INDEX BASED A-STAR ALGORITHM

INDEX

INTRODUCTION

SHORTEST PATH PROBLEM

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

VARIATIONS OF SHORTEST PATH PROBLEMS

- **Single-Pair Shortest Path:** It is a shortest path problem where the shortest path between a given pair of vertices is computed. A* Search Algorithm is a famous algorithm used for solving single-pair shortest path problem.

- **Single-Source Shortest Path:** It is a shortest path problem where the shortest path from a given source vertex to all other remaining vertices is computed. Dijkstra's Algorithm and Bellman Ford Algorithm are the famous algorithms used for solving single-source shortest path problem.

- **Single-Destination Shortest Path:** It is a shortest path problem where the shortest path from all the vertices to a single destination vertex is computed. By reversing the direction of each edge in the graph, this problem reduces to single-source shortest path problem. Dijkstra's Algorithm is a famous algorithm adapted for solving single-destination shortest path problem.

- **All Pairs Shortest Path:** It is a shortest path problem where the shortest path between every pair of vertices is computed. Floyd-Warshall Algorithm and Johnson's Algorithm are the famous algorithms used for solving all pairs shortest path problem.

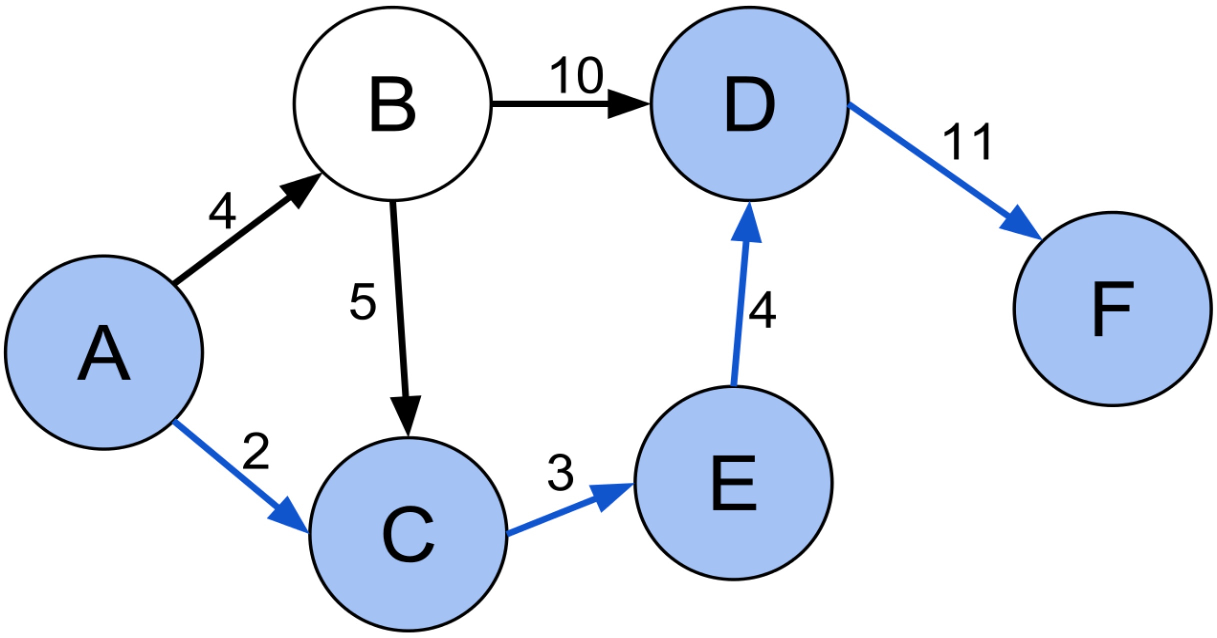


Figure 1: Single Pair Shortest Path in Weighted Directed Acyclic Graph

INDEX BASED A-STAR

The **A-Star Algorithm** is an efficient classical algorithm for solving the shortest path problem. The efficiency of the algorithm depends on the evaluation function, which is used to estimate the heuristic value of the shortest path from the current vertex to the target. When the vertex coordinates are known, the heuristic value of the shortest path is usually generated by the distance. **Index Based A-Star algorithm (IBAS)** aims to solve the shortest path problem in a weighted directed acyclic graph with unknown vertex coordinates.

This algorithm constructs three indexes for each vertex, i.e., the earliest arrival index, reverse earliest arrival index, and latest arrival index. The IBAS algorithm not only makes use of the earliest arrival index to construct the evaluation function of the A-star algorithm but also utilizes the three indexes to prune useless vertices, so as to improve the performance of the algorithm.

BACKGROUND KNOWLEDGE

DIRECTED ACYCLIC GRAPH (DAG)

In mathematics, particularly graph theory, a **Directed Acyclic Graph (DAG)** is a directed graph with no directed cycles. That is, it consists of vertices and edges (also called arcs), with each edge directed from one vertex to another, such that following those directions will never form a closed loop. A directed graph is a DAG if and only if it can be topologically ordered, by arranging the vertices as a linear ordering that is consistent with all edge directions.

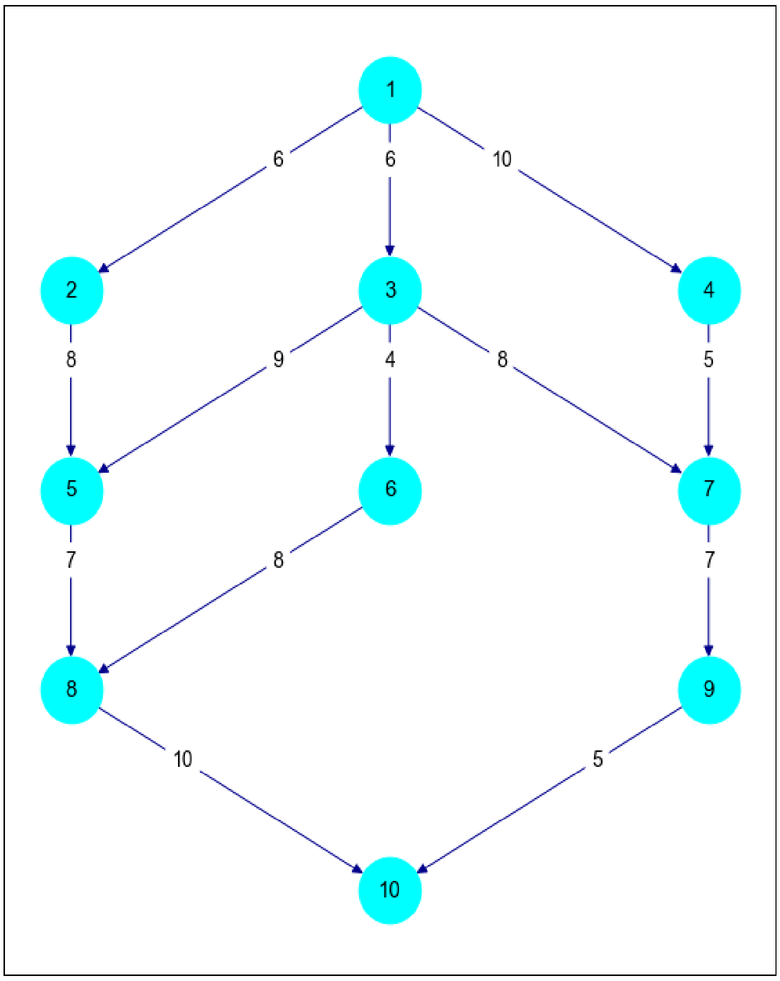


Figure 2: An Example of Directed Acyclic Graph

APPLICATIONS OF DAG

- Scheduling Tasks
- Data Processing Network
- Version History
- Citation Graphs
- Data Compression

A-STAR ALGORITHM

A* is a graph traversal and path search algorithm, which is often used in many fields of computer science It is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

At each iteration, A* determines which of its paths to extend based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. **Specifically, A* selects the path that minimizes**

$$f(n) = g(n) + h(n)$$

where, n is the next node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal**.**

