An exact algorithm for *k*-cardinality degree constrained clustered minimum spanning tree problem

To cite this article: T Jayanth Kumar and S Purusotham 2017 *IOP Conf. Ser.: Mater. Sci. Eng.* **263** 042112

IOP Conf. Series: Materials Science and Engineering **263** (2017) 042112 doi:10.1088/1757-899X/263/4/042112

An exact algorithm for *k*-cardinality degree constrained clustered minimum spanning tree problem

T Jayanth Kumar and S Purusotham
Department of Mathematics, School of Advanced Sciences, VIT University, Vellore-632014, Tamil Nadu, India

E-mail: drpurusotham.or@gmail.com

Abstract. The *k*-cardinality degree constrained clustered minimum spanning tree problem (*k*-DCCMST) aims to determine a *k*-node (out of *n* nodes) spanning tree of minimum weight defined on a complete weighted undirected graph, where the node set is partitioned into set of clusters such that except the root node, the degree of other nodes in the resultant spanning tree does not exceed the predefined degree limit. The *k*-DCCMST model has significant applications in the context of designing of networks and is then formulated as a zero-one integer linear program. To solve this problem optimally, an exact Lexi-search algorithm (LSA) is developed. The developed LSA is subjected in Matlab, tested on some benchmark as well as randomly generated test instances and computational results are reported. Numerical experimental results demonstrate the efficiency of proposed LSA on dense graphs.

1. Introduction

The minimum spanning tree problem (MST) is an acyclic sub graph, which connects all the vertices with minimum total weight for its edges. The classical MST is one of the significant fundamental problems in network combinatorial optimization and it has wide variety of practical applications in designing of networks including telecommunications, computer networks, transportation, water supply networks, drain systems, and power grids [1]. Due to its wide applicability, it has gained much attention over the past few decades and many efficient approaches have been emerged. The polynomial time greedy algorithms were developed by [2-3]. The classical MST has been diversified into many variants which were proved to be NP-hard [4].

Degree constrained minimum spanning tree problem (DCMST) was introduced by [5] and is one of the problem that received a great consideration in the sense of solution procedures.An application of this problem can be observed in the context of design of electrical grids, in which all the terminal vertices are connected using least amount of wire, such that no vertex is incident to other vertices with more than the given number of wires [5]. More recently, a learning based automata heuristic approach [6] and a branch and cut algorithm [7]has been developed for solving DCMST. In addition to that, some of the well-studied variants of classical MST such as Hop-constrained minimum spanning tree problem [8], Leaf-constrained minimum spanning tree problem [9], Minimum diameter spanning tree [10]. Clustering is the process of partitioning the vertex set into set of clusters. The classical MST has been played a significant role and become a powerful tool for performing clustering analysis [11]. Several clustering algorithms based minimum spanning tree can be found in the literature [11-14].

The *k*-cardinality minimum spanning tree problem (*k*-MST) is a variant of the classic MST, first introduced by[15] and was proven to be NP-hard [16]. The *k*-MST is formally defined as follows: Let

$G=(N,E)$ be a weighted, connected and undirected graph having $|N|$ vertices and $|E|$ edges, the *k*-MST aims to find the minimum weight spanning tree with exactly $k\,(k \le |N|)$ vertices and $k-1$ edges. Since the *k*-MST has potential applications in oil-field leasing, telecommunications, open pit mining and image processing [16-18], it has received a great attention in the research community. The constant factor approximation approach for solving the *k*-MST was first presented by[19]. Subsequently, *k*-MST problem has been studied by [17] and suggested an *O(log k)* approximation algorithm. A 2.5-factor heuristic algorithm [20] has been developed for solving *k*-MST. Three meta-heuristics algorithms namely Tabu Search (TS), Ant Colony Optimization (ACO) and Evolutionary Computation (EC) have been proposed [21]for*k*-MST problem and showed that the ACO and TS approach works better for lower and higher cardinalities respectively.

Recently, a new solution procedure using Tabu search [22] proposed 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 [23] for solving the *k*-MST problem has been developed. A hybrid heuristic approach based on the memetic algorithm and Tabu search algorithm [24] has been developed for the *k*-MST problem. Obviously, most of the above cited works on *k*-MST problem concerns approximate solution procedures. However, these approximate solution methods may not assure the exact solutions and developing exact solution methods is again a challenging problem.

This paper considers an extension to the *k*-MST namely the *k-cardinality degree constrained clustered minimum spanning tree problem* (*k*-DCCMST). We develop an exact Lexi-search algorithm (LSA) for solving the *k*-DCCMST model optimally. The remainder of the paper is structured as follows. Section 2 provides the problem description and its mathematical formulation as well. The preliminaries and developed algorithm are presented in Section 3. Computational results are reported in Section 4. Finally, conclusions are drawn in Section 5.

