The Use of Minimal Spanning Tree for Optimizing Ship Transportation

Korištenje minimalnog razgranatog stabla radi optimizacije brodskog prijevoza

Karel Antoš
Department of Informatics and Natural Sciences
Institute of Technology and Business in České Budějovice, Czech Republic
e-mail: antos.vste@seznam.cz

DOI 10.17818/NM/2016/SI1
UDK 656.615
656.629
Preliminary communication / *Prethodno priopćenje*
Paper accepted / *Rukopis primljen:* 21. 3. 2016.

Summary

This article describes a design of solving ship transport optimization using tools from graph theory. The tool minimal spanning tree (denote MST) is suitable for searching ideal transport routes between the ports. The principle of the MST problem is that it describes various kinds of situations where it is necessary to use this theoretical instrument and how to use this tool for finding a solution. Graph theory knows several algorithms for searching the minimum spanning tree and this article compares two of them, in terms of their different approaches, their complementarity, and their assessment, and which of these two methods can find a feasible solution faster. To model the situation of ship transportation we use connected weighted graph where vertices represent sea ports and the edges represent the transport routes between the ports. The weight of an edge represents energy consumed to drive the boat between two ports. A theoretical discussion and a model example are carried out to compare the two methods.

KEY WORDS

ship transportation
graph theory
minimum spanning tree
Joseph Kruskal
Reverse-Delete Algorithm

Sažetak

Ovaj članak opisuje nacrt rješenja optimizacije brodskog prijevoza koristeći alate iz teorije grafa. Alat minimalno razgranatog stabla (MST) je prikladan za traženje idealnih prijevoznih ruta između luka. Princip MST problema je taj da on opisuje različite vrste situacija gdje je potrebno koristiti ovaj teoretski instrument i znati kako koristiti ovaj alat za pronalazak rješenja. Teorija grafa poznaje različite algoritme za traženje minimalno razgranatog stabla, a ovaj članak uspoređuje dva od njih, po pitanju različith pristupa, njihove komplementarnosti i njihove procjene i koja od ove dvije metode može naći održivo rješenje što brže. Da bismo oblikovali situaciju prijevoza koristimo povezane opterećene grafove gdje vrhovi predstavljaju morske luke, a rubovi predstavljaju prijevozne rute između luka. Opterećenje ruba predstavlja energiju koja je konzumirana da bi vozila čamac između dvije luke. Teoretska diskusija i ogledni primjerak izvršeni su da bi se usporedile dvije metode.

KLJUČNE RIJEČI

prijevoz brodom
teorija grafa
minimalno razgranato stablo
Joseph Kruskal
Algoritam Reverse-Delete

1. INTRODUCTION

Organization of ship transportation is associated with the solution of many problems [8]. One of them might be requirement to solve shipping transportation of goods from the central warehouse to all ports so as to ensure connection to all ports and the costs for transport connection to be as low as possible.

Imagine entering when we have sea ports, which are connected by ship transport routes so that there is sufficient connection to each port. Transport hubs are ports and transport routes are shipping lanes.

To solve such a problem we can use tools from the graph theory [7]. To model this situation we create a connected weighted graph where vertices represent sea ports and the edges represent the transport routes between the ports through which ships transport goods. The weight of an edge connecting two vertices represents the energy consumed to drive the boat between these two ports.

At the beginning there is a situation where ships transport goods between hubs over many different routes and in different ways but the transport links are inefficient and expensive as a whole.

Our task now is to optimize the connections between hubs so as to ensure connection to all transportation hubs and the transport costs between all hubs were minimal.

To search for optimal transport connection we can use the tool spanning tree from the graph theory [9]. This tool ensures

connection between all hubs by the only way. The minimum spanning tree then ensures that this unique connection will be the least expensive.

2. STARTING CONDITIONS

In the beginning we set the input conditions.

- All graphs in this article are finite, connected and simple.
- To model this situation will create a connected graph $G = (V, E)$ with weighted edges.
- V is the set of vertices of the graph G, E is the set of edges of the graph G.
- For each edge $e \in E$ is given a number $w(e)$ which we call the weight of an edge e.
- The condition to connect each shipping port with boat transport route and together to reduce costs for transport connection to minimum satisfies the minimum spanning tree $T = (V, E')$.

The spanning tree of a connected graph G is a connected subgraph of G which does not contain any cycles and it is the tree [11].

Let $G = (V, E)$ be a connected graph with n vertices. The spanning tree of the graph G is the tree $T = (V', E')$, where $V' = V$ and $E' \subseteq E$, where E' is the set of $n - 1$ edges of the spanning tree. The weight of the spanning tree of the weighted graph G we mean the sum of the weights of all edges of the spanning tree $w(E') = \sum_{e \in E'} w(e)$.

If the weight of each edge is the same, each spanning tree will have the same weight, because any spanning tree has $n - 1$ edges, and the sum of the same number of edges is therefore always the same [10]. If, however, weights of individual edges varies, different spanning trees may have different weights. Our task will be to find the spanning tree whose weight $w(E')$ is the smallest possible.

Minimum spanning tree (denote the abbreviation MST) [1] we mean the spanning tree $T = (V', E')$ in a graph G, where $V' = V$ and $E' \subseteq E$, and which is of the smallest value $w(E') = \sum_{e \in E'} w(e)$ [6], therefore, the sum of the valuations of its edges is minimal.

For searching the minimum spanning tree there are several algorithms. To solve our problem, we use the Kruskal algorithm and compare it with the Reverse-Delete Algorithm, which was discovered by Kruskal, too. Both algorithms we use to solve the given example and compare their effectiveness.

In the literature known and cited method of Kruskal algorithm for looking for the minimum spanning tree [2] is based on the principle, that we gradually assign edges into the spanning tree, starting with the edge with the lowest weight, and after this starting edge we gradually add other edges in order of increasing weights so long as all the vertices are connected.

For this algorithm, there is also the opposite method, which is in English literature named as the "Reverse-Delete Algorithm" [3], which creates the minimum spanning tree just from the opposite end, namely so that we gradually remove the edges starting with the edge of the highest weight and then you gradually remove edges in decreasing sequence with exception that you can not remove any edge in case that it would disconnect the graph. In certain cases this algorithm may lead to solving the problem more efficiently.

3. DESCRIPTION OF KRUSKAL (ASSIGNING) ALGORITHM

- given a connected graph $G = (V, E)$ with n vertices and m edges such that $w(e) \geq 0$ for all $e \in E$
- at first arrange them into increasing sequence $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
- in the beginning put the empty set $E' = \{\}$ for the spanning tree.
- now for $i = 1, 2, \ldots, m$ take the edges e_i and try to add them to the set E'. If set $E' \cup \{e_i\}$ does not creates a cycle, add the edge e_i to the set E', if it creates a cycle, do not use it and make another step and test another edge from the sequence
- the algorithm stops after maximum of m-th step, when the set E' contains $n - 1$ edges of the minimum spanning tree of the weighted graph $G = (V, E)$.

4. EXAMPLE OF USING KRUSKAL (ASSIGNING) ALGORITHM FOR SEARCHING THE MINIMUM SPANNING TREE

Example of a graph (Figure 1), for which we are searching the minimum spanning tree by using Kruskal´s algorithm [5].

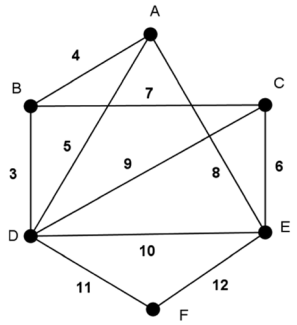

Figure 1 Example of a graph for looking for the MST

The procedure of the algorithm:

We mark the edges and arrange them in increasing order beginning with the edge with the smallest weight e_1 = {B,D}, e_2 = {A,B}, e_3 = {A,D}, e_4 = {C,E}, e_5 = {B,C}, e_6 = {A,E}, e_7 = {C,D}, e_8 = {D,E}, e_9 = {D,F}, e_{10} = {E,F}.

$w(e_1)$=3, $w(e_2)$=4, $w(e_3)$=5, $w(e_4)$=6, $w(e_5)$=7, $w(e_6)$=8, $w(e_7)$=9, $w(e_8)$=10, $w(e_9)$=11, $w(e_{10})$=12

We have the sequence of edges with $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_{10})$ and we start with the empty set of edges $E' = \{\}$. The following are the individual steps of testing edges of our sequence. These steps are displayed in Figure 2.

- start with the first edge of our sequence e_1 and because $E' \cup \{e_1\}$ does not create a cycle assign it into E'. $E' = \{e_1\}$. See Figure 2 „step 1"
- take another edge from the sequence e_2 and because $E' \cup \{e_2\}$ does not create a cycle, assign it into E'. $E' = \{e_1, e_2\}$. See Figure 2 „step 2"
- take another edge from the sequence e_3 and because $E' \cup \{e_3\}\}$ creates a cycle, do not assign it.
- take another edge from the sequence e_4 and because $E' \cup \{e_4\}$ does not create a cycle, assign it into E'. $E' = \{e_1, e_2, e_4\}$. See Figure 2 „step 4"
- take another edge from the sequence e_5 and because $E' \cup \{e_5\}$ does not create a cycle, assign it into E'. $E' = \{e_1, e_2, e_4, e_5\}$. See Figure 2 „step 5"

