Comparison of Shortest Path Finding Algorithms for Cable Networks on Industrial Construction Projects

F. Alsakka[a], S. Khalife[a], M. Nomir[a], Y. Mohamed[a], and R. Hermann[b]

[a] Department of Civil and Environmental Engineering, University of Alberta, Canada
[b] *PCL Industrial*, Canada
E-mail: falsakka@ualberta.ca, khalife@ualberta.ca, nomir@ualberta.ca, Yasser.Mohamed@ualberta.ca, rhhermann@pcl.com

Abstract –

Cable path optimization is a commonly encountered problem in industrial construction projects. Numerous designs are technically feasible but are associated with varying construction costs and effort. Hence, selecting the right network is deemed a critical decision. Multiple shortest path algorithms could be deployed to identify the optimal solution. Although they could potentially identify solutions of equal values, there could be differences among their performance which become more significant as the project size increases. Thus, this study compares the results of applying three shortest path algorithms, Dijkstra, A* and Bellman-Ford, in finding the shortest paths for power and instrument cables in an industrial facility project. Moreover, it presents an integrated methodology that combines different software to help practitioners experiment with different scenarios in a fast and systematic manner and make decisions accordingly: (1) Build a 3D model of the project under study, (2) Design a database for data retrieval and management, (3) Import data from the model into the database, (4) Develop a code that reads from the database, finds the optimal paths for the planned cables in the facility, and writes the results back into the database, and (5) Import the obtained results into NavisWorks for visualization and validation purposes. The results have shown that the three algorithms identified the same paths for six cables. Yet the optimal path found using Dijkstra and A* for the seventh cable was one node longer than that identified using Bellman-Ford, but the three paths were of equal weights. Generally, Dijkstra and A* exhibited a close performance. Meanwhile, the main difference between Dijkstra and A*, on one hand, and Bellman-Ford, on the other hand, lied in fact in the time needed to solve the optimization problem. The percent difference between Dijkstra and Bellman-Ford's runtimes for one of the cables reached 4,800% making Dijkstra superior with respect to runtime. Even though the impact of the runtime difference is considered insignificant within the scope of this study as it is in the range of milliseconds, its criticality would increase as the size of the network increases. Therefore, a proper selection of the optimization algorithm would support a rapid and efficient decision-making process.

Keywords –

Cable Network; Shortest path algorithms; Optimal path; Nodes; Edges

1 Introduction

Construction projects entail critical decisions in connection with planning, developing and executing major engineering elements, which would have a severe impact on cost and time savings if not dealt with in a responsible manner. In particular, the issue of planning and designing cable networks on industrial projects is of great concern for construction managers and owners, specifically with respect to cost implications. The main problem encountered in such networks is their complexity as deciding on the optimal solution with respect to different considerations is not a straightforward decision. Cable network optimization is, thus, one of the several optimization problems in construction that need supportive methods, such as mathematical techniques, to help in identifying the most feasible solutions [1]. Examples of similar optimization problems comprise time-cost trade-off [2], resource levelling [3], site layout optimization [4,5], planning and allocating equipment [6], among others.

Mathematical algorithms have a great potential in solving complex problems, specifically those including a large number of alternative solutions given the numerous possible permutations [2]. For cable network problems, specific algorithms are found that deal with identifying the optimal path for a given connection. Three of the most

prevalent algorithms used to determine the shortest path between two nodes are Dijkstra, A* (A-star) and Bellman-Ford [7]. Similar algorithms prove to be highly useful for various applications given their ability to account for factors other than distance when identifying the optimal path [8]. For instance, the Dijkstra and A*, along with other algorithms, were evaluated for their performance and accuracy in finding the optimal path on construction sites considering multi criteria requirements, including travel distance, safety and visibility [8]. Such studies offer a detailed comparison between the different algorithms to help specify what type of problems best fit each algorithm.

