

A SAMPLE RUN

by
M.TOYGAR KARADENIZ

Bogazici University
1996

TABLE OF CONTENTS

1. INTRODUCTION	3
1.1 A FIRST WORD ON DEGREE CONSTRAINED MST PROBLEM	3
1.2 DEFINITION OF THE PROBLEM	3
2. RESULTS OF THE BRIEF SURVEY ABOUT THE PROBLEM	4
3. DEVELOPED ALGORITHMS AND BENCHMARKS	6
3.1 DESCRIPTION OF THE TECHNIQUES USED FOR SOLVING THE PROBLEM	6
3.2 PSEUDO-CODE OF THE ALGORITHMS USED	7
3.3 EVALUATING THE QUALITY OF OUR SOLUTIONS	8
3.4 TEST PROBLEMS AND RESULTS	9
4. CONCLUSION	14
5. REFERENCES	15
6. APPENDIX	16

1. Introduction

1.1 A First Word On Degree Constrained MST Problem

Surely, it is an easy and well understood problem to find a minimum cost spanning tree (MST) for any given graph with edge weights attached. But, it is always more difficult to satisfy the constraints on the vertex degrees of a MST. In fact, the problem of decreasing the maximum degree of a MST to a certain bound is known to be NP-Complete. The NP-Completeness covers even the basic version of this problem which is known as the 'Hamiltonian Path Problem'. For degree values greater than the Hamiltonian Path problem, the problem is proven to be NP-Hard. Following is a report which presents the basic definition of the problem at hand and a algorithm consisting of two new heuristics intended to attack the problem. The problem is considered in a general manner in this particular attempt. So, the approaches given here, can be basically used on any kind of network structures. It may be a better choice to customize some heuristics for some kind of structure rather than a generic one used for all. The main advantage of this choice will, of course, be faster algorithms in the first place. But, this is not done here and the heuristics developed are all general purpose.

1.2 Definition Of The Problem

Degree constraint minimum spanning tree problem, is one of the well known problems which reside in the NP-Complete set. Discussing the NP-Completeness, is rather beyond the purposes of this report, but, the interested reader is invited to Garey and Johnson book on computers and intractability [6]. Defining the problem will, surely, make it clear. So, the definition which is given by Garey and Johnson, is presented below.

- **Name** : Degree Constrained Spanning Tree
- **Instance** : Graph $G = (V,E)$, positive integer $K \leq |V|$
- **Question** : Is there a spanning tree for G in which no vertex has degree larger than K ?
- **Comment** : Remains NP-Complete for any fixed $K \geq 2$.

2. Results Of The Brief Survey About The Problem

Unfortunately, there are not so many papers written on the degree constrained MST problem. In this brief survey, five related papers which are listed in the references part of this paper, are examined and hopefully, enough information is gathered.

In these papers mainