Genetic Programming Based Degree Constrained Spanning Tree Extraction

Zaheer Babar, Muhammad Waqas, Zahid Halim
GIK Institute, Topi, Pakistan.
{gcs1225, gcs1224, zahid.halim}@giki.edu.pk

Muhammad Arshad Islam
Muhammad Ali Jinnah University, Islamabad.
arshad.islam@jinnah.edu.pk

***Abstract*—The problem of extracting a degree constraint spanning tree deals with extraction of a spanning tree from a given graph. Where, the degree of each node is greater than or equal to a thread hold value. Genetic Programming is an evolution based strategy appropriate for optimization problems. In this paper we propose a genetic programming based solution to the degree constraint spanning tree extraction. The individuals of the population are represented as a tree and mutation is applied as the only genetic operator for evaluation to occur. We have further tested our proposed solution on different graph and found it to be suitable for degree constraint spanning tree extraction problem**

***Keywords*—Degree Constraint Spanning Tree; Spanning Tree; Minimum Spanning Tree; Cyclic Interchange.**

I. INTRODUCTION

Minimum Spanning Tree(MST) of a graph can be used to solve various problems related to graph theory[1].There are several known solution to compute minimum spanning tree (MST) for a given graph such as Prim's algorithm [2] or Kruskal algorithm [3]. However, computing a degree constrained spanning tree has been focal problem for researchers for some time now [1].Moreover, finding degree constraint spanning tree (DCST) is known to be an NP-problem [2]. A DCST is utilized in scenarios where we have to find a solution to a spanning tree problem that satisfies certain conditions.

One way of approaching such problems is to create a set or sample space of all possible spanning trees of a given graph and then a subset is created consisting of only those spanning trees that satisfy the conditions posed due to the nature of the problem. We may consider the members of the subset as possible solutions for a given graph and select those solutions that are better than the rest of the set. Genetic algorithms are traditionally used where we have a large sample space of possible solution and, an appropriate fitness function is created to select the optimum solution for the given problem. In the context of finding spanning trees in a graph, a good number of random candidate spanning trees are generated and then fitness of all the candidates is computed. In addition to other constraints the fitness function may represent a condition such as degree of every node in the given tree must be greater than any integer *d.*

In this paper, we use genetic programming to improve K number of DCST, with degree d, to an optimal form while maintaining their degree bound and parent graph edge node structure invariance. We have also applied this technique with a population of simple spanning trees without degree bound. The obtained results are analyzed with respect to success rate of the evolved solutions. Initial population is created by first extracting a random tree from a given graph with degree bound *d*, later a population of size *k* is initialized by constructing several trees through application of cyclic interchange on the aforementioned random tree. Mutation is the only genetic operator used to generate new trees during evolutionary process.

Remaining paper is organized as follow: Section 2 covers literature survey, Section 3 introduces the proposed technique, Section 4 list experiments and results and Section5 concludes the paper.

II. LITERATURE REVIEW

Extracting a degree constrained minimum spanning tree (DCMST) from a graph has been a problem of interest due to its application in various fields like telecommunication networks and integrated circuits. [4]. This section provides a brief survey of the existing solution while highlighting their strengths and weaknesses.

Markus Behly et al. [5] have proposed the primal branch-and-cut algorithm. They also mention difference between two techniques (a) standard separation problem and(b) primal separation problem. Gunther R. Raidl et al. [6] obtained a DCMST of a graph using an efficient evolutionary algorithm that represent the main candidate tree by the set of edges. New valid DCMST are generated from the existing set of candidate solutions by using the crossover and mutation operators. The initialization of the candidate solutions take O(|V|) computational time and then new generation is formed in O (|V|) time. One of the drawback of this approach is that variance of the graph cannot be maintained.