While the literature offers a plenty of studies comparing shortest path algorithms in different contexts, the construction industry literature remains in shortage for research discussing opportunities for the application of shortest path algorithms in a systematic manner specifically for cable networks. Given the potential cost and performance impact of cable network design on construction projects, this paper aims at presenting an application of the shortest path method to find optimal paths for a set of pairs of start and destination nodes for power and instrument cables in an industrial facility. The optimal path in the context of this paper represents the shortest path for connecting the cables in the facility while achieving minimum costs that are dictated by the type of trays that cables pass through. The paper achieves this by utilizing the commonly used algorithms: Dijkstra, Bellman-Ford and A* algorithms. The performance of these algorithms is also compared based on the number of connection nodes in the identified shortest paths, the total weight of the paths, and the computation duration (i.e. runtime) to recommend the most suitable algorithm for this type of problem.

The main contribution lies in presenting a convenient approach for solving a critical practical problem on industrial projects while analysing some key points of departure whenever considering similar problems on different projects.

2 Literature Review

2.1 Overview of the Selected Algorithms

Although various algorithms could yield similar results, they exhibit distinct properties such as speed, efficiency, and computation method. This section provides a brief and general overview of the three algorithms employed in this study.

2.1.1 Dijkstra Algorithm

Theoretically, Dijkstra is considered the most common algorithm for finding the shortest path in network problems having a single-source (i.e. one source node) [7]. It is a straightforward and simple algorithm which has gained popularity in network optimization field [9]. In Dijkstra, the number of nodes is specified, and the node which represents the source is just one specific vertex while the destination could include several vertices [10]. Thus, this algorithm is beneficial when the destination is unknown [11]. The function of this algorithm f(n) is equal to g(n), where g(n) is the cost of the path from the source node to the destination node, n [11].

Dijkstra's time complexity is computed by O(n2), where n is the total number of nodes [9], [12]. Hence, in case of problems that are relatively large, Dijkstra is successful as it has a time complexity of an order of n^2. However, the algorithm is generally considered to take a long time and waste resources due to the blind search it performs [11], [13]. Dijkstra adopts the "Greedy Best First Search" approach, and it takes a big search area prior to reaching the destination [11]. It works by going equally in all directions [11] and terminates after visiting all the nodes [14].

2.1.2 A* (A-Star) Algorithm

Another common algorithm for solving the shortest path problem is A*. A* comprises the summation of two functions; a function, g(n), that constitutes the exact cost of the path extending from the start node to node n, and a heuristic function, h(n), that represents an acceptable estimated cost to reach the destination node [12]. The heuristic function provides an estimate of the best path from any node to the destination node, where the order of this estimate defines how the nodes will be visited [11]. A* uses the "Best First Search" approach in which the node with the best heuristic value is chosen to be visited first [15]. The algorithm focuses only on the region in the direction of the goal [12]. However, there is no assurance that optimal solutions will be obtained with the use of heuristics [16].

The performance of A* is significantly impacted by the choice of the heuristic function. In fact, heuristics are used as guidance to improve performance, and they affect the time complexity of the algorithm [12]. If h(n) is exactly equal to the cost required to move from node n to the destination node, the algorithm will only follow the optimal path, without expanding into anything else, resulting in a high search speed. The path to the goal node can be discovered even faster if h(n) overestimates the cost required to reach the destination node from node n. Yet the cost of the identified path might not be the most favorable one in this case [12]. The time complexity is O(n log n), where n is the number of nodes [11]. A* is usually efficient when both the start node and the target node are known [11].

Overall, it can be said that the main difference between Dijkstra and A* algorithms is the heuristic

function h(n) used in A*. In fact, if the value of h(n) is set to zero, A* will give the same outcomes as Dijkstra [12]. Since A* uses the "Best First Search", it is considered faster than Dijkstra which adopts the "Greedy Best First Search" approach [11].

