(v)$ is the known cost of the path from the source to node v; $h(v)$ is the heuristic estimate of the remaining cost to the destination (e.g. Euclidean, Manhattan, or Diagonal distance); and $f(v)$ is the total estimated cost of a path through v.

At each step, A* expands the node with the lowest $f(v)$ value, prioritising nodes that are both close to the source (via $g(v)$) and close to the destination (via $h(v)$). This directed search prevents the algorithm from exploring large portions of the graph that are unlikely to contribute to the shortest path [10, 4].

An intuitive analogy: imagine a mouse navigating a maze toward cheese it can smell. The mouse follows the scent — the strongest smell corresponds to the lowest $h(v)$ — while also tracking how far it has walked ($g(v)$). If it ventures down a dead end, the increasing $g(v)$ raises $f(v)$, eventually redirecting it back toward more promising routes.

2.2 Dijkstra's Algorithm

Dijkstra's Algorithm was developed in 1956 by Dutch computer scientist Edsger W. Dijkstra, originally to demonstrate the computational power of the ARMAC computer [15]. It

finds the shortest path from a single source node to *every* other node in a graph, making it particularly well-suited to weighted graphs in which edges carry attributes such as cost, travel time, or physical distance.

Unlike A*, Dijkstra's algorithm possesses no heuristic; it explores the graph outward from the source in all directions, always processing the lowest-cost unvisited node next [8]. This exhaustive strategy guarantees globally optimal paths but at the cost of significantly more iterations than A*.

The algorithm maintains three core data structures: a **distance array** initialised to ∞ for all nodes except the source (set to 0); a **predecessors list** enabling path reconstruction; and a **min-heap priority queue** that always yields the unvisited node with the smallest known distance.

Illustrative Example: Cheapest Flight from Madrid to London

To illustrate Dijkstra's process concretely, consider a weighted node-graph representing seven European airports (MAD, FCO, CDG, ZRH, BRU, BER, LDN) with edge weights denoting flight prices. Figure 1 shows the geographical connections; Figure 2 presents the corresponding weighted node-graph.

Figure 1. Geographical connections between seven European airports. Source node: MAD (Madrid); destination node: LDN (London). Author's own figure, created using Google Drawings.

Figure 1: image by author, using Google Drawings

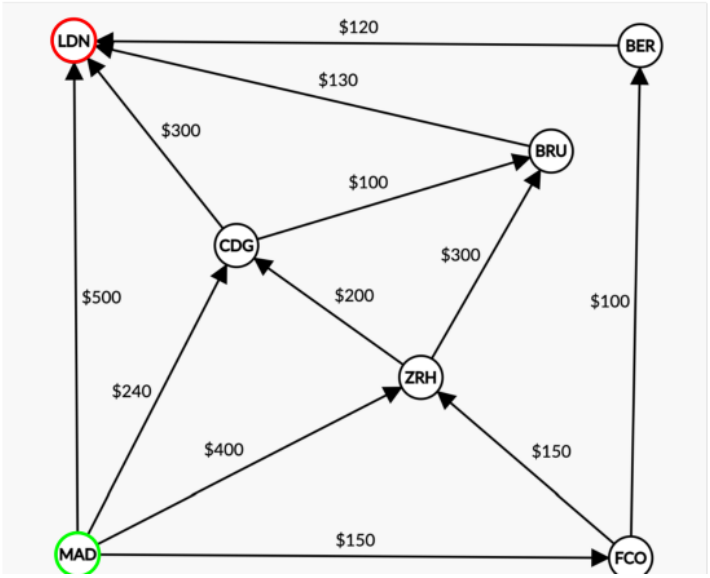


Figure 2. Weighted node-graph derived from the airport map. Edge weights represent estimated flight prices in USD. Source: MAD; destination: LDN. Author's own figure, created using CSAcademy Graph Editor.

Dijkstra's Algorithm proceeds as follows:

1. Initialise: $d(\text{MAD}) = 0$; $d(v) = \infty$ for all other nodes v.

2. Extract MAD from the min-heap. Relax[1] all its neighbours: $d(\text{FCO}) = 150$, $d(\text{CDG}) = 240$, $d(\text{ZRH}) = 400$, $d(\text{LDN}) = 500$. Figure 3 shows the state of the predecessors list, distance array, and min-heap after this step.

[1]To *relax* a node is to update its tentative distance if a shorter path through the current node is found [8].

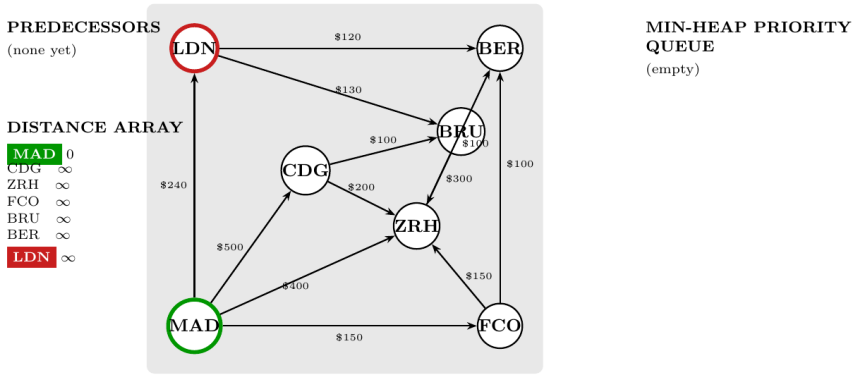


Figure 3. State of the algorithm after processing the source node MAD: initial relaxation of all neighbours, with updated distance array and min-heap priority queue. Author's own figure.