978-1-4799-0615-4/13/$31.00 ©2013 IEEE

Approach in [6] may affect the graph invariance our proposed technique caters for it, in term of edge-node structure, by using only mutation as a genetic operator. R. Ravi et al. [7] presented the partition algorithm by using Steiner tree. Proposed algorithm in [7] is more appropriate for network design problem. Kenneth Sörensen et.al. [11] has proposed an algorithm to generate the all spanning trees in the graph. Starting from a minimum spanning tree, this algorithm generates spanning trees in order of increasing cost. The space complexity of this algorithm is O (N x |V|) and the time complexity of this algorithm is O (|E| log |E|). Mohit Singh et.al. [13] proposed an algorithm for the minimum bounded degree spanning tree (MBDST). They have extended the iterative rounding framework to obtain the best possible guarantee for the MBDST problem. However, it cannot guarantee, that the average cost of the newly formed spanning tree will improved.

Marco Dorigo et al.[8]have used ant based algorithm for solving the travelling salesman problem (TSP). They have utilized the behavior of ants, for finding the shortest path, using the pheromone level. Every ant visits each city exactly once to comply with conditions of TSP problem statement. Their algorithm works better compared to other ant based algorithms for the symmetric and asymmetric TSP's. Son Hong Ngo et. al.[9] have proposed the ant based algorithm for the dynamic routing and wavelength assignment (RWA) in optical WDM (wavelength Division Multiplexing) network under the wavelength continuity constraint. In their algorithm, the agent (ant) automatically updates the routing table according to the network state. They utilized the foraging technique used by the real ants. An ant selects the next path stochastically in terms of path length and the degree of congestion along this path. In their approach an ant would die if it reaches its destination, or it cannot find the path or if it exceeds the specified lifetime. The approach proposed by Priya Vaijayanthi et. al. [10] combines the ant colony optimization with Tabu search for the document clustering problem. They have compared their result with K-means algorithm which shows that this algorithm is better. Thang N. Bui et. al. [15] proposed an ant based algorithm for the DCMST, consisting of the 3 main phases: initialization phase, exploration phase and construction phase. All of the aforementioned approached generating DCST or DCMST except in [6] which shows algorithm for improvement of spanning trees. The approach in [6] can have an effect on graph invariance due to edge crossover. Our presented approach is different from this in term initial population generation through cyclic interchange and using mutation as reproduction operator to protect graph invariance. Some related work on MST can be seen in [17-18]

III. PROPOSED APPROACH

The proposed approach has two phases. In the first phase, sample population is generated by extracting simple MST independent of any degree conditions. In the second phase, we apply the proposed mechanism of selecting and refining only those MSTs that satisfy the constraints of our problem. Following sub-sections describe the both phases in detail. Phase-II is the main proposed solution and contribution of this work to DCMST problem.

A. Phase-I

This phase considers a population of spanning trees without any degree bound on nodes. Following steps are executed during phase-I on the given graph.

- Extract a random spanning tree from input graph. Extraction can be done by breadth first search or Prims algorithm.
- A finite population of spanning trees is produced (to apply Genetic Programming) using cyclic interchange over the initial spanning tree.
- The evolution process using GP is applied on the population with the mutation as the genetic operator. Mutation is applied on a set of branch edges B of spanning tree T against the set of chord edges C regarding T, which is generated as

$$Edges(C) \leftarrow Edges(G) - Edges(T) \quad (1)$$

1) Initial Spanning Tree Extraction: We start with input graph G in the form of adjacency matrix. A spanning tree can be created by using either Prim's algorithm or Breadth First Search (BFS). Prim's algorithm will provide us an initial Tree as MST and then we could apply further operations on it. Whereas, BFS will provide random spanning tree, as BFS traverses whole the graph in form of a tree. Though choosing either Prim's algorithm or BFS will have not alter the working our method, it may be convenient to monitor population fitness convergence for the spanning trees provided by BFS because evolutionary process will start with fittest tree in its initial population. In our experiments, we use both Prim's algorithm and BFS in an alternate manner, and we obtain an arbitrary spanning tree $Temp_T$ using. The pseudo code of algorithm is listed in fig-1.Time complexity for generating the initial tree will be same as that of Prim's algorithm or BFS i.e. $O(|V|^2)$ or $O(|V|+|E|)$ respectively.

