Periodica Polytechnica Transportation Engineering

https://doi.org/10.3311/PPtr.11171

The Use of Weighted Adjacency Matrix for Searching Optimal Transportation Routes

Karel Antos[1*]

RESEARCH ARTICLE

Received 21 February 2017; accepted 11 July 2017

Abstract

This article provides a new approach to searching solutions of the ship transport optimalization problems. It brings a new variant of one algorithm of searching for the Minimum Spanning Tree. The new element in the algorithm is that it uses the Weighted Adjacency Matrix. This Weighted Adjacency Matrix is suitable for searching for the Minimum Spanning Tree (MST) of the graph. It shows how it could be used in cases where weighted edges of the graph are given. This creates a new procedure of searching for the MST of the graph and completes previously known algorithms of searching for the MST. In the field of transportation it could be succesfully used for solutions of optimizing transportation routes where smallest costs are wanted. Proposed Weighted Adjacency Matrix could be used in similar issues in the field of the graph theory, where graphs with weighted edges are given. The procedure is shown on the attached example.

Keywords

graph theory, minimum spanning tree, Joseph Kruskal, reverse algorithm

[1] Department of Informatics and Natural Sciences,
Institute of Technology and Business in České Budějovice,
Okružní 517/10, 30 01 České Budějovice, Czech Republic

[*] Corresponding author, e-mail: antos.vste@mail.vstecb.cz

1 Introduction

One of the important aims in the field of shipping traffic is to find the ideal combination of shipping traffic routes so as to ensure the serviceability of all places and to reduce costs of transport connections as low as possible. It is necessary to reach each hub and to reduce transport costs to the minimum. Hubs are ports and transport routes are shipping lanes.

Graph theory offers useful tools for solving problems in this area. To model this situation we create a connected weighted graph where vertices represent sea ports and the edges represent the routes between the ports through which ships transporting goods. The weight of an edge between two vertices represents the energy consumed to drive the boat between these 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.

The task of our algorithm is now to optimize the connections between hubs, so that the cost of transport links between all ports were minimal with the condition that every port is reachable through traffic routes.

To search for optimal transport connection we can use the tool spanning tree from the graph theory. This tool is useful to optimize the connections between all hubs to be as simple as possible. Another tool is the minimum spanning tree, which ensures that this unique connection will be the least expensive. For searching the minimum spanning tree we offer here a new algorithm, which complements the previously known algorithms and demonstrates new and original approach.

2 Description of the MST issues

All graphs in this article are finite, simple and connected. The system of shipping traffic routes we can transform into the graph where vertices represent sea ports, edges represent transport routes and weights of edges represent the energy consumed to drive the boat between two ports. To model this situation we create a connected graph $G = (V, E)$ with weighted edges. The optimal traffic connection of the system is represented by the spanning tree of the graph. And the problem of the cheapest traffic system means that we must find the minimum spanning tree.

The spanning tree of a connected graph G is a subgraph G' which connects all vertices and which does not contain any cycles (Kleinberg and Tardos, 2006). The minimum spanning tree we denote $T = (V, E')$, where $V' = V$ and E' is the set of $n - 1$ edges of the minimum spanning tree, and it applies that $E' \subseteq E$. In the subsequent text we use the abbreviation MST (short for the Minimum Spanning Tree) (Jackson and Read, 2010). The sum of the weights of edges of MST is minimal.

For searching for the minimum spanning tree there are several obviously known algorithms which search for the MST in different ways. For example The Kruskal's algorithm, Prim's algorithm or Borůvka's algorithm are the generally known. In the article we use some principles of Prim's algorithm for searching the MST (Kruskal, 2004). But this article presents a new procedure for searching for the MST, which is Weighted Adjacency Matrix.

Let $G = (V, E)$ be a connected, finite and non-oriented graph with positively weighted edges, where V is a set of n vertices and E is the set of m edges. The set of vertices V we denote $V = \{v_1, v_2, \dots, v_n\}$ and the set of edges we denote E, where e_{ij} denotes the edge between vertices v_i and v_j, then it is $e_{ij} = \{v_i, v_j\} \in E$. $W(e_{ij})$ denotes the weight of the edge connecting vertices v_i and v_j, where $e_{ij} = \{v_i, v_j\} \in E$.

The spanning tree of a connected graph G is a subgraph G' which connects all vertices and which does not contain any cycles (Fredman and Willard, 1984). For this subgraph G' it holds that $G' = (V', E')$, where $V' = V$ and $E' \subseteq E$. Note that the E' set contains $n - 1$ edges (Ch[illegible] et al., 20[illegible]).

For subgraph $G' = (V', E')$ of the graph G we put $w(G') = \sum_{e \in E'} w(e)$. Because of the spanning tree is a tree we denote it T.

The spanning tree $T_1 = (V, E_1)$ of the graph G we call the minimum spanning tree if for each spanning tree $T_2 = (V, E_2)$ of the graph G it holds that $w(T_1) \leq w(T_2)$.

The minimum spanning tree we denote $T = (V, E')$, where $V' = V$ and E' is the set of $n - 1$ edges of the minimum spanning tree, and it applies that $E' \subseteq E$. In the subsequent text we use the abbreviation MST (short for the Minimum Spanning Tree). If we define the function $w: E \to R$ (ie. evaluation of edges), then the minimum spanning tree is such a spanning tree for which it holds that the sum of the weights of edges of MST is minimal, i.e. $w(T) = \sum_{e \in E'} w(e)$ is minimal.

In the following capture there is displayed a new algorithm which uses some new elements for searching the MST and adopts them to one of the previously mentioned, to the Prim's algorithm.

3 Weighted Adjacency Matrix

At first in the proposed algorithm we create a modified adjacency matrix, which we call "Weighted Adjacency Matrix". This matrix is similar to the Adjacency Matrix where in positions of elements of the matrix are either 1 or 0 if there is an edge between vertices v_i and v_j or not. In this modified Weighted Adjacency Matrix the positive number w_{ij} on the position of the element v_i and v_j indicates the weight of the edge connecting vertices v_i and v_j, if the edge between vertices v_i and v_j exists. A value of 0 indicates that there is no edge between vertices v_i and v_j (Goldberk, 1987).

Weighted Adjacency Matrix (Table 1) is thus a square matrix $W = n \times n$, where n denotes the number of vertices and the value of the element at the position w_{ij} corresponds to the weight of the edge between vertices v_i and v_j.

$$w_{ij} = \begin{cases} w(e_{ij}) & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Table 1 Weighted Adjacency Matrix

	v_1	…	v_i	…	v_j	…	v_n
v_1	0	[illegible]	w_{1i}	…	w_{1j}	…	w_{1n}
…			[illegible]	…	…	…	…
v_i			0	…	w_{ij}	…	w_{in}
…					…	…	…
v_j					0	…	w_{jn}
…							…
v_n							0

Weighted Adjacency Matrix is symmetric with respect to the main diagonal, the diagonal elements have a value of 0, the algorithm will only use the elements of the triangle above the main diagonal. The algorithm of searching for the MST works in the Weighted Adjacency Matrix and works with elements in the triangle above the main diagonal.

4 Algorithm procedure

Search through the elements of the matrix and find the one with the smallest positive value w_{ij}. Denote chosen matrix element in bold and underlined, then mark the rows v_i, v_j and columns v_i, v_j (Denote the columns and rows with arrows at the top of the table). If there is more than one element with the same smallest positive value, it is possible to choose arbitrary one of these. Then more than one MST exists.

Search again through the elements of the matrix and find another smallest positive element, search between elements in the marked rows and columns (Table 2). Chosen element denote in the matrix in bold and underlined. Let the new element be w_{jk}. According to the index position of the element mark the row v_k and column v_k. Rows and columns marked in the previous steps remain marked.

This step ensures the connection of the generated MST because the connecting edge has one of the indexes the same as the previous selected element, so this element connects to any of previously connected vertices.

Table 2 WAM 1

	v_1	…	v	v	…	v_k	…	v_n
v_1	**0**	…	$\boldsymbol{w_{1i}}$	$\boldsymbol{w_{1j}}$	…	$\boldsymbol{w_{1k}}$	…	$\boldsymbol{w_{1n}}$
…		**0**	…	…	…	…	…	…
v_i			**0**	$\underline{\boldsymbol{w_{ij}}}$	…	$\boldsymbol{w_{ik}}$	…	$\boldsymbol{w_{in}}$
v_j				**0**	…	$\boldsymbol{w_{jk}}$	…	$\boldsymbol{w_{jn}}$
…					**0**	…	…	…
v_k						**0**	…	$\boldsymbol{w_{kn}}$
…							**0**	…
v_n								**0**

Table 3 WAM 2

	v_1	…	v_i	v_j	…	v_k	…	v_n
v_1	**0**	…	$\boldsymbol{w_{1i}}$	$\boldsymbol{w_{1j}}$	…	$\boldsymbol{w_{1k}}$	…	$\boldsymbol{w_{1n}}$
…		**0**	…	…	…	…	…	…
v_i			**0**	$\underline{\boldsymbol{w_{ij}}}$	…	$\boldsymbol{x}$	…	$\boldsymbol{w_{in}}$
v_j				**0**	…	$\underline{\boldsymbol{w_{jk}}}$	…	$\boldsymbol{w_{jn}}$
…					**0**		…	…
v_k						**0**	…	$\boldsymbol{w_{kn}}$
…							**0**	…
v_n								**0**

Furthermore delete (ie. replace by the cross) all the elements in positions where newly marked row and column intersect with rows and columns previously marked. Here delete the element w_{ik}. This step prevents creating cycles.

Search again through the elements of the matrix and find another smallest positive element, search between elements in the marked rows and columns. Chosen element denote in the matrix in bold and underlined (Table 3). Let the new element be w_{1i}. According to the index position of the element mark the row v_1 and column v_1. Rows and columns marked in the previous steps remain marked. Furthermore delete all the elements in positions where newly marked row and column intersect with rows and columns previously marked. Here delete the elements w_{1j} and w_{1k} (Table 4).

Suppose that our algorithm makes k steps.

- If $k = n - 1$, algorithm stops, we have made all steps
- If $k < n - 1$, we make $(k + 1)$-th step analogously

After we make the $(n - 1)$-th step in our Weighted Adjacency Matrix $(n - 1)$ chosen elements are labeled (in bold and underlined), the other elements (which were not chosen) are replaced by a cross. At the same time all rows and columns in our matrix are labeled (Table 5). Elements denoted in the matrix in bold and underlined are values of weights of edges of the MST. Labeling the rows and columns of the selected element indicates the vertices v_i and v_j that the edge on this position connects. The sum of the values of all chosen elements gives the total weight of the MST.

Table 4 WAM 3

	v_1		v_i		…	v_k	…	v_n
v_1	0		$\underline{\boldsymbol{w_{1i}}}$	x	…	x	…	w_{1n}
…			…	…	…	…	…	…
v_i			0	$\underline{\boldsymbol{w_{ij}}}$	…	x	…	w_{in}
v_j				0	…	$\underline{\boldsymbol{w_{jk}}}$	…	w_{jn}
…					0	…	…	…
v_k						0	…	w_{kn}
…							0	…
v_n								0

Table 5 Final Matrix

	v_1	…	v_i	v_j	…	v_k	…	v_n
v_1	0	…	$\underline{\boldsymbol{w_{1i}}}$	x	…	x	…	$\underline{\boldsymbol{w_{1n}}}$
…			…	…	…	…	…	…
v_i			0	$\underline{\boldsymbol{w_{ij}}}$	…	x	…	x
v_j				0	…	$\underline{\boldsymbol{w_{jk}}}$	…	x
…					0	…	…	…
v_k						0	…	$\underline{\boldsymbol{w_{kn}}}$
…							0	…
v_n								0

5 Verification of the algorithm

1. Continuity of generated MST is guaranteed by the fact that newly connected edge has one of the indices the same as the indices of previously selected elements (Kruskal, 2000). Therefore, it connects to one of the previously connected vertices.

