Decision Science Letters 7 (2018) 301–310

Contents lists available at GrowingScience

Decision Science Letters

homepage: www.GrowingScience.com/dsl

The degree constrained *k*-cardinality minimum spanning tree problem: a lexi-search algorithm

Thenepalle Jayanth Kumar[a] and Singamsetty Purusotham[a*]

[a]*Department of Mathematics, School of Advanced Sciences, VIT University, Vellore-632014, Tamil Nadu, India*

C H R O N I C L E

Article history:
Received January 16, 2017
Received in revised format:
May 22, 2017
Accepted July 23, 2017
Available online
July 25, 2017

Keywords:
k-cardinality minimum spanning tree
Lexi-search algorithm
Degree constraint

A B S T R A C T

This paper deals with the degree constrained *k*-cardinality minimum spanning tree (*k*-MSTPD) problem defined on a connected, edge weighted and undirected graph. The aim of this problem is to determine the least weighted spanning tree with exactly *k* vertices such that except the root vertex, no other vertex in the resultant spanning tree exceeds the specified degree limit. The *k*-MSTPD problem has many practical applications for the design of electric, communication, and transportation networks. The problem is then formulated as a zero-one programming. In this paper, an exact algorithm known as Lexi-search algorithm (LSA) is developed to tackle the *k*-MSTPD problem. Furthermore, the developed LSA is programmed in Matlab and tested on some benchmark instances as well as on random instances and the respective results are reported. The obtained experimental results showed that the developed LSA takes significantly less time to find the optimal solutions.

1. Introduction

The minimum spanning tree (MST) problem is a significant network combinatorial optimization problem. The MST problem is an acyclic subgraph, which connects all the vertices/nodes with the minimum overall associated weight to its edges. The *k*-cardinality minimum spanning tree problem (*k*-MST) is a variant of the classic MST problem and formally defined as follows: Let $G = (N, E)$ be a weighted, connected and undirected graph having $|N|$ vertices and $|E|$ edges, the *k*-MST problem determines the least weight spanning tree with exactly k $(k \le |N|)$ vertices and $k-1$ edges. The *k*-MST was first coined by Hamacher et al. (1991) and it was proven to be NP-hard by Ravi et al. (1996). It has several potential applications like oil-field leasing, telecommunications, open pit mining and image processing, (see Ravi et al., 1996; Garg & Hochbaum, 1997; Philpott & Wormald, 1997; Ma et al., 2000) thus it has received a great attention from the researchers in recent years.

* Corresponding author.
E-mail address: drpurusotham.or@gmail.com (S. Purusotham)

doi: 10.5267/j.dsl.2017.7.002

Since the *k*-MST is NP-hard, developing exact solution methods is again a challenging problem. Due to this reason, most of the existing works on *k*-MST problem concerns approximate solution procedures. However, these approximate solution methods may not assure the exact solutions. The constant factor approximation approach for solving the *k*-MST problem was first given by Blum et al. (1995). Subsequently, Garg and Hochbaum (1997) studied the *k*-MST problem and suggested an O(log k) approximation algorithm. A 2.5-factor heuristic algorithm for *k*-MST has been developed by Arya and Ramesh (1998). Three meta-heuristics algorithms namely Tabu Search (TS), Ant Colony Optimization (ACO) and Evolutionary Computation (EC) have been proposed by Blum and Blesa (2005) for the *k*-MST problem and showed that the ACO and TS approach works better for lower and higher cardinalities respectively. Recently, Katagiri et al. (2010) proposed a new solution procedure using Tabu search for *k*-MST and shown that this method performs better than the existing methods. An efficient hybrid approximation algorithm using Tabu search and Ant colony optimization for solving the *k*-MST problem was proposed by Katagiri et al. (2012). Purusotham and Sundara Murthy (2012) suggested the iterative techniques to solve some of the variants of network problems. A hybrid heuristic approach based on the Memetic algorithm and Tabu search algorithm was developed by Katagiri and Guo (2013) for the *k*-MST problem.

