QuickNews

Local news as they happen

Ekfontan, Lilliawa, and Merijnjb

Scenario Overview

- Supplying camera drones for a big city news agency.
- Wants to use drones to take better and quicker pictures when action happens.
- The drones need to be able to get from **A** to **B** as quickly as possible.
- Drones should have built in avoidance and the algorithms should be able to run directly on the drone, therefore computation speed is a critical factor.
- That is why we are looking for the best pathfinding algorithm for the drones within a 3d environment.

Algorithms

Dijkstra

Expanding area that keeps track of the total cost of the different paths. Returns the path with least cost.

Breadth-First Seach

Expanding area that selects the first path to the goal.

A* (octile)

Similar to Dijkstra, but uses a Heuristic, which incentives selection of nodes closer to the goal.

Bi A* (octile)

Bi directional A*, essentially expanding from both start->end and end->start at the same time.

Theta*

an any angle Line of Sight algorithm which uses LoS to create the most direct path, and is thus not limited to grid based movement.

Octile is a heuristic which approximates the euclidean distance to the target using the formula: (where dx is the distance to the target in the x axis, etc.)
SQRT2 and SQRT3 are pre calculated values, and it thus doesn't use any SQRT while running the computations and is therefore generally faster than pythagorean.

```
dmax = max(dx, dy, dz)
dmin = min(dx, dy, dz)
dmid = dx + dy + dz - dmax - dmin

return dmax + SQRT2_MINUS_1 * dmid + SQRT3_MINUS_SQRT2 * dmin
```

Metrics:

Steps	*Total amount of nodes in the path*
Distance_Cost	*Total euclidean distance travelled, calculated by summing the distance between the nodes in the path using pythagorean theorem*
Operations	*Total amount of searches the algorithm does before finding a path*
Computation_seconds	*Total time, in seconds, each algorithm used to compute a path*

Hypothesis

Expectations regarding algorithms' performance

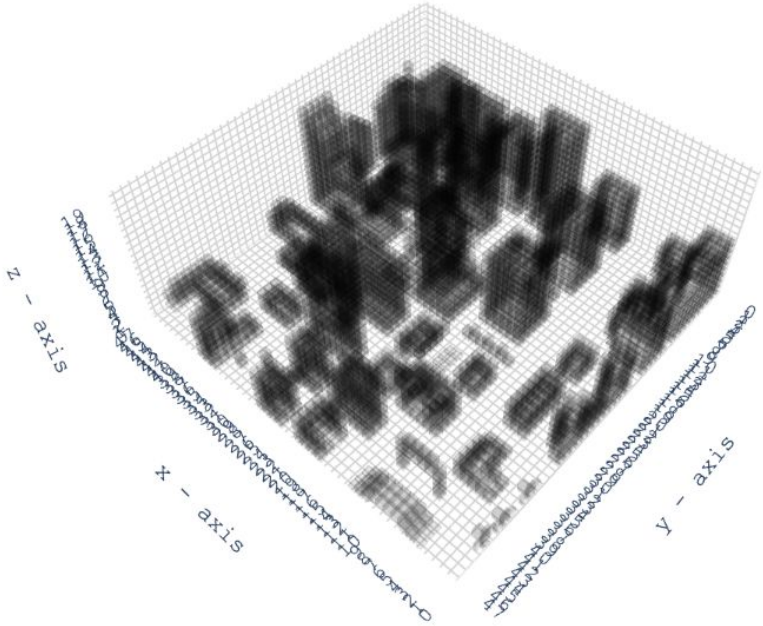
H1. We expect **Dijkstra** will use the most time and operations.

H2. **Theta*** will find the shortest path.

H3. With focus on relation between processing time and path cost we anticipate that **A*** will be the over-all most optimal algorithm for our purpose.

The Map

For the creation of the map we found a Minecraft .nbt to .json converter and used air for walkable, green_wool for startpoints, red_wool for endpoints, and everything else for obstacles.