2. Avoiding the cycles is ensured by deleting all the elements in positions where newly marked row and column intersect with rows and columns previously marked (Cormen, et al., 2001).
3. The algorithm is a variant of the Prim´s algorithm, with the difference that in the first step we do not begin by selecting the arbitrary initial vertex, but in our Weighted Adjacency Matrix we begin by selecting the edge with the smallest weight. From the second step our algorithm works analogously as in the Prim´s algorithm (which has been proven, see (Harris et al., 2000)). This guarantees selection of the minimum spanning tree.

6 Demonstration of solved Example

Imagine the system of the sea transport routes. At first we transform the system of transport routes into the weighted graph (Fig. 1). There are 6 ports represented by 6 vertices of the graph $v_1, v_2, \dots, v_6$, connections between the ports are represented by the edges in the graph and numbers belonging to the edges represent the costs of energy consumed to drive the boat between two ports.

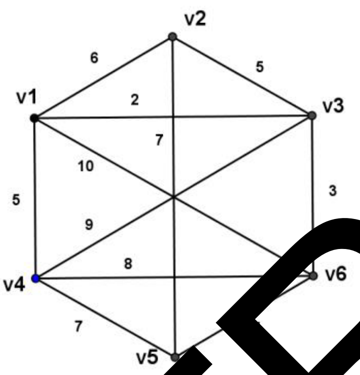

Fig. 1 Given Graph

Corresponding Weighted Adjacency Matrix is (Table 6):

Table 6 Given Matrix

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	6	2	5	0	10
v_2		0	5	0	7	0
v_3			0	9	0	3
v_4				0	7	8
v_5					0	4
v_6						0

7 Steps of algorithm

1. Search through the elements of the matrix and find the one with the smallest positive value $w_{13} = 2$. Denote chosen matrix element in bold and underlined, then mark the rows v_1, v_3 and columns v_1, v_3 (Table 7).

Table 7 WAM 4

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	6	[illegible]	5	[illegible]	0
v_2		0	5	0	7	[illegible]
v_3			0	[illegible]	0	3
v_4				0	7	8
v_5					0	4
v_6						0

2. Search again through the elements of the matrix and find another smallest positive element $w_{36} = 3$, search between elements in the marked rows and columns. Chosen element denote in the matrix in bold and underlined. According to the index position of the element mark the row v_6 and column v_6. Rows and columns marked in the previous steps remain marked (Table 8). Furthermore we delete (ie. replace by the cross) all the elements in positions where newly marked row and column intersect with rows and columns previously marked. Here we delete the element w_{16}.

Table 8 WAM 5

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	6	**2**	5	0	x
v_2		0	5	0	7	0
v_3			0	9	0	**3**
v_4				0	7	8
v_5					0	4
v_6						0

3. Search again through the elements of the matrix and find another smallest positive element $w_{56} = 4$, search between elements in the marked rows and columns. Chosen element denote in the matrix in bold and underlined.

According to the index position of the element mark the row v_5 and column v_5. Rows and columns marked in the previous steps remain marked. Furthermore delete all the elements in positions where newly marked row and column intersect with rows and columns previously marked. Here delete the elements, $w_{34} = x$, $w_{35} = x$ (Table 9).