2.1.3 Bellman-Ford Algorithm

Similar to Dijkstra algorithm, Bellman-Ford is a traditional algorithm used for solving shortest path problems from a single-source node [17]. However, Bellman-Ford algorithm can run in a distributed manner unlike the Dijkstra algorithm which is a global algorithm and cannot run easily in a distributed manner [18]. In this sense, distributed algorithms, being subtypes of parallel algorithms, are characterized by the ability of the nodes to keep track of the shortest distances between themselves and the other nodes, while communicating with its neighbour nodes by transmitting messages over the links [19]. Thus, unlike Dijkstra, parallelization takes place in Bellman-Ford easily [20], [21]. Additionally, Bellman-Ford has a looping behaviour, where iterations occur over all edge connections in order to continuously update the nodes until the final distances are reached. However, this reduces Bellman-Ford's efficiency in comparison to Dijkstra's [20], and this is considered a major drawback of the distributed Bellman-Ford algorithm [19]. As for the runtime of Bellman-Ford algorithm, it is O(nm), where n is the number of nodes and m is the number of edges [22]. Bellman-Ford is also dominant in solving the majority of routing problems which have various constraints and occur in a flat network found in an autonomous system, where the primary objective function is the minimum node count [23].

2.2 Some Applications of Shortest Path Algorithms in Construction

Various optimization problems that require the use of shortest path algorithms are found in the construction industry. Material flow on construction sites is an example of similar problems. Optimizing material flow and reducing travel time are valuable as cost and time are impacted by the path that the materials go through [8]. Moreover, optimizing site layout to minimize travel distances can boost production rate as wastage and working time are reduced [8].

Shortest path algorithms are also used in transportation applications. Related examples include in-vehicle route guidance systems that require an immediate response upon request for information and identify vehicle routing and scheduling. Accordingly, a rapid identification of the shortest paths is needed [24]. Moreover, with the use of intelligent robots in construction, there is a need for planning their movement path on site. The focus is on finding an effective and short path that is also collision-free from the initial position to the final position, by avoiding both stationary and movable obstacles [25].

Other applications are focused on optimizing some more important aspects of construction operations while trying to minimize the travel distance. Lei et al. [26] developed a generic algorithm to manage the movement of large mobile cranes used to lift prefabricated units on site while taking into account the site constraints including the congestion of different site areas, the geometry of lifted items and the configuration of cranes. This aims at saving time, satisfying safety and efficiency requirements as well as minimizing the risks of failure and accidents' occurrence on site.

3 Problem Description and Modelling

Designing paths for power and instrument cables within a project requires identifying the shortest feasible paths in order to minimize costs accompanied with their installation. The project presented in this paper deals with this problem of finding the optimal solution for connecting five power cables and two instrument cables between pre-specified source and destination points. The project under study is an industrial facility of which the network consists of a total of 6,155 nodes. These nodes are connected via different types of trays which are modelled as edges. Edges are grouped into three different categories based on their type:

- Category 1: It accommodates the extension of power cables only and is, thus, designated as "CTP".
- Category 2: It accommodates the extension of instrument cables only and is, thus, designated as "CTI".
- Category 3: It accommodates the extension of both types of cables and is, thus, designated as "All".

The nodes and edges of the network are modelled as shown in Figure 1 below. In the 3D model, a CTP is represented by a red line, a CTI is represented by a blue line, and the "All" type is represented with an orange line. The total number of edges is 9,711.

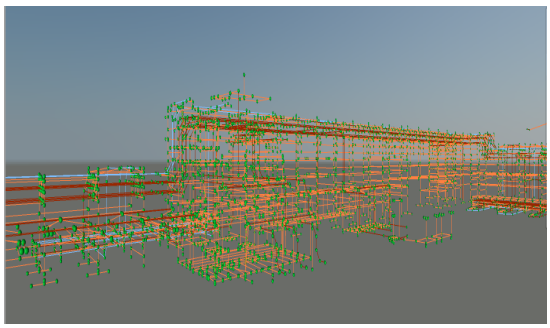

Figure 1. Network Model

Given equal lengths, different types of trays/edges differently impact the final cost. Thus, the type of each edge must be considered when finding the optimal path. This is achieved by factoring a cost-related weight into the total weight. Accordingly, for each tray type, a weight multiplier is computed to reflect its cost. This multiplier is used to adjust the distances between the nodes reflecting cost considerations for each edge type. Different edge types and their corresponding multipliers are summarized in Table 1 below.