Size is 48x48x20

Tool Overview

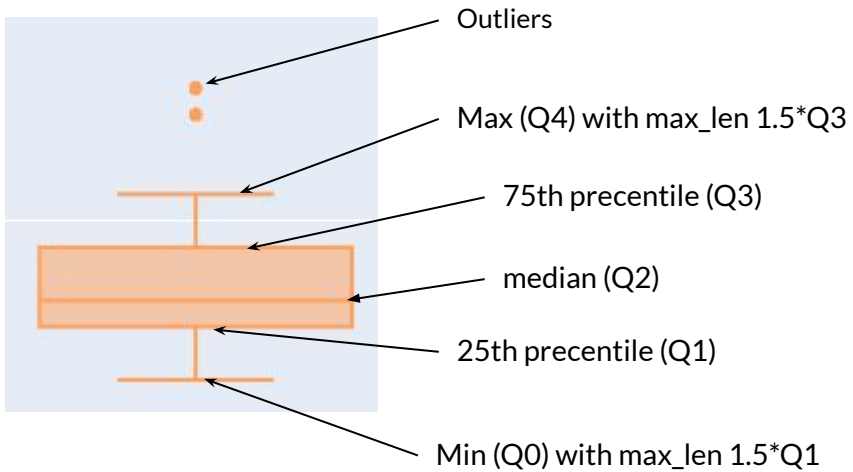
The tools used to run the algorithms:

- Python
- Numpy
- External pathfinding library
  - github.com/harisankar95/pathfinding3D
  - Seems pretty reliable

Visualisation of statistics

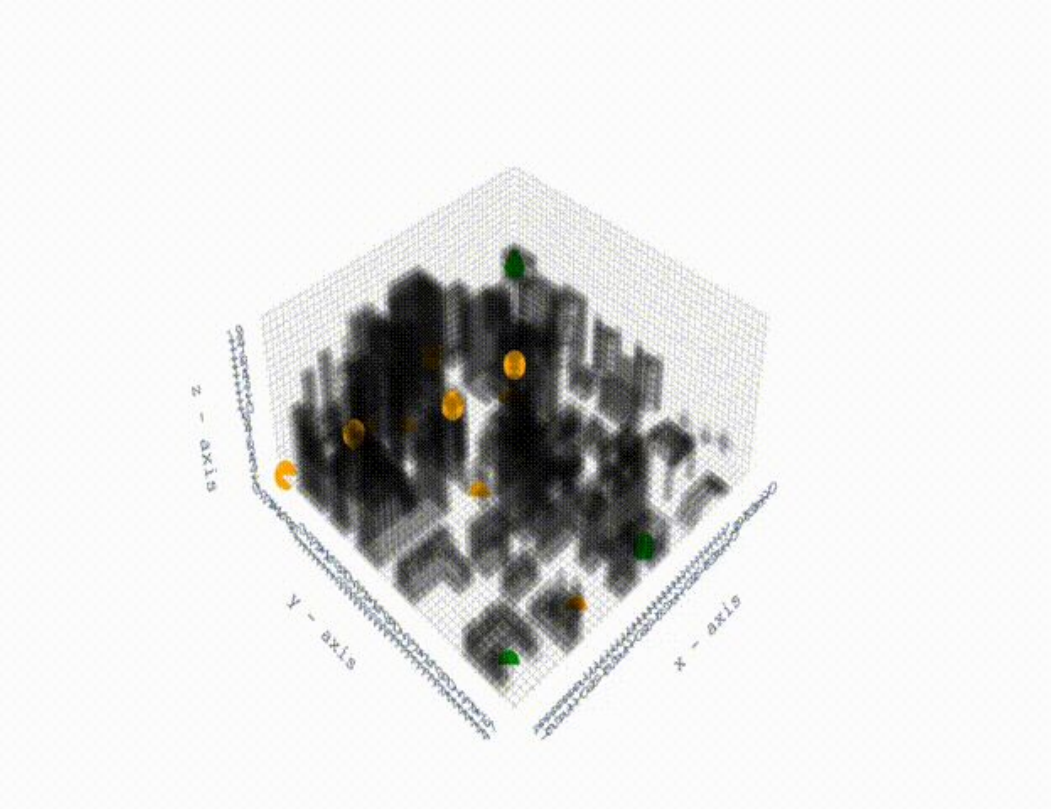
- Plotly
  - Boxplots and tables
  - mesh3d for 3d visualisation

Boxplot explanation:

