An Improved Dijkstra-based Algorithm for Resource Constrained Shortest Path

Pan Liu[1*], Yihao Li[2], Shili Ai[1], Cong Luo[1], and Chengjian Yang[1]
[1]Faculty of Business Information, Shanghai Business School, Shanghai 201400, China
[2]School of Information and Electrical Engineering, Ludong University, Yantai, China
panl008@163.com, yihao.li@ldu.edu.cn, 2115258607@qq.com, 2680211421@qq.com, 530590767@qq.com
*corresponding author

***Abstract*—As a variation of the shortest path problem, the resource constrained shortest path problem (RCSPP) has been widely studied in academia and industry. Many related algorithms and Lagrange relaxation technologies have been proposed to solve the RCSPP. The paper presents a new algorithm to solve the RCSPP by improving Dijkstra's algorithm. Firstly, we illustrate the process of solving the RCSPP based on the idea of Dijkstra's algorithm through a simple example. Then, we design an algorithm to realize this process. To improve the running efficiency of the proposed algorithm, we give three empirical hypotheses as Lagrangian relaxation techniques. Next, an improved algorithm is presented to solve the RCSPP based on these three hypotheses. Finally, an example is given to demonstrate the improved algorithm. Compared with Dijkstra's algorithm for the RCSPP, our proposed algorithm can reduce the number of visits to nodes in the network.**

Keywords-resource constrained shortest path; Dijkstra's algorithm; shortest path; relaxation techniques

I. INTRODUCTION

The resource constrained shortest path problem (RCSPP) [1, 2] is derived from the shortest path problem. This problem is to find a shortest path from the graph which needs to satisfy certain constraints, such as travel time and travel cost. Because the RCSPP is more suitable for complex real conditions than shortest path problem, dealing with the RCSPP has more practical value and application signification. In practice, the application scenarios of the RCSPP include the travelling salesman problem [3], the route planning problem [4], the bus route problem [5], and the vehicle routing problem [6-8]. Due to the constraints of multiple resource constraints and basic path constraints, the RCSPP is significantly more difficult to solve than the shortest path problem with a single constraint. So, the problem is also a strong NP hard problem [9].

When the map is small in size, we can enumerate all feasible solutions to find a resource constrained shortest path in the map. However, the method of exhaustion [10] is brutal and the amount of computation can soar quickly as the size of the data increases. Therefore, in most cases, this method is not suitable for solving the RCSPP because it cannot quickly find a suboptimal or feasible solution. In the past, several approaches have been proposed to solve the RCSPP problem. For example, Handler and Zang [11] relaxed the capacity constraint by Lagrange relaxation, and obtained the optimal solution by solving the dual problem of the original problem. By using the cost reduction obtained from the optimal Lagrange multiplier to run the path ranking algorithm, Aneja et al. [12] presented a preprocessor for the RCSPP, and then Beasley and Christofides [13] improved the preprocessor for the RCSPP.

The paper analyzes the process of dealing with the RCSPP based on Dijkstra' algorithm. Then, to improve its efficiency in real applications, three hypotheses as three Lagrange relaxation techniques are proposed. Next, combining Dijkstra-based algorithm for the RCSPP and these three hypotheses, we design a new algorithm to solve the RCSPP. Through a simple example, we illustrate the process of our algorithm to deal with the RCSPP. Compared with Dijkstra-based algorithm for the RCSPP, the efficiency of our algorithm is improved by more than 32% in our experiments.

The structure of this paper is as follows: In Section 2, two hypotheses are proposed and the idea of resource shortest path problem is presented based on Dijkstra's algorithm. A simple example illustrates the processing of our proposed method. In Section 3, our algorithm implements the algorithm proposed in Section 2. Section 4 analyzes the algorithm proposed in Section 3 and presents an improved algorithm to deal with resource-constrained shortest path problems based on a new assumption. Section 5 uses a simple example to demonstrate the processing of our improved algorithm.