2. Mathematical formulation

The *k*-DCCMST is defined on the complete, undirected, connected and weighted graph $G=(V,A)$, where $V=\{1,2,3,...,n\}$ be the vertex/node set and $A=\{(i,j):i,j\in V, i\neq j\}$ be the arc/edge set. With each edge of A is associated with non-negative symmetric cost/distance c_{ij} is equal to cost from i^{th} node to j^{th} node. The node set V is partitioned into set of m clusters which must be satisfied $V=\bigcup_{p=0}^{m} V_p$, $V_i \cap V_j=\varnothing$, where $i,j=0,1,...,m;\ i\neq j$. Given the non-negative integers k,d and size of the clusters, the *k*-cardinality degree constrained clustered minimum spanning tree (*k*-DCCMST) aims to find a $k\ (k\le|V|)$ node spanning tree of optimal cost such that except the root node, no other vertex in the resultant spanning tree has more than the degree limit d and subject to the cluster constraints. The*k*-DCCMST is formulated as a zero-one integer linear program. The notations which are used to model the problem are given as follows:

Notation	
V	set of all nodes i.e. $V=\{1,2,3,...,n\}$
$\|V\|=n$	number of nodes
P	Set of clusters, $P=\{0,1,,...,m\}$
m	number of clusters
d	degree threshold
c_{ij}	associated symmetric distance/cost between i^{th} node to j^{th} node, $i,j\in N$
V_p	node set of p^{th} cluster
V_0	root node cluster; its cardinality being one i.e. $\|V_0\|=1$

α	root node, where $\alpha \in V_0$
k	cardinality on vertices
x_{ij}	a binary variable, where $x_{ij} \in \{0,1\}$

The zero-one linear integer programming model is given below

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

Subject to

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

$$\sum_{j=1}^{n} x_{ij} \le d;\ i \ne j,\ i \in N \setminus \{V_0\} \tag{3}$$

$$\sum_{j \in \cup V_p} x_{\alpha j} = |P|,\ \alpha \in V_0,\ p \in P \tag{4}$$

$$\sum_{j \in V_p} x_{\alpha j} = 1,\ \alpha \in V_0,\ p \in P \tag{5}$$

$$\text{If } x_{ij} = 1 \text{ and } i, j \notin V_0 \text{ then } i, j \in V_p, \text{where } p \in P \setminus \{0\} \tag{6}$$

$$+ \text{Sub tour elimination constraints} \tag{7}$$

$$x_{ij} \in \{0,1\},\ \forall i, j \in V \text{ and } i \ne j \tag{8}$$

The objective function is given in constraint (1) denotes the sum of the total associated cost of the spanning tree. The constraint (2) implies that resultant spanning tree must have $k-1$edges. The constraint (3) guarantees that except the root node, no other node in the resultant spanning tree is more than a predefined degree limit. The constraints (4-5) ensure that satisfies the clustering requirements. The constraint (6) imposes, if there is an edge from i^{th} node to j^{th} node, where $i, j \notin V_0$ then both the nodes should be in the specific cluster. The constraint (7) eliminates sub tours. Finally, the constraint (8) represents the binary variable.

3. Lexi-search algorithm

The word Lexicographic is observed from Lexicography, the science of effective storage and recovery of data. The name Lexicographic itself propose that the search for an optimal solution brought about in a systematized manner.A systematized branch and bound approach [25] called Lexi-search algorithm was developed to tackle the loading problem. The approach is depending on following grounds [25]

a. *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.*
b. *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 symbolic representation of two dimensional array $X = [x_{ij}]$ that corresponds to the spanning tree is called a pattern. If X has a feasible solution, then the pattern is said to be feasible. The value of the pattern is measured using the formula given in (9).

$$Vc(X)=\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}\,x_{ij} \tag{9}$$

The value $Vc(X)$ gives the total associated cost of the spanning tree expressed by X .

3.2. Alphabet table

According to [26], for any two dimensional array, there exist $M=n\times n$ ordered pairs, which are organized in increasing order corresponding of their costs and indexed from 1 to M . The set SN constitutes M indices. Let $L_r=(\beta_1,\beta_2,...,\beta_r),\beta_i\in SN$ be an ordered sequence of r indices from SN . The pattern represented by the ordered pairs whose indices are indicated by L_r , which is independent of the order of β_i in the sequence. The indices in L_r can be arranged in non-decreasing order such that $\beta_i<\beta_{i+1};i=1,2,...,r-1$. If a word L_r satisfies all the constraints and if $r<k-1$, then it is said to be the partial feasible word.If $r=k-1$, then the word L_r is said to be full length feasible word and it provides the feasible solution. If the block of words involved at least one feasible word then the leader L_r is said to be feasible.