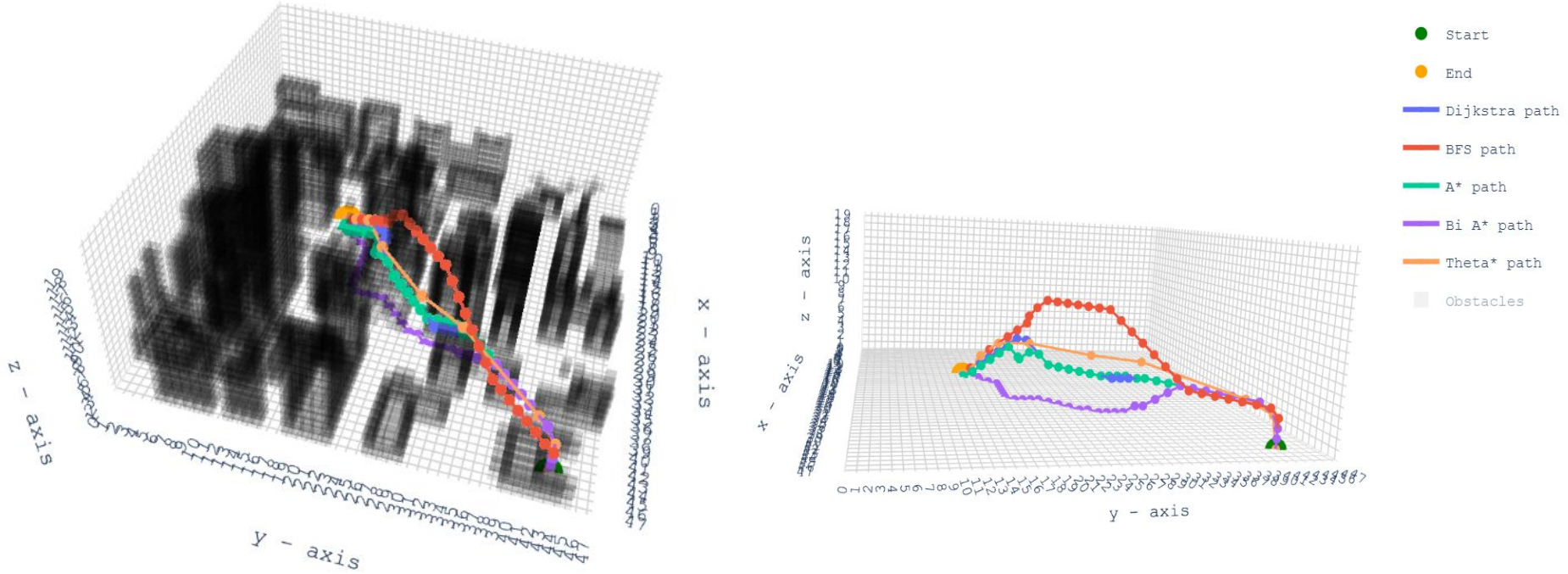


Start and endpoints

- 3 Start points
  - Green points
- 10 Endpoints
  - Orange points
  - Some might be hidden down alleyways and “inside” buildings
- **30 Total paths combinations**