Degree constrained minimum spanning tree (DMST) problem is one of the variants of classic MST problem and has been proved to be NP-hard by Karp (1972). The DMST problem has many practical applications such as design of computer and communication networks, electric circuits, design of integrated circuits, road networks and energy networks (Martins & De souza, 2009). Several solution methods have been developed in order to tackle DMST problem, for few see, Genetic algorithm by Hanr and Wang (2006), an Ant based algorithm by Doan (2007), Learning based automata approach by Torkestani (2013), and Branch and cut algorithm by Martinez and Cunha (2014).

However, the above cited literature revealed that a limited consideration has been given to the variants of the *k*-MST problem due to NP-hard nature. Infact, many practical scenarios can be modeled as variants of *k*-MST problem. In this paper, a variant of the *k*-MST problem called the *degree constrained k-cardinality minimum spanning tree problem* (*k*-MSTPD) is considered and an exact solution procedure known as Lexi-search algorithm (LSA) is developed, which provides the optimal solutions effectively. Without loss of generality, relaxing the degree constraint and assuming $k = n,$ the *k*-MSTPD corresponds to classic MST problem. Our aim is to construct the model and develop an exact Lexi-search algorithm for the *k*-MSTPD.

The rest of the paper is structured as follows: The problem is described and then it is formulated as a zero-one integer programming in Section 2. Section 3 includes the concepts and solution procedure of the proposed LSA algorithm. A numerical example is given in Section 4 to demonstrate the proposed LSA. Section 5 presents the computational results and subsequently, the conclusions are discussed in Section 6.

2. Mathematical Model

Let $G = (N, E)$ be a connected, edge weighted and undirected graph, where N and E be the vertex/node set and edge set respectively. Each edge in E is associated with a non-negative weight known as cost. Given the specified positive integers k and D, the *k*-MSTPD aims to find the least cost spanning tree

with exactly k $(k \le |N|)$ vertices such that except the root vertex, no other vertex has more than the predefined degree D. The notations and parameters which are useful for formulating the model *k*-MSTPD are given as follows:

$|N| = n$: Number of vertices/nodes.

c_{ij} : Associated cost from i^{th} node to j^{th} node, where $c_{ij} = c_{ji}$.

k : Cardinality i.e. specified number of vertices, where $k \le n$

D : Specified degree limit on vertices.

α : Root vertex.

x_{ij} : Binary variable.

The mathematical model for *k*-MSTPD using zero one programming is given as follows:

$$\min Z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x_{ij} \tag{1}$$

subject to

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} = k-1 \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} \le D;\ i \ne j, i \ne \alpha,\ i \in N \tag{3}$$

$$+ \text{ Subtour elimination constraints} \tag{4}$$

$$x_{ij} \in \{0,1\};\ \forall i, j \in N \tag{5}$$

In this formulation, the constraint (1) represents the objective function that minimizes the overall cost of the spanning tree. The constraint (2) tells the resultant spanning tree possess *k-1* edges. The constraint (3) ensures that except the root vertex, no other vertex in the spanning tree is more than a predefined degree limit. The constraint (4) eliminates the subtours. Finally, the integrality constraints are given in (5).

3. Lexi-search Algorithm

The Lexi-search algorithm (LSA) is a well organized Branch and Bound (B&B) algorithm suggested by Pandit (1962). In principle, the LSA has similar features of the Branch and Bound (B&B) algorithm such as checking the feasibility, computing the bounds, generating a feasible solution. The LSA has been proved to be productive than the B&B approach and is based on the following grounds Pandit (1962):

"It is possible to list all the solutions in a structural hierarchy which also reflects a hierarchical ordering of the corresponding values of these configurations.
Effective bounds can be set to the values of the objective function, when structural combinatorial restraints are placed on the allowable configurations".