Table 1. Weight Multipliers for Weight Groups

Weight Group	**Multi-plier**	**Weight Group**	**Multi-plier**
Cable Tray	1	Field Run to Tag	25
Cable Tray Angled 3D or Rotated	1.5	Ignore	1000
Cable Tray Angled East Vertical	1.5	Jumper Manual	15
Cable Tray Angled Horizontal	1.5	Steel Jumper	15
Cable Tray Angled North Vertical	1.5	Steel Tray Connect	1.1
Cable Tray East	1	Structural Steel	10
Cable Tray Elbow	1.5	Structural Steel Angled East Vertical	15
Cable Tray North	1	Structural Steel Angled North Vertical	15
Cable Tray Vertical	2	Structural Steel East	10
Cable Tray Jumper	1.5	Structural Steel North	10
Cable Tray Steel Jumper	1.5	Structural Steel Vertical	20

4 Methodology

The study presents a structured methodology that integrates multiple software to help decision makers find optimal network solutions quickly and efficiently. The methodology has been devised based on the following steps:

1. Build a 3D model of the cable network under study.
2. Design a database to store relevant data and identify the properties of nodes and edges. Microsoft Access database is used in this study. The database comprises five different tables as summarized in Table 2.

Table 2. Access Tables and Attributes

Table	Attributes
Nodes	ID, x, y, and z coordinates
Edges	ID, start node, end node, edge length (i.e. distance between nodes), edge type, edge category
Weight Groups	Edge type, weight multiplier
End Points	Source node, destination node
Shortest Paths	Tag (i.e. cable), start node, end node, sequence number (i.e. the position of the connection edge (node1, node2) in the path)

2.1 Extract the (x, y, z) coordinates of nodes from the 3D model of the facility and import them into the database.

2.2 Compile data on the edges which are represented by the nodes they connect. Identify and record the types of different edges as well as their lengths. The lengths of edges are computed using the Euclidean distance (Equation 1). Note that this distance is to be multiplied by the weight multiplier to obtain the total weight used for path optimization.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (1)$$

2.3 Identify the source node and destination node for each cable of the seven cables.

3. Develop a program that retrieves data from the database, identifies the shortest path, and exports the optimal path back into the database. Python programming language is used in this study; Python is widely used in numeric computation and includes libraries that help in data analysis and modelling of data [27]. The problem presented in this paper benefits from the built-in libraries, mainly NetworkX and pyodbc. NetworkX is a package used for the creation and the study of complex networks and includes standard shortest path algorithms [28]. pyodbc is an open-source Python module that allows accessing ODBC databases [29] and is used to establish a direct connection to the Access database. For each cable, the developed program performs the following tasks:

3.1 Retrieve relevant information from the database using SQL queries. Based on the cable type (power vs. instrument), the queries select the edges belonging to the categories that are compatible with the cable type. They also select a total weight column calculated by multiplying the distance by the weight multiplier.

3.2 Build a weighted graph from the nodes and the

available weighted edges.

3.3 Identify the shortest path using Dijkstra, A*[1], and Bellman-Ford algorithms from Networkx library.

3.4 Compute the total weight of the identified paths using the three algorithms.

3.5 Compute the time taken by each algorithm to identify the shortest path (i.e. runtime). The runtime is computed to compare the performance of the algorithms.

3.6 Select the path with the lower total weight if there is a difference between the weights.

4. Store the nodes of the identified shortest path back into the database.

5. Store the shortest path nodes in XML structure format that is readable by NavisWorks to visualize and verify the shortest path.

The presented methodology is illustrated in Figure 2.

