# Theoretical Time Complexity of Dijkstra's Algorithm Variants: A Review

Khalid Nooruddin Charan[1]*, Shahid Munir Shah[2], Lachhman Das Dhomeja[3], Shahzad Ahmed Memon[4], Nisar Ahmed Memon[5], Mahmood Aljawarneh[6]

***Abstract*— Complex networks formed by people, cities, computers, genes, and financial transactions, etc. as nodes and their interdependent associations as links exist in real world. Often, multiple links connect two nodes in these networks. It is essential in many applications to discover shortest path between the nodes. Shortest path problems are computation resources intensive particularly the computational time, therefore, the shortest path algorithms are analyzed to predict their computational time complexity. Plethora of literature presents comparative studies of shortest path algorithms; however, the compared algorithms provide the solutions to dissimilar settings of input network graphs. State-of-the-art Dijkstra's algorithm efficiently solves single source shortest path problem for non-negative weighted directed graphs. The algorithm selects a node with minimum distance from start node and traverse input graph. Different techniques used to implement queue resulted in different time complexity of variants of Dijkstra's algorithm. The scope of our work is to analyze time complexity of the classical Dijkstra's algorithm variants and highlight the bottlenecks that set computational upper bound of their time complexities. The outcome of this article is to recap time complexity improvements of the Dijkstra's algorithm introduced by various advancements in the data structures. This may assist the research to provide some more efficient solution for single source shortest path problems.**

***Keywords*— Graph theory; shortest path; time complexity; heap data structures; Dijkstra's algorithm.**

## INTRODUCTION

Complex networks of people, cities, airports, electronic device, genes, file servers and financial transactions etc. are very common in real life. These networks are formed by a set of nodes (independent entities) based upon any of their association/interdependency. Often, multiple paths connect one entity to another in the networks; therefore, in various applications there arises a genuine need to ascertain shortest path between entities of a network. A shortest path is a path with a minimum total cost among all the existing alternate paths.

Computation of the shortest path, a scientific activity, depends on mathematical modeling of real-life networks [1]. Graph is an abstract mathematical model to express complex network models in a concise pictorial language with clarity and precision [2].

[1-2]Hamdard University, Karachi
[3-5] University of Sindh, Jamshoro
[4]School of Architecture, Computing and Engineering, University of East London, UK
[6]Applied Science Private University Amman, Jordan
Country: Pakistan, United Kingdom, Jordan
Email: * raniya844@yahoo.com

Graphs contain set of vertices, and set of edges joining the vertices; therefore can be transformed in graphs by referring to objects as vertices and associations/interdependencies as edges, respectively, a sample network of airports is shown in Figure 1.

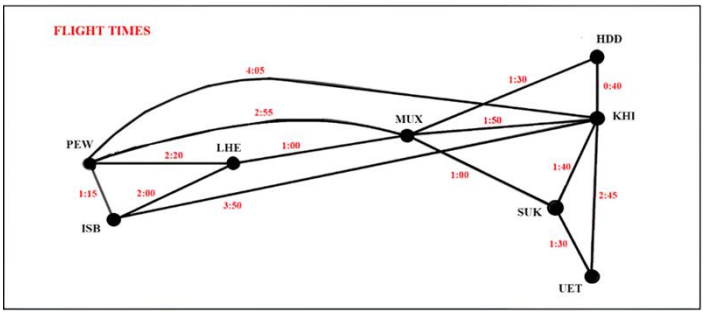


**Figure 1. A network of airports (Nodes) as vertices and Fight Time (association) as edges.**

The Shortest path algorithms work on input graphs to identify an optimal route with a minimum cost from a given starting vertex to target vertex, depending on constraints posed by the problem being studied [3]. Different approaches are implemented to find the minimum distance (cost) path and each approach generates different performance aspects. Shortest path algorithms are broadly categorized in two groups i.e. Single Source Shortest Path (SSSP) algorithms and All-Pairs Shortest Path (APSP) algorithms. The SSSP algorithms discover shortest path from a start vertex to all other vertices, and the APSP algorithms discover shortest paths among all connected vertices available in a graph [4].

Shortest path algorithms are required in various fields including social sciences, biology, chemistry, neuroscience, software engineering, Artificial intelligence (AI), security, logistics & planning, transportation planning, finance, networking, document formatting, compilers, and dataflow analysis. They appear as sub-problems of several other combinatorial optimization problems as well [5, 6, 7].

Shortest path problems are computation resources such as memory, bandwidth, or hardware but most often the computational time intensive. Literature reviews shortest path algorithms from different aspects about the greediness for such resources. The scope of our work is to review theoretical time complexity of the variants of classical Dijkstra's algorithm for SSSP problems of non-negative, weighted directed graphs. This article presents time complexity analysis of the variants of Dijkstra's algorithm. The time complexity variations in Dijkstra's variants is due to the use of different data structures. Our work highlights the critical reasons causing the variation in time complexity achievements of the different variants of the algorithm. The methodology applied to accomplish the task within its scope is mentioned in Section 2. This paper provides related work in section 3, and Section 4 provides theoretical analysis of the time complexity of Dijkstra's algorithm variants. Finally, the work is concluded in section 5.

## RELATED WORK

Some related work available in contemporary literature is stated in this section.

Kumawat et al. [16] discussed the time complexity of various shortest path algorithms. The researcher provided time complexity comparison of different solutions for shortest path problem available in the literature with concluding remarks that shortest path problem algorithms' time complexity needed to be reduced in order to enhance their utilization.

The time complexity of some and classical shortest path algorithms is compared in [17]. The work provides classic Dijkstra, Bellman-Ford, and Johnson algorithms and heuristic Dinitz, Scaling, and Genetic algorithms' time complexity comparisons. However, the competing algorithm possess different characteristics.

Barkund et al. [18] present a survey on shortest path algorithms. The survey covered SSSP and APSP problems algorithms. The survey compared many attributes of the algorithms but time complexity attribute is not included in the comparison table.

Verma et al. [19] provided empirical performance comparison of four Dijkstra's algorithm variants, i.e. the simple array data structure, Fibonacci heap, Binary heap and Bi-directional implementation variants. The research concluded that Bidirectional Dijkstra's algorithm outperformed the competing variants empirically. The researchers mentioned the theoretical time complexities of simple array, the Fibonacci, and the Binary heap variants, however, did not mentioned the theoretical asymptotic time complexity for the Bi-directional variant of Dijkstra's algorithm. It may be because of the reason that the theoretical time complexity of Bi-Directional implementation will be similar to either Binary or Fibonacci heap, depending on either of the data structures used.

Qureshi et al. [20] provided theoretical time complexity of three variants of Dijkstra's algorithm i.e. simple array and the Fibonacci and Binary heap variants. The article discussed the time complexity of individual heap operations and data structures, and reviewed and discussed the techniques to understand the impact of efficient data structures on time complexity improvements in Dijkstra's algorithm. However, did not included the other variants of Dijkstra's algorithm.

## METHODOLOGY

Plethora of literature presents comparative studies of shortest path algorithms. Eneh & Arinze [8] presented comparison amongst the main shortest path algorithms, shown in Table 1. Similar type of time complexity comparisons are provided in [9, 10, 11, and 12] as well. However, it is pertinent to mention that the algorithms compared in Table 1 provide solutions to different input graph settings. Therefore, it seems inappropriate to compare the characteristics of the algorithm providing solutions to different kinds of problems. Well-known Dijkstra's algorithm [13], a labeling method, addresses SSSP problem of directed and non-negative weighted graphs, since its inception [14 and 15]. Therefore, we are comparing the time complexity feature of variants of Dijkstra's algorithm, as all its variants solve identical problems.

**Table 1. Time complexity comparison of the shortest path algorithms**

| Algorithms' Name | Weight | SSSP/APSP | Time Complexity |
|---|---|---|---|
| Dijkstra's | +(ve) | SSSP | $\mathbf{O}(E + V \log V)$ |
| Bellman-Ford | −(ve) | SSSP | $\mathbf{O}(V * E)$ |
| Floyd-Warshall | −(ve) | APSP | $\mathbf{O}(V^3)$ |
| A* | +(ve) | SSSP | Depends on ***f(h)*** |
| Johnson's | −(ve) | APSP | $\mathbf{O}(V^2 \log V + VE)$ |
| Prim's | +(ve) | SSSP /APSP | $\mathbf{O}((V + E) \log V)$ |
| Kruskal's | +(ve) | SSSP /APSP | $\mathbf{O}(E \log E)$ |