3.3. Bound settings

Initially, the upper bound (UB) of a partial word can be taken as infinity as a trial solution. The lower bound (LB) for the values of the blocks of words denoted by L_r can be determined by using the formula $LB(L_r)=Vc(L_r)+Cd(\beta_r+k-1-r)-Cd(L_r)$, where $Vc(L_r)$ gives value of the partial word and is determine using $Vc(L_r)=Vc(L_{r-1})+c(\beta_r)$ with $Vc(L_0)=0$.

3.4. Proposed Lexi-search algorithm

The steps involved in Lexi-search algorithm towards the optimal solution are given as follows:

Step1 **Initialization**

a. Cost matrix (C)
b. Number of nodes (n) and number of clusters (m)
c. Cardinality on vertices (k)
d. Degree limit (d), size of different clusters (q)
e. Set the upper bound $UB=VT=9999$ as a trial value.

Step 2 **Alphabet table**

Generate the alphabet table by sorting the elements of two dimensional cost matrix $C=[c_{ij}]$ as discussed in the Section (3.2) and go to Step 3.

Step 3 **Bound settings**

a. Initially, the algorithm starts with a partial word $L_r=(\beta_r)=1$ where r = 1.
b. Compute the lower bound (LB) of a partial word L_r as discussed in Section (3.3).
c. If $LB(L_R)<VT$ then go to Step 4.
d. If $LB(L_R)\geq VT$, then discard the partial word L_r and reject the block of word with L_r as leader, since it does not provide optimal solution and thus, dismiss all the partial words of the order r that succeeds L_r and go to Step 6.

Step 4 **Feasible checking**

a. If the partial word L_r satisfying the cardinality constraint, degree constraint, acyclic and balancing the size of the clusters, then it is feasible otherwise infeasible.
b. If L_r is feasible, then we accept it and continue for next partial word of order $r+1$ and go to Step 5.

c. If L_r is infeasible, consider the next partial word of order by taking another letter that succeeds β_r in its r^{th} position and go to Step 3.

Step 5 **Concatenation**

a. If L_r is a full length feasible word, then $VT = LB(L_r)$ and for further improvement go to Step 7.

b. If L_r is a partial word, then this can be concatenated by using $L_{r+1} = L_r * (\beta_{r+1})$ where * represents the concatenation operation and go to Step 3.

Step 6 If all the words of order r are exhausted and length of the word L_r is 1, then go to Step 9, otherwise, go toStep 7.

Step 7 **Backtracking**

a. To explore the solution space, backtracking is performed; the current VT is taken as an upper bound and continues the search with the next letter of the partial word of order $r-1$ and go to Step 3.

b. On repeating the Steps 3 to 7 and eliminates the feasible/infeasible solutions which are not included in the optimal solution.

c. Continue the process, until VT has no further improvement and go to Step 8.

Step 8 **Output**

Record VT and L_r, go to Step 9.

Step 9 **Stop**

At the end of the search, *VT* provides an optimal solution and the word L_r gives the complete schedule for minimum connectivity of the clustered nodes.

4. Computational analysis

The main objective of this section is to measure the efficiency of the developed LSA for *k*-DCCMST. The developed LSA was implemented in Matlab 2016a and is tested on a PC with 2.0 GHz processor Intel Core i3 and 4 GB of RAM running Microsoft Windows 2010 operating system. In order to assess the LSA performance, the LSA experimented over benchmark as well as randomly generated data instances. All the computational experiments were carried out on dense graphs (i.e. the maximum number of possible edges for the dense graph is $\frac{1}{2}n(n-1)$).

The exact algorithm for *k*-DCCMST was tested over six benchmark data instances from TSPLIB [27] by assuming vertex 1 as the root node. The benchmark instances include *gr17*, *gr21*, *gr24*, *fri26*, *bayg29*, and*dantzig42*. For each instance, we have considered four cases with distinct values of cardinality on vertices (*k*), degree threshold (*d*), number of clusters (*m*) and size of the clusters (*q*) and each experiment was executed independently 10 runs. Overall, for 6 benchmark data instances, 24 cases were tested. The obtained results include best found solution using LSA, worst time as well as best time to obtain the best solution are summarized in Table 1. From the results in Table 1, it is observed that the LSA produces the best solutions in practically considerable CPU run times. It is interesting to note that there are inconsistent CPU run times due to the dependence of the LSA in the structure of the data instance.