3.1. Pattern

A two dimensional array $X = [x_{ij}]$ that corresponds to the *k*- cardinality spanning tree is referred to as a pattern. The value of the pattern is evaluated by using the Eq. (6).

$$V(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij} \tag{6}$$

3.2.Alphabet Table

For any given two dimensional cost array $C = [c_{ij}]$, there exists $M = n \times n$ ordered pairs, which are organized in non-decreasing order and are indexed from *1* to M, stored in an array SN (Sundara Murthy, 1979). Let $L_p = (\beta_1, \beta_2, ..., \beta_p)$, $\beta_i \in SN$ be an arranged sequence of p indices from SN. The pattern indicated through the ordered pairs whose indices are represented by L_p, which is independent of the order of β_i in the sequence. The indices in the sequence L_p can be arranged in increasing order such that $\beta_i < \beta_{i+1}$, $i = 1, 2, ..., p-1$. The ordered sequence L_p is said to be a word of length p. The set SN is called an *Alphabet table.* A word L_p is called sensible word, if $\beta_i < \beta_{i+1}$, $i = 1, 2, ..., p-1$; in any other case, the word L_p becomes non-sensible word.

3.3. Word and Partial Word

If $p = k-1$, then L_p is said to be a full length word or simply word. If $p < k-1$, then L_p indicates partial word. Every full length word L_p gives a feasible solution.

3.4. Upper and Lower Word

The upper bound (*UB*) of a partial word is assumed to be a very higher value. The lower bound of a partial word ($LB(L_p)$) for the values of the blocks of words indicated by $L_p = (\beta_1, \beta_2, ..., \beta_p)$ can be determined as follows:

$LB(L_p) = V(L_p) + CC(\beta_p + k - 1 + p) - CC(L_p)$; where $V(L_p)$ provides the value of the partial word and is measured by using $V(L_p) = V(L_{p-1}) + C(\beta_p)$ with $V(L_0) = 0$.

3.5. Proposed Lexi-search Algorithm

The proposed Lexi-search algorithm (LSA) is mainly based on the features such as *alphabet table*, *setting effective bounds*, *checking the feasibility* and *backtracking*. The algorithmic steps of the proposed LSA for *k*-MSTPD problem are given as follows:

Step 1: ***Initialization***

Input: $C = [c_{ij}] \rightarrow$ Cost matrix; *n*→ number of vertices; *k*→ specified number;

$D \rightarrow$specified degree limit; *UB*→ an upper bound (99999, say)

Step 2: ***Alphabet Table***

Create the alphabet table using the given cost matrix C as discussed in the Section 3.2 and goto Step 3.

Step 3: ***Setting effective bounds***

Usually, the search starts with a partial word $L_p = (\beta_p) = 1,$ where $p = 1.$

a. Compute the lower bound of the partial word $(LB(L_p))$ as discussed in the Section 3.4
b. If $LB(L_p) < UB,$ then go to Step 4.

c. If $LB(L_p) \geq UB$, then eliminate the partial word L_p and subsequently remove the blocks of words with L_p as the leader. Go to Step 6.

Step 4: ***Feasibility checking***

a. If the partial word L_p satisfies the degree constraint, acyclic and balancing the size of the cardinality *k*, then it is feasible and infeasible, otherwise.

b. If the partial word L_p is feasible, then accept it and proceed for the subsequent partial word of order $p+1$ and go to Step 5. If L_p is infeasible, pick up the consecutive partial word of same order by considering the letter that succeeds β_p in its p^{th} position and go to Step 3.

Step 5: ***Concatenation***

a. If L_p is a partial word, then this will be concatenated by using $L_{p+1} = L_p * \beta_{p+1}$, where * refers the concatenation operation and go to Step 3.

b. If L_p is a full length word replace by $LB(L_p)$ and go to Step 7.

Step 6: If all the words of order p are exhausted and length of the word L_p is 1 then move to Step 9 and go to Step 7, otherwise.

Step 7: ***Backtracking***

Backtracking helps to improve the solution. The current value of UB is assumed as the upper bound and proceed the search by taking the next letter of the partial word of order $p-1$ and go to Step 3.

Step 8: Repeat the Step 3 to 7 and discard the feasible or infeasible solutions which are not necessary to obtain the optimal solutions. Continue this process until UB has no further improvement and go to Step 9.

Step 9: ***Output***

Record current UB and its corresponding full length word L_p and go to Step 10.

Step 10: ***Stop***

Once the search gets over, the current UB gives an optimal solution and the word L_p provides the complete schedule for *k*-MSTPD problem.

4. Numerical Illustration

In order to illustrate the proposed LSA, a numerical example is considered for which $|N| = n = 9$, $k = 7$ and $D = 3$. Since the problem defined on undirected graph, an upper triangular matrix is considered and the cost matrix $C = [c_{ij}]$ is represented in Table 1. Note that the components of the cost matrix C assume the non-negative integral values. For convenience, the partial alphabet table for the cost matrix in Table 1 is shown in Table 2. In Table 2, the arrays SN, C, CC, R and CL represents the serial number, elements of the cost matrix organized in ascending order, cumulative cost for the elements of the cost matrix, row and column indices respectively. The search table reported in Table 3 represents the details about how the proposed LSA computes the feasible solution and converges to the optimal solution. In Table 3, the column indexed by *SN* represents the serial number, the subsequent columns indexed by 1, 2, 3, 4, 5 and 6 indicates the first, second, third, fourth, fifth and sixth position of the letters in a partial word respectively. The next columns labeled by *v* and *LB* tells that value of the partial word and lower bound of a partial word respectively. The successive columns R and CL represent the row and column indices respectively. Finally, the last column *Remark* gives whether the partial word is accepted or rejected.

Since the problem considered in Table 1 aims to find the least cost spanning tree with exactly $k = 7$ (out of $n = 9$ vertices) vertices and $k = 6$ edges. Thus, the length of the feasible word could be 6 i.e. 6 edges. In Table 3, the LSA algorithm starts with a partial word $L_1 = 1,$ and it satisfies the feasibility conditions, thus the partial word is accepted and denoted by '*A*'. It is seen that some of the partial words failed in satisfying the feasibility conditions and hence rejected partial words are represented by '*R*'. Proceeding in this manner it lead to a full length feasible word $L_6 = (1, 2, 3, 4, 5, 11)$ which possess the objective value *UB*=52 (feasible solution) and it is observed at 11th row of the Table 3. The graphical representation of the feasible solution can be shown in Fig.1. Once the feasible solution is obtained, the current *UB* is assumed to be the upper bound and proceed for the backtracking in order to enhance the search space. It is observed that the improved solution *UB*=49 is observed at 14th row of the Table 3 and its corresponding full length word could be $L_6 = (1, 2, 3, 4, 6, 8)$. Since no other solution is found to be better than the current solution, the current found solution i.e. *UB*=49 could be the optimal solution and further, the algorithm terminates at 19th row of the Table 3. The graphical representation of optimal solution is given in Fig.2.

Table 1

Cost matrix of the k-MSTPD problem

	999	29	24	42	46	9	35	38	5
		999	22	28	1	45	65	12	31
			999	14	36	18	50	26	40
				999	32	15	21	39	20
C					999	10	25	39	33
						999	56	7	40
							999	55	30
								999	17
									999

Table 2

Alphabet Table

SN	C	CC	R	CL
1	1	1	2	5
2	5	6	1	9
3	7	13	6	8
4	9	22	1	6
5	10	32	5	6
6	12	44	2	8
7	14	58	3	4
8	15	73	4	6
9	17	90	8	9
10	18	108	3	6