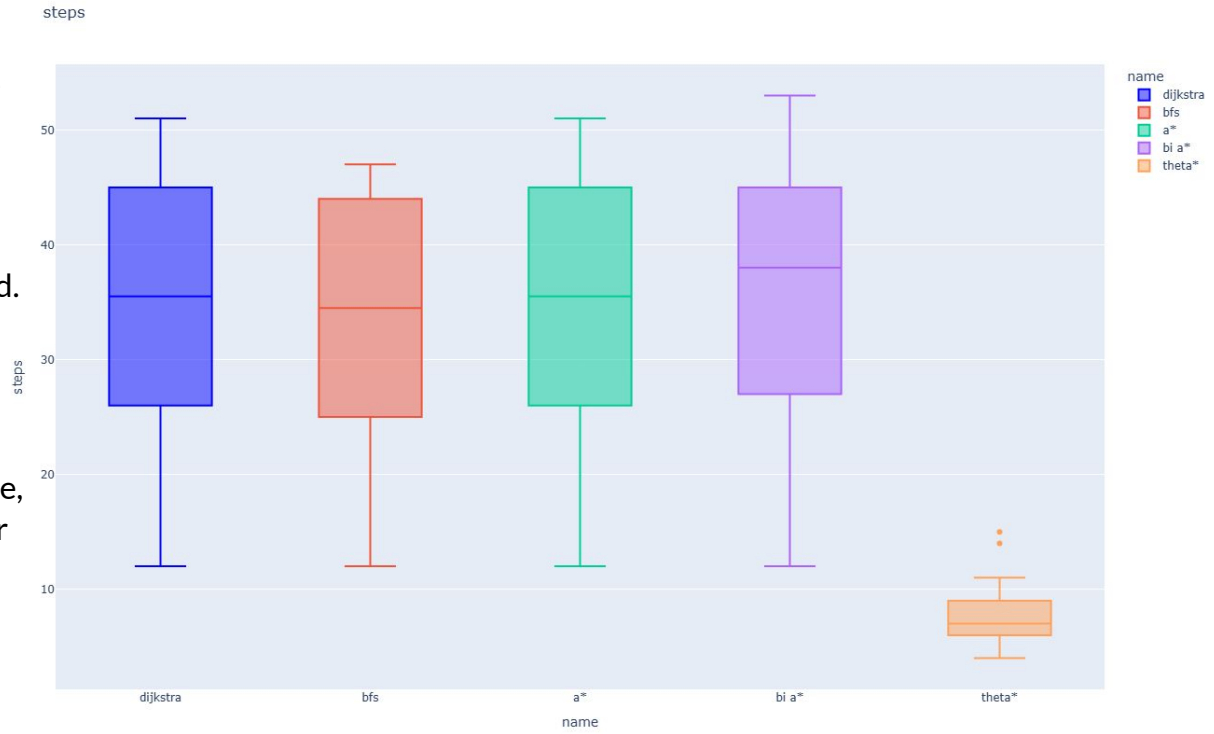


Running the algorithms



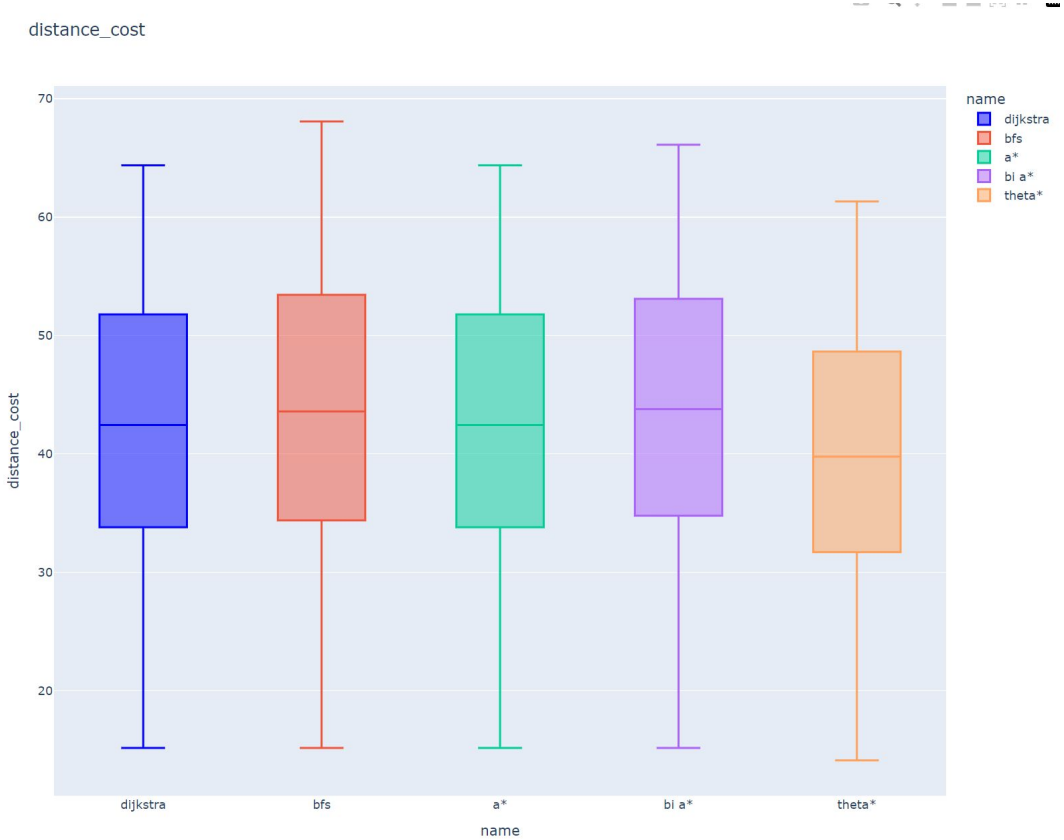
Steps

- Theta* has very **few steps** compared to the other algorithms.
    - This result comes from that Theta*'s steps may traverse any distance and at any angle thus leaving less nodes within the grid.
- Out of the grid-based algorithms **BFS** has the **lowest overall steps**.
- Bi A* median appear to be slightly worse, but its Q1 and Q3 are still pretty similar to A* and Dijkstra.