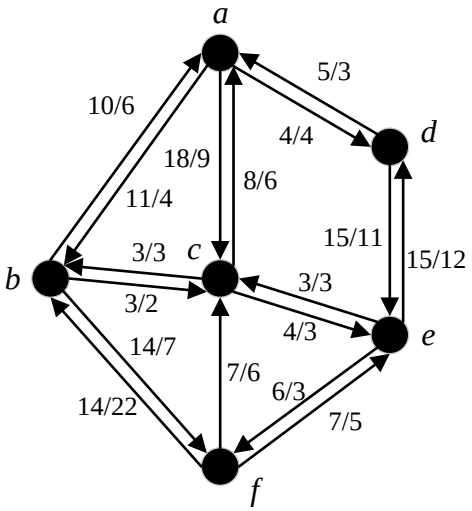

Fig. 1. A directed graph

2767-6684/22/$31.00 ©2022 IEEE
DOI 10.1109/DSA56465.2022.00056

II. DIJKSTRA-BASED ALGORITHM

Dijkstra's algorithm was designed by Dijkstra to solve the shortest path problem with single constraint [14]. We can obtain a shortest path satisfying a resource constraint by adding a resource constraint to Dijkstra's algorithm. Fig. 1 is a directed graph consisting of 6 nodes and 19 directed edges. The numbers on these edges in Fig. 1 represent the path length/number of resource consumption. Assuming node *a* is an initial node. Now, we need to find a shortest path that consumes resource is not greater than 12 from node *a* to node *f* in Fig.1.

The method of Dijkstra's algorithm is to traverse Fig. 1 starting from node *a*. If the resource consumption on a path is greater than 12 during the traversal of Fig. 1, the path is abandoned and the next path continues until a shortest path satisfying resource consumption 12 from node *a* to node *f* is found in Fig.1.

TABLE I. THE PROCESS OF FINDING SHORTEST PATH SATISFY RESOURCE CONSUMPTION 12 ACCORDING TO DIJKSTRA'S ALGORITHM.

S	U	dist[]						path[]					
		a	b	c	d	e	f	a	b	c	d	e	f
<a>	{b,c,d,e,f}	0/0	11/4	18/9	4/4	-/-	-/-	-	a	a	a	-	-
<a,d>	{b,c,e,f}	0/0	11/4	18/9	4/4	19/15	-/-	-	a	a	a	d	-
<a,d,b>	{c,e,f}	0/0	11/4	14/6	4/4	19/15	25/11	-	a	b	a	d	b
<a,d,b,c>	{e,f}	0/0	11/4	14/6	4/4	18/9	-/-	-	a	b	a	c	-
<a,d,b,c,e>	{f}	0/0	11/4	14/6	4/4	18/9	24/12	-	a	b	a	c	e

Table I shows the process of finding a resource constraint shortest path from node *a* to node *f* in Fig.1 based on the above method. In Table 1, S is a list of visited nodes, U is a set of unvisited nodes, array dist[] records the distance and resource consumption from node *a* to other nodes, array path[] records the precursor node of a node on the shortest path, and the symbol - indicates that it does not exist.

In the past, Dijkstra' algorithm can obtain dist[] according to the following formula.

$$dist[i,k] = \min\left(dist[i-1,k-1], dist[i,j] + w[j]\right) \quad (1)$$

In Eq. (1), the variable *i* denotes the i*th*, the variable *k* denotes the k*th* element in U, and *w*[*j*] is a distance weight on j*th* edge. For example, in Fig. 2, there are three nodes and three edges. According to Eq. (1), dist[i] corresponding to node *c* is min(9, 5+3)=8 in Fig. 2.