## THEORETICAL TIME COMPLEXITY ANALYSIS OF DIJKSTRA'S ALGORITHM

***Pseudo-code of Dijkstra's Algorithm***

**DIJKSTRA'S ALGORITHM**

```
DA (Graph G, Source s)
1 for each vertex u ∈ V                          } IG-A
2 u.π = Nil                                      }
3 u.d = ∞{unknown / unreached}                   }
4 s.d = 0
5 S ← ∅
6 Q ← G.V                                        } IG-B
7 while Q ≠ ∅                                    } IG-E
8 Delete-min form the Q (returns a               } IG-C
    vertex with minimum u.d)                     }
9 S = S U [u]                                    } IG-E
10 for each vertex v ∈ G.Adj[u]                  } IG-D
11 if v.d > u.d + f(v, w)  (f(v, u):             } IG-D
     a weight function for edge(u, v))           } IG-D
12 v.d = u.d + f(v, w)                           } IG-D
13 v.π = u                                       } IG-D
14 Rearrange Q                                   } IG-D
```

Primitive operations, available in pseudocode, are grouped to estimate running time growth rate of Dijkstra's algorithm [21] for theoretical analysis of its time complexity. Instructions Groups (IG) are marked as *IG*-A, *IG*-B and so on as shown Pseudocode. The *IG*-A initializes shortest-path-estimates (*u.d*) and predecessors ($u.\pi$) in O(*V*) time. *IG*-B builds minimum-priority-queue. The selection of a node with minimum values for subsequent iteration is performed at *IG*-C. Scanning of adjacent vertices for selected vertex *Decrease-key* operation is performed at *IG*-D for a total of |*E*| times. It is important to note that *IG*-C and *IG*-E operations are performed inside **while** loop at *IG*-E. Therefore worst-case cost will be computed as:

⇨ *T*(*V*) = *IG*-A + *IG*-B + *IG*-E (*IG*-C) + *IG-E* (*IG*-D)

***Time Complexity Analysis***

The breadth first search (BFS) technique best suits for computing the shortest path in non-weighted or identical weighted edges input graphs. Whereas, a sorted list (priority queue) is essential in arbitrary weighted graphs to find the shortest path, efficiently. Binary Search Tree (BST), linked lists, arrays, and heaps data structures provide mechanism to implement priority queue. Nevertheless, heaps are considered efficient for priority queue implementations. Dijkstra's algorithm solves shortest path problem for input graphs with arbitrary edge weights, thus select a node with smallest distance at its every iteration. Efficient asymptotic time behavior of Dijkstra's algorithm correlates with arrangement of vertices and selection of a vertex with smallest distance value.

Dijkstra's algorithm variants implement priority queue using different methods for better asymptotic performance. Variations in the running time complexity of Dijkstra's algorithm depend on the data structures utilized for priority queue implementations. For example, in simple array based implementation, selection of vertices with minimum distance label will cost O(V2) operations of priority queue for |V| vertices, and scanning of adjacent vertices will cost |E| operations for |E| edges, in worst-case scenario. The total asymptotic cost of the algorithm will be O(V2 + E) [4].

The heap data structure based priority queue implementation in Dijkatra's algorithm correlates the efficient asymptotic time behavior with following heap operations:

- Insert: inserts new element in queue in an ordered sequence,
- Delete-min: retrieves minimum value element and subsequent delete,
- Decrease-key: modifies values of an element and rearrange the queue.

Every heap variant has different cost for these operations thus causes the varying asymptotic behavior of Dijkstra's algorithm variants.