Table 1. Results of the LSA for *k*-DCCMST over benchmark instances.

Instance	|*V*|	|*A*|	*k*	*d*	*m*	*q*	Best sol.	CPU Runtime	
								Worst time	Best time
gr17	17	136	12	3	2	(7, 4)	788	0.1957	0.0848
			11	3	2	(6, 4)	667	1.1708	0.1998
			12	3	3	(5, 3, 3)	890	0.2902	0.0878
			10	3	2	(5, 4)	534	0.4208	0.0939
gr21	21	210	16	3	3	(6, 5, 4)	1507	1.6868	0.1380
			14	3	3	(6, 4, 3)	1222	0.9207	0.3852
			11	3	2	(6, 4)	755	3.0234	1.2902
			18	3	3	(8, 5, 4)	1810	1.7523	0.9223
gr24	24	276	14	4	2	(8, 5)	559	1.9220	0.8825
			16	3	4	(5, 4, 3,3)	874	2.2050	1.0712
			14	3	3	(6, 4, 3)	641	3.7142	1.0980
			20	3	2	(12, 7)	907	6.1070	2.0654
fri26	26	325	13	3	3	(5, 4, 3)	494	1.1024	0.4269
			16	3	3	(7, 5, 3)	587	0.7850	0.1932
			18	3	3	(8, 6, 3)	640	1.6880	0.4135
			20	4	3	(8, 6, 5)	725	14.1942	12.0510
bayg29	29	406	20	4	4	(8, 5, 3, 3)	902	8.1002	3.4560
			23	4	4	(8, 6, 4, 4)	1097	7.2432	4.5622
			18	3	4	(6, 4, 4, 3)	826	8.0541	4.7854
			20	4	3	(10, 5, 4)	868	8.6366	5.9432
dantzig42	42	861	16	4	3	(6, 5, 4)	231	4.5462	3.0432
			20	4	3	(8, 7, 4)	277	15.0720	12.1022
			24	4	3	(10, 8, 5)	327	25.2280	19.0762
			24	3	3	(8, 8, 7)	343	44.8234	37.0986

Instance − benchmark instances taken from TSPLIB [27]; |*V*| − cardinality of the node set in the given graph; |*A*|− cardinality of the edge set in the given graph; *k* – cardinality of the resultant spanning tree; *d*– degree limit on the vertices;*m*– number of clusters; *q*– size of the clusters(sum of the sizes of all clusters and root node equals to *k*); Best sol. – the optimal solution by the proposed LSA; Worst time – worst computational runtime (in CPU seconds) required to find the best solution; Best time – best computational runtime (in CPU seconds) required to find the best solution.

Additionally, to measure the LSA performance, we extend the computational experiments by testing the LSA on randomly generated test instances. A class of 10 randomly generated small and medium size test instances ranging from 20 to 110 was considered for our computational experiments. 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} takes the random values over the range [1, 300].

The numerical experimental results reported in Table 2 show the mean results of each size using LSA. Table 2 also includes the standard deviation (SD) of CPU run times. In Table 2, the average CPU run times for the problems of distinct sizes from 20 to 110 with distinct parametric values of *k*, *d*, *m*, and *q* ranges from 0.4920 seconds to 48.9804 seconds. It is observed that average CPU run times start increasing when the problems of size 60 or higher are tested. However, these computational runtimes are fairly reasonable in the sense of finding exact solutions.

The overall results summarized in Table 1 and Table 2 shows that LSA can efficiently solve the *k*-DCCMST for the small and medium size of instances within practically allowable CPU run times. For higher dimensional instances, the CPU run times may also be higher.

Table 2. Mean results of the LSA for *k*-DCCMST over random instances.

SN	|*V*|	|*A*|	*k*	*D*	*m*	*q*	NPT	CPU Runtime			SD
								Min.	Max.	Avg.	
1	20	190	15	3	2	(8, 6)	10	0.3416	0.5012	0.4920	0.0196
2	30	435	24	3	3	(10, 8, 5)	10	0.4510	0.5864	0.5266	0.0343
3	40	780	35	4	3	(15, 10, 9)	10	0.7008	0.9089	0.8920	0.0232
4	50	1225	40	4	3	(15, 15, 9)	10	0.1017	0.1290	0.1100	0.0075
5	60	1770	45	4	4	(15, 10, 10, 9)	10	3.1227	5.2169	4.1658	0.0336
6	70	2415	55	4	4	(18, 15, 13, 8)	10	4.0042	7.2644	6.5224	0.0750
7	80	3160	60	5	5	(18, 17, 10, 10, 4)	10	8.2102	10.4621	9.4234	0.0620
8	90	4005	70	6	5	(25, 15, 12, 10, 8)	10	14.0542	22.4468	16.7524	0.1556
9	100	4950	90	6	5	(30, 25, 20, 10, 4)	10	18.1790	31.0098	25.9806	0.0942
10	110	5995	90	6	5	(25, 25, 15, 14, 10)	10	33.0892	56.3402	48.9804	0.1964