2) Population Generation: We apply cyclic interchange on the $Temp_T$ by replacing one member of the branch set with a member of the chord set. Branch set contains all the edges that belong to

Main Algorithm

$DCST - GP(G)$ // **for DCST** $\underline{DCST - GP(G,d)}$

Input: G= (V, E) a connected Graph in the form of adjacency matrix. All of the edges are having positive values. For DCST d is degree bound on each node of Tree having value d>1

Output: After applying genetic programming an optimum cost population with minimum average cost

Begin

1. Initialize an Object type array T for storing multiple spanning trees
   a. $T = [T_0, T_1, T_2, \dots, T_n]$
2. Initialize and integer i with 0
   a. $i \leftarrow 0$
3. $T_{Temp} \leftarrow ConstructInitialSpanningTree(G)$
4. $T_i \leftarrow T_{Temp}$
5. If (nullity(T_{Temp})>0)
   { Construct Population (G, t, i)//for DCST (G,T,d,i)
   Evol (G,T) }// for DCST Evol(G,T,d)
   Else Display ("Incompatible")

End

Fig.1. Main Algorithm

the initial spanning tree while the rest of the edges belong to the chord set. If we replace a member of branch set member with chord set member in such a way that all there is not partition in the graph, we obtain a new spanning tree. Details of this process (also known as cyclic interchange) are presented below.

Main Algorithm Phase II

$\underline{DCST - GP(G,d)}$

Input: G= (V, E, d) a connected Graph in the form of adjacency matrix. All of the edges are having positive values. For DCST d is degree bound on each node of Tree having value d>1

Output: After applying genetic programming an optimum cost population with minimum average cost

Begin

1. Initialize an Object type array T for storing multiple spanning trees
   $T = [T_0, T_1, T_2, \dots, T_n]$
2. Initialize and integer i with 0; $i \leftarrow 0$
3. $T_{Temp} \leftarrow ConstructInitialSpanningTree(G,d)$
4. $T_i \leftarrow T_{Temp}$
5. If (nullity(T_{Temp})>0)

{Construct Population (G,T,d,i)
Evol(G,T,d)
Else Display ("Incompatible")

End

Fig. 2. Main Algorithm Phase II

For cyclic interchange, we denote B as the set of branch edges in initial tree and C as the set of chord edges obtained from the initial tree. It means that for each chord edge in C we will replace individually each branch in $Temp_T$. Fig-2 lists the algorithm for the population generation. We can start by inserting an edge from set C into Spanning Tree $Temp_T$, which will create a circuit in the tree. Afterwards, we will remove an edge belonging to set B from spanning tree that is also part of the circuit created in the previous step. The resultant tree is stored as a candidate solution with provision that resultant would be a valid tree or connected graph with V-1 edges. We will remove one branch at a time and validate it. Once validated, store and place this branch in $Temp_T$ and then remove another branch from that cycle, repeat this process for whole tree. The same process is repeated for each chord edge in chords set C. Ultimately, we have k (k is the cardinality of chord set) spanning trees in set T. Time complexity of cyclic interchange is O(CxB) which is in polynomial in nature.

Algorithm for Population Generation

$ConstructPopulation(G,T,i)$

// **for DCST** $\underline{ConstructPopulation(G,T,d,i)}$

Begin

1. Initialize an array C for storing chord edges
   $C = G - T_0$
2. For each edge c:C do
3. { $T_{Temp} \leftarrow T_0$ // make a copy of T_0 Store it into T_{Temp}
4. For each edge e:T_{Temp} do
5. { Insert (T_{Temp}, c) //Insert edge c in Temp Tree T_{Temp}
6. Remove (T_{Temp}, e)// remove branch b of T_{Temp}
7. If (ISConnected(T_{Temp}))

// for DCST include in above && DCTest(T_{Temp}, d)
// returns about disconnection of T_{Temp} Degree bound Test

8. { $T_i \leftarrow T_{Temp}$
9. $i \leftarrow i + 1$
10. Reverse (T_{Temp}, e, c) }
11. Else Reverse (T_{Temp}, e, c)
12. }}

End

Fig. 3. Algorithm for Population Generation

3) Evolution: We can represent all k trees obtained in the previous step as input for genetic algorithm before starting the process of evolution. We use mutation as a reproduction operation on population set T. In addition to reproduction we maintain a global best *GB* value of best fitted spanning tree in terms of cost across the whole operation and local best *LB* value of best fitted spanning tree of that particular iteration. For evolution to be successful, all spanning trees in T are sorted in increasing order. This process has a time complexity of O(VlogV). After sorting, an object array of chord sets against each of the spanning tree, of set T, is generated which takes $O(|T|V^2)$ where V is the number of nodes. Fig. 3 lists the algorithm. Mutation is applied on spanning trees by first finding the minimum cost chord edge and then finding the maximum cost branch edge. The extracted chord edges are placed in the spanning tree and extracted branch edges are removed from

this spanning tree (If the cost of chord edge is less than extracted branch edge). Once a new spanning tree is created, a validation check is performed to make sure that it exists in actual graph. If validation conditions are satisfied, the offspring replaces its parent (provided its fitness is better) otherwise, second maximum branch is selected in the tree and the same process is repeated until offspring is validated. The evolutionary process is run for 1000 iterations in the experiments performed for this work. At the end of each iteration, object array T is sorted and after that object array C is again filled up.

Evolution Algorithm

$\boldsymbol{Evol(G,T)}$ **// for DCST** $\boldsymbol{\underline{Evol(G,T,d)}}$

1. Sort Object array T in term of total cost of edges of each Tree in t
   $Sort(T)$
2. Initialize an Object Array C which will store Matrices of chord edges against each Tree $T_i$in T where $i = 0,1,2,3 \ldots k$ // k is the number of spanning trees in T
   $C = [C_0, C_1, C_2, \ldots, C_k]$
3. Make chord matrices against each $T_i$and store in C through make_Chords(T,C)
4. After sorting T, current T_0 declared as having minimum cost than other Trees so cost of T_0 declared as initial Global Best cost
   $GB \leftarrow cost(T_0)$
5. Initialize an integer array LB[] for storing Iteration best cost
6. For j=0 to 100
   { For i=0 to k
   { $mutation(T_i, C_i)$
   // for DCST $mutation(T_i, C_i, d)$ }
   Sort (T)
   make_Chords(T,C)
   $LB_j \leftarrow cost(T_0)$
   If $(LB_j < GB)$
   $GB \leftarrow LB_j$}
7. End

Fig. 4. Algorithm for evolution

B. Phase II

In Phase-II we apply the same algorithms as in Phase-I with constraints on node degree which should be less thannode degree *d*, where *d*>1. (Consider mentioned DCST modifications in algorithm figures)

1) Initial Spanning Tree Extraction: In this part we use same technique mentioned in section 3.1.1 except one alteration. In this phase we extract degree constrained spanning tree (DCST) from graph G.

We have used a modified form of Prim's algorithm which provides a spanning tree having degree constraint of *d* on each node of the tree (as shown in fig-2). Once the tree is extracted it is verified to be connected this is done to avoid any forest due to degree constraints.