***Array Indexed-by-Vertex***

Let's represent graph *G*(*V*, *E*) of set *V* of vertices and set *E* of edges, as an adjacency matrix where the element A[*i*,*j*] holds information about edge (*i*,*j*). Priority queue of vertices is implemented using array indexed by numbers from 1 to |*V*|, then updating of *v.d* for each neighbor of node *i* will take O(1) time. However, the time taken at IG-C to select a node with smallest distance from the priority queue would cost O(*V*) time within each iteration of the while loop, as there is 1 iteration for each vertex therefore there would be |*V*| number of iteration that will lead to O($V^2$) time complexity. Scanning of neighborhood at each vertex takes O(*E*) time. Thus, the total time complexity of an algorithm would be the sum of running time of each instruction group for a graph of |*V*| vertices and |*E*| edges [4, 32]:

- $T(V) = O(V) + O(V) + O(V\ O(V)) + O(E)$ (adjacency scanning may reach to *E*)
- $T(V) = O(V^2 + E)$
- O($V^2$) according to the order of the growth [4].

## *Binary heap*

The graph G, defined above, is represented as adjacency list and Binary heap data structure is utilized to implement priority queue. All vertices of the graph will be traversed in O(E) time. The Binary heap, such as [22], takes O(log V) for each Delete-min and Decrease-key operations on a set with |V| vertices, so, the maximum O(E log V) operation will be required for Decrease-key in worst-case [23, 24]. The overall time complexity would be:

- $T(V) = O\left(V\right) + O\left(V\ log\ V\right) + O\left(V\ O(log\ V)\right) + O\left(E\ O(log\ V)\right)$
- $T(V) = O(V\ log\ V) + O(E\ Log\ V)$ (($E\ \ log\ \ V$) for *Decrease-key* of each adjacency)
- $O(E\ log\ V)$

***Fibonacci heap***

Fredman and Tarjan [25,30], implemented priority queue with Fibonacci heap data structure. Fibonacci heap cost amortized O(1) for *Insert* and *Decrease-key* operations, however, *Delete-min* cost O(*log V*) for each operation on a heap of |*V*| vertices. The maximum size of a heap would be |*V*|-1, for a graph with *V* vertices. The Fibonacci heap will take O(*V*) for *Insert* and (*V*) for heap operations and O(*V* + *E*) time for other tasks. Thus running time of Fibonacci heap variant of Dijkstra's algorithm is reduced to [4, 25, 31]:

- T(V)=O(V)+O(V)+ O(V O(log V))+O(E) (E times O(1) for Decrease-key)
- T(V)= O(V log V)+O(E)
- O(E+V log V)

***d-ary heap***

d-ary heap is an extension of Binary heap data structure. A parent has two children in the Binary heap, whereas in d-ary a parent has *d* number of children, thus the height of *V* element tree is reduced to O($log_d$ V). d-ary heap based priority queue cost O(log *V* / $log_d$) for each *Insert* and *Decrease-key* operations and O(*d log* V / $log_d$) for *Delete-min* operation, therefore the total running time for |*V*| vertices is [26]:

- $T(V) = O(V) + O\left(\frac{V\ log\ V}{\log d}\right) + O\left(V.d\ O\left(log\ \frac{V}{\log d}\right)\right) + O\left((E)O\left(log\ \frac{V}{\log d}\right)\right)$
- $T(V) = O\left(V.d\ O\left(log\ \frac{V}{\log d}\right)\right) + O(E\left(log\ \frac{V}{\log d}\right))$ (*E* times *log V* / $log_d$ for *Decrease-key* of each adjacency)
- $O(V.d\ ) + E\left(log\ \frac{V}{\log d}\right))$

***Radix heap***

Originally, Johnson [27] proposed one-level Radix heap data structure. The heap achieved $O(E\ log\ log\ C + V\ log\ C\ log\ log\ C))$ upper bound time complexity for Dijkstra's algorithm. Ahuja et al. [28], with a slight amendment in the Johnson work, introduced a collection of buckets in radix heap and cost O(*log C*) *Insert* and *Delete-min* operations and takes O(1) cost for *Decrease-key* operation thus reduced Dijkstra's algorithm upper bound time complexity to:

- $T(V) = O(V) + O(V\ log\ C) + O(V\ O(log\ C)) + O(E\ O(1))$
- $T(V) = O(V\ log\ C) + O(E)$
- $O(E + V\ log\ C)$