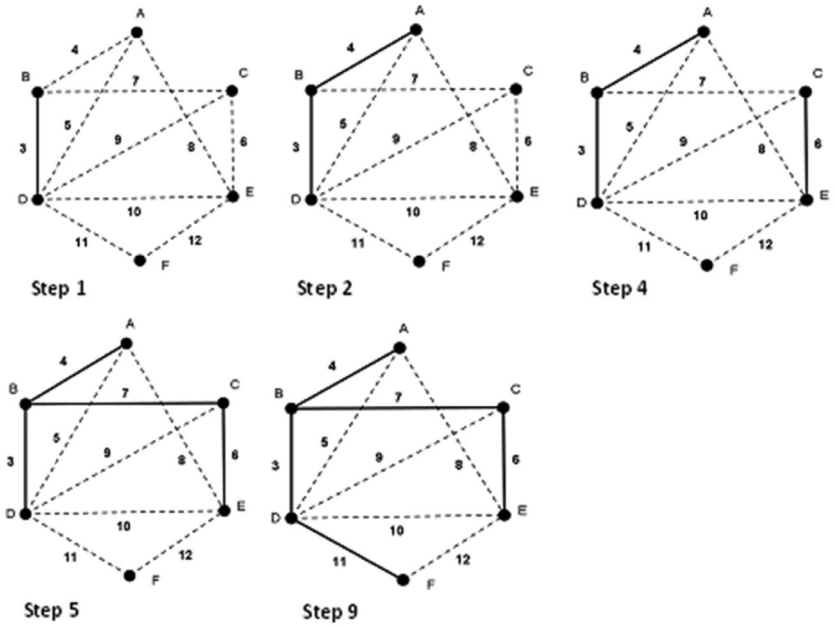

Figure 2 Steps of algorithm

- take another edge from the sequence e_6 and because $E' \cup \{e_6\}$ creates a cycle, do not assign it
- take another edge from the sequence e_7 and because $E' \cup \{e_7\}$ creates a cycle, do not assign it
- take another edge from the sequence e_8 and because $E' \cup \{e_8\}$ creates a cycle, do not assign it
- take another edge from the sequence e_9 and because $E' \cup \{e_9\}$ does not create a cycle, assign it into E'. $E' = \{e_1, e_2, e_4, e_5, e_9\}$. See Figure 2 „step 9"

So now, $|E'| = n - 1 = 6 - 1 = 5$ edges and $T = (V, E')$ is the spanning tree of the graph G and process of algorithm stops. The remaining edge e_{10} of the sequence does not need to be tested. For construction of the MST we made 9 steps.

The dashed lines in Figure 2 are the edges of the graph G that do not belong to the minimum spanning tree, the bold solid lines are the edges of the minimum spanning tree.

5. DESCRIPTION OF THE KRUSKAL´S REVERSE – DELETE ALGORITHM

This algorithm is in English called "Reverse - Delete Algorithm" [3].

- let $G = (V, E)$ be a connected graph with weighted edges $w: E \to \mathbb{R}_0^+$
- the edges of this graph arrange in a decreasing sequence $w(e_{m1}) \geq w(e_{m-1}) \geq \cdots \geq w(e_1)$
- let's start with the set of m edges of the graph G. From this set of edges remove the edge with the highest weight e_m. If removing it causes that the graph is divided into two components, this edge remains in the graph.
- in the next step remove another edge from the sequence with the second largest value, again with the condition that after its removal the graph may not be divided into two components.
- so the algorithm works until we remove $m - (n - 1)$ edges. Then we get the minimum spanning tree. From the original set E with m edges we get the set E' with $n - 1$ edges, which is our minimum spanning tree.

6. EXAMPLE OF USING "REVERSE - DELETE ALGORITHM" FOR SEARCHING THE MINIMUM SPANNING TREE

To demonstrate the use of this algorithm we use the same graph as in the previous example (Figure 1).

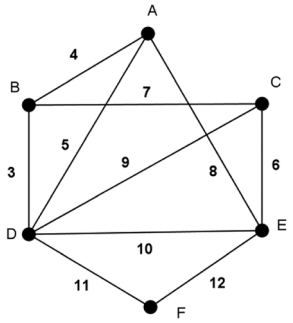

Figure 3 Example of the same graph for using the „Reverse - Delete Algorithm"

If the starting graph is the same graph as in our previous example, we will also work with the same set of edges $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$.