5 Results and Comparison

The three algorithms found the same shortest paths for six sets of source and destination nodes out of the seven sets. Nevertheless, the shortest path identified for one set using Bellman-Ford passes through 100 nodes while those found using Dijkstra and A* require 101 nodes. However, the three of the identified paths have the same total weight indicating that both solutions are equally favourable given the imposed optimization criterion (i.e. minimal total weight).

The identified shortest path nodes for each set were imported into NavisWorks and highlighted. Figure 3 illustrates the shortest paths for three of the cables. The paths are highlighted in blue. It could be noted that the optimum path between the source and destination nodes is not necessarily the path with the shortest distance between them. As illustrated, the first path bypasses the location of the destination point before it returns back to it. This is a result of factoring costs of different types of edges in the total weight of edges.

The results of the three algorithms for each set are summarized in Table 3. Dijkstra and A* algorithms generally showed close performances with respect to runtime. Meanwhile, Bellman-Ford exhibited a lower performance in terms of computation speed as compared to the other two algorithms. The percentage difference between Dijkstra and Bellman-Ford's runtimes for the first cable reached 4,800% as it took 0.0980 and 0.00200 seconds to identify the optimal path using Bellman-Ford and Dijkstra, respectively.

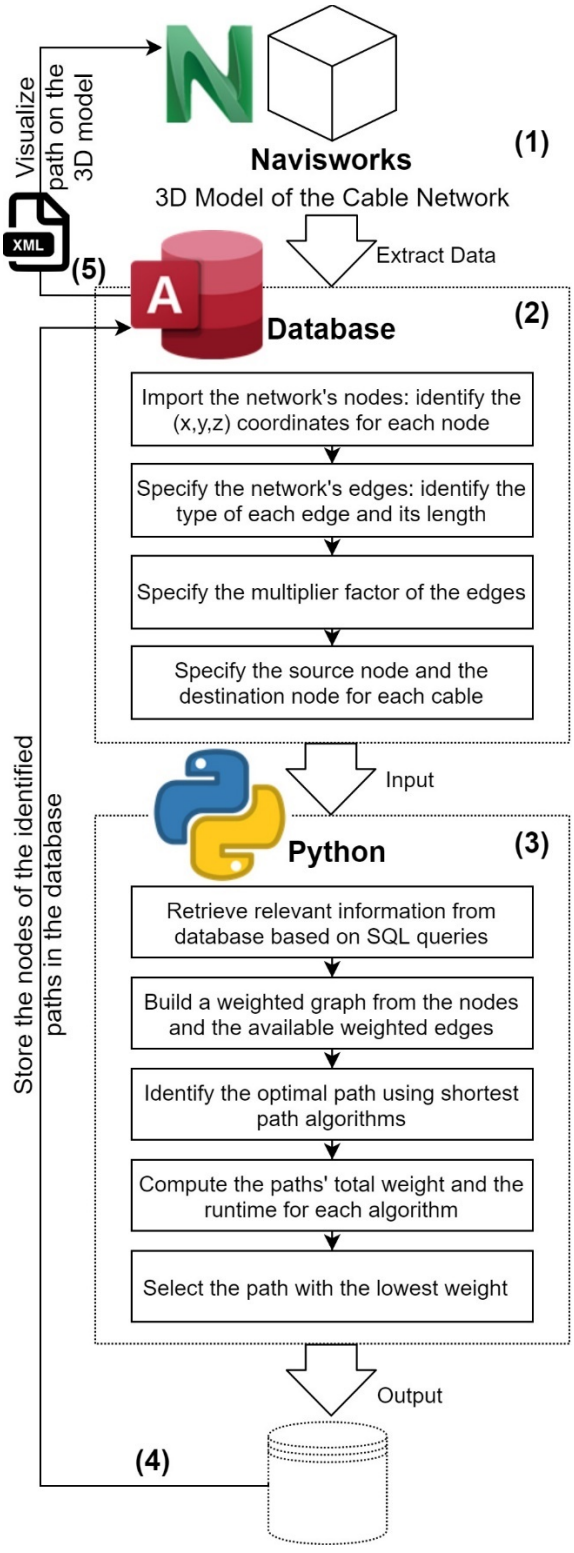

Figure 2. Methodology

[1] The Euclidean distance is used as the heuristic function for A* algorithm

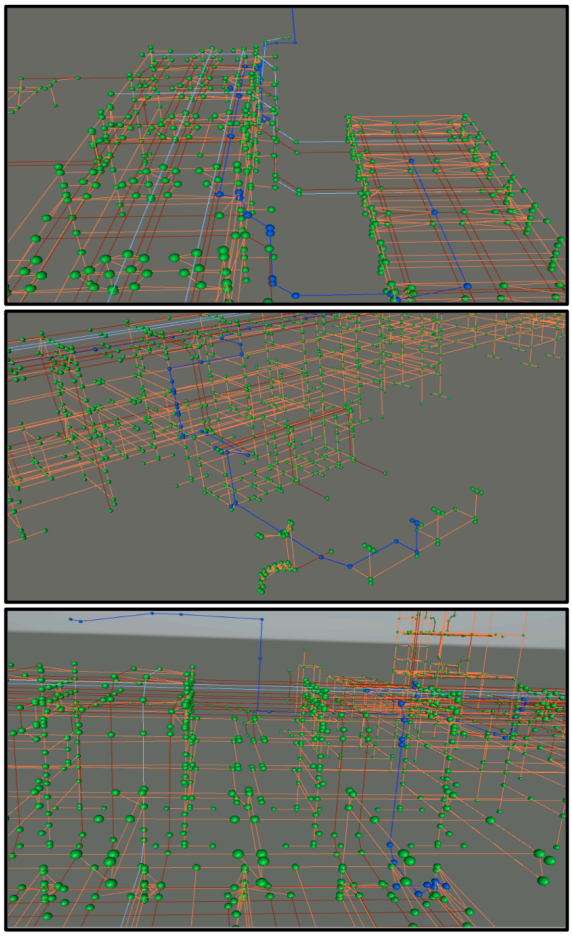

Figure 3. Shortest Paths in NavisWorks

The obtained results have shown that there is no significant difference between Dijkstra and A* algorithms. This could be attributed to the fact that the heuristic estimates (the Euclidean distance) of the A* algorithm are lower than the actual cost (the weighted distance) of moving from the nodes to the destination nodes. Consequently, the algorithm expands more nodes and, hence, takes a longer computation time. On the other hand, Bellman-Ford's underperformance in terms of speed was anticipated as a result of its looping behaviour. Although the difference between their performances is in the order of milliseconds, it becomes more significant as the size of the problem increases (i.e. as the number of nodes and edges increase). As the algorithm runtime is considered a major determinant in selecting the most convenient algorithm for a specific application [30], it could be concluded that Dijkstra and A* algorithms are deemed more suitable for solving a similar type of problems.

Table 3. Shortest Paths Results

Dijkstra			
Cable	Number of Nodes	Runtime	Weight
Test 1	45	0.00200	611.929
Test 2	78	0.0150	467.658
Test 3	101	0.0240	654.499
Test 4	41	0.00400	317.676
Test 5	55	0.00800	564.203
Test 6	84	0.0220	435.403
Test 7	78	0.0220	502.488
A*			
Cable	Number of Nodes	Runtime	Weight
Test 1	45	0.00400	611.929
Test 2	78	0.0300	467.658
Test 3	101	0.0480	654.499
Test 4	41	0.00400	317.676
Test 5	55	0.00900	564.203
Test 6	84	0.0270	435.403
Test 7	78	0.0350	502.488
Bellman-Ford			
Cable	Number of Nodes	Runtime	Weight
Test 1	45	0.0980	611.929
Test 2	78	0.0510	467.658
Test 3	100	0.114	654.499
Test 4	41	0.0620	317.676
Test 5	55	0.0670	564.203
Test 6	84	0.0720	435.403
Test 7	78	0.0480	502.488