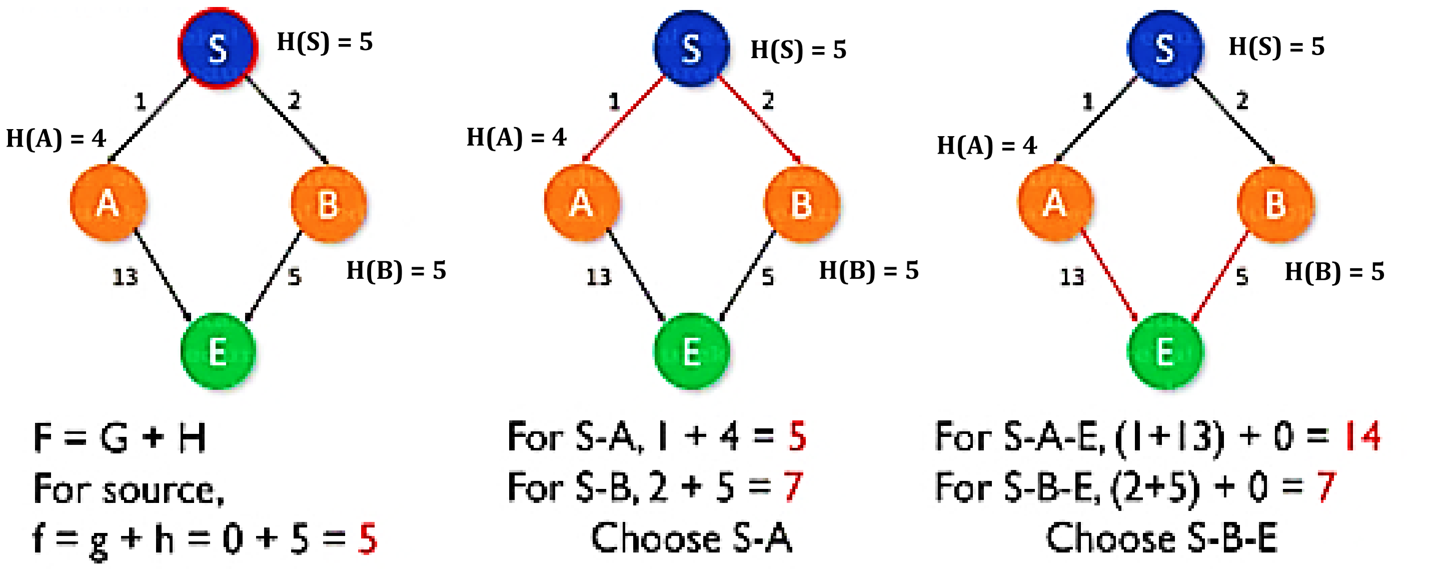


Figure 3: An Example of A* algorithm

The selection of h(n), which needs to be estimated in advance, significantly influences the efficiency of the A-star algorithm. Although there exist a variety of computing methods, such as the Manhattan distance, diagonal distance, and Euclidean distance, these are all established with known vertex coordinates. In the case with unknown vertex coordinates, the problem of determining h(n) requires urgent attention. In addition, in the process of computing the shortest path, it traverses some useless vertices that can significantly reduce the solving speed of the shortest path problem.

A* is often used for the common path-finding problem in applications such as video games. It finds applications in the problem of parsing using stochastic grammars in NLP. Other cases include an Informational search with online learning.

DIJKSTRA'S ALGORITHM

Dijkstra's Algorithm is another algorithm for finding the shortest paths between nodes in a graph, which may represent a road network. The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road (for simplicity, ignore red lights, stop signs, toll roads and other obstructions), Dijkstra's algorithm can be used to find the shortest route between one city and all other cities.

It is important to note that Dijkstra's algorithm is only applicable when all weights are positive because, during the execution, the weights of the edges are added to find the shortest path. Therefore, if any of the weights are introduced to be negative on the edges of the graph, the algorithm would never work properly.

A widely used application of shortest path algorithms is network routing protocols, most notably IS-IS (Intermediate System to Intermediate System) and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnson's. In some fields, artificial intelligence in particular, Dijkstra's algorithm or a variant of it is known as uniform cost search and formulated as an instance of the more general idea of best-first search.

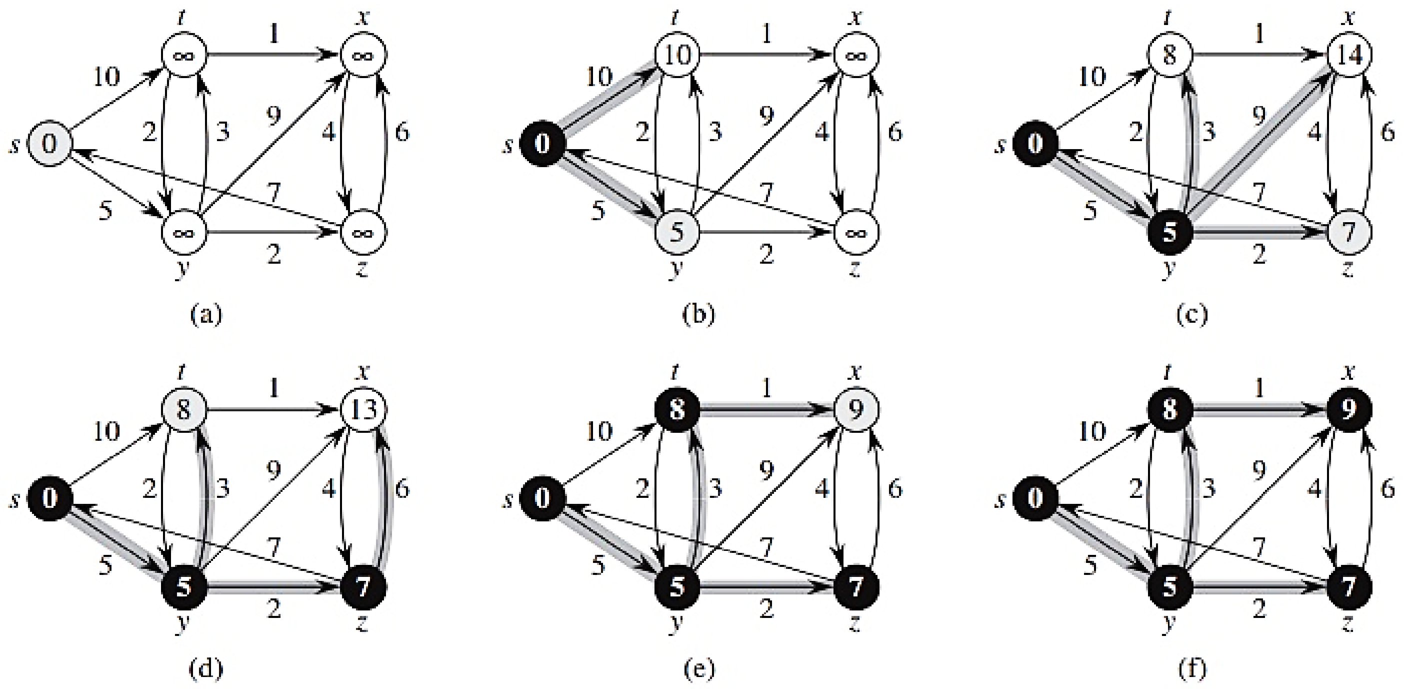


Figure 4: An Example of Dijkstra's algorithm

APPROACH AND IMPLEMENTATION

Index Based A-Star Algorithm aims to solve the shortest path problem in a weighted directed acyclic graph with unknown vertex coordinates. It mainly focuses on the following steps:

- Construction of three indexes namely **Earliest Arrival Index (*E*)**, **Reverse Earliest Arrival Index (*R*)** and **Latest Arrival Index (*L*)** in a weighted DAG.
- Construction of **heuristic evaluation function** of A* algorithm using Earliest Arrival Index and **pruning of vertices** based on all three indexes constructed in step 1.

Following sections explain the above steps in detail with the help of a DAG given below:

The sample DAG contains 23 vertices and 27 edges.