3. Extract FCO (cost 150). Relax its neighbours: $d(\text{ZRH}) = \min(400,\ 150{+}150) = 300$; $d(\text{BER}) = 250$. Update predecessors accordingly. Figure 4 shows the updated state.

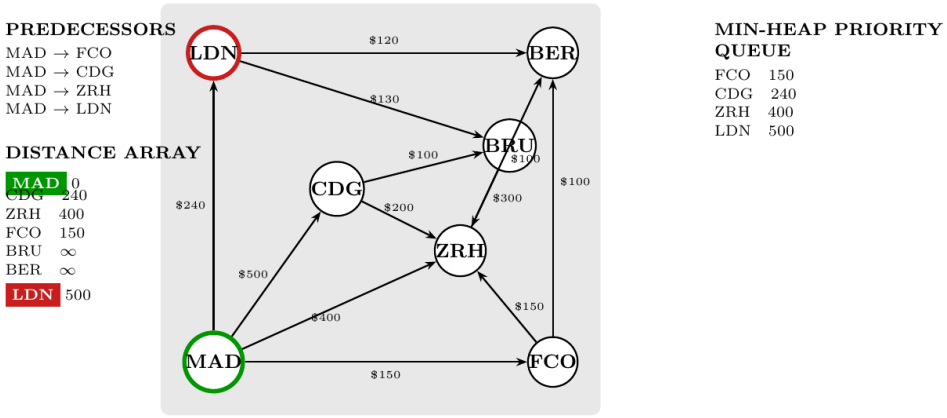


Figure 4. State after processing MAD's neighbours: distance array and min-heap updated, with FCO next in the queue. Author's own figure.

4. Continue extracting nodes in min-heap order — CDG (240), BER (250), ZRH (300), BRU (340) — relaxing neighbours at each step. Figure 5 shows the state after processing FCO.

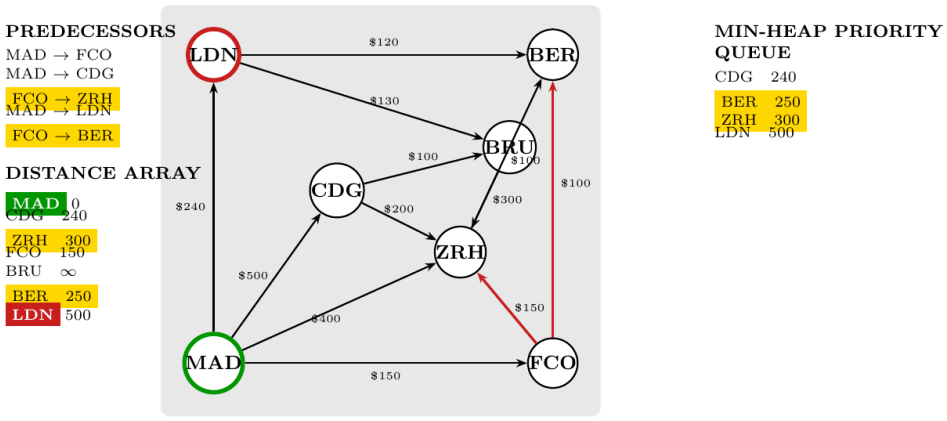


Figure 5. State after processing FCO: BER and ZRH have been relaxed to cheaper costs; predecessors list updated. Author's own figure.

5. Candidate path MAD → CDG → LDN costs $240 + 300 = 540 > 500$; LDN is not relaxed.

6. Algorithm terminates when all nodes are processed. Figure 6 shows the completed distance array and predecessors list.

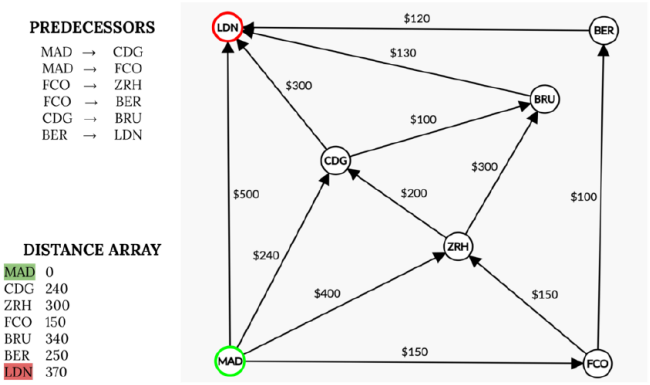


Figure 6. Final state of Dijkstra's Algorithm after termination: complete distance array with optimal costs to all nodes, and full predecessors list enabling path reconstruction. Author's own figure.

7. Optimal path reconstructed via the predecessors list: MAD → FCO → BER → LDN at a total cost of $370. Figure 7 highlights the identified path on the graph.

ɔndon's
und by
t. The
is FCO,
e it to
he trip

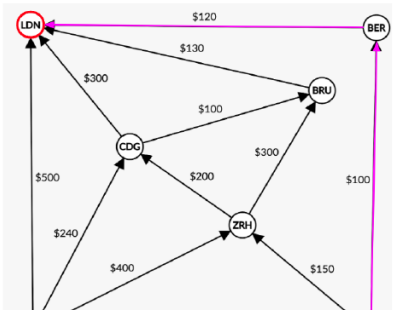


Figure 7. Shortest path identified by Dijkstra's Algorithm: MAD → FCO → BER → LDN at a cost of $370, highlighted in pink. Author's own figure.

2.3 Parameter Definitions

Iterations

An iteration denotes one node-expansion step: the algorithm extracts the highest-priority node from its queue, processes its neighbours, and updates the relevant data structures [5]. Iteration count is used as a proxy for both computational complexity and memory usage, since each iteration requires storing and updating queue and array entries. Fewer iterations indicate lower resource consumption [10].

Distance

Distance is the total length of the identified path, measured either as the number of edges traversed (unweighted graphs) or as cumulative edge weights (weighted graphs). Path length is the primary measure of accuracy [5].

Weights

Edge weights encode the cost of traversal — time, price, physical distance, or any scalar metric. In unweighted graphs all edges are equivalent (weight = 1). Weights substantially increase the complexity of pathfinding and influence which algorithm is most appropriate [10].

3. Hypothesis

Based on the theoretical properties of both algorithms, the following hypotheses are proposed:

- **H1 (Accuracy — unweighted):** In unweighted graphs, both algorithms will return paths of identical length, as the absence of weights neutralises Dijkstra's primary advantage.
- **H2 (Accuracy — weighted):** In weighted graphs, Dijkstra's algorithm will return a more time-optimal path than A*, because A*'s heuristic optimises for spatial distance rather than edge weight.
- **H3 (Speed and memory):** A* will require significantly fewer iterations in all environments due to its directed heuristic search, making it more memory-efficient.

4. Experiment 1 — Small Scale & Unweighted

4.1 Methodology

In 2018, four informatics lecturers at Universitas Trilogi, South Jakarta — Silvester Dian Handy Permana, Ketut Bayu Yogha Bintoro, Budi Arifitama, and Ade Syahputra — conducted a controlled experiment comparing A*, Dijkstra's, and Breadth-First Search (BFS) on a 20×20 unweighted maze [10]. Figure 8 shows the maze structure used.

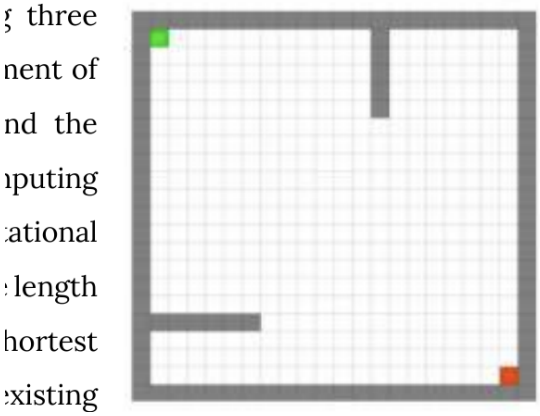


Figure 8. Example 20×20 unweighted maze used in Experiment 1. The green block denotes the source node; the red block denotes the destination. Grey blocks are impassable obstacles. Reproduced from Permana et al. (2018).

Each algorithm was run three times on three distinct maze configurations with increasing obstacle density. Performance was measured across: processing time (ms), path length (blocks), and number of blocks visited (iteration count). A brief note on BFS: Breadth-

First Search explores all nodes at the current depth level before proceeding to the next [5]. It guarantees finding the shortest path in unweighted graphs but is computationally expensive due to its exhaustive, non-discriminating search.

4.2 Data Processing

Figure 9 shows the visual output of each algorithm across the three mazes: column (a) shows A*, column (b) Dijkstra's, and column (c) BFS.

computationally expensive due to its exhaustive search strategy.

5.2 Data Processing

A* (a), Dijkstra's (b), and BFS (c) algorithms were run on the same
below displays the result of these run processes, the table below display
from the processes.

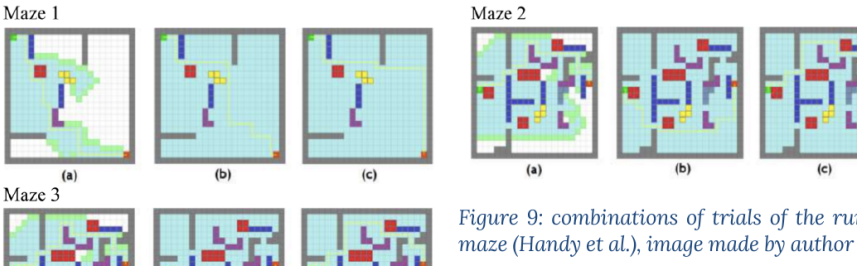


Figure 9: combinations of trials of the ru
maze (Handy et al.), image made by author

Figure 9. Visual output of A* (a), Dijkstra's (b), and BFS (c) on Mazes 1, 2, and 3 respectively. Explored cells are shaded; the identified path is highlighted in yellow. Image by author, adapted from Permana et al. (2018).

Table 1. Results from Permana et al. (2018): A*, Dijkstra's, and BFS on three unweighted mazes.

Maze	Algorithm	Iterations	Path length (blocks)	Time (ms)
1	A*	323	38	0.3500
	Dijkstra's	738	38	0.3000
	BFS	738	38	0.8000
2	A*	415	34	0.4000
	Dijkstra's	618	34	0.6000
	BFS	618	34	0.8000
3	A*	504	58	1.1000
	Dijkstra's	624	58	0.9000
	BFS	624	58	1.0000

4.3 Results and Analysis

Accuracy

All three algorithms returned paths of identical length across all three mazes, confirming Hypothesis H1. In an unweighted graph there is no cost differentiation between edges, so all three algorithms converge on the same optimal path. This is consistent with the theoretical guarantee that any correct shortest-path algorithm will find the same path in unweighted environments [10].

Speed

A* required substantially fewer iterations than Dijkstra's and BFS in every maze: approximately 56% fewer in Maze 1, 33% fewer in Maze 2, and 19% fewer in Maze 3. This efficiency improvement narrows as obstacle density increases, because denser mazes force A* to explore more detours before the heuristic can confidently direct it toward the goal.

Despite fewer iterations, A*'s wall-clock processing time was higher than Dijkstra's in

Maze 3 (1.1 ms vs. 0.9 ms). This apparent paradox arises because each A* iteration is computationally more expensive: the algorithm must evaluate $h(v)$ at every node expansion — overhead absent in Dijkstra's simpler relaxation step. In mazes with many obstacles, this per-iteration cost accumulates.

Memory Usage

A* consistently required fewer iterations, implying lower memory usage. Both Dijkstra's and BFS visited the same number of nodes in each maze, confirming that without a heuristic, both algorithms explore equivalent portions of the graph. BFS's higher processing times despite identical iteration counts reflect its use of a simpler queue structure compared to Dijkstra's min-heap [5].

Summary

A* is the most memory-efficient algorithm for unweighted mazes. Dijkstra's strikes a favourable balance between processing time and iteration count. BFS is the least efficient on both dimensions despite identical accuracy.

5. Experiment 2 — Large Scale & Weighted

5.1 Methodology

In February 2024, Santiago Fiorino, an AI researcher completing a master's degree at the University of Buenos Aires, conducted a comparative simulation of A* and Dijkstra's algorithm on the real street network of Buenos Aires, Argentina [3].

Fiorino used OSMnx — a Python package developed by Dr. Geoff Boeing (University of Southern California) for modelling urban networks from OpenStreetMap data [1] — to convert the city's road network into a weighted node-graph. Each street intersection is represented as a node; each connecting road as a directed edge, with weight equal to the travel time required to traverse that segment given the road type and speed limit. This formulation captures the insight that two roads of identical length may have very different travel times (e.g. a freeway versus a cobbled side street).

Figure 10 shows the Buenos Aires node-graph before experimentation, with source and

destination nodes marked.

map, Fiorino ran both A* and Dijkstra's Algorithm to find the shortest p
as shown in Figure 12.

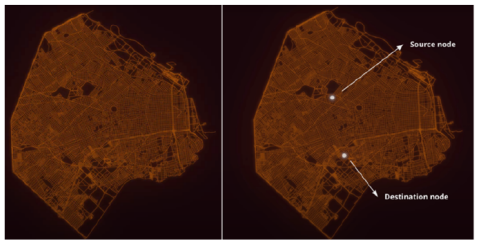


Figure 10: node graph of Bu source and destination no

[15] Buenos Aires is the capital city of Argentina. It has a population of 3.1 millio (Wikipedia 2024)

Figure 10. Node-graph of Buenos Aires, Argentina, generated from OpenStreetMap data using OSMnx. Each node represents a street intersection; each edge a road segment weighted by travel time. Source and destination nodes are marked in white. Reproduced from Fiorino (2024).

Seven independent simulations were conducted, each using a distinct pair of randomly selected source and destination nodes.

5.2 Data Processing

Simulation 1

Figures 11 and 12 show the exploration progress and final path for A* in Simulation 1. After 699 iterations, A* returned a route of 5.94 km with an average speed of 42.3 km/h and a travel time of 8.4 minutes.

Simulation 1 for A*

Below, in Figure 13, the progress of A* pathfinding between the source node and the destination node is displayed.

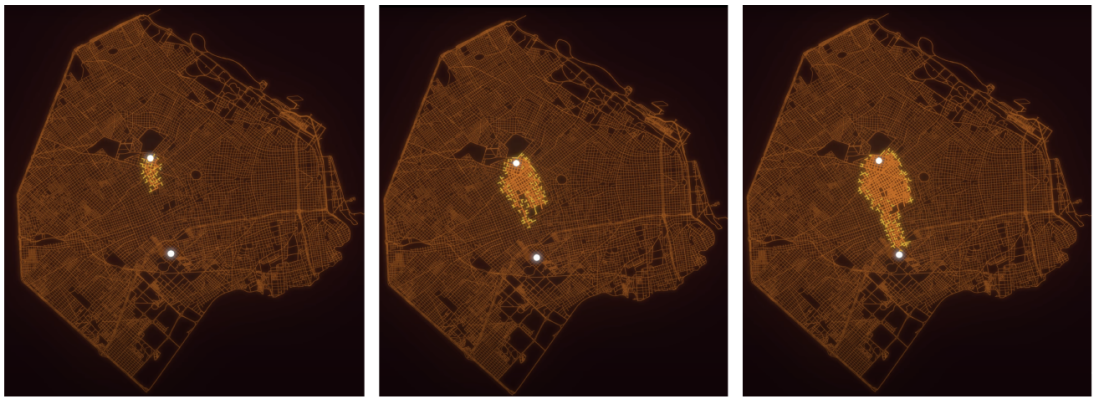
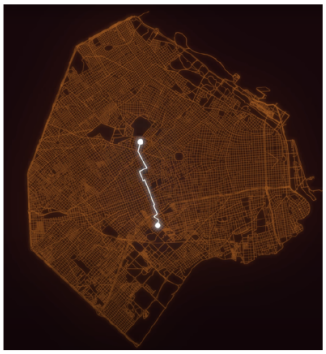


Figure 11. Sequential snapshots of A* exploring the Buenos Aires node-graph during Simulation 1. The highlighted region expands directionally toward the destination, reflecting the heuristic's guidance. Reproduced from Fiorino (2024).



e following
the source
an average
time of 8.4

Figure 12: shortest path generated by A - simulation 1*

Figure 12. Shortest path identified by A* in Simulation 1: 5.94 km, travel time 8.4 minutes. Reproduced from Fiorino (2024).

Figures 13 and 14 show the equivalent results for Dijkstra's Algorithm. After 9,298 iterations, Dijkstra's returned a route of 6.1 km with an average speed of 50.3 km/h and a travel time of 7.2 minutes.