6 Conclusion

Shortest path algorithms have been proven effective in providing support for decision makers when solving complex path-defining problems. On construction projects, these algorithms were used for diverse problems to attain optimal solutions. This paper considers the problem of selecting the shortest path for power and instruments cables in an industrial facility. The Dijkstra, Bellman-Ford, and A* algorithms were adopted to find the shortest path while considering the total weight of the edges. The total weight for this problem is dependent on both the distance and the cost subject to the type of each edge/tray.

Using the developed Python program to identify the shortest paths, results showed that all algorithms gave the same paths for six cables out of the seven cables. Additionally, the total weight is found to be the same across all tests indicating that these methods were equally efficient in finding the optimal solution given the optimization criteria. However, the results showed that the Dijkstra and A* algorithms had a better performance than that of Bellman-Ford with respect to runtime in all of the cases. Specifically, the difference between

Dijkstra's and Bellman-Ford's runtime reached 1,524% in one of the cases. Although for this case the difference is only in milliseconds, the difference between their performance becomes more significant in larger scale problems.

The paper also presented an integrated approach that combined different software platforms to automate the process of inputting data, computing the shortest path and then plotting this path in 3D format for visualization. The problem represented a case on the importance of using the shortest path method when dealing with decisions that are complicated by nature. Construction projects could benefit from using such algorithms and integrated approach for helping in various decision-making processes.

The methodology presented in this study could be extended to address the previously mentioned applications of shortest-path algorithms in construction. Today, with the continuously increasing level of automation in construction, the problem of finding the collision-free shortest paths for mobile robots on construction sites is of a particular importance. Hence, a case study on mobile construction robots shall be addressed in a future study. Efforts shall be devoted to select a case study of a larger size to demonstrate the impact of the performance of the different algorithms on the efficiency of solving the problem.

7 Acknowledgement

The authors wish to acknowledge the technical support and assistance of Nehme Roumani in this study.

8 References

[1] Al-Tabtabai H. and Alex AP. Using genetic algorithms to solve optimization problems in construction. *Engineering Construction and Architectural Management*, 6(2):121-32, 1999.

[2] Hegazy T. Optimization of construction time-cost trade-off analysis using genetic algorithms. *Canadian Journal of Civil Engineering*; 26(6):685-97, 1999.

[3] Damci A., Arditi D. and Polat G. Multiresource leveling in line-of-balance scheduling. *Journal of Construction Engineering and Management*, 139(9):1108-16, 2013.

[4] Lien LC. and Cheng MY. A hybrid swarm intelligence based particle-bee algorithm for construction site layout optimization. *Expert Systems with Applications*, 39(10):9642-50, 2012.

[5] Wong CK., Fung IW. and Tam CM. Comparison of using mixed-integer programming and genetic algorithms for construction site facility layout planning. *Journal of construction engineering and management*, 136(10):1116-28, 2010.

[6] Al Hattab M., Zankoul E. and Hamzeh FR. Near-real-time optimization of overlapping tower crane operations: a model and case study. *Journal of Computing in Civil Engineering*, 31(4):05017001, 2017.

[7] Yin C. and Wang H. Developed Dijkstra shortest path search algorithm and simulation. In *Proceedings of the 2010 International Conference on Computer Design and Applications,* pages V1-116, Qinhuangdao, China, 2010.

[8] Soltani AR., Tawfik H., Goulermas JY. and Fernando T. Path planning in construction sites: performance evaluation of the Dijkstra, A∗, and GA search algorithms. *Advanced engineering informatics*, 16(4):291-303, 2002.

[9] Magzhan K. and Jani HM. A review and evaluations of shortest path algorithms. *International journal of scientific & technology research*, 2(6):99-104, 2013.

[10] Zhang Z. and Zhao Z. A multiple mobile robots path planning algorithm based on A-star and Dijkstra algorithm. *International Journal of Smart Home*, 8(3):75-86, 2014.

[11] Chen JC. Dijkstra's shortest path algorithm. *Journal of Formalized Mathematics*, 15(9):237-47, 2003.

[12] Goyal A., Mogha P., Luthra R. and Sangwan N. Path finding: A* or Dijkstra's?. *International Journal in IT & Engineering*, 2(1):1-5, 2014.

[13] Ramadani E., Halili F. and Idrizi F. Tailored Dijkstra and Astar Algorithms for Shortest Path Softbot Roadmap in 2D Grid in a Sequence of Tuples. *International Journal of Science and Engineering Investigations*, 8(92), 2019.

[14] Gogoncea V., Murariu G. and Georgescu L. The use of Dijkstra's algorithm in waste management problem. *The Journal The Annals of "Dunarea de Jos" University of Galati, Fascicle IX. Metallurgy and Materials Science*, 28(2):125-127, 2010.

[15] Singh G. and Chopra V. An analysis of various techniques to solve travelling salesman problem: A Review. *International Journal of Advanced Research in Computer Science*, 3(5), 2012.

[16] Goldberg A. and Radzik T. *A heuristic improvement of the Bellman-Ford algorithm*. Computer Science Department, Stanford University, Stanford, CA 94305, 1993.

[17] Zou B., Hu J., Wang Q. and Ke G. A distributed shortest-path routing algorithm for transportation systems. In *Proceedings of the Seventh International Conference on Traffic and Transportation Studies*, pages 494-500, Kunming, China, 2010.

[18] Humblet PA. Another adaptive distributed shortest path algorithm. *IEEE transactions on communications*, 39(6):995-1003, 1991.

[19] Busato F. and Bombieri N. An efficient implementation of the Bellman-Ford algorithm for Kepler GPU architectures. *IEEE Transactions on Parallel and Distributed Systems*, 27(8):2222-33, 2015.

[20] Weber A., Kreuzer M. and Knoll A. A generalized Bellman-Ford algorithm for application in symbolic optimal control. In *Proceedings of the European Control Conference*, 2001.06231, Saint Petersburg, Russia, 2020.

[21] Bannister, M.J. and Eppstein, D. Randomized speedup of the Bellman–Ford algorithm. In *Proceedings of the Ninth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 41–47, Kyoto, Japan, 2012.

[22] Dinitz, Y. and Itzhak, R. Hybrid Bellman–Ford–Dijkstra algorithm. *Journal of Discrete Algorithms*, 42:35–44, 2017.

[23] Cavendish, D. and Gerla, M. Internet QoS routing using the Bellman-Ford algorithm. In *Proceedings of the International Conference on High Performance Networking*, pages 627–646, 1998.

[24] Fu, L., Sun, D. and Rilett, L.R. Heuristic shortest path algorithms for transportation applications: state of the art. *Computer and Operations Research*, 33(11): 3324–3343, 2006.

[25] Kim, S.K., Russell, J.S. and Koo, K.J. Construction robot path-planning for earthwork operations. *Journal of Computing in Civil Engineering*, 17(2):97–104, 2003.

[26] Lei, Z., Han, S., Bouferguène, A., Taghaddos, H., Hermann, U. and Al-Hussein, M. Algorithm for mobile crane walking path planning in congested industrial plants. *Journal of Construction Engineering and Management*, 141(2):5014016, 2015.

[27] Python.org. Applications for Python. On-line: https://www.python.org/about/apps/, Accessed: 03/06/2020.

[28] PyPI.org. Network X. On-line: https://pypi.org/project/networkx/, Accessed: 03/06/2020.

[29] PyPI.org. pyodbc. On-line: https://pypi.org/project/pyodbc/, Accessed: 03/06/2020.

[30] Zhan, F.B. and Noon, C.E. Shortest path algorithms: an evaluation using real road networks. *Transportation Science*, 32(1): 65–73, 1998.