***Integer priority queues***

Delete-min operation cost O(log log V) time and other operations cost O(1) in integer priority queue [29], the time complexity is achieved as:

$$\Rightarrow T(V) = O(V) + O(V) + O\big(V\, O(log\ log\ V)\big) + O(E\, O(1))$$

$$\Rightarrow T(V) = O(V\ log\ log\ V) + O(E)$$

$$\Rightarrow O(E + V\ log\ log\ V)$$

Time complexities of the variants of Dijkstra's algorithm using different data structures for priority queue implementation are appended in Table 2:

**Table 2. Time complexity comparison of Dijkstra's algorithm variants**

| Variant | Total running time complexity |
|---|---|
| Node-indexed array | $O(V^2)$ |
| Binary heap | $O(E\ log\ V)$ |
| Fibonacci heap | $O(E + V\ log\ V)$ |
| d-ary heap | $O(V.d + E\ log\ V / log_d)$ |
| Radix heap | $O(E + V\ log\ C)$ |
| Integer priority queue | $O(E + V\ log\ log\ V)$ |

## CONCLUSION

Several algorithms solve shortest path problems. These algorithms are analyzed theoretically to predict their computational time complexity. The literature presents comparative studies on time complexities of shortest path algorithms; however, the competing algorithms solve problems of input graphs with different settings. This paper presents review of well-known Dijkstra's algorithm for SSSP problems of non-negative, weighted directed graphs. Dijkstra's algorithm select a node with smallest distance at its every iteration. Efficient asymptotic time behavior of Dijkstra's algorithm correlates with arrangement of vertices and selection of a vertex with smallest distance value. Simple array based arrangement of vertices led to an upper bound time complexity of $O(V^2)$ due to selection of vertices with smallest distance values. Heap data structures based priority queues provided flexibility to researchers to linearize the time complexity of the algorithm to $O(E + V\ log\ log\ V)$ and $O(E + V\ Log\ V)$. Variants of Dijkatra's algorithm utilized different heaps for priority queue implementations. The dissimilarities in time complexity of Dijkstra's algorithm variants is because of difference in cost of *Insert*, *Delete-min* and *Decrease-key* operations of different heaps that is a critical reason causing variations in the time complexity achievements of different Dijkstra's algorithm variants. Advancement in data structures used for queuing purposes or any proposal for reductions in primitive operations may reduce running time cost of the algorithms and in turn may result in better solution of single source shortest path problem.

## REFERENCES


[1] Robert Sedgewick and Philippe Flajolet (2013). Introduction to the Analysis of Algorithms, 2nd Edition. Addison-Wesley Professional.

[2] Kairanbay M, and Mat Jani H (2013). A Review And Evaluations Of Shortest Path Algorithms. Int. J. of Sci. & Tech.Res.2 6 99.

[3] K. Erciyes (2018). Guide to Graph Algorithms - Sequential, Parallel and Distributed. © Springer International Publishing AG, part of Springer Nature 2018.

[4] Thomas H. Corman, Charles E. Leiserson, Ronald L. Rivest, Clifford stein (2009). INTRODUCTION TO ALGORITHMS 3RD Ed The MIT Press Cambridge, Massachusetts London, England.

[5] Kao, M. Y. (Ed.). (2008). *Encyclopedia of algorithms*. Springer Science & Business Media.

[6] Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., & Hwang, D. U. (2006). Complex networks: Structure and dynamics. *Physics reports*, *424*(4-5), 175-308.S.

[7] Costa, L. D. F., Oliveira Jr, O. N., Travieso, G., Rodrigues, F. A., Villas Boas, P. R., Antiqueira, L., ... & Correa Rocha, L. E. (2011). Analyzing and modeling real-world phenomena with complex networks: a survey of applications. *Advances in Physics*, *60*(3), 329-412.

[8] Eneh, A. H., & Arinze, U. C. (2017). Comparative analysis and implementation of dijkstra's shortest path algorithm for emergency response and logistic planning. Nigerian Journal of Technology, 36(3), 876-888.

[9] Susanto, W., Dennis, S., Handoko, M. B. A., & Suryaningrum, K. M. (2021, October). Compare the path finding algorithms that are applied for route searching in maps. In *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)* (Vol. 1, pp. 178-183). IEEE.

[10] Manan, A., Imran, S., & Lakyari, A. (2019). Single source shortest path algorithm Dijkstra and Bellman-Ford Algorithms: A Comparative study. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND EMERGING TECHNOLOGIES (IJCET)*, *3*(2), 25-28.