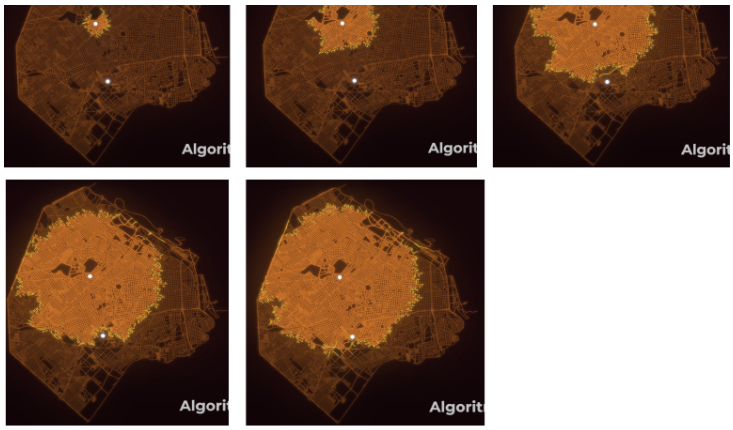


Figure 13. Sequential snapshots of Dijkstra's Algorithm exploring the Buenos Aires node-graph during Simulation 1. The highlighted region expands radially from the source with no directional preference, covering a far larger portion of the graph than A*. Reproduced from Fiorino (2024).

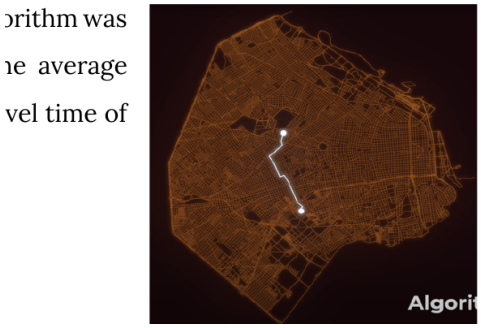


Figure 14. Shortest path identified by Dijkstra's Algorithm in Simulation 1: 6.1 km, travel time 7.2 minutes. Reproduced from Fiorino (2024).

Table 2. Results from Fiorino (2024): Simulation 1 on the Buenos Aires street network.

Measurement	**A***	**Dijkstra's**
Iterations	699	9,298
Path length (km)	5.94	6.10
Avg. speed (km/h)	42.3	50.3
Travel time (min)	8.4	7.2

Simulations 2–7

Figure 15 shows the six additional simulation setups used to broaden the dataset, and Figures 16 and 17 show the exploration progress and resulting paths for all additional

simulations.

To have more accurate conclusions, Santiago Fiorino conc
with six more random source and destination nodes to experime
Plotting 12 more random points, source, and destination.

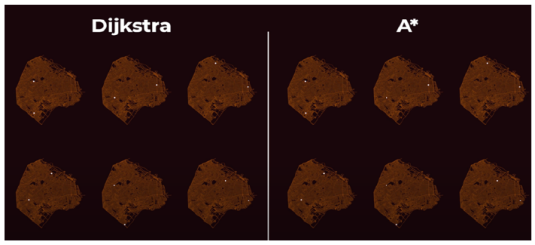


Figure 15: six
Dijkstra's Algorit

Figure 15. Six additional source–destination node pairs used in Simulations 2–7. Dijkstra's pairs are shown on the left; A* pairs on the right. Reproduced from Fiorino (2024).