- A* and Dijkstra are identical.



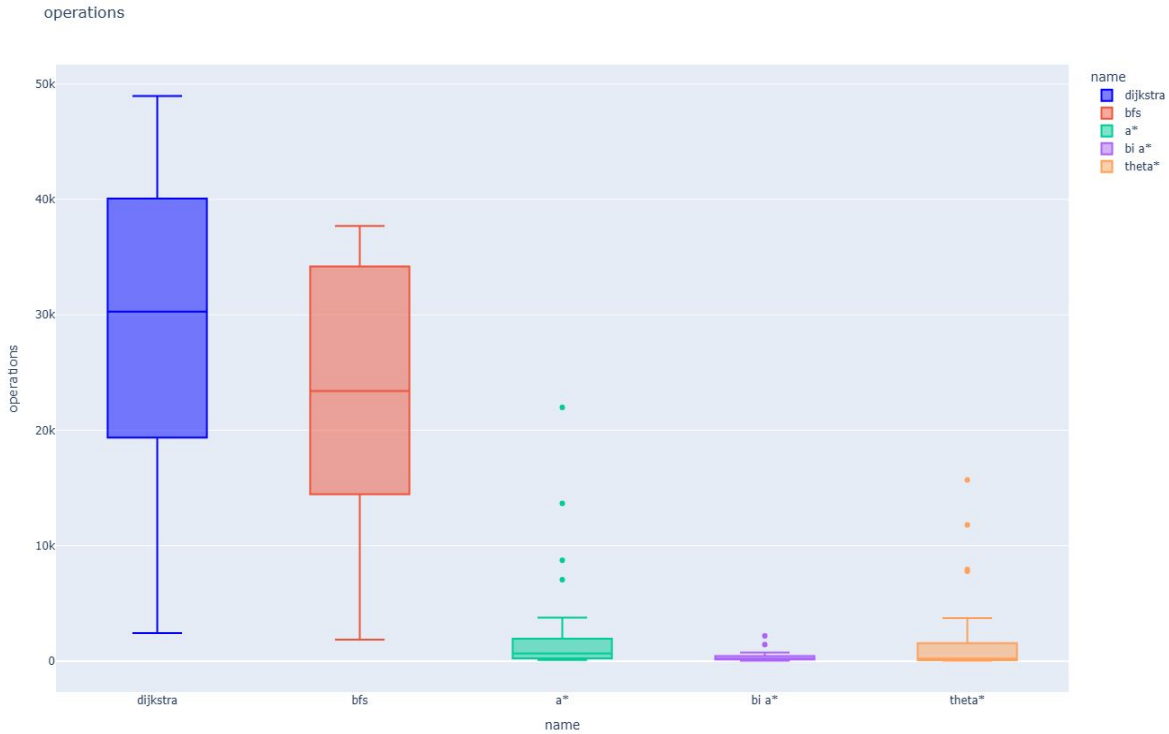
Distance cost

- Theta* has the **lowest median and Q4 cost**.
  - This makes sense due to Theta* isn't limited to grid-based movement and may therefore take "short-cuts" at angles that the other algorithms cannot travers.
- Even though BFS has the least amount of steps among the grid based algorithms, it still has the **highest average distance cost**.
- Dijkstra and A* also found paths of identical cost.