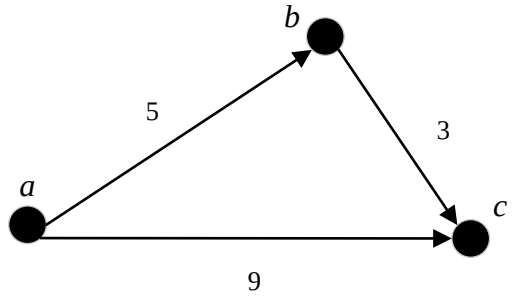

Fig. 2. An example for calculating dist[]

To solve the RCSPP, we change Eq. (1) to Eq. (2) as follows.

$$dist[i,k] = \begin{cases} \min\left(dist[i-1,k-1], dist[i,j] + w[j]\right), & resource[i,j] + w_r[j] \le RC \\ dist[i-1,k-1], & otherwise \end{cases} \quad (2)$$

where *resource*[*i*, *j*] denotes the resource consumption on the path from the source to node *j*, $w_r[j]$ is a resource consumption on jth edge, and *RC* is the resource constraint on paths.

From Table I, we visit Fig.1 from node *a*. So, there are S = <*a*>, U = {*b*, *c*, *d*, *e*, *f*}, dist[]={0/0, 11/4, 18/9, 4/4, -/-, -/-}, and path[]={-, *a*, *a*, *a*, -, -}. According to Dijkstra's algorithm, we can find a minimum distance from dist[] that is 4 in 4/4. Thus, node *d* in U is the next visited node and the shortest path from node *a* to node *d* is 4 with resource consumption 4. If we take node *d* as a starting node and then repeat the above process, node *b* in U can be selected. Similarly, we can repeat the above process until node *f* is selected.

Now, let's change resource consumption from 12 to 11 on the shortest path from node *a* to node *j*. Based on Dijkstra's algorithm and Eq. (2), we can construct Table II. Although the distance on the shortest path from node *a* to node *j* is 24 in Table I, the resource consumption is greater than the given resource constraint 11. Thus, we retain the path *a*-*b*-*f* with distance 25 and resource consumption 11 as a resource constrained shortest path.

TABLE II. THE PROCESS OF FINDING SHORTEST PATH SATISFY RESOURCE CONSUMPTION 11 ACCORDING TO DIJKSTRA'S ALGORITHM.

S	U	dist[]						path[]					
		a	b	c	d	e	f	a	b	c	d	e	f
<a>	{b,c,d,e,f}	0/0	11/4	18/9	4/4	-/-	-/-	-	a	a	a	-	-
<a,d>	{b,c,e,f}	0/0	11/4	18/9	4/4	19/15	-/-	-	a	a	a	d	-
<a,d,b>	{c,e,f}	0/0	11/4	14/6	4/4	19/15	25/11	-	a	b	a	d	b
<a,d,b,c>	{e,f}	0/0	11/4	14/6	4/4	18/9	-/-	-	a	b	a	c	-

<a,d,b,c,e>	{f}	0/0	11/4	14/6	4/4	18/9	25/11	-	a	b	a	c	b

Combining Dijkstra's algorithm and Eq. (2), we can construct Algorithm 1 to solve the TCSPP as follows.

Algorithm 1:

Input: A map G=(V, E), where V is a set of nodes and E is a set of edges, and resource constraint T;
Output: A shortest path from s0 to sn in G.

```
1  Initial a list S=<>, a set U=V, two arrays dist[]=-/- and path[]=-;
2  S=<s0>, U=U-{s0},dist[0]=0/0,path[0]=s1;
3  while(U!={}) do
4    Get the last element si of S;
5    For each edges from si to other nodes in G;
6      Get the weight w on the edge from si to sk;
7      if(dist[k]==-/-) then
8        dist[k]=w;
9        path[k]=si;
10     else if(dist[k]!=-/-) then
11       Get distance d and resource consumption t from
         dist[k] of sk;
12       Get distance d1 and resource consumption t1 from
         dist[i] of si;
13       Get distance d2 and resource consumption t2 from
         w;
14       if(d>d1+d2 && t1+t2<=T) then
15         dist[k]=(d1+d2)/(t1+t2);
16         path[k]=si;
17       endif
18     endif
19   endfor
20   Get a node sj with the minimum dist[j];
21   U=U-{sj};
22   Add sj to the end of S;
23 endwhile
24 return a shortest path from path[];
```

Similar to Dijkstra's algorithm, the time complexity of Algorithm 1 is $O(n^2)$.