This set of edges is turned in decreasing order from the edge with the highest weight to the edge with the lowest weight $\{e_{10}, e_9, e_8, e_7, e_6, e_5, e_4, e_3, e_2, e_1\}$.

The Reverse - Delete Algorithm works with edges in chosen order and gradually removes $m - (n - 1)$ edges of the graph G (see in Figure 4). For each edge, check if deleting the edge will further disconnect the graph. Then this edge remains in the graph. Work with one edge is one step of the algorithm.

- Step 1: remove edge e_{10}
- Step 2: edge e_9 remains in the graph because deleting this will disconnect the graph
- Step 3: remove edge e_8
- Step 4: remove edge e_7
- Step 5: remove edge e_6
- Step 6: edge e_5 remains in the graph because deleting this will disconnect the graph

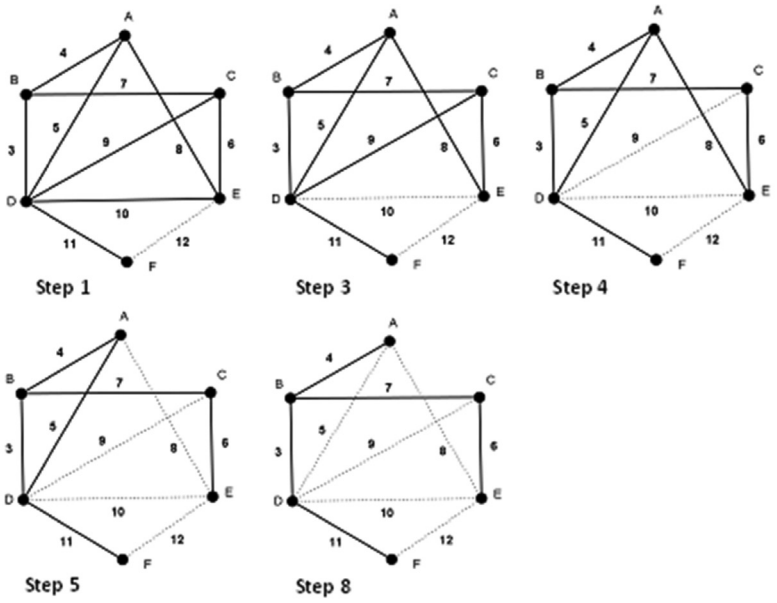

Figure 4 Steps of „Reverse-Delete Algorithm"

- Step 7: edge e_4 remains in the graph because deleting this will disconnect the graph
- Step 8: remove edge e_3

The algorithm stops, because we have removed $m-(n-1) = 10-(6-1) = 5$ edges and the minimum spanning tree with $n-1 = 6$ edges left; i.e. $E' = \{e_1, e_2, e_4, e_5, e_9\}$.

When the process of algorithm stops we get the same minimum spanning tree as using the previous algorithm (Figure 4, step 8) but the solution is found from the opposite way.

7. DISCUSSION OF RESULTS

7.1. Comparison of „Reverse - Delete Algorithm" with Kruskal´s (assigning) algorithm

- if we have a connected graph $G = (V, E)$, for which we are looking for the minimum spanning tree, there is the set of vertices V with n vertices and the set of edges E with m edges.
- Kruskal´s algorithm creates the minimum spanning tree after adding $n-1$ edges, because any spanning tree includes just this number of edges.
- „Reverse - Delete Algorithm" creates the minimum spanning tree after removing edges.
- the number of steps that Kruskal's algorithm must make until it finds the minimum spanning tree depends on the construction of the graph G; if the edge with the greatest weight can not be removed because of loss of the connectivity of the graph, the algorithm must go through all the edges until the last one is added into the spanning tree and therefore the algorithm makes m steps.
- the number of steps that „Reverse - Delete Algorithm" must make until it finds the minimum spanning tree again depends on the structure of the graph G, maximum number of edges that this algorithm must test is $m-2$, because from the decreasing sequence $\{e_m, e_{m-1}, \ldots e_2, e_1\}$ the edges e_1 and e_2 can not be removed because the last two edges from the given nonincreasing sequence belong to the MST because they do not make a cycle and their value is the smallest.
- the speed of searching the minimum spanning tree depends on the structure of the graph G for which we search the MST. The speed means the number of steps which the algorithm must make.

8. COMPARISON OF KRUSKAL´S (ASSIGNING) ALGORITHM WITH KRUSKAL´S „REVERSE - DELETE ALGORITHM" WHEN WE USE OUR EXAMPLE GRAPH

We marked the edges of our graph $G = (V, E)$ and arranged them in an increasing sequence according their weights $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_{10})$.