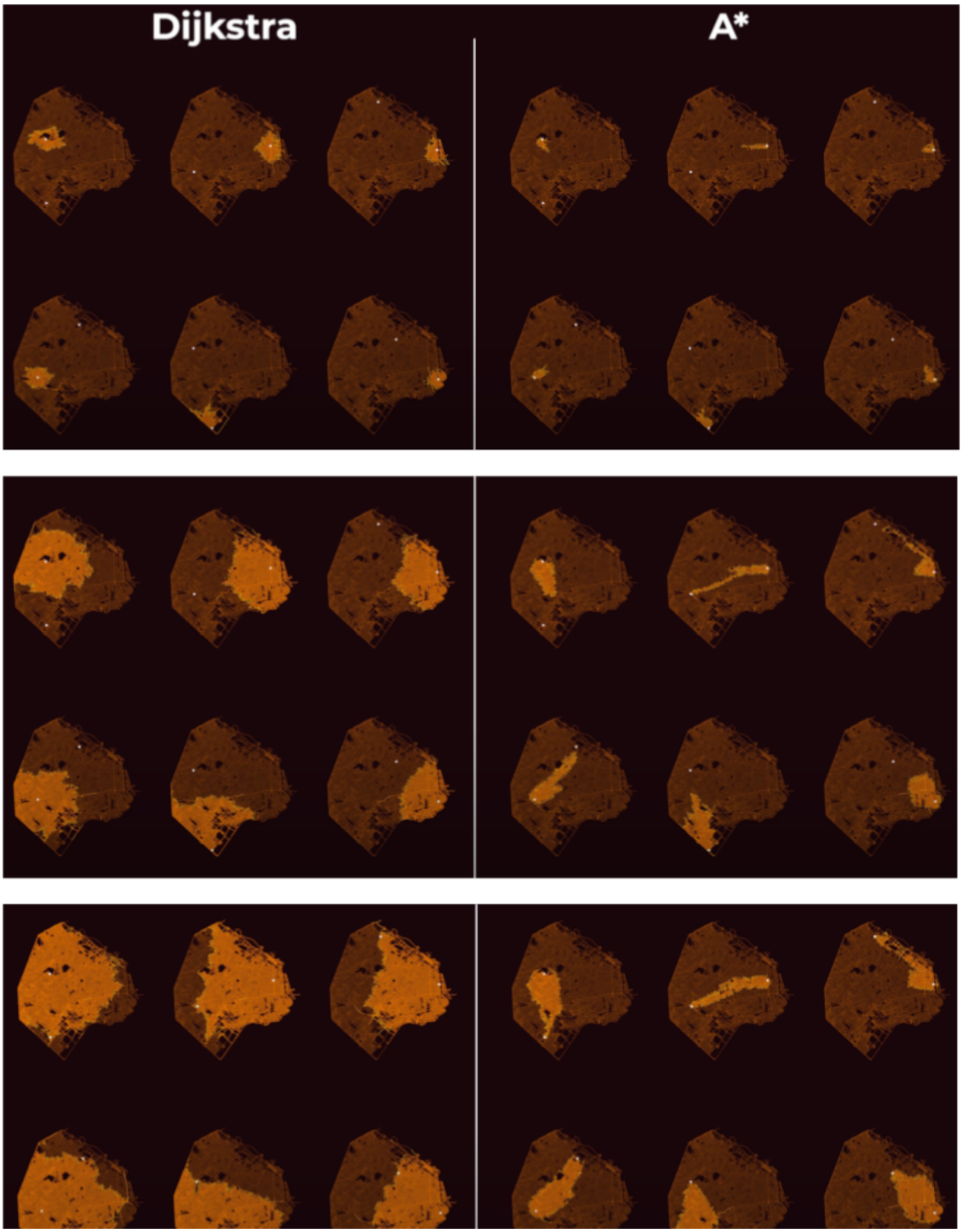


Figure 16. Exploration progress for Simulations 2–7: Dijkstra's Algorithm (left column) and A* (right column). Dijkstra's consistently explores a far larger fraction of the graph before terminating. Reproduced from Fiorino (2024).

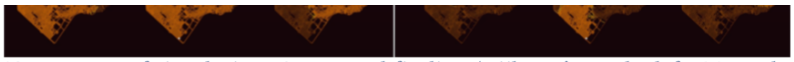


Figure 16: progress of simulations 2-7 at pathfinding (Dijkstra's on the left, A on the right)*

In the figure above the progress of the simulations 2-7 can be seen. The shortest paths found by Dijkstra's algorithm and A* can be seen to the below.

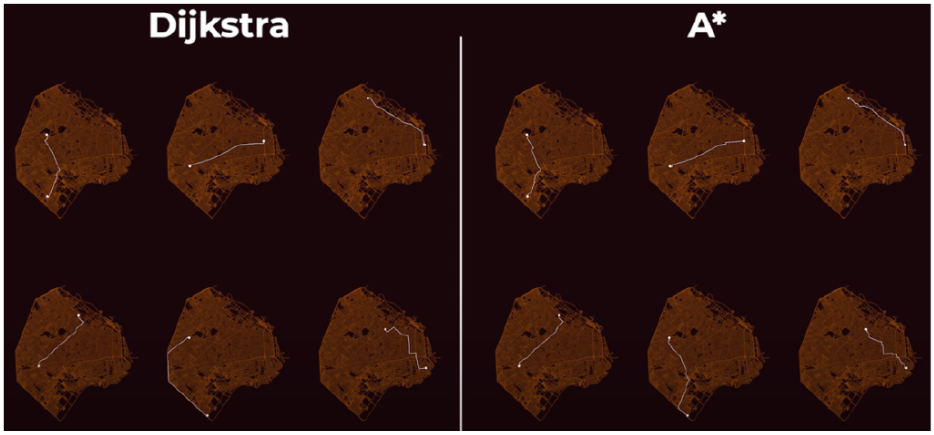


Figure 17. Final paths returned for Simulations 2–7: Dijkstra's Algorithm (left column) and A* (right column). Dijkstra's paths are systematically longer in distance but faster in travel time. Reproduced from Fiorino (2024).

Table 3. Raw results for all seven simulations on the Buenos Aires weighted node-graph (Fiorino, 2024).

Sim.	Algorithm	Iterations	Distance (km)	Avg. speed (km/h)	Time (min)
1	A*	699	5.94	42.3	8.4
	Dijkstra's	9,298	6.10	50.3	7.2
2	A*	2,082	10.40	49.2	12.7
	Dijkstra's	14,041	10.40	58.0	11.2
3	A*	921	11.80	51.5	13.7
	Dijkstra's	11,503	11.90	58.5	12.2
4	A*	1,104	11.80	55.7	12.7
	Dijkstra's	8,362	11.60	62.3	11.1
5	A*	2,471	10.60	55.2	11.6
	Dijkstra's	13,462	10.70	57.9	11.1
6	A*	3,720	13.50	46.9	17.3
	Dijkstra's	10,130	15.20	63.0	14.4
7	A*	2,332	10.20	39.7	15.5
	Dijkstra's	6,879	10.90	57.3	11.4

Table 4. Normalised comparison of A* and Dijkstra's across all seven simulations (A* = 100%).

Metric	A*	Dijkstra's
Iterations	100%	674%
Path length	100%	103%
Travel speed	100%	121%

5.3 Analysis and Evaluation

Memory Usage

The iteration differential is the most striking finding: Dijkstra's required on average 674% more iterations than A* — nearly seven times as many node expansions, confirming Hypothesis H3. The difference arises because A*'s heuristic $h(v)$ constrains exploration to a directed corridor toward the destination, whereas Dijkstra's expands radially from the source with no directional preference, visiting large portions of the graph irrelevant to the chosen destination.

Accuracy

Dijkstra's paths were on average 3% longer by distance but 21% faster by travel time, confirming Hypothesis H2. The apparent paradox — a longer route arriving faster — is explained by the algorithm's exploitation of weighted edge information. Dijkstra's consistently identifies and uses high-speed road segments even when they require a longer total distance.