III. IMPROVED ALGORITHM

In Algorithm 1, we find that algorithm's time consumption is related to the **while** statement in line 3 and the **for** statements in line 5. If we can reduce the number of loops in the **while** and **for** statements, the time consumption of Algorithm 1 will be reduced. Because we're solving for the resource constrained shortest path between two nodes s_0 and s_n in a network, we can modify the **while** statement in line 3 of Algorithm 1 to:

while(sn in U) **do**

Let p be sn in U and q be U!={}. Then, there is $p \rightarrow q$. Thus, our modification to line 3 of Algorithm 1 actually replaces a looser judgement condition with a stricter judgement condition, which will result in fewer loops of the **while** statement in line 3 of Algorithm 1 in practice.

The function of the **for** statement in line 5 of Algorithm 1 is to access the edges of the graph. Therefore, if the number of visits to the edges of the graph can be reduced, the number of loops in line 5 of the **for** statement can be reduced. To achieve this goal, we propose three empirical assumptions as Lagrange relaxation techniques to solve TCSPP.

Assumption 1: In a network, all edges ending at node s_0 must not be in any resource constrained shortest path from s_0 to s_n.

According to Assumption 1, we can delete three edges (b, a), (c, a), and (d, a) in Fig. 1, and then get Figure 3 (a).

Assumption 2: In a network, all edges starting from s_n must not be in any resource constrained shortest path from s_0 to s_n.

According to Assumption 2, we can delete three edges (f, b), (f, c), and (f, e) in Fig. 3 (a), and then get Fig. 3 (b).

Assumption 3: In a network, if the resource constrained shortest path from s_0 to s_i has been found by Algorithm 1, then those edges ending at s_i and starting at nodes in U of Algorithm 1 must not be in any resource constrained shortest path from s_0 to s_n.

Assumption 3 is a generalization of Assumption 1. In the process of solving the shortest path from node s_0 to node s_n, algorithm 1 will visit many intermediate nodes. If these intermediate nodes are regarded as initial nodes and s_n is regarded as terminal node, hypothesis 3 can be obtained according to assumption 1.

By Assumption 3，we can reduce some edges during the execution of algorithm 1. For example, let's consider the construction process for obtaining the shortest path that satisfies resource 12 in Table 1. When we select node d from U, the resource constrained shortest path from node a to node d has been obtained. Thus, the edge from node e to node d can be deleted from Fig. 3 (b) according to Assumption 3, and then Fig. 3 (c) is obtained. Then, after we select node b from U, the resource constrained shortest path from node a to node b has been obtained. Thus, the edge from node c to node b can be deleted from Fig. 3 (c) according to Assumption 3, and then Figure 3 (d) is obtained. Similarly, we can delete the edge from node e to node c when we select node c from U, and then Fig. 3 (e) is obtained. We select node e from U and then obtain Figure 3 (f). When we select node f from U, Figure 3 (g) is obtained. Finally, we can obtain a resource constrained shortest path in Figure 3 (h).