SN	C	CC	R	CL
11	20	128	4	9
12	21	149	4	7
13	22	171	2	3
14	24	195	1	3
15	25	220	5	7
16	26	246	3	8
–	–	–	–	–
–	–	–	–	–
–	–	–	–	–
–	–	–	–	–

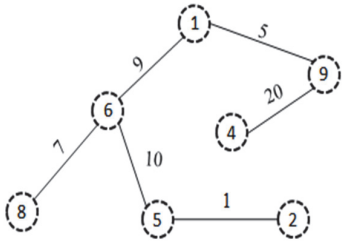

Fig.1. Feasible solution for the *k*-MSTPD problem given in Table 1

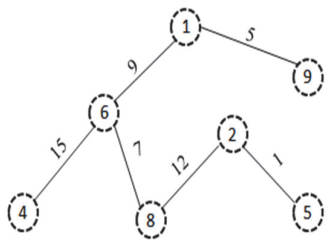

Fig.2. Optimal solution for the *k*-MSTPD problem given in Table 1

Table 3
Search Table

SN	1	2	3	4	5	6	V	LB	R	CL	Remark
1	1						1	44	2	5	A
2		2					6	44	1	9	A
3			3				13	44	6	8	A
4				4			22	44	1	6	A
5					5		32	44	5	6	A
6						6	44	44	2	8	R
7						7	46	46	3	4	R
8						8	47	47	4	6	R
9						9	49	49	8	9	R
10						10	50	50	3	6	R
11						11	52	52	4	9	A, UB=52
12					6		34	48	2	8	A
13						7	48	48	3	4	R
14						8	49	49	4	6	A, UB =49
15					7		36	51	3	4	> UB, R
16				5			23	49	5	6	> UB, R
17			4				15	51	1	6	> UB, R
18		3					8	53	6	8	> UB, R
19	2						5	57	1	9	> UB, R

5. Computational Results

This section reports the computational details of the proposed LSA for the *k*-MSTPD. The LSA was coded in Matlab 2016a and the experimental analysis was carried out on a 64-bit Microsoft Windows 2010 operating system with 4 GB of RAM PC and processor Intel Core i5 running at a 2.10 GHz. We evaluated the performance of the proposed LSA by varying distinct parameters in terms of a number of vertices, the cardinality of the tree, the structure of the cost matrix and degree limit on the vertices.

To assess the efficiency of the proposed LSA for the *k*-MSTPD, seven benchmark instances taken from TSPLIB which include *gr17*, *gr21*, *gr24*, *fri26*, *bayg29*, *dantzig42*, and *gr48*. The extensive computational details are reported in Table 4. From the results in Table 4, it is seen that for all the 23 cases of seven instances, the exact solutions were found by the LSA within considerable CPU runtime ranging from 0.0542 seconds to 12.8818 seconds. From these results, it is seen that the LSA is capable in finding the optimal solutions for all the cases within practically allowable runtimes. Note that for all the cases the root vertex is assumed as 1.

Furthermore, to measure the effectiveness of the proposed LSA, the computational experiments were carried out on a class of 10 randomly generated small and medium test instances ranging from 15 to 100. Each class included 10 instances and overall, a total of 100 test instances with symmetric edge weights are generated and tested. The cost c_{ij} assumes the randomly generated values over the range 1 to 500, i.e. $1 \le c_{ij} \le 500$. The experimental results summarized in Table 5 concerns the mean results of each size of the instance by the proposed LSA. Table 5 also contains the standard deviation (SD) of CPU runtimes. The results given in Table 5 based on five independent runs. In Table 5, the problems with 15 to 80 vertices with distinct cardinalities (*k*) could be solved in very less CPU runtimes. The CPU runtimes start rising when the problems of size 90 or higher are tested. However, these computational runtimes are fairly reasonable.

The overall results summarized in Table 4 and Table 5 reveal that proposed LSA could efficiently solve the *k*-MSTPD for the small and medium size of instances. For the problems of higher dimensions, the CPU runtimes may also be higher.