[11] Wang, X. Z. (2018, September). The comparison of three algorithms in shortest path issue. In *Journal of Physics: Conference Series* (Vol. 1087, No. 2, p. 022011). IOP Publishing.

[12] AbuSalim, S. W., Ibrahim, R., Saringat, M. Z., Jamel, S., & Wahab, J. A. (2020, September). Comparative analysis between dijkstra and bellman-ford algorithms in shortest path optimization. In *IOP Conference Series: Materials Science*

*and Engineering* (Vol. 917, No. 1, p. 012077). IOP Publishing.

[13] E. W. Dijkstra (1959). A Note on Two Problems in Connexion with Graph. Numerische Mathematlk l, 269 - 27 I.

[14] Yue, Y. (1999). An efficient implementation of shortest path algorithm based on dijkstra algorithm. Journal of Wuhan Technical University of Surveying & Mapping.

[15] Gass, S. I., & Harris, C. M. (1997). Encyclopedia of operations research and management science. *Journal of the Operational Research Society*, *48*(7), 759-760.

[16] Kumawat, S., Dudeja, C., & Kumar, P. (2021, May). An extensive review of shortest path problem solving algorithms. In *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)* (pp. 176-184). IEEE.

[17] Bamboat, M. A., Laghari, A. A., Li, H., Khan, A. A., & Qaimkhani, M. A. (2024, August). Quality of Experience Assessment of Over the Top Services. In 2024 4th International Conference on Blockchain Technology and Information Security (ICBCTIS) (pp. 145-149). IEEE.

[18] Güneri, Ö. İ., & Durmuş, B. (2020). Classical and heuristic algorithms used in solving the shortest path problem and time complexity. *Konuralp Journal of Mathematics*, *8*(2), 279-283

[19] Barkund SH, Sharma A, and Bhapkar HR (2022). Survey of Shortest Path Algorithms. JJTU Journal of Renewable Energy Exchange (Volume 10, Issue 11 (2022), pp:46-57), doi: 7, https://doi.org/10.58443/IJREX.10.11.2022.46-57.

[20] Verma, D., Messon, D., Rastogi, M., & Singh, A. (2021, February). Comparative study of various approaches of Dijkstra algorithm. In *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)* (pp. 328-336). IEEE.

[21] Qureshi, M. A. (2021). Dijkstra's Algorithm-A Case Study to Understand How Algorithms are improved. VFAST Transactions on Software Engineering, 9(3), 48-56.

[22] Laghari, A. A., Li, H., Khan, A. A., Shoulin, Y., Karim, S., & Khani, M. A. K. (2024). Internet of Things (IoT) applications security trends and challenges. Discover Internet of Things, 4(1), 36.

[23] Goodrich, M. T., & Tamassia, R. (2015). Algorithm design and applications (Vol. 363). Hoboken: Wiley.

[24] J. W. J. Williams (1964). Algorithm 232.: 'Heapsort. Commun,". Commun. ACM, vol. 7, no. 6, pp. 347–349, Jun. 1964, doi: 10.1145/512274.512284.

[25] Edelkamp and Stefan (2012). Heuristic Search || Basic Search Algorithms. doi:10.1016/b978-0-12-372512-7.00002-x.

[26] Mikkele Thorup (1999). Undirected SSSP with positive integer weights in linear time.AT & T Labs Research.

[27] Khani, M. A. K., Khan, A. A., Brohi, A. B., & Shaikh, Z. A. (2022). Designing Mobile Learning Smart Education System Architecture for Big Data Management Using Fog Computing Technology. International Journal of Imaging and Sensing Technologies and Applications (IJISTA), 1(1), 1-23.

[28] Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, *34*(3), 596-615.

[29] Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (2008). Algorithms (p. 336). New York: McGraw-Hill Higher Education.

[30] JOHNSON, D. B. (1977). Efficient special-purpose priority queues. In Proceedings of the 15th Annual Allerton Conference on Communications, Control, and Computing (pp. 1-7).

[31] Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, *34*(3), 596-615.

[32] Mikkel Thorup (2004). Integer priority queues with decrease key in constant time and the single source shortest paths problem. Journal of Computer and System Sciences 69 330–353.