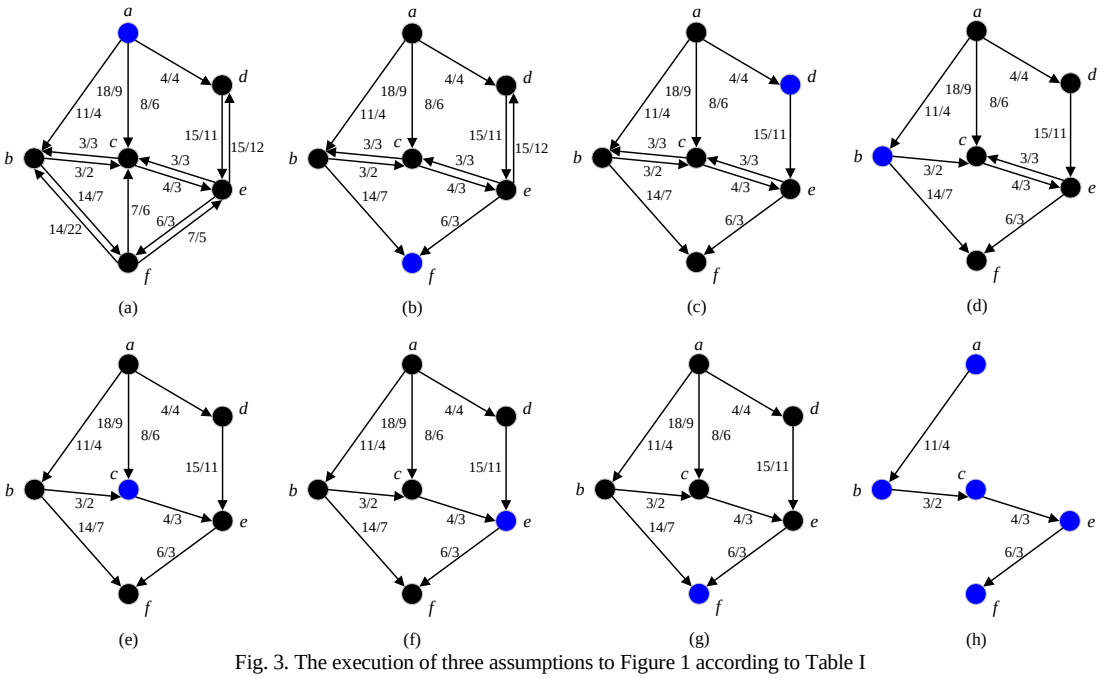

Fig. 3. The execution of three assumptions to Figure 1 according to Table I

Now, we need to design pseudocodes to realize assumptions 1, 2, and 3. For Assumption 1, we design the following statement.

Delete those edges in G by assumption 1;

For Assumption 2, we have designed the following statement.

while(sn in U) **do**

For Assumption 3, we can insert a new statement between lines 23 and 24 of algorithm 1 as follows:

Delete those edges from nodes in U to sj;

Therefore, Algorithm 1 can be improved to Algorithm 2 as follows.

Algorithm 2:

Input: A map G=(V, E), where V is a set of nodes and E is a set of edges, and resource constraint T;
Output: A shortest path from s0 to sn in G.

```
1  Initial a list S=<>, a set U=V, two arrays dist[]=-/- and path[]=-;
2  S=<s0>, U=U-{s0},dist[0]=0/0,path[0]=s1;
3  Delete those edges in G by assumption 1; //For assumption 1.
4  while(sn in U) do //For assumption 2.
5    Get the last element si of S;
6    For each edges from si to other nodes in G;
7      Get the weight w on the edge from si to sk;
8      if(dist[k]==-/-) then
9        dist[k]=w;
10       path[k]=si;
11     else if(dist[k]!=-/-) then
12       Get distance d and resource consumption t from
         dist[k] of sk;
13       Get distance d1 and resource consumption t1 from
         dist[i] of si;
14       Get distance d2 and resource consumption t2 from
w;
15       if(d>d1+d2 && t1+t2<=T) then
16         dist[k]=(d1+d2)/(t1+t2);
17         path[k]=si;
18       endif
19     endif
20   endfor
21   Get a node sj with the minimum dist[j];
22   U=U-{sj};
23   Add sj to the end of S;
24   Delete those edges from nodes in U to sj; //For
       assumption 3.
25 endwhile
26 return a shortest path from path[];
```

Although the worst time complexity of algorithm 2 is still $O(n^2)$, it can greatly reduce the number of edge accesses in practical applications. In the network, if the distance between the initial node s_0 and the terminal node s_n is k, and each node on the shortest path has an average of i edges out and j edges in, the time complexity of algorithm 2 is $O(k * i)$.