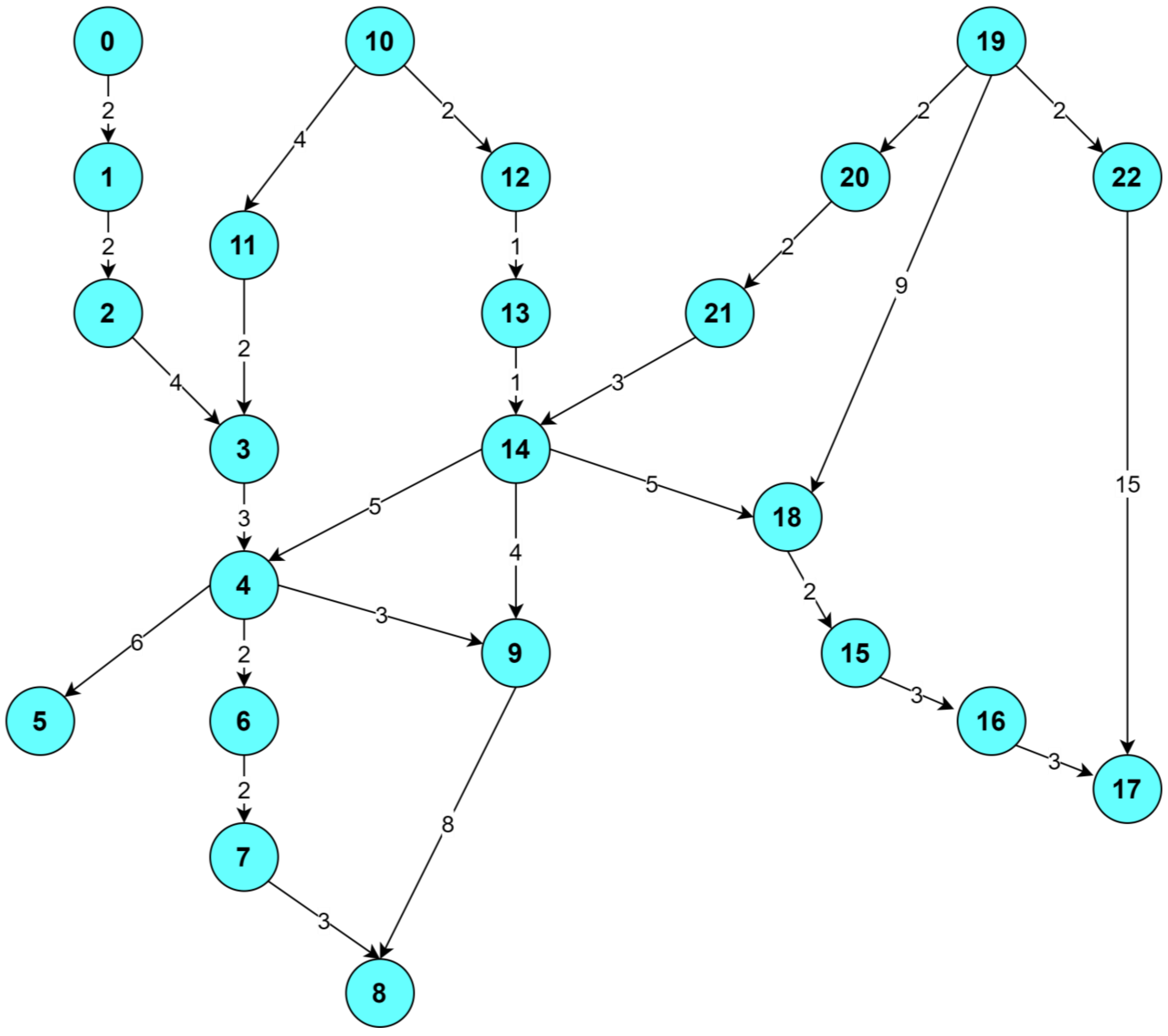


Figure 5: Sample DAG

CONSTRUCTION OF THREE INDEXES

Suppose G = (V, E, W) is a DAG, where V is a set of vertices, E $\subseteq$ V $*$ V is a set of arcs, and W is a set of arc weights. The three indexes for each vertex u ∈ V are calculated as follows:

- **Earliest Arrival Index**

$$E(u) = \begin{cases} 0 & \textit{the in}-\textit{degree of } u \textit{ is } 0 \\ \min\{E(t) + c\} & \textit{otherwise} \end{cases}$$

- **Reverse Earliest Arrival Index**

$$R(u) = \begin{cases} 0 & \textit{the out}-\textit{degree of } u \textit{ is } 0 \\ \min\{R(t) + c\} & \textit{otherwise} \end{cases}$$

- **Latest Arrival Index**

$$L(u) = \begin{cases} 0 & \textit{the in}-\textit{degree of } u \textit{ is } 0 \\ \max\{L(t) + c\} & \textit{otherwise} \end{cases}$$

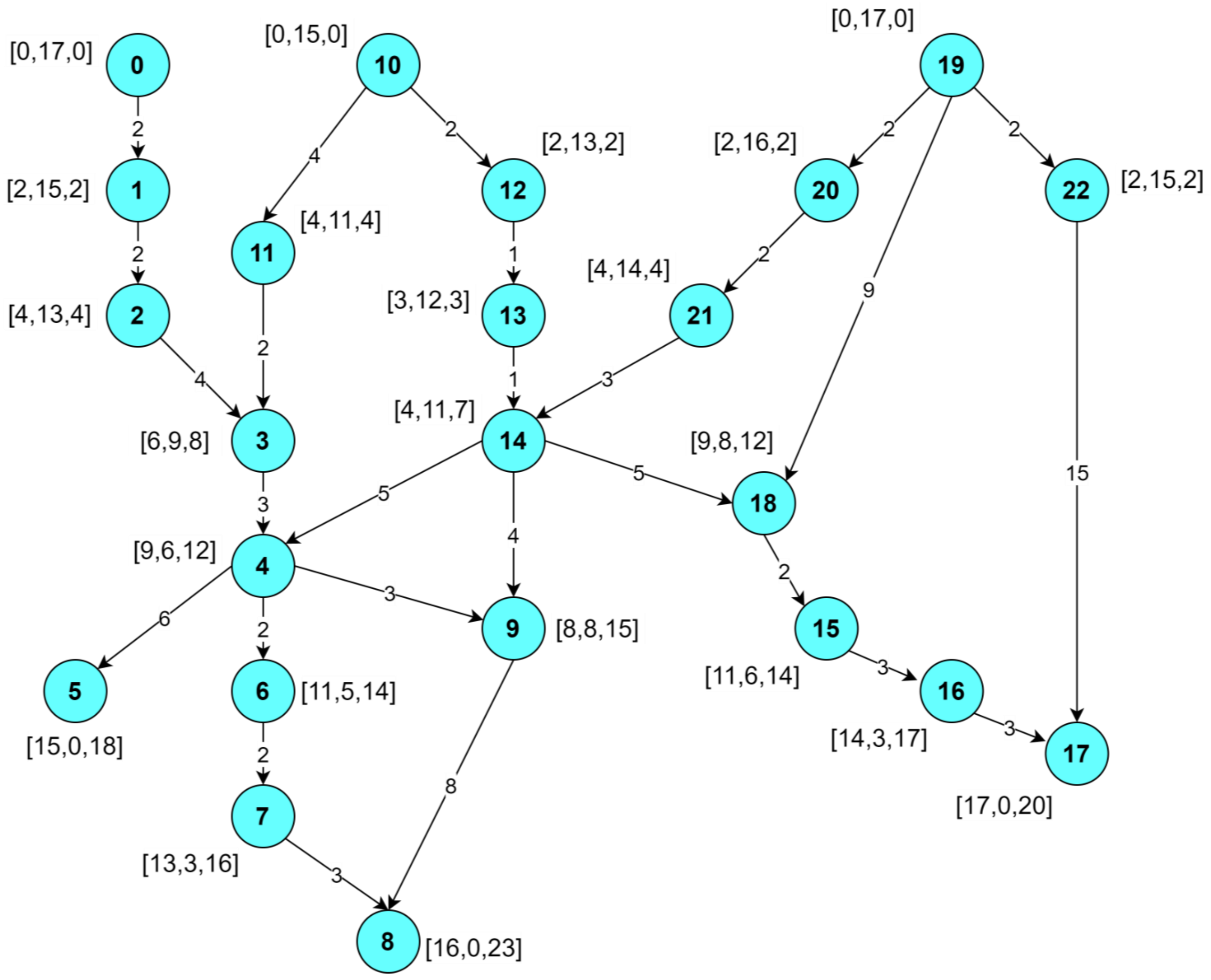
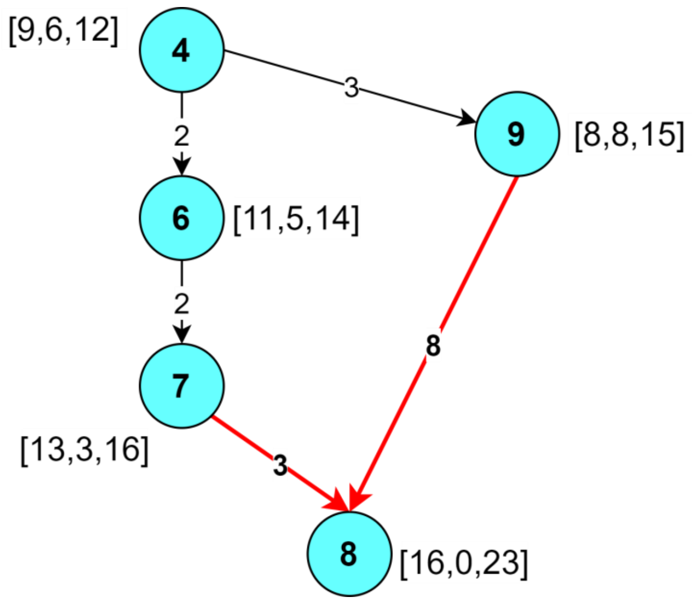


Figure 6: Calculation of three indexes for sample DAG

CALCULATION OF THREE INDEXES

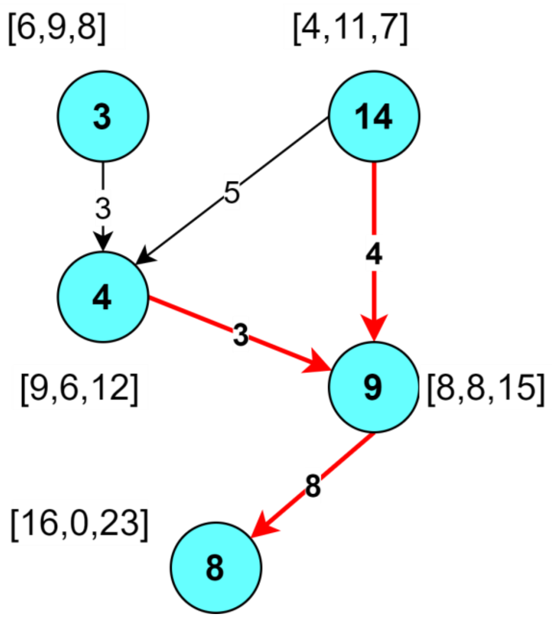
A. Vertex 8

- **Earliest Arrival Index (E):**
  = Min { E(7) + 3 , E(9) + 8 }
  = Min { 13 + 3 , 8 + 8 }
  = Min { 16 , 16 } = 16
- **Reverse Earliest Arrival Index (R):**
  = 0
  Since, the out-degree of Vertex 8 is 0
- **Latest Arrival Index (E):**
  = Max { L(7) + 3 , L(9) + 8 }
  = Max { 16 + 3 , 15 + 8 }
  = Max { 19, 23 } = 23



B. Vertex 9

- **Earliest Arrival Index (E):**
  = Min { E(4) + 3 , E(14) + 4 }
  = Min { 9 + 3 , 4 + 4 }
  = Min { 12 , 8 } = 8
- **Reverse Earliest Arrival Index (R):**
  = Min { R(8) + 8 }
  = Min { 0 + 8 }
  = Min { 8 } = 8
- **Latest Arrival Index (E):**
  = Max { L(4) + 3 , L(14) + 4 }
  = Max { 12 + 3 , 7 + 4 }
  = Max { 15 , 11 } = 15



PRUNING OF VERTICES

Pruning of vertices in a DAG refers to the reduction in number of vertices considered in the evaluation of single pair shortest path. The three indexes are used in order to achieve efficient pruning in DAG. Pruning of vertices can be achieved in two manners:

- **Static Pruning**: It refers to the pruning done before the execution of shortest path finding algorithm between a particular source (**u**)-destination (**v**) pair. In static pruning, a vertex **t** is pruned if one of the following conditions are met:
  - E(t) = 0 and t != u
  - R(t) = 0 and t != v
  - E(t) – E(u) > d or E(v) – E(t) > d or R(u) – R(t) > d or R(t) – R(v) > d
    where d = L(v) – L(u).

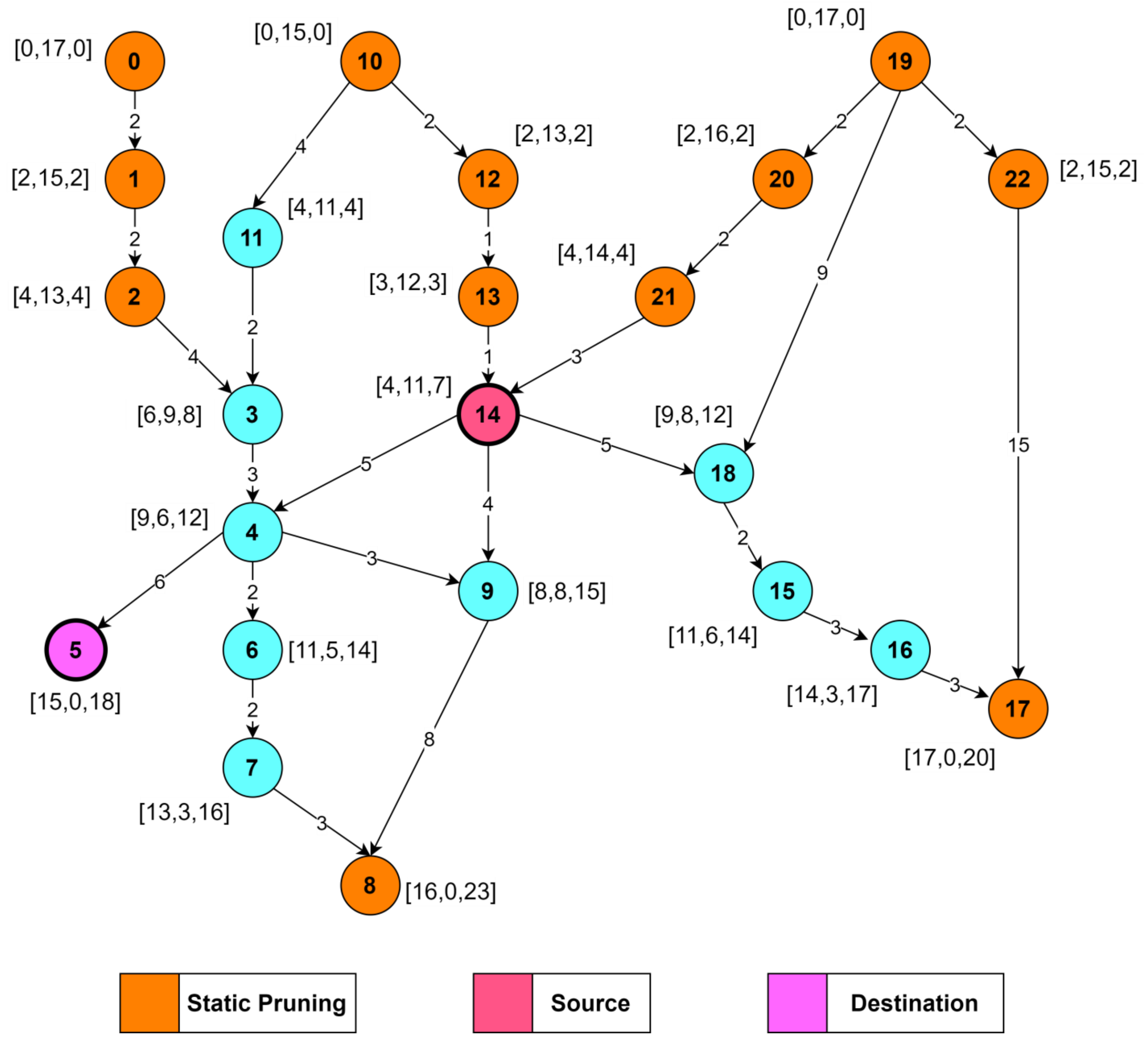


Figure 7: Static Pruning on sample DAG

STATIC PRUNING ON SAMPLE DAG

Consider Vertex 14 as Source and Vertex 5 as Destination.

L(v) – L(u) = d = 11

A. Vertex 12

- E(12) = 2
- R(12) = 13
- E(12) – E(14) = -2 , E(5) – E(12) = 13 , R(14) – R(12) = -2 , R(12) – R(5) = 13
  Since, E(5) – E(12) > d , Vertex 12 is pruned for given source-destination pair.

B. Vertex 19

- Since, E(19) = 0 and 19 != 14 , Vertex 19 is pruned for given source-destination pair.

Total 12 vertices pruned in Static Pruning: 0, 1, 2, 8, 10, 12, 13, 17, 19, 20, 21, 22

Index Based A-Star with Static Pruning (IBAS-S)

IBAS-S can be used a performance driven variant of IBAS algorithm. Since it involves static pruning only, which can be achieved using pre-computation and does not involve dynamic pruning, it can achieve faster results as compared to IBAS algorithm in scenarios where dynamic pruning is not dominant.

- **Dynamic Pruning**: It refers to the pruning done during execution of shortest path finding algorithm between a particular source (**u**)-destination (**v**) pair. In dynamic pruning, a vertex **t** is pruned if one of the following conditions are met:
    - $dis(u, r) + w[r][t] + E(v) - E(t) > d$
    - $dis(u, r) + w[r][t] + R(t) - R(v) > d$

  where dis(u, r) is the shortest distance between source **u** and intermediate vertex **r**, w[r][t] is the weight of the edge between vertex r and t, and d = L(v) – L(u).

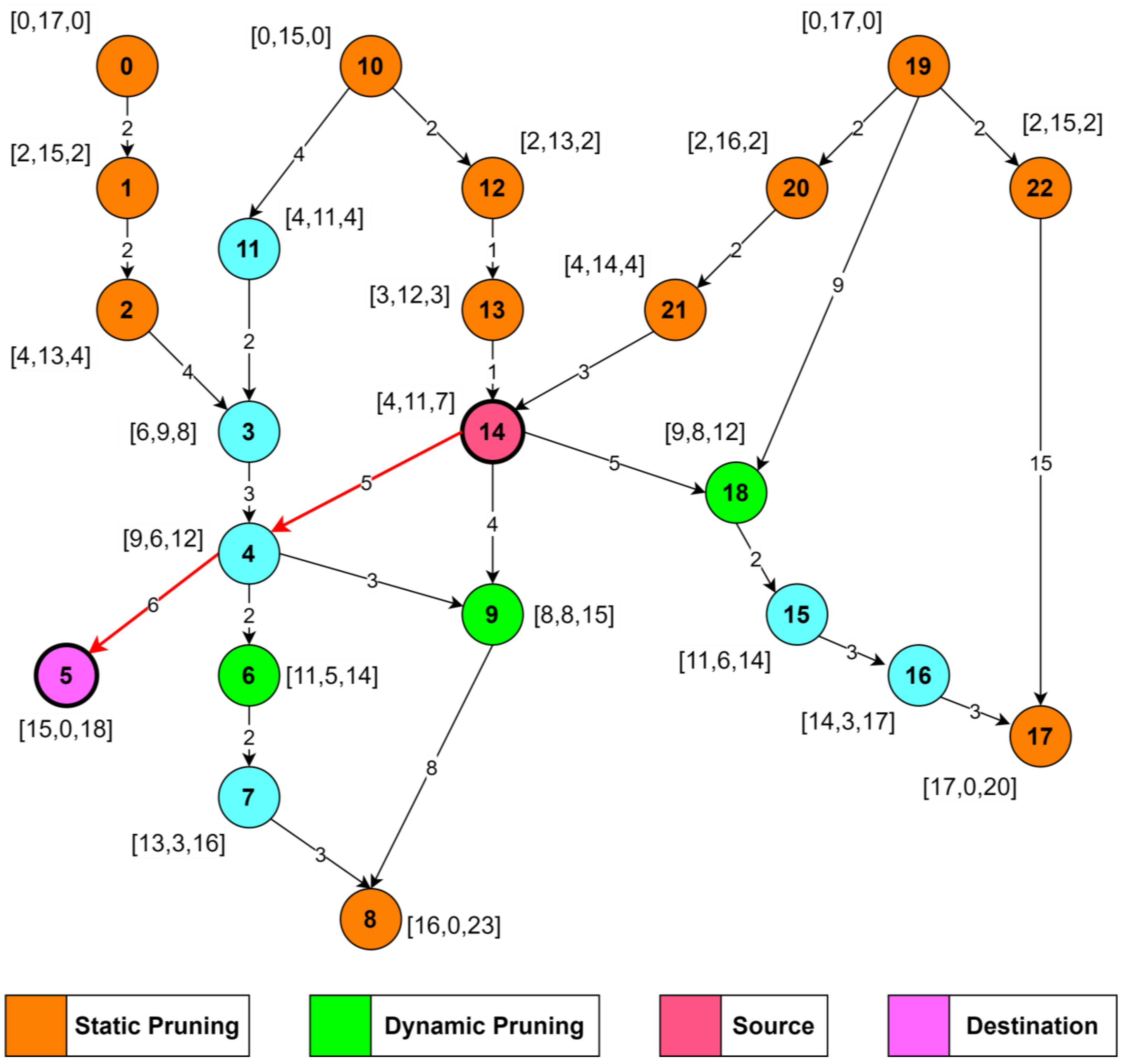


Figure 8: Static and Dynamic Pruning on sample DAG

DYNAMIC PRUNING ON SAMPLE DAG

Consider Vertex 14 as Source and Vertex 5 as Destination.

L(v) – L(u) = d = 11

A. Vertex 6

Consider r = 4

- dis(14, 4) + w[4][6] + E(5) – E(6) = 5 + 2 + (15 – 11) = 11
- dis(14, 4) + w[4][6] + R(6) – R(5) = 5 + 2 + (5 – 0) = 12

Since, dis(14, 4) + w[4][6] + R(6) – R(5) > d , Vertex 6 is pruned for given source-destination pair.

B. Vertex 9

Consider r = 14

- dis(14, 14) + w[14][9] + E(5) – E(9) = 0 + 4 + (15 – 8) = 11
- dis(14, 14) + w[14][9] + R(9) – R(5) = 0 + 4 + (8 – 0) = 12

Since, dis(14, 14) + w[14][9] + R(9) – R(5) > d , Vertex 9 is pruned for given source-destination pair.

Total 3 vertices pruned in Dynamic Pruning: 6, 9, 18

Figure 8 shows the static and dynamic pruning on the sample DAG with 14 as source vertex and 5 as destination vertex. Fifteen vertices have been pruned during static and dynamic pruning. The effect of pruning is observed as number of iterations to find the shortest path from vertex 14 to vertex 5 reduces from 13 to 6.

- ***Resultant Path*:** 14 -> 4 -> 5
- ***Resultant Cost*:** 11

COMPARATIVE RESULTS

IBAS performs well for the example taken in previous sections. It is able to reduce the number of iterations required to find the shortest path between source-destination pair. In order to further establish the effectiveness of IBAS over other shortest path finding algorithms, a comparative study has been done. **For comparison purposes, two well-known algorithms have been used which are:**

- **A-Star Algorithm**
- **Dijkstra's Algorithm**

IBAS-S is also included in the comparative study so as to analyse the performance-complexity trade off. **The basis of comparison of all four algorithms is taken as the number of iterations required by each algorithm in order to find the single pair shortest path.**

Following are the sample DAGs used for comparative purpose:

A.

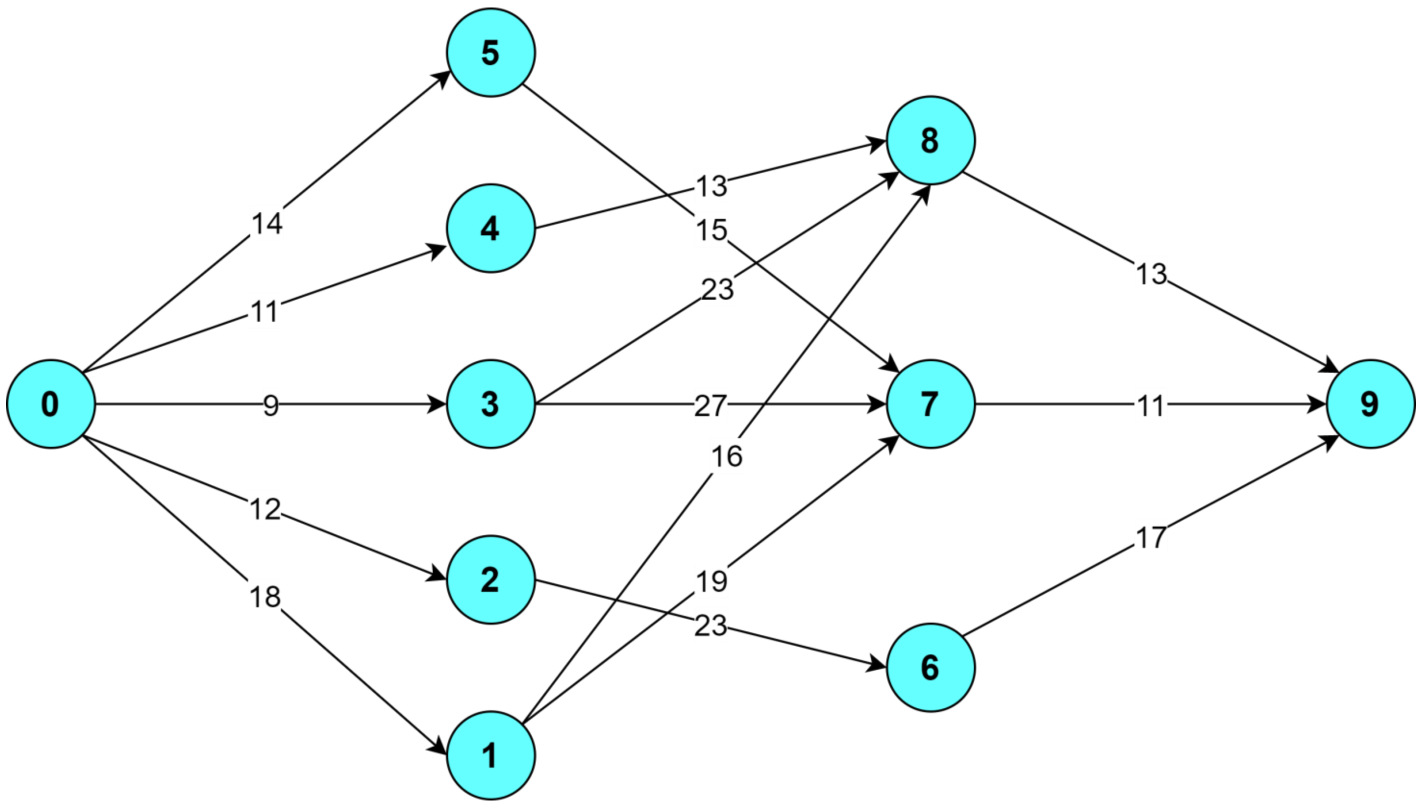


Figure 9: DAG with 10 vertices

The given DAG contains 10 vertices and 15 edges. Figure 8 shows the structure of DAG. Figure 9 shows the total number of iterations required by each algorithm to find all single pair shortest paths that exists within the DAG.

IBAS requires least number of iterations i.e. 157 iterations and thus out-performs all other algorithms. IBAS-S slots in nicely between IBAS and A-Star algorithm with 195 iterations and represents the effect of static pruning. A-Star performs worse than IBAS due to absence of pruning with 246 iterations. Dijkstra's algorithm takes the highest number of iterations i.e. 510 iterations and performs the worst out of all algorithms. IBAS is over three times faster than the Dijkstra's algorithm.

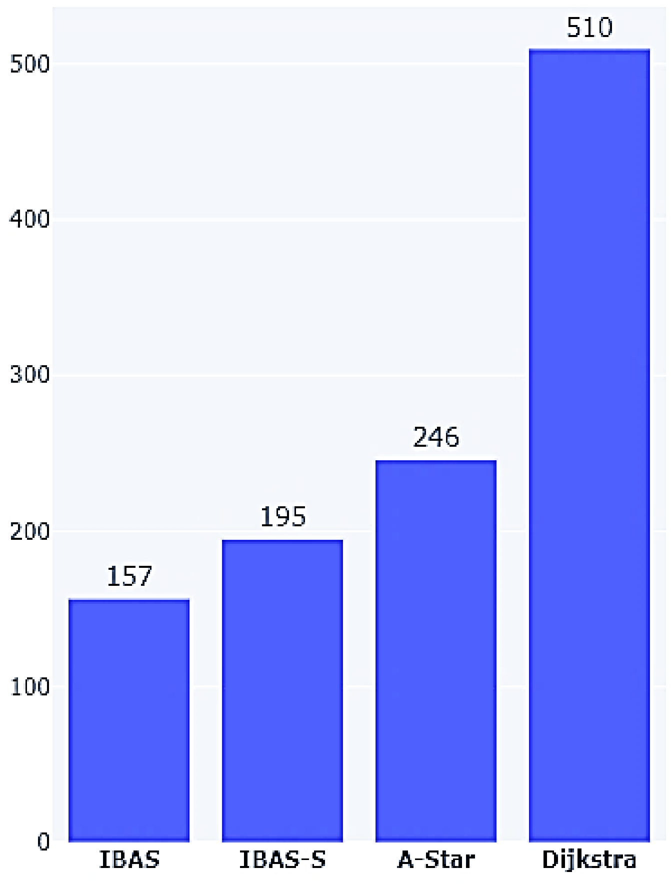


Figure 10: Comparison of Total Number of Iterations

B.

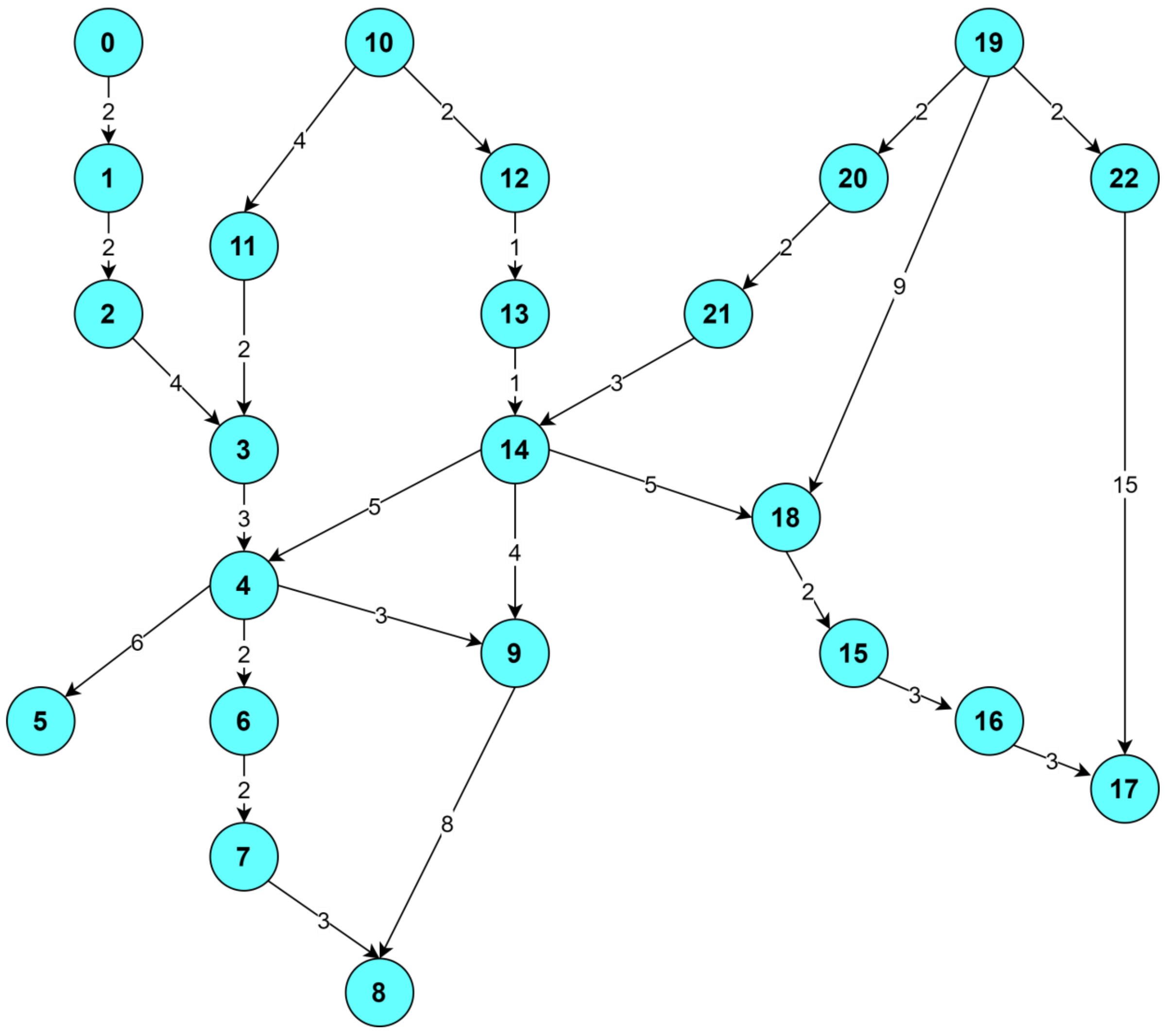


Figure 11: DAG with 23 vertices

The given DAG contains 23 vertices and 27 edges. Figure 10 shows the structure of DAG. Figure 11 shows the total number of iterations required by each algorithm to find all single pair shortest paths that exists within the DAG.

IBAS requires only 1438 iterations and again out-performs all other algorithms. IBAS-S performs better than A-Star algorithm with 1691 iterations. A-Star performs worse than IBAS due to absence of pruning with 2025 iterations. Dijkstra's algorithm takes the highest number of iterations i.e. 4347 iterations and performs the worst out of all algorithms. IBAS, IBAS-S and A-Star gives very comparable performance.

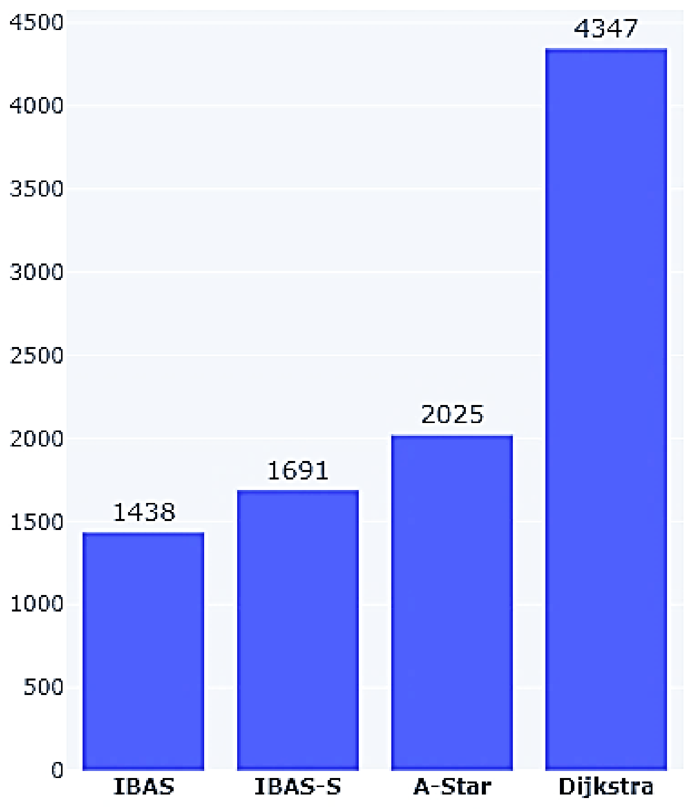


Figure 12: Comparison of Total Number of Iterations

Figure 12 shows the comparison of all four algorithms on the basis of iterations required to find all single pair shortest paths as the number of vertices in DAG grows. IBAS performs the best among all algorithms and performs four times better than Dijkstra's algorithm for large number of vertices. IBAS-S gives performance comparable to IBAS even for large number of vertices and thus can be useful in scenarios where time complexity is of major concern. A-Star performs much better than Dijkstra's algorithm but requires twice the number of iterations as that of IBAS for large number of vertices.

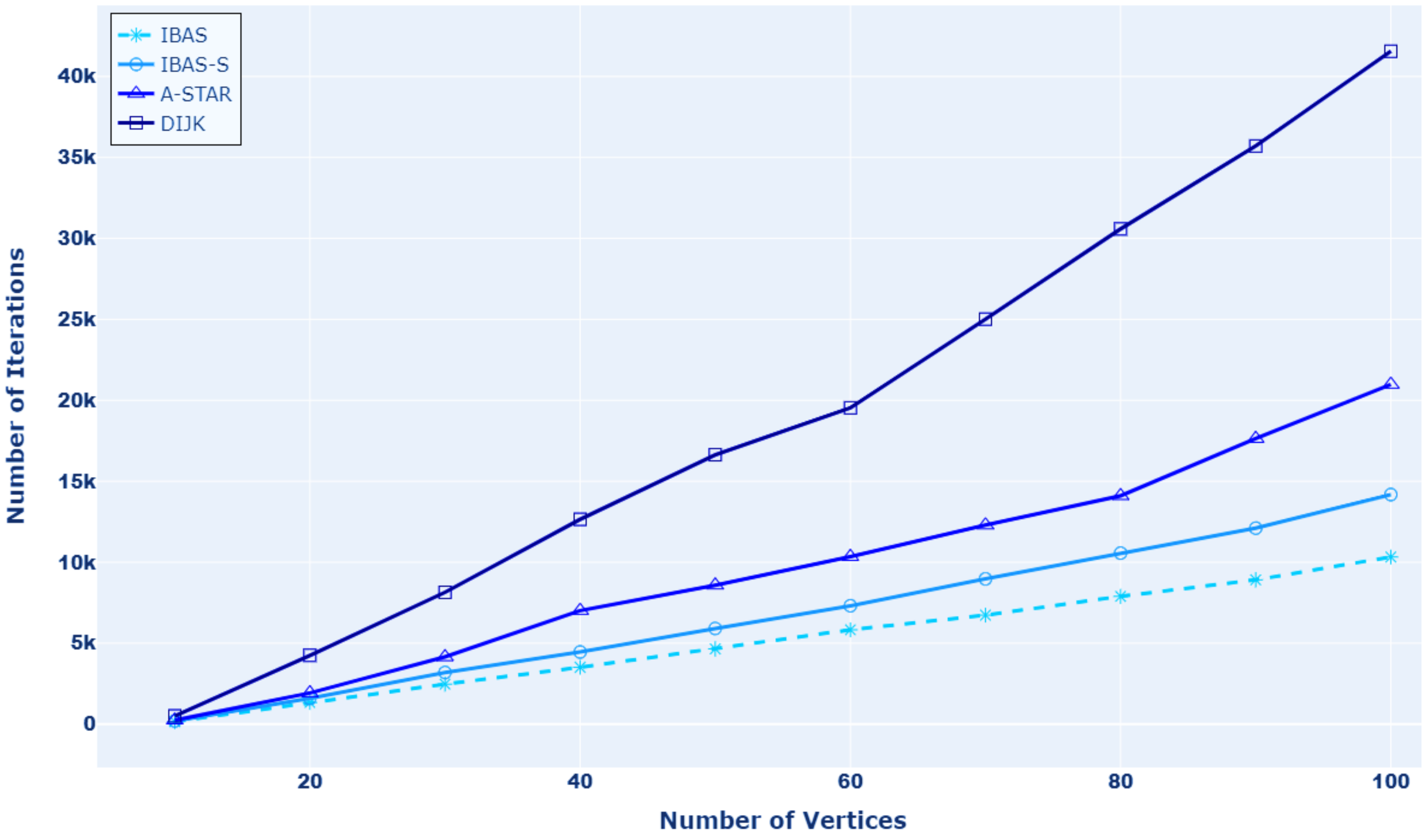
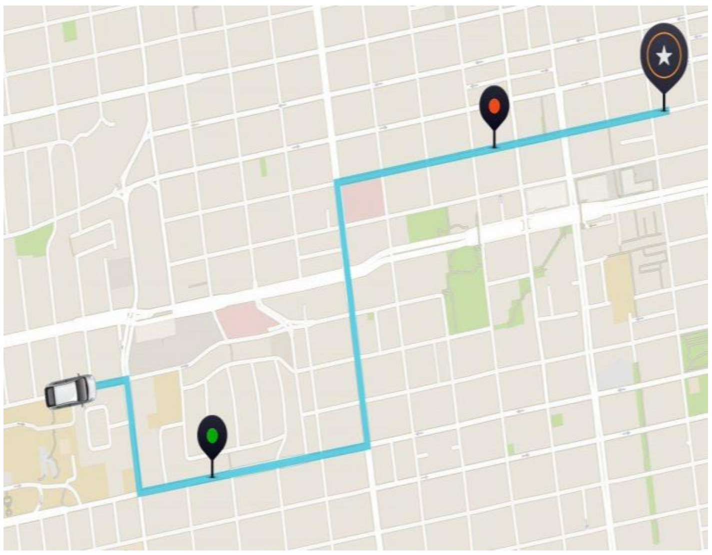