$\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$

By using Kruskal (assigning) algorithm we gradually added edges into the spanning tree T of the graph G which we labeled by bold solid lines; edges which we did not use because making cycles we labeled by dashed lines.

In the „Reverse - Delete Algorithm" we arranged the same edges just in opposite order in a decreasing sequence

$\{e_{10}, e_9, e_8, e_7, e_6, e_5, e_4, e_3, e_2, e_1\}$

and gradually removed edges from the original set of edges E; removed edges we labeled by dashed lines, edges which could not be removed because of loss of connectivity we labeled by bold solid lines; these edges belong to seeking minimum spanning tree T of the graph G.

The final set of edges E' obtained from the both algorithms is always the same

$E' = \{e_1, e_2, e_4, e_5, e_9\}$

Kruskal´(assigning) algorithm made 9 steps until the minimum spanning tree was found. „Reverse - Delete Algorithm" made 8 steps until the minimum spanning tree was found. Processing one edge is considered to be one step.

In our graph example, the „Reverse - Delete Algorithm" was faster for finding the minimum spanning tree because it made only 8 steps until the MST was found.

9. CONCLUSION

In the article we showed that organization of ship transportation can be modeled and solved using tools from the graph theory. System of ship transport routes we converted to the graph

where routes are the edges and ports are vertices. The weight of an edge connecting two vertices represents the energy consumed to drive the boat between two ports.

For given task to find the only ideal and cheapest connection between ale ports we used the tool minimum spanning tree. We explained that the minimum spanning tree finds the only ideal connection to each port and that this connection is the cheapest.

We displayed two algorithms of searching minimum spanning tree, Kruskal´s assigning algorithm and Kruskal´s Reverse - Delete Algorithm.

We showed how these two algorithms work both in theoretical level and on the given example.

We made comparison of the choice of one or the other algorithm and explained that there is no definite conclusion which algorithm finds the MST faster since the structure of the graph may vary from case to case. This means that the decision, which algorithm will be optimal for given graph cannot be done depending on knowledge of parameters *n* or *m* of the graph *G*.

The reason is that we can imagine a graph in which the edges of the MST will be included in the spanning tree in the first steps but when any edge with the highest weight is placed in such a position where it could not be removed then the algorithm will have to go through all the edges until this edge will come to process. In this case the algorithm will make the same number of steps as the number of edges.

Reverse is also true, when the „Reverse-Delete Algorithm" gradually removes edges and because of the structure of the graph the last two edges from the non-increasing sequence will not be removed then this removing algorithm makes maximum steps.

In conclusion, it is necessary to say that the speed of searching the MST depends on the structure of the graph and not on the choice of any type of algorithm.

REFERENCES

[1] Harris J. M., Hirst J. L., Hossinghofer M. J., Combinatorics and Graph Theory, Springer, New York, (2000). http://dx.doi.org/10.1007/978-1-4757-4803-1

[2] J. Matoušek , J. Nešetřil, Kapitoly z diskrétní matematiky, vydání čtvrté (4th edition), Prague, Charles University in Prague, (2009).

[3] J. Kleinberg, E. Tardos, Algorithm Design, New York: Pearson Education, Inc. (2006).

[4] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, P Am Math Soc 7 (1956), 48–50., http://dx.doi.org/10.1090/S0002-9939-1956-0078686-7

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, (2001).

[6] T. S. Jackson, and N. Read, Theory of minimum spanning trees, Phys Rev E 81 (2010) 021130.

[7] Brázdová M., Využití některých metod teorie grafů při řešení dopravních problémů, Univerzita Pardubice

[8] Tuzar, A.; Maxa, P., Svoboda, V. Teorie dopravy. Praha: ČVUT, 1997. ISBN 80-01-01637-4

[9] Kolář, J.; Štěpánková, O.; Chytil, M. Logika, algebry a grafy. Praha: SNTL, 1989.

[10] Peterková A., Využití metod teorie grafů pro hledání nejspolehlivější cesty vdopravní síti, LOGVD 2012-Žilina 20.-21.9.2012

[11] K. H. Rosen, Discrete Mathematics and Its Applications – 6th ed., McGraw-Hill, New York NY, (2007), ISBN-10 0-07-288008-2.

[12] Krile, S., Krile, M., Prusa, P., Non-Linear Mimimax Problem For Multi-Stop Flight Routes, Transport, Vilnius, Litvania, 2015, Vol. 30., No3., pp. 361-371

[13] Krile, S., Efficient Heuristic for Non-linear Transportation Problem on the Route with Multiple Ports, Polish Maritime Research, Gdansk, Poland, 2013, Vol. 20, No 4, pp. 80-86