Operations

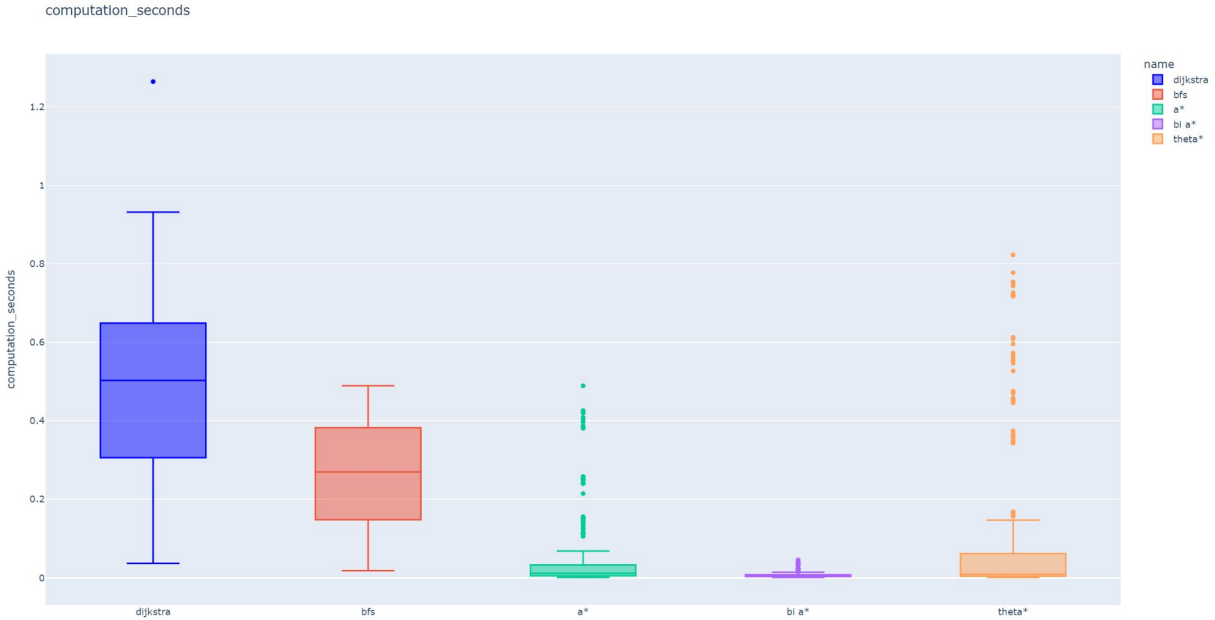
- Dijkstra very clearly does the **most operations**.
  - This is due to the algorithm checks every available path to the target.
- BFS appears to also do **a lot of operations**.
  - Due to the algorithm searching every available path in an expanding area with no guiding heuristic.
- A* and Theta* have similar amount of operations.
- Bi A* has by far the fewest.



Time (seconds)

- Dijkstra is the s**lowest of the algorithms**.
  - Expected due to all the operations it does
- BFS also appears **significantly slower** than the other algorithms.
  - But unlike the other algorithms has no clear outliers
- A* and Theta* has a very **low median**, but have a lot of **slower outliers**
  - Even though they had similar amount of operations, A* is clearly faster on average.
- Similarly to operations Bi A* is very fast.
  - It's outliers also appears to be a lot closer to it's median