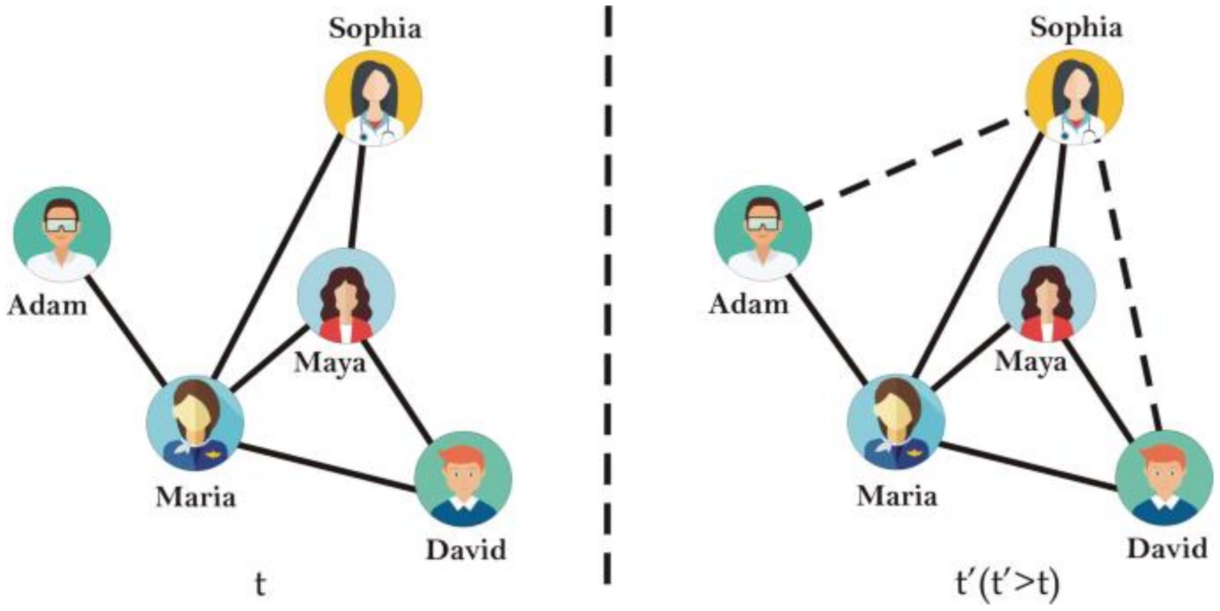
Figure 13: Number of Vertices v/s Number of Iterations

APPLICATIONS

1. **Digital Mapping Services in Google Maps:** Shortest Path Algorithms are required to find the distance in Google Maps, from one city to another, or from a person's location to the nearest desired location. There can be various routes/paths connecting them but the minimum distance is desired and thus, Index based A-Star (IBAS) algorithm can be used to find the minimum distance between two locations along the path.



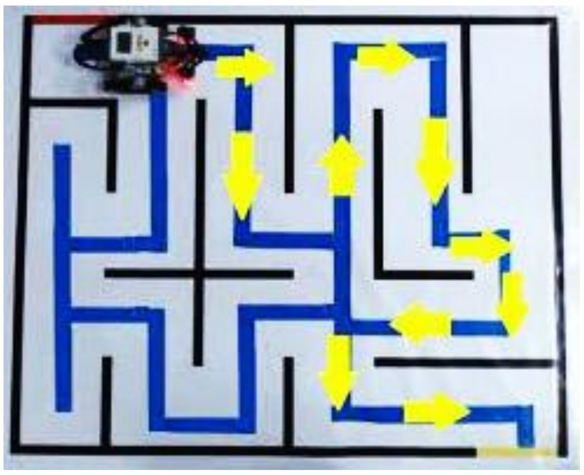
2. **Social Networking Applications:** In many applications, suggestions of friends are given that a particular user may know. Shortest path algorithms help social media companies implement this feature efficiently, especially when the system has over a billion users. When the social networking graph is very small, it uses standard Dijkstra's algorithm to find the shortest paths, but however, when the graph becomes bigger, alternate advanced algorithms are used. For the purpose of larger social networking graphs, IBAS algorithm can be used.



3. **Telephone Network:** In a telephone network, each line has a bandwidth, 'b'. Bandwidth represents the amount of information that can be transmitted by the line. Imagine a city to be a graph, the vertices represent the switching stations, and the edges represent the transmission lines with 'b' as the weight of the edges. This problem of optimal use of bandwidth in a network can be solved efficiently using IBAS algorithm.

4. **Flighting Agenda:** Specialized softwares are required for making an agenda of flights for customers. The agent has access to a database with all airports and flights. Besides the flight number, origin airport, and destination, the flights have departure and arrival time. Specifically, the agent wants to determine the earliest arrival time for the destination given an origin airport and start time. IBAS algorithm comes into use for this specific purpose.

5. **Robotic Path:** Nowadays, drones and robots have come into existence, some of which are automated. They are used to deliver the packages to a specific location. They are loaded with algorithm module which when provided with source and destination, they move in the ordered direction by following the shortest path to deliver the package in a minimum amount of time. The algorithm module can take advantage of IBAS algorithm for this task.



CONCLUSION

In this project, we have studied **Shortest Path Problem** and various algorithms used to tackle this problem. Firstly, we have implemented a new Shortest Path Algorithm namely: **Index Based A-Star** and then compared its performance with two well-known algorithms:

- **A-Star Algorithm**
- **Dijkstra's Algorithm**

IBAS-S is also included in the comparative study so as to analyse the performance-complexity trade off. IBAS-S acts as lite version of original IBAS algorithm and thus can be used when time complexity is of major concern.

Index Based A-Star algorithm aims to solve the shortest path problem in a weighted directed acyclic graph with unknown vertex coordinates. It mainly focuses on the following steps:

- Construction of three indexes namely Earliest Arrival Index (E), Reverse Earliest Arrival Index (R) and Latest Arrival Index (L) in a weighted DAG.
- Construction of heuristic evaluation function of A* algorithm using Earliest Arrival Index and pruning of vertices based on all three indexes constructed in step 1.

Comparative results shows that IBAS performs the best among four algorithms and requires least number of iterations to find all single pair shortest paths. As the number of vertices in DAG grows, the performance difference between the algorithms become more and more apparent.

LEARNING

In this project, we studied an interesting branch of **Graph Theory** known as **Shortest Path Problem**. By analyzing the algorithms used to solve shortest path problems, we came to know how simple concepts could be used and implemented for solving complex real-life problems.

During the course of this project, we used various Python3 libraries such as **NumPy, Matplotlib** and **Plotly** etc. We implemented Index Based A-Star Algorithm and along with A-Star and Dijkstra's Algorithms for comparison purposes.

Generally, this project helped us to learn how to break a natural and complex phenomenon into simpler algorithms that can be understood by people from different domains. We also learnt how to write modular codes, which in turn makes debugging easier and makes the presentation more sophisticated.

REFERENCES

- https://ieeexplore.ieee.org/document/8299426
- https://hal.archives-ouvertes.fr/hal-02084671/document
- https://www.geeksforgeeks.org/a-search-algorithm/
- https://en.wikipedia.org/wiki/Shortest_path_problem
- https://en.wikipedia.org/wiki/Directed_acyclic_graph
- https://rdrr.io/cran/pcalg/man/randomDAG.html
- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- https://brilliant.org/wiki/dijkstras-short-path-finder/