Table 9 WAM 6

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	6	**<u>2</u>**	5	x	x
v_2		0	5	0	7	0
v_3			0	9	x	**<u>3</u>**
v_4				0	7	8
v_5					0	**<u>4</u>**
v_6						0

4. Search again through the elements of the matrix and find another smallest positive element $w_{14} = 5$, search between elements in the marked rows and columns. Chose[illegible]le-ment denote in the matrix in bold and underlined. Ac[illegible]rding to the index position of the element mark the ro[illegible] v_4 and column v_4. Rows and columns marked in the previo[illegible] steps remain marked. Furthermore del[illegible] all [illegible] elements in positions where newly marked [illegible]w and co[illegible]n intersect with rows and columns pre[illegible]usly mark[illegible] Here delete the elements $w_{34} = x$, w[illegible] x, w[illegible] x [illegible]able 10).

Ta[illegible] WAM 7

	v_1	v_2	v_3	[illegible]	v_5	v_6
v_1	[illegible]	6	**<u>2</u>**	**<u>5</u>**	x	x
[illegible]		0	[illegible]	0	7	0
v_3	[illegible]	[illegible]	0	x	x	**<u>3</u>**
v_4	[illegible]			0	x	x
v_5					0	**<u>4</u>**
v_6						0

5. Search again through the elements of the matrix and find another smallest positive element $w_{23} = 5$, search between elements in the marked rows and columns. Chosen element denotes in the matrix in bold and underlined. According to the index position of the element mark the row v_2 and column v_2. Rows and columns marked in the previous steps remain marked. Furthermore delete all the elements in positions where newly marked row and column intersect with rows and columns previously marked. Here delete the elements $w_{12} = x$, $w_{24} = x$, $w_{25} = x$, $w_{26} = x$ (Table 11).

Table 11 WAM 8

	v_1	v_2	v_3	v_4	v_5	[illegible]
v_1	0	x	**<u>2</u>**	**<u>5</u>**	x	x
v_2		[illegible]	**<u>5</u>**	[illegible]	x	x
[illegible]	[illegible]	[illegible]	[illegible]	x	x	**<u>3</u>**
v_4	[illegible]	[illegible]	[illegible]	[illegible]	x	x
v_5		[illegible]			0	**<u>4</u>**
v_6		[illegible]				0

8 Te[illegible]ination of th[illegible] algorithm

Whe[illegible] graph[illegible] has n vertices then the MST has $n - 1$ [illegible]ges (Kleinberg and Tardos, 2006). At each step of the algorit[illegible]dd to the gradually rising MST one edge, then [illegible]gorithm makes $n - 1$ steps. In our example, the graph has 6 vertices, then the MST has 5 edges. That is the reason why [illegible]lgorithm makes 5 steps.

After the final step, all the elements in the Weighted Adjacency Matrix went through processing, ie. the edges chosen for the MST are denoted in the agreed way, i.e. here in bold and underlined, deleted edges are marked by symbol x.

At the same time, after the last step all the rows and columns of the matrix are marked with the arrows next to the rows and above columns.

9 Final graph of the MST is here

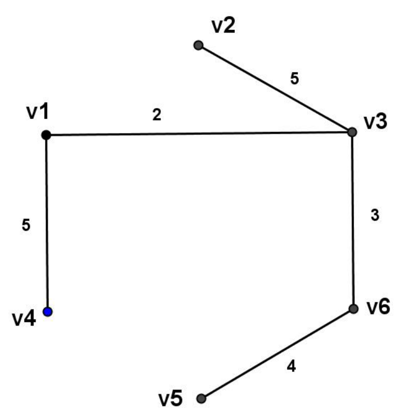

Fig. 2 Final Graph of the MST

Value of the final MST is:

$$\sum_{e \in E'} w(e) = (2+3+4+5+5) = 19. \tag{2}$$

10 Conclusion

This article describes the proposal of the algorithm for searching for the minimum spanning tree. The proposed algorithm is similar to the Prim´s algorithm (Harris et al., 2000), which creates the minimum spanning tree as gradually growing set of edges of the MST. In this regard there is a compliance with Prim's algorithm. The Prim's algorithm starts with the arbitrary vertice. Here, however, the first element the algorithm starts with is the edge with the lowest weight (Jackson and Read, 2010).

