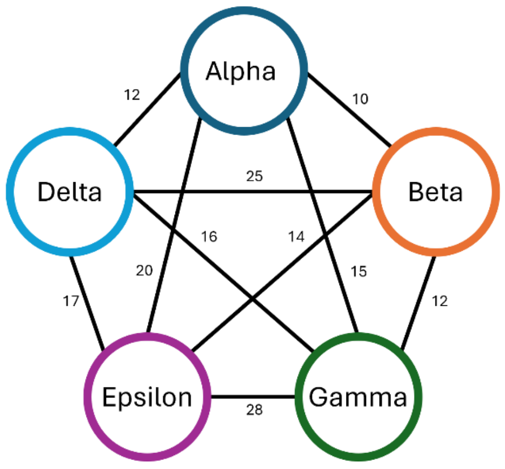
Computational Methods Assignment

Task 3

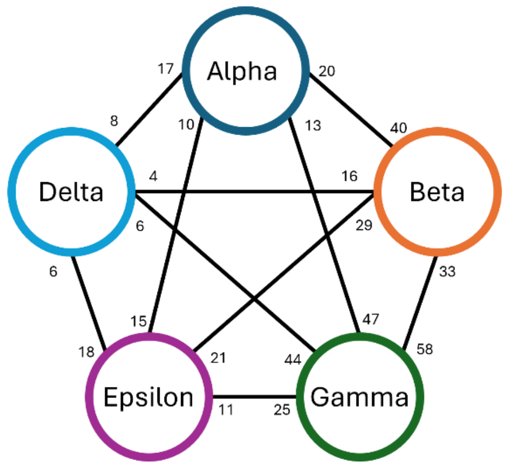
For this task, I've chosen to use the greedy Dijkstra's Algorithm. Dijkstra's Algorithm works by finding the shortest path from one node to all other nodes. It does so by repeatedly selecting the nearest unvisited node and calculating the distance to all unvisited neighbouring nodes. *(W3Schools DSA Dijkstra's Algorithm, 2024)*

Originally, I planned to use just the planetary distances, but I believe the algorithm will work better if I create weighted distances, to get these weighted distances I divided each planet's respective load by the planetary distances between two planets, then multiplied that by 10 and rounded them. Solely dividing load by distance results in small decimal numbers which lose accuracy when rounded, I multiply by 10 and round to increase the weight's accuracy to one decimal place.