Mutation Algorithm

$\boldsymbol{mutation(T_i, C_i)}$ **// for DCST** $\boldsymbol{mutation(T_i, C_i, d)}$

Begin

1. make a copy of Tree $T_i asTemp_i$
   $Temp_i \leftarrow T_i$
2. while(!chordsDone){
   //Extract Min cost edge
3. edge $ce \leftarrow MinCostEdge(C_i)$
   //removing to avoid same selection next time
4. $remove(C_i, ce)$
   // Extract Max Cost edge
5. Edge $be \leftarrow MaxCostEdge(Temp_i)$
6. While(cost(be)!=0 && !branchesDone){

If (cost(be)<cost(ce) {chordsDone=branchesDone=true
break;}
// removing to avoid same selection next time
remove $(be, Temp_i)$
//operation on real Tree
remove (be, T_i)
$insert(ce, T_i)$
If (IsConnected (T_i))
// for DCST IsConnected (T_i, d) && DCTest(T_i, d)
chordsDone=branchesDone=true
Else
Reverse(be, T_i)
// Extracted Current Max in $Temp_i$
$be \leftarrow MaxCostEdge(Temp_i)$}}

End

Fig. 5. Algorithm for mutation

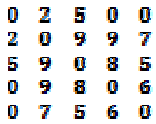
2) Population Generation: Population is generated using a method similar to described in section 3.1.2 however only those members are inducted in the population set that satisfy the degree constraint. For this purpose we have a check on each newly created spanning tree for degree constraints. Each node of the tree is verified to have degree greater than or equal to *d* additionally the resulting tree must be connected. (As shown in Fig. 2)

3) Evolution: Evolution process in phase-II is also similar to the method mentioned in section 2.1.3with additional modification for our degree conditions. Each successful mutation is also checked for degree constraints too. Fig-4 and 5 list the algorithms.

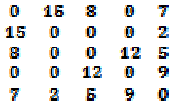
IV. EXPERIMENTATION AND RESULTS

The proposed algorithm is implemented in java and is tested ona number of different graphs with 5, 7 and 10 nodes.

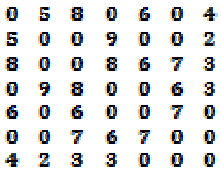
Case 1: In first case we had conducted test on a 5 × 5 matrix graph which is given as follows.



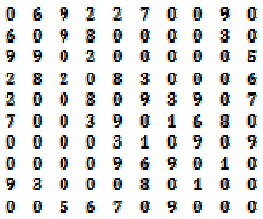
Case 1



Case 2



Case 3



Case 4

In this case for simple spanning trees problem we can observe in Table 1. The initial cost of the extracted spanning tree was 23. After cyclic interchange 5 more spanning trees are generated having best case of 20 and worst case as 30. After evolution we may observe that average fitness level of whole population is enhanced in term of best fittest 18 and worst 27.As listed in Table 2 cost of initial spanning tree was 21 and then after applying cyclic interchange we got 5 more spanning trees in the population. Prior to applying mutation best fittest individual had cost 18 and worst with cost of 26. After Applying Evolution we can observe that best case cost remains same but worst cost is improvement which is 21.

Case 2: In this case we again have a graph with dimensions 5 × 5 given as below

In this case for simple spanning trees problem (refer to table 1)the initial cost of the extracted spanning tree was 29. After cyclic interchange 11 more spanning trees were generated having best case of 29 and worst case as 44. After evolution we may observe that average fitness level of whole population enhanced in terms of fittest 26 and worst 39. For a DCST referent to Table 2 cost of initial spanning tree was 31 and then after applying cyclic interchange we got 11 more spanning trees as population. Before apply mutation best fitness level was with cost of 29 and worst cast with cost of 44. After Applying Evolution we can observe that best case cost remains same but improvement in worst cost which is 34.

Case 3: In this case we have taken graph having dimensions 7 × 7. Matrix is as given below

In this case for simple spanning trees problem the initial cost of the extracted spanning tree was 24. After cyclic interchange 8 more spanning trees were generated having best case of 24 and worst case as 31. But right after evolution we may observe that average fitness level of whole population remains same in terms of fittest 24 and worst 31. For a DCST case again was unable to succeed so this case failed.

Case 4: In this case we have taken graph having matrix dimension 10 × 10. Matrix is given below.