The new tool here is the "Weighted Adjacency Matrix (abr. WAM)". It follows the principle of adjacency matrix known known in the graph theory, but in the positions of matrix elements there are values of weights of edges connecting the vertices. The vertices denote the rows and columns of the matrix.

The whole process of searching MST begins with choosing the smallest element of the matrix, representing the edge with the lowest weight. Gradually we add elements so that another new element has one index same as some of the elements that have been chosen in previous steps. This step guarantees the continuity of MST.

Elements which in denoted rows and columns are not chosen, must be removed because these edges would create cycle. The entire process takes place in WAM, the original graph is not needed.

Benefits of the proposed algorithm is that the searching for MST by using WAM is efficient and fast. According to my knowledge the searching for the MST by using WAM is a new tool and it can be assumed that the WAM could be used for solving other similar problems in the graphs where weighted edges are given. Working of the algorithm is shown on solved example.

The proposed algorithm is suitable for optimizing the ship transport because the system of the ship traffic routes can be easibly transformed into the Weighted Adjacency Matrix which is clear representation of the graph with weighted edges. Solving the problem of searching for the minimum spanning tree goes in this matrix quickly and is illustratively presented in the solved example.

References

Chazelle, B. (2000). A minimum spanning tree algorithm with inverseackermann type complexity. *Journal of the ACM*. 47(6), pp. 1028-1047. https://doi.org/10.1145/355541.355562

Chong, K. W., Han, Y., Lam, T. W. (2001). Concurrent threads and optimal parallel minimum spanning trees algorithm. *Journal of the ACM*. 48(2), pp. 297-323. https://doi.org/10.1145/375827.375847

Cormen, T. H., Leiserson, C. E., Rivest, R. L, Stein, C. (2001). *Introduction to Algorithms*. 2nd Edition. MIT Press and McGraw-Hill. ISBN 0262033844 9780262033848

Fredman, M. L., Willard, D. E. (1993). Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*. 47(3), pp. 424–436. https://doi.org/10.1016/0022-0000(93)90040-4

Fredman, M. L., Willard, D. E. (1994). Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*. 48(3), pp. 533–551. https://doi.org/10.1016/S0022-0000(05)80064-9

Goldberg, A. W. (1987). Efficient Graph Algorithms for Sequential and Parallel Computers. PhD thesis, Department of Electrical Engineering and Computer Science, MIT.

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

Jackson, T. S., Read, N. (2010). Theory of minimum spanning trees. *Physical Review E*. 81, 021130. https://doi.org/10.1103/PhysRevE.81.021130

Kleinberg, J., Tardos, E. (2006). *Algorithm Design*. Pearson Education, Inc., New York.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*. 7, pp. 48–50. https://doi.org/10.1090/S0002-9939-1956-0078686-7

Maren, M. (2008). Graph Algorithms (The Saga of Minimum Spanning Trees). PhD thesis, Charles University, Prague, Czech Republic. [Online]. Available from: http://mj.ucw.cz/papers/saga/ Accessed: 1st April 2017]

Matousek J., Nesetril, J. (2009). *Kapitoly z diskrétní matematiky*. (The Chapters from discrete Mathematics.), 4th edition, Prague, Charles University in Prague. ISBN 978-80-246-1740-4 (in Czech).

Pettie, S., Ramachandran, V. (2002). An optimal minimum spanning tree algorithm. *Journal of the ACM*. 49(1), pp. 16-34. https://doi.org/10.1145/505241.505243

Prim, R. C. (1957). Shortest connection networks and some generalizations. *The Bell System Technical Journal*. 36(6), pp. 1389–1401. https://doi.org/10.1002/j.1538-7305.1957.tb01515.x