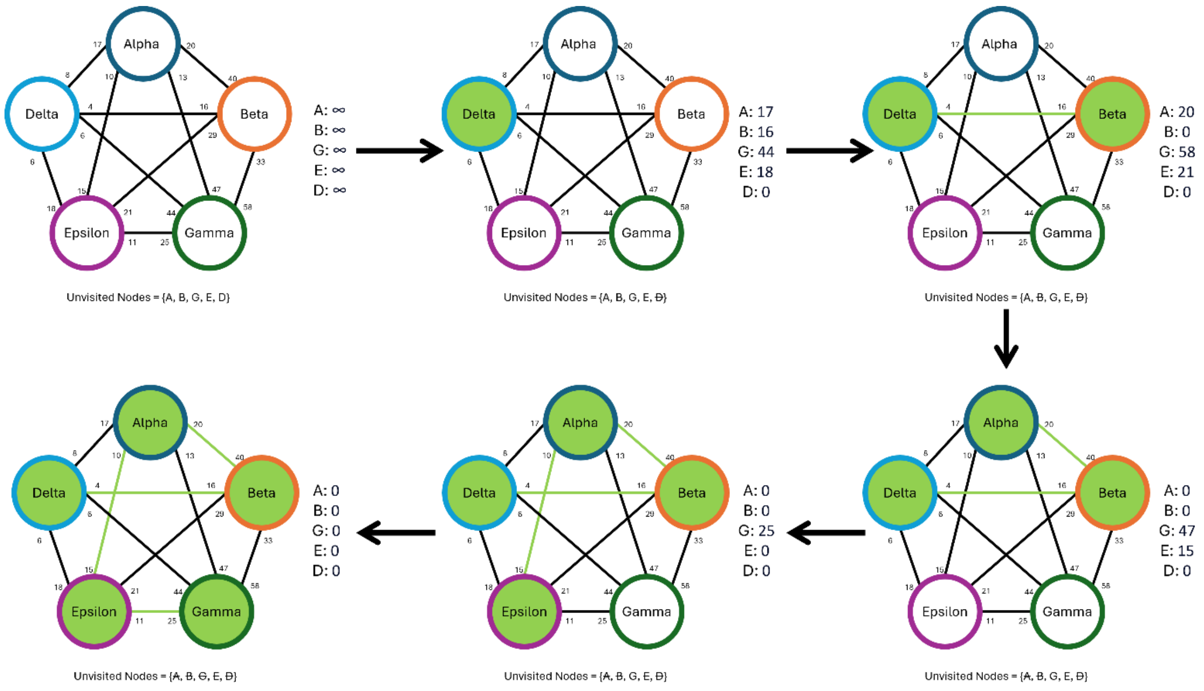
Using Planetary Distance

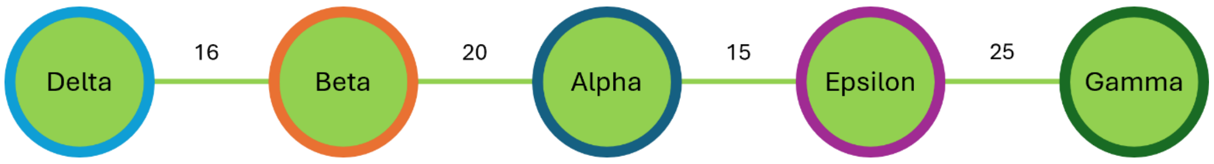


Using Weighted Distances
(distance / load * 10)



I've chosen Planet Delta as my graph's origin point due to it having the smallest average weight, alongside my prior knowledge of knowing that Delta → Alpha → Epsilon → Beta → Gamma is the fastest route through my brute forcing task.





Diagrammatic Form created with help from (*Michael Sambol 2014*)

With Dijkstra's Algorithm applied above we can see that the final cost of this route is

Planet	Delta	Beta	Alpha	Epsilon	Gamma	Total
Distance	0	25	10	20	28	83
Weight	10	40	20	30	70	170
Fuel Cost	0	6250	12500	35000	70000	123,750

This is a good example to show the downsides of greedy algorithms, because we're being greedy, although Dijkstra's algorithm leads to the shortest current path, it might prematurely choose a path that seems optimal in the short term but worse in the long term. We can clearly see this in the above example now we already know the best path from brute forcing. In the short term, it seems like beta is the best path to follow, however we read Alpha and find out from now on the path is suboptimal. We are forced to follow this path as we cannot backtrack upon ourselves. In large graphs, simple implementations of Dijkstra and lead to this algorithm becoming inefficient.

For Dijkstra's Psuedocode, we begin with initialising all nodes with an infinite distance and initialise the starting node with 0, then we mark the distance of the starting node as permanent, and all other distances as temporary. We calculate all the temporary distances of every node neighbouring the active node, if the calculated distance of a node is smaller than the current one, it’s distance is updated. Afterwards we set the node with the minimum temporary distance as active and it’s distance as permanent, lastly we repeat this until there aren’t any nodes left with a permanent distance that has neighbours with temporary distances. (*Dijkstra Algorithm: Short terms and Pseudocode, 2016)*

```
function dijkstra(Graph, source):
    for each vertex v in Graph:
        dist[v] = infinity
        previous[v] = undefined
    dist[source] = 0
    Q = all nodes in Graph
    while Q has nodes
        u = node in Q with smallest dist[]
        remove u from Q
        for each neighbour v of u:
            alt = dist[u + dist_between(u, v)
            if alt < dist[v]
                dist[v] = alt
                previous[v] = u
    return previous[]
```

The time complexity for Dijkstra’s algorithm is $O(n^2)$. This is because the node with the lowest distance must search for the next node and that takes O(n) time. Since this must be done for every node connected to the source, when factored in results in $O(n^2)$.

However, when using alternative data structures for the distances, such as Min-heap or Fibonacci-heap structures, the time needed to search for the minimum distance node is reduced from O(n) to O(log n). Resulting in O(n · log n + e) where n is the number of nodes and e is the number of edges, so we multiply the number of nodes by the logarithm of n + e. The improvement we get from using a Min-heap data structure is good especially when you have a graph with a large number of nodes, but not as many edges. Implementing the Fibonacci-heap data structure is better for dense graphs where each node has an edge to almost every other node, which is the situation we are in. (Time Complexity for Dijkstra's Algorithm, *W3Schools DSA Dijkstra's Algorithm (2024)*)

Bibliography

W3schools.com DSA Greedy Algorithms (2024) Available at: https://www.w3schools.com/dsa/dsa_ref_greedy.php (Accessed: 26 November 2024).

W3Schools DSA Dijkstra's Algorithm. (2024) Available at: https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php (Accessed: 26 November 2024).

GeeksforGeeks What is Dijkstra’s algorithm (2024) Available at: https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/ (Accessed: 26 November 2024).

Michael Sambol (2014) Dijkstra's algorithm in 3 minutes. 16th September 2014. Available at: https://www.youtube.com/watch?v=_lHSawdgXpI (Accessed: 26 November 2024).

Thomas Grossmann (Overall), H.F. (Specials) (2016) Dijkstra Algorithm: Short terms and Pseudocode, GITTA - Geographic Information Technology Training Alliance. Available at: http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html (Accessed: 26 November 2024).