This behaviour is most clearly illustrated in Simulation 6, shown in Figure 18. A* selected a direct 13.5 km route taking 17.3 minutes. Dijkstra's took a 15.2 km route via *Avenida General Paz* — the beltway freeway encircling Buenos Aires — completing the journey in 14.4 minutes, saving 2.9 minutes despite the 1.7 km detour. Figure 18 cross-references Dijkstra's routes with the official Buenos Aires freeway map, confirming the algorithm correctly identified and exploited the high-speed corridor. Simulation 4 shows analogous behaviour via *Autopista Arturo Illia*.

Aires found on the government website (buenosaires.gob.ar).

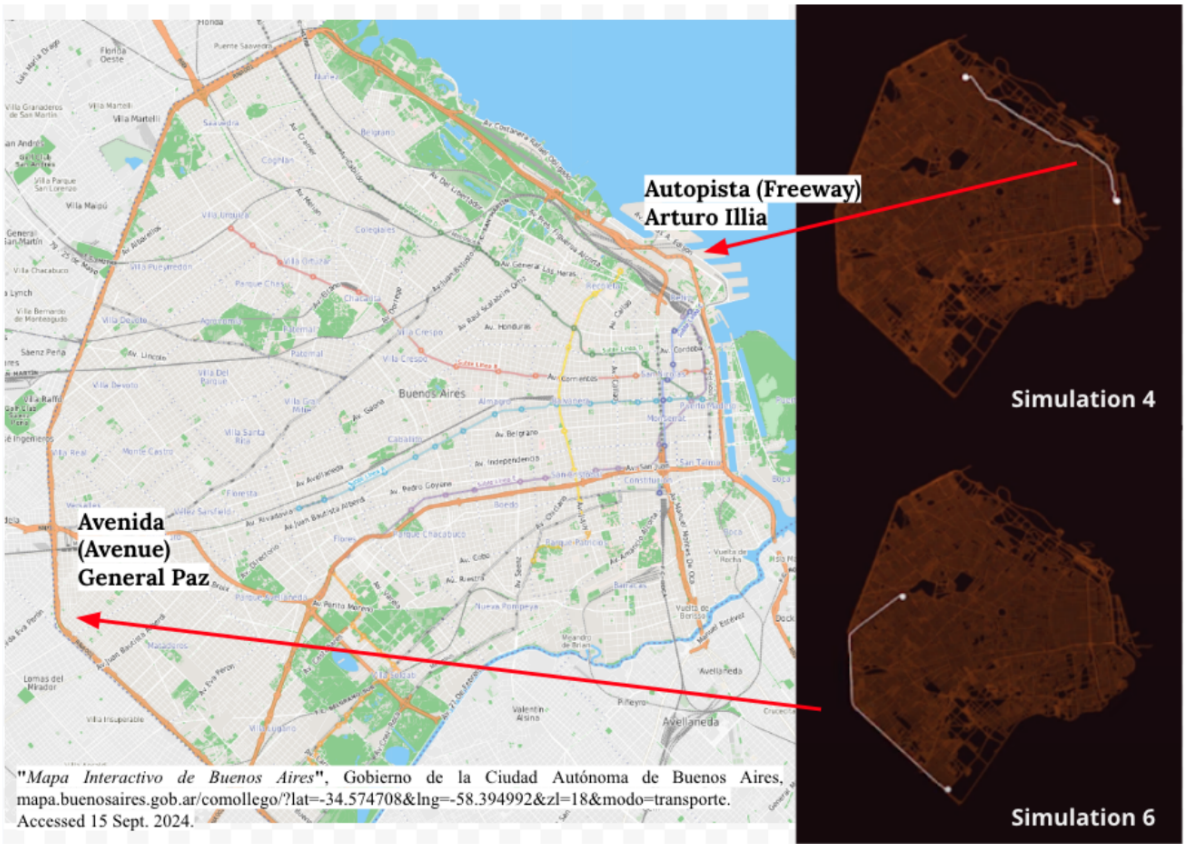


Figure 18. Side-by-side comparison of the official Buenos Aires freeway map (left) and Dijkstra's Algorithm routes for Simulations 4 and 6 (right). Dijkstra's paths closely follow the Autopista Arturo Illia (Simulation 4) and Avenida General Paz beltway (Simulation 6), demonstrating the algorithm's correct identification of high-speed road segments. Left: "Mapa Interactivo de Buenos Aires", Gobierno de la Ciudad Autónoma de Buenos Aires, accessed 15 Sept. 2024. Right: Fiorino (2024).

Speed

Although Dijkstra's requires many more iterations, the per-iteration cost is lower than A* (no heuristic evaluation). Nevertheless, the sheer volume of additional iterations means Dijkstra's overall computation time is substantially higher than A*'s in large-scale graphs, consistent with the theoretical time complexity of $O((V + E) \log V)$ for both algorithms with a binary heap, but with Dijkstra's exploring a much larger fraction of V and E.

6. Conclusion

6.1 Summary of Results

The two experiments together provide a coherent picture of the complementary strengths of A* and Dijkstra's algorithm.

In **small-scale, unweighted environments** (Experiment 1), A* demonstrated superior memory efficiency, requiring approximately 40–56% fewer iterations than Dijkstra's. Both algorithms achieved identical path accuracy, as the absence of weights renders the heuristic's directional advantage irrelevant to path optimality. A*'s per-iteration overhead occasionally made it slower in wall-clock time (Maze 3), but its lower iteration count makes it preferable for memory-constrained applications. These findings make A* particularly suited to video game pathfinding and robotics, where repeated rapid computation within bounded environments is required.

In **large-scale, weighted environments** (Experiment 2), Dijkstra's algorithm outperformed A* on the primary optimisation criterion — travel time — by an average of 21% across seven simulations. It did so by exploiting weighted edge information to identify faster road segments, even at the cost of greater total distance. This came at the price of 674% more iterations. These properties make Dijkstra's the superior choice for GPS navigation, transport logistics, and network routing, where optimal path cost is paramount.

6.2 Comparison to Literature Values

The results align with prior theoretical and empirical literature. The efficiency of A*'s heuristic in sparse or bounded graphs is well documented [4, 16]. Dijkstra's dominance in weighted graph optimisation is equally established [5, 8]. The specific quantitative finding — that Dijkstra's requires approximately 6–13× more iterations than A* in a large urban graph — is consistent with the $O(|V|)$ exploration difference predicted when an effective heuristic eliminates non-shortest-path node expansions [4].

6.3 Scientific Explanation of Results

A*'s efficiency derives directly from equation (1). The term $g(v)$ prevents the algorithm from wandering arbitrarily far from the source; $h(v)$ prevents it from exploring nodes far

from the destination. Together, they focus computation on a narrow band of promising nodes.

Dijkstra's algorithm, lacking $h(v)$, expands outward uniformly from the source. Its exhaustiveness is simultaneously its weakness (high iteration count) and its strength: because it considers every reachable node and every edge weight without directional bias, it cannot overlook a time-saving detour that A*'s Euclidean heuristic would discount.

An important additional property of Dijkstra's: a single run produces shortest-path distances from the source to *every* node in the graph [12]. If the application requires shortest paths to multiple destinations, Dijkstra's amortises its high iteration count across all queries simultaneously, whereas A* must be re-run independently for each destination.

7. Evaluation

7.1 Strengths

- The investigation draws on two methodologically distinct sources — a peer-reviewed academic paper [10] and an independent graduate-level simulation [3] — reducing the risk of source bias.
- The use of a real urban road network (Buenos Aires) ensures ecological validity: results reflect algorithm behaviour on graphs of realistic size, density, and weight distribution.
- The three-axis evaluation framework (speed, memory, accuracy) provides a structured and comprehensive basis for comparison.
- Seven independent simulations in Experiment 2 provide a sufficient sample for meaningful averaging and cross-simulation pattern identification.

7.2 Weaknesses

- The investigation examines only two graph scales (small unweighted, large weighted), omitting intermediate cases such as medium-sized weighted graphs or graphs with dynamically changing weights — environments where the relative performance of the

two algorithms may differ.

- Experiment 2 relies entirely on secondary data reported by a single researcher. Independent replication with controlled random-seed selection and larger node-pair samples would strengthen the conclusions.
- The heuristic used in A* (Euclidean distance) is one of several options. Experiments with Manhattan or Diagonal distance heuristics on the same graphs could reveal whether the observed accuracy gap between A* and Dijkstra's is heuristic-dependent.
- Iteration count is used as a proxy for memory usage, but does not capture peak heap size or predecessor-list length, which may differ even when iteration counts are similar.

References

[1] Boeing, G. "Modeling and Analyzing Urban Networks and Amenities with OSMnx." *Working Paper*, 2024. https://geoffboeing.com/publications/osmnx-paper/

[2] Boragano, Susana H. "General Paz." *Historias de la Ciudad*, no. 11, Sept. 2001, p. 1.

[3] "Comparing Algorithms: A* vs Dijkstra, in the City's Map." *YouTube*, uploaded by Santiago Fiorino, 9 Feb. 2024. https://www.youtube.com/watch?v=oMgfGkFSgI0

[4] "Comparison of A* and Other Pathfinding Algorithms." *Stanford University Game Programming*. https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

[5] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009.

[6] Cui, Xiao, and Hao Shi. "A*-based Pathfinding in Modern Computer Games." *International Journal of Computer Science and Network Security*, vol. 11, no. 1, 2011, pp. 125–130.

[7] Duchoň, František, et al. "Path Planning with Modified A Star Algorithm for a Mobile Robot." *Procedia Engineering*, vol. 96, 2014, pp. 59–69.

[8] "Data Structures and Algorithms: 10.2.1 Predecessor Lists." *Algorithm Animations*, University of Auckland, 1998. www.cs.auckland.ac.nz/software/AlgAnim/dij_pred.html

[9] "Heaps and Priority Queues." *HackerEarth*. www.hackerearth.com/practice/notes/heaps-and-priority-queues/

[10] Permana, Silvester Handy, et al. "Comparative Analysis of Pathfinding Algorithms A*, Dijkstra, and BFS on Maze Runner Game." *International Journal of Information Systems and Technology (IJISTECH)*, vol. 1, no. 2, 2018, pp. 1–6.

[11] Ravulakollu, Kiran Kumar, and Shailashree K. Sheshadri. "Ds* Heuristic Approach using 'Safety Distance' for Agent Path Planning." *IOSR Journal of Computer Engineering*, vol. 16, 2014, pp. 109–115.

[12] Sambol, Michael. "Dijkstra's Algorithm in 3 Minutes." *YouTube*, 16 Sept. 2014. www.youtube.com/watch?v=QO0wokRL0aI

[13] Shahi, Gurpreet Singh, et al. "A Comparative Study on Efficient Path Finding Algorithms for Route Planning in Smart Vehicular Networks." *International Journal of Computer Networks and Applications*, vol. 7, no. 5, 2020, pp. 157–166.

[14] "Shakey the Robot." *SRI International*. https://www.sri.com/hoi/shakey-the-robot/

[15] The Editors of Encyclopaedia Britannica. "Edsger Dijkstra." *Encyclopaedia Britannica*, 2 Aug. 2024. https://www.britannica.com/biography/Edsger-Dijkstra

[16] "Understanding Heuristics in Pathfinding." *Stanford University Game Programming*. https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html