IV. Experiment

The purpose of our experiment is to compare the number of the maximum number of cycles of Algorithm 1 and Algorithm 2. The experiment is carried out on a traffic map of Chongming Island in China. The traffic map has 78 nodes and 264 roads. For experimental comparison, we number all nodes on the traffic map from 0 to 77. We then randomly pick 10 pairs of nodes as the sources and destinations from the map, and randomly get resource constraints between 100

and 350. Then, we call Algorithm 1 and Algorithm 2 respectively to compare their maximum cycles, and the experimental result is shown in Table 3.

TABLE III. COMPARISON OF ALGORITHM 1 AND ALGORITHM 2 FOR THE MAXIMUM CYCLES.

Experiment			Maximum cycles (times)	
source	**destination**	**resource**	**Algorithm 1**	**Algorithm 2**
28	64	304	3003	2832
14	10	129	3003	77
67	71	272	3003	2967
27	13	267	3003	858
66	62	263	3003	302
11	40	311	3003	2142
46	43	181	3003	77
47	32	308	3003	1518
12	52	259	3003	2568
50	40	273	3003	588

From Table 3, the maximum cycles of Algorithm 2 are smaller than those of Algorithm 1 under the same sources, destinations, and resource constraints. Therefore, the efficiency of Algorithm 2 is higher than that of Algorithm 1 in practical application. In addition, the value of Algorithm 1 is stable, while that of Algorithm 2 is not stable. Moreover, the value of Algorithm 2 is sometimes much less than that of Algorithm 1, which indicates that in some cases, the efficiency of Algorithm 2 is much better than that of Algorithm 1.

To quantitatively compare the efficiency of the two algorithms, we randomly select 10,000 times of source, destination, and resource constraint. Then we count the average cycles of Algorithm 1 and Algorithm 2 in this experiment. The experimental result is shown in Fig. 4. From Fig. 4, the average number of cycles for Algorithm 1 is 3003, while that for Algorithm 2 is 2020. Therefore, the efficiency of Algorithm 2 is 32.7% higher than that of Algorithm 1 in the experiment.

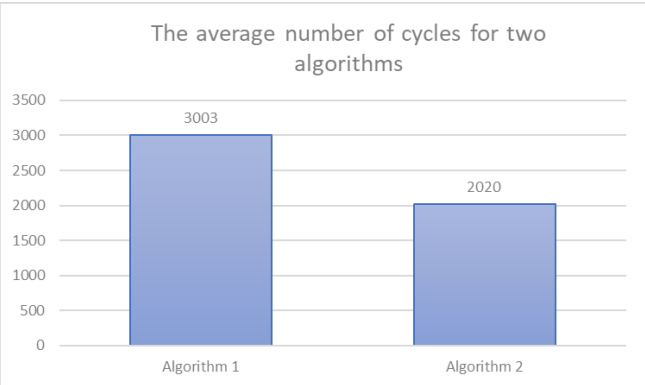

Fig. 4. Experimental results of algorithm 1 and 2 after 10,000 times random selection of sources, destinations, and resource constraints

Note that because Algorithm 1 is to solve the resource constrained shortest path from one node to all nodes, while Algorithm 2 is to solve the resource-constrained shortest path between two nodes. To realize the function of Algorithm 1, we simply need to change the sentence in Line 4 of Algorithm 2 back to "**while**(U!={}) **do**" in Line 4 of algorithm 1.

V. CONCLUSION

The TCSPP is a common problem in the field of transportation. To solve this problem, many related algorithms and Lagrange relaxation techniques have been proposed in the past. This paper first designs an algorithm for solving the TCSPP based on Dijkstra's algorithm. Then, we analyze the number of cycles of the algorithm and point out the possibility of reducing the number of cycles. Next, we propose three empirical assumptions as Lagrange relaxation techniques. Based on these three assumptions, we improve the algorithm based on Dijkstra's algorithm. Then, we use a simple example to demonstrate the execution of the improved algorithm. In the experiment, we compare the number of cycles of this improved algorithm with Dijkstra-based algorithm. Experimental result shows that our improved algorithm can reduce the number of cycles by more than 32% compared with Dijkstra's algorithm. Therefore, our algorithm has potential application value.

REFERENCES

[1] N. Tanoumand and T. Ünlüyurt, " An Exact Algorithm for the Resource Constrained Home Health Care Vehicle Routing Problem," *Annals of Operations Research*, vol. 304, no. 1, pp. 397-425, 2021.

[2] M. Ruß, G. Gust, and D. Neumann, "The Constrained Reliable Shortest Path Problem in Stochastic Time-Dependent Networks," *Operations Research*, vol. 69, no. 3, pp. 709-726, 2021.

[3] K. Panwar and K. Deep, " Discrete Grey Wolf Optimizer for Symmetric Travelling Salesman Problem," *Applied Soft Computing*, vol. 105, p. 107298, 2021.

[4] X. Wang, L. Wang, S. Wang, J.-f. Chen, and C. Wu, " An Xgboost-enhanced Fast Constructive Algorithm for Food Delivery Route Planning Problem," *Computers & Industrial Engineering*, vol. 152, p. 107029, 2021.

[5] X. Chen, Y. Wang, Y. Wang, X. Qu, and X. Ma, " Customized Bus Route Design with Pickup and Delivery and Time Windows: Model, Case Study and Comparative Analysis," *Expert Systems with Applications*, vol. 168, p. 114242, 2021.

[6] A. Agárdi, L .Kovács, and T. Bányai, "Using Time Series and Classification in Vehicle Routing Problem," *International Journal of Performability Engineering*, vol. 17, no. 1, pp. 14-25, January 2021.

[7] I. Kucukoglu, R. Dewil, and D. Cattrysse, " The Electric Vehicle Routing Problem and Its Variations: a Literature Review," *Computers & Industrial Engineering*, vol. 161, p. 107650, 2021.

[8] A. Agárdi, L .Kovács, and T. Bányai, " Neutrality of Vehicle Routing Problem," *International Journal of Performability Engineering*, vol. 17, no. 10, pp. 848-857, October 2021.

[9] P. Festa, " Constrained Shortest Path Problems: State-of-the-art and Recent Advances," in *2015 17th International Conference on Transparent Optical Networks (ICTON)*, 2015: IEEE, pp. 1-17.

[10] A. Minkin, O. Nikolaeva, and A. Russkov, " Hyperspectral Data Compression Based Up-on the Principal Component Analysis," *Computer Optics*, vol. 45, no. 2, pp. 235-244, 2021.

[11] G. Y. Handler and I. Zang, " A Dual Algorithm for the Constrained Shortest Path Problem," *Networks*, vol. 10, no. 4, pp. 293-309, 1980.

[12] Y. P. Aneja, V. Aggarwal, and K. P. Nair, " Shortest Chain Subject to Side Constraints," *Networks*, vol. 13, no. 2, pp. 295-302, 1983. [Online]. Available: https://doi.org/10.1002/net.3230130212.

[13] J. E. Beasley and N. Christofides, " An Algorithm for the Resource Constrained Shortest Path Problem," *Networks*, vol. 19, no. 4, pp. 379-394, 1989. [Online]. Available: https://doi.org/10.1002/net.3230190402.

[14] M. Barbehenn, " A Note on the Complexity of Dijkstra's Algorithm for Graphs with Weighted Vertices," *IEEE transactions on computers*, vol. 47, no. 2, p. 263, 1998. [Online]. Available: https://doi.org/10.1109/12.663776.