Table 4
Results of the LSA for k-MSTPD over benchmark instances

Instance	|N|	|E|	k	D	Best sol	CPU Runtime (In seconds)	
						Worst time	Best time
gr17	17	136	9	3	389	0.0887	0.0542
			10	3	442	0.0720	0.0598
			13	3	767	0.1919	0.0678
gr21	21	210	10	3	583	0.1240	0.0539
			13	3	919	0.1868	0.0830
			17	3	1396	0.2097	0.0852
gr24	24	276	14	3	451	0.2234	0.0902
			17	3	597	1.7253	0.5223
			18	4	645	1.9920	0.5425
			20	3	759	2.2302	0.6012
fri26	26	325	15	3	297	1.1742	0.4980
			18	3	380	2.1070	0.6054
			21	4	459	1.1024	0.4269
bayg29	29	406	17	4	670	0.7850	0.1932
			20	3	807	0.6880	0.1135
			24	3	1003	0.1942	0.0510
			26	4	1110	0.1002	0.0456
dantzig42	42	861	16	4	151	7.2432	4.5622
			20	4	196	8.0541	4.7854
			24	4	240	8.6366	5.7432
gr48	48	1128	32	4	2190	10.0720	8.1022
			36	4	2587	10.1472	9.0862
			40	4	3018	12.8818	9.0565

Instance − benchmark instances taken from TSPLIB; |N| − number of vertices in the given graph; |E|− number of edges in the given graph; k – number of vertices in the resultant spanning tree; D – degree limit on the vertices; Best sol. – the optimal solution by the proposed LSA; Worst time – worst computational runtime (in CPU seconds) for the best found solution; Best time – best computational runtime (in CPU seconds) for the best found solution.

Table 5
Mean results of the LSA for k-MSTPD over random instances

SN	|N|	|E|	k	D	NPT	CPU Runtime			SD
						Min.	Max.	Avg.	
1	15	105	10	3	10	0.0343	0.0502	0.04192	0.0049
2	20	190	14	3	10	0.0428	0.0480	0.0456	0.0014
3	30	435	20	4	10	0.0450	0.0586	0.0526	0.0043
4	30	435	25	4	10	0.0527	0.0635	0.0564	0.0038
5	40	780	30	4	10	0.0708	0.1089	0.0820	0.0132
6	50	1225	25	4	10	0.1017	0.1290	0.1100	0.0075
7	60	1770	40	4	10	0.1117	0.2139	0.1658	0.0336
8	80	3160	55	5	10	0.2818	0.4530	0.3763	0.0583
9	90	4005	60	6	10	0.8024	1.4125	1.1075	0.0465
10	100	4950	70	6	10	1.5432	6.0300	4.2353	0.0064

SN – problem number; |N| − number of vertices in the given graph; |E|− number of edges in the given graph; k – number of vertices in the resultant spanning tree; D – degree limit on the vertices; NPT– number of problems tested on each dimension; Min. – minimum CPU runtime (in seconds) to find best found solution for the ten runs; Max. – maximum CPU runtime (in seconds) to find best found solution for the ten runs; Avg. – mean CPU runtime (in seconds) to find best found solution for the ten runs; SD – standard deviation of CPU runtimes.

6. Conclusions

In this paper, a practical variant of MST problem reffered to as a degree constrained *k*-cardinality minimum spanning tree problem (*k*-MSTPD) has been taken into consideration and the model was then

modeled using zero-one programming. The *k*-MSTPD may have several practical applications, particularly in the areas like designing of networks, electric circuits, and transportation networks. We have proposed an exact solution procedure namely Lexi-search algorithm for *k*-MSTPD. Through numerical experiments for various benchmark and randomly generated instances, the performance of the developed LSA was measured. The experimental results reveal that the proposed LSA for the *k*-MSTPD is found to be productive to obtain the optimal solutions within considerable CPU runtimes.

References

Arya, S., & Ramesh, H. (1998). A 2.5-factor approximation algorithm for the k-MST problem. *Information Processing Letters*, *65*(3), 117–118.

Blum, A., Ravi, R., & Vempala, S. (1996, July). A constant-factor approximation algorithm for the k MST problem. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (pp. 442–448). ACM.

Blum, C., & Blesa, M. J. (2005). New metaheuristic approaches for the edge-weighted k- cardinality tree problem. *Computers & Operations Research*, *32*(6), 1355–1377.

Doan, M. N. (2007, September). An effective ant-based algorithm for the degree-constrained minimum spanning tree problem. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on* (pp. 485–491). IEEE.

Garg, N., Hochbaum, D. (1997). An O(log *k*) approximation algorithm for the *k* minimum spanning tree problem in the plane. *Algorithmica*, *18*(1), 111–121.

Hamacher, H.W., Jorsten, K., & Maffioli, F. (1991). Weighted *k*-cardinality trees. *Technical Report*, 91.023, Politecnico di Milano, Dipartimento di Elettronica, Italy.

Hanr, L., & Wang, Y. (2006). A novel genetic algorithm for degree-constrained minimum spanning tree problem. *International Journal of computer science and Network Security*, *6*(7A), 50–57.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85–103). Springer US.

Katagiri, H., Hayashida, T., Nishizaki, I., & Ishimatsu, J. (2010). An approximate solution method based on tabu search for k-minimum spanning tree problems. *International Journal of Knowledge Engineering and Soft Data Paradigms*, *2*(3), 263–274.

Katagiri, H., Hayashida, T., Nishizaki, I., & Guo, Q. (2012). A hybrid algorithm based on tabu search and ant colony optimization for k-minimum spanning tree problems. *Expert Systems with Applications*, *39*(5), 5681–5686.

Katagiri, H., & Guo, Q. (2013). A Hybrid-Heuristics Algorithm for k-Minimum Spanning Tree Problems. In *IAENG Transactions on Engineering Technologies* (pp. 167–180). Springer Netherlands.

Ma, B., Hero, A., Gorman, J., & Michel, O. (2000). Image registration with minimum spanning tree algorithm. In *Image Processing, 2000. Proceedings. 2000 International Conference on* (Vol. 1, pp. 481–484). IEEE.

Martinez, L. C., & Da Cunha, A. S. (2014). The min-degree constrained minimum spanning tree problem: Formulations and Branch-and-cut algorithm. *Discrete Applied Mathematics*, *164*, 210–224.

Martins, P., & de Souza, M. C. (2009). VNS and second order heuristics for the min-degree constrained minimum spanning tree problem. *Computers & Operations Research*, *36*(11), 2969–2982.

Pandit, S. N. (1962). The loading problem. *Operations Research*, *10*(5), 639–646.

Purusotham, S. & Sundara Murthy, M. (2012). A new approach for solving the network problems, OPSEARCH, 49(1), 1-21.

Philpott, A. B., & Wormald, N. C. (2000). *On the Optimal Extraction of Ore from an Open Cast Mine*. Department of Engineering Science, University of Auckland.

Ravi, R., Sundaram, R., Marathe, M. V., Rosenkrantz, D. J., & Ravi, S. S. (1996). Spanning trees—short or small. *SIAM Journal on Discrete Mathematics*, *9*(2), 178–200.

Sundara Murthy, M. (1979). Combinatorial programming problems - A pattern recognition approach. *Ph.D Thesis,* REC, Warangal, India.
Torkestani, J. A. (2013). Degree constrained minimum spanning tree problem: a learning automata approach. *The Journal of Supercomputing*, *64*(1), 226–249.
TSBLIB: https://github.com/pdrozdowski/TSPLib.Net/tree/master/TSPLIB95/tsp