In this case for simple spanning trees problem the initial cost of the extracted spanning tree was 52. After cyclic interchange 9 more spanning trees were generated having best case of 44 and worst case as 59. But right after evolution we may observe that average fitness level of whole population enhanced in term of fittest 33 and worst 59. Interestingly for a DCST problem (Table 2) cost of initial spanning tree was 48 and then after applying cyclic interchange we got 10 more spanning trees as population. Before apply mutation best fitness level was with cost of 42 and worst cast with cost of 59. After applying evolution we can observe that best case cost gets lower to 36 and also an improvement is seen in worst cost which is 49.

Table 1. DCST results

Cases	Nodes	Graph Cost	Initial spanning tree cost	No. of Spanning trees (after cyclic interchange)	Best case before Evolution	Worst cost before Evolution	Best Cost case (after evolution)	Worst cost(after evolution)
1	5	51	21	5	18	26	18	21
2	5	53	31	11	29	44	29	34
4	10	142	48	10	42	56	36	49

Table 2. Simple Spanning Trees results

Cases	Nodes	Graph Cost	Initial spanning tree cost	Spanning trees (after cyclic interchange)	Best case before Evolution	Worst cost before Evolution	Best Cost case (after evolution)	Worst cost(after evolution)
1	5	51	23	5	20	30	18	27
2	5	53	29	11	29	44	26	39
3	7	72	24	8	24	31	24	31
4	10	142	52	9	44	59	33	59

In addition to the aforementioned test cases we have conducted experimentation on a total of 15 graphs. The details of these graphs (included the already mentioned test cases) are listed in table 3.

Table 3. Results of test on 15 graphs

Case No.	Nodes	Graph Cost	Degree Constraint (d)	Initial Spanning Tree cost	Number of Spanning Trees (after cyclic interchange)	Best cost case before evolution	Average cost before evolution	Worst cost case before evolution	Best cost case after evolution	Average cost after evolution	Worst cost case after evolution	Result status (P=Pass, F=Failed, N = No change)
1	5	51	2	21	15	18	22	26	18	19	21	P
2	5	58	2	31	11	29	36	44	29	30	31	P
3	7	74	2	*Disconnected Graph generated in case of d=2*								F
4	10	141	2	48	31	42	50	56	36	45	49	P
5	10	144	4	20	111	20	26	33	20	21	23	P
6	13	175	4	52	101	52	58	65	52	53	59	P
7	12	132	4	69	63	69	76	82	69	70	76	P
8	11	146	4	71	63	71	76	82	71	75	82	N
9	12	201	4	53	109	53	59	66	53	54	56	P
10	11	137	4	53	77	53	60	66	53	55	62	P
11	10	124	4	32	57	32	36	39	32	33	36	P
12	15	136	4	70	73	70	73	77	70	71	76	P
13	10	139	4	47	65	47	52	60	47	49	55	P
14	8	74	2	*Disconnected Graph generated in case of d=2*								F
15	8	74	4	29	29	29	32	36	29	30	31	P

The proposed algorithm was tested on machine an Intel C2D 2.2 GHz with 2GB RAM. A total of 15 cases (generated graphs) were tested. Cases 1, 2, 3, 4 and 14 were tested for degree constrained d of 2 and rest of the ten cases were tested for degree constrained d of 4. Based upon the experiments we observe that all the cases, except case 8, algorithm show improvement in worst cost solution trees. Since in case 8 results remain same so no failure occur as this would be the case where the initial population was the best possible population. For average case proposed algorithm gives improvement in average cost of the population. As in case 5 average cost improved from 26 to a lower value of 21. The proposed approach performs better in 92 % cases.

V. CONCLUSION

This paper presented a new evolutionary based approach using genetic programming for DCMST. Mutation is the only reproduction operation used. Through an input graph we have created an input tree and then through cyclic interchange generated population for applying genetic programming. Mutation was applied on each of the candidate spanning tree. The evolutionary process was run for 1000 iterations. Our focus was on DCST for this purpose the proposed algorithm as test on four different graphs. The proposed approach performs better in 92 % cases.

REFERENCES

[1] Helmi, B. Hoda, Martin Pelikan, and Adel T. Rahmani. "Linkage learning using the maximum spanning tree of the dependency graph." Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion. ACM, 2012.

[2] Binh, Thanh Thi Huynh, et al. "New heuristic and hybrid genetic algorithm for solving the bounded diameter minimum spanning tree problem." Proceedings of the 11th Annual conference on Genetic and evolutionary computation. ACM, 2009.

[3] Katsigiannis, Anastasios, et al. "An Approach to Parallelize Kruskal's Algorithm Using Helper Threads." *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 2012.

[4] Julstrom, Bryant A. "Codings and operators in two genetic algorithms for the leaf-constrained minimum spanning tree problem." *International Journal of Applied Mathematics and Computer Science* 14.3 (2004): 385-396.

[5] Markus Behle, Michael J¨unger, and Frauke Liers, (2007) A Primal Branch-and-Cut Algorithm for the Degree Constrained Minimum Spanning Tree Problem. WEA'07 Proceedings of the 6th international conference on Experimental algorithms Pages 379-392

[6] G¨unther R. Raidl ,An Efficient Evolutionary Algorithm for the Degree-Constrained Minimum Spanning Tree Problem, Proceedings of the conference on the Evolutionary Computation, 2000 , Volume 1, page 104- 111,

[7] R. Ravi madhav v. Marathe s. S. Ravi daniel j. Rosenkrantz harry b. Approximation Algorithms for Degree-ConstrainedMinimum-Cost Network-Design Problems, Fundamental Problems in Computing. ISBN 978-1-4020-9687-7. Springer Netherlands, 2009, p. 241

[8] Dorigo, M. and Gambardella, L.M.. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1(1):53-- 66, 1997.

[9] Son Hong Ngo, Xiaohong Jianget al. An Ant-based Approach for Dynamic RWA in Optical WDM Networks, Photonic Network Communications January 2006, Volume 11, Issue 1, pp 39-48

[10] PriyaVaijayanthi R.High Dimensional Data Clustering using Ant Based Algorithm. European Journal of Scientific Research ISSN 1450-216X Vol.73 No.3 (2012), pp. 364-372

[11] Kenneth Sörensen, Gerrit K. Janssens, An Algorithm To Generate All Spanning Trees Of A Graph In Order Of Increasing Cost , Pesquisa Operacional, v.25, n.2, p.219-229, Maio a Agosto de 2005

[12] Hans-georg beyer and Hans-paul schwefel, Evolution strategies A comprehensive introduction, Kluwer Academic Publishers, Natural Computing 1: 3–52, 2002.

[13] Mohit Singh, Lap Chi Lau, Approximating Minimum Bounded Degree Spanning Trees to within One of Optimal, STOC'07, June 11–13, 2007, San Diego, California, USA.

[14] Waldemar Hummer, Constrained Minimum Spanning Tree Algorithms, December 18, 2008

[15] Bui, T. N. and Zrncic, C. M. (2006). An ant-based algorithm for finding degree-constrained minimum spanning tree.GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 11{18, New York, NY, USA.

[16] Bui, T. N. and Zrncic, C. M. (2006). An improved ant-based algorithm for finding degree-constrained minimum spanning tree. IEEE Transactions on Evolutionary Computation, VOL. 16, NO. 2, April 2012.

[17] Jamil, S., Khan, A., Halim, Z., & Baig, A. R. (2011, August). Weighted MUSE for Frequent Sub-graph Pattern Finding in Uncertain DBLP Data. In Internet Technology and Applications (iTAP), 2011 International Conference on (pp. 1-6). IEEE.

[18] Aniq, M., Halim, Z., & Baig, R. (2008, May). MST and SFMST based Clustering. In 1st National Conference on Security, Computing, & Communication (p. 68).