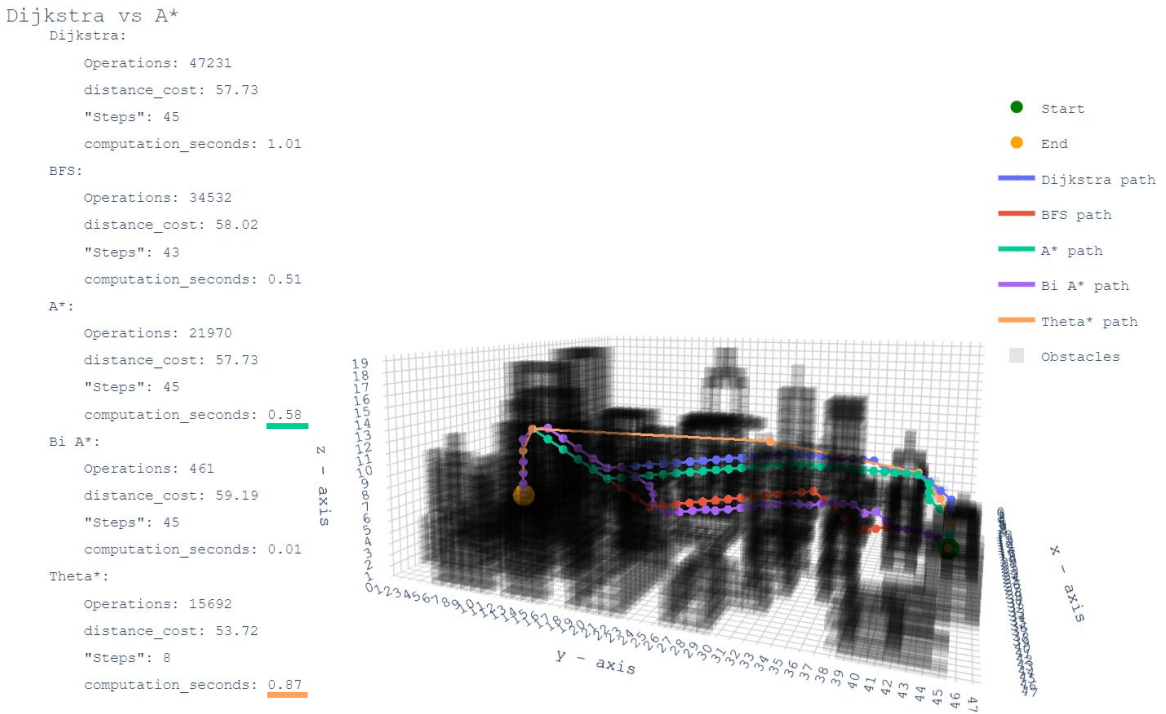
Disclaimer: There are lot of factors that may influence the processing time. Thus for this boxplot we ran the algorithms through the 10 times to get a bigger dataset.



Outliers

A* and Theta* both generally used a lot of computation_seconds with this endpoint, which is deep inside a building.

We assume this is because the heuristic leads the algorithm to the bottom of the building since that is technically the closest point. The heuristic then slowly leads it around the building upwards (and outwards?) since that is still “closer” than going directly up to the entrance of the building. This requires further experimentation.



Final conclusion

Table over deciding mean values:

name	mean distance_cost	mean computation_seconds
Dijkstra	41.90557790688593	0.47559383074442546
BFS	43.40635659314286	0.2596064551671346
A*	41.90557790688593	0.04583266814549764
Bi A*	42.73806762337555	0.0073943678538004555
Theta*	39.52582918190519	0.09190219402313232

The parts of our hypothesis which was confirmed:

- **H1**. Dijkstra took the **longest time.**
- **H2**. Theta* produced the **shortest paths on average**.

Observations:

- A*, due to its **low mean cost and relatively low mean time,** appears to be a optimal option
- Bi A*, due to its **shorter mean time and relative low mean cost,** appears almost equally as optimal as A*.
- Theta* has a **shorter distance** cost than A*, but also **doubles the computation time**.
    - We believe this difference in computation time is too detrimental for real time obstacle avoidance in our scenario.

In conclusion, for **H3**, the most optimal algorithm for our purpose comes down to either Bi A*, which uses very little time and thus may be better suited for chaotic environments and avoiding sudden obstacles, or A* with its slightly lower mean distance_cost but still fast time.

Further development

- Trajectory planning
  - Travel time instead of distance traveled?
    - Will Theta* make an even shorter path?
- Dynamic environment
  - How will moving parts affect our results? What effect will weather have?
  - Will Bi A* have even more distance to traverse?
- Bigger and more advanced map
  - How will the size of the map, air restrictions, battery, etc. affect the results?
  - Weights next to buildings, increasing safety?
  - More start and end points.
- Claustrophobic map, endpoints in tunnels & buildings, etc.
  - A* and Theta* showed quite a few bad outliers when navigating to our end inside a building. Is this a coincidence or weakness?
- Bi Theta* (if feasible), Lazy Theta*, Path Smoothing Bi A*?
  - Less outliers, shorter distance?

Thanks for us :)

https://github.com/Mebu98/pathfinding3D

(Bit of a mess)