SN – problem number; |*V*| − cardinality of the node set in the given graph; |*A*|− cardinality of the edge set in the given graph; *k* – number of vertices in the resultant spanning tree; *d*– degree threshold on the nodes;*m*– number of clusters; *q*– size of the clusters(sum of the sizes of all clusters and root node equals to *k*);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 a best found solution for the ten runs; Avg. – mean CPU runtime (in seconds) to find a best found solution for the ten runs; SD – standard deviation of CPU run times.

5. Conclusion

In this study, we investigated an NP-hard *k*-DCCMST model and presented as a zero-one integer linear program. To find exact solutions for *k*-DCCMST, an efficient exact Lexi-search algorithm (LSA) was developed. The LSA performance has been shown through numerical experiments for some benchmark test instances. Furthermore, the mean computational results obtained over the randomly generated test instances with the LSA are quite promising with practically considerable CPU run times. Although LSA is time consuming as the problem dimension increases, the quality of the solutions produced by LSA is consistent. However, the LSA could be made more capable by enforcing data guided strategy, as well as effective bound settings, would remain as future considerations.

References

[1] Krishnamoorthy M, Ernst AT and Sharaiha YM 2001 *J. Heuristics* **7** 587– 611
[2] Kruskal JB 1956 *Proceedings of the American Mathematical society***7**48–50
[3] Prim RC 1957 *Bell Labs Techn. J.***36**1389–1401
[4] Karp RM 1972 In *Complexity of computer computations* pp. 85-103 Springer US
[5] Narula SC and Ho CA 1980 *Comput. Oper. Res.***7** 239–249
[6] Torkestani JA 2013 *J Supercomput.***64** 226–249
[7] Martinez LC and Da Cunha AS 2014 *Discrete Appl. Math.***164** 210–224
[8] Gouveia L, Paias A and Sharma D 2011 *J. Heuristics***17** 23–37
[9] Singh A 2009 *Appl. Soft Comput.***9** 625–631
[10] Hassin R and Tamir A 1995 *Inform. Process. Lett.***53** 109–111
[11] Wang GW, Zhang CX and Zhuang J 2014 *Appl. Math. Comput.***247** 521–534
[12] Paivinen N 2005 *Pattern Recognit. Lett.***26** 921–930
[13] Zhong C, Miao D and Wang R 2010 *Pattern Recognit.***43** 752–766
[14] Wang GW, Zhang CX, Zhuang J and Yu DH 2011 In *Wavelet Analysis and Pattern Recognition (ICWAPR)* pp. 132–137 IEEE
[15] Hamacher HW, Jorsten K and Maffioli F 1991 *Technical Report* 91.023 Politecnico di Milano, Dipartimento di Elettronica, Italy.
[16] Ravi R, Sundaram R, Marathe MV, Rosenkrantz DJ and Ravi SS 1996 *SIAM J. Discrete Math.***9** 178–200
[17] Garg N and Hochbaum D 1997 *Algorithmica**18*** 111–121

[18] Ma B, Hero A, Gorman J and Michel O 2000 In *Image Processing*, Vol.**1** pp. 481-484 IEEE
[19] Blum A, Ravi R and Vempala S 1996 In *Proc. of the twenty-eighth annual ACM Symp. on Theory of computing* pp. 442-448 ACM
[20] Arya S and Ramesh H 1998 *Inform. Process. Lett.***65** 117–118
[21] Blum C and Blesa MJ 2005 *Comput. Oper. Res.***32** 1355–1377
[22] Katagiri H, Hayashida T, Nishizaki I and Ishimatsu J 2010 *Int. J. Knowl. Eng. Soft Data Paradig.***2** 263–274
[23] Katagiri H, Hayashida T, Nishizaki I and Guo Q 2012 *Expert Syst. Appl.***39** 5681–5686
[24] Katagiri H and Guo Q 2013 In *IAENG Transactions on Engineering Technologies* pp. 167-180 Springer Netherlands
[25] Pandit SNN 1962 *Oper. Res.***10** 639–646
[26] Sundaramurthy M 1979*Ph.D Dissertation* REC Warangal India
[27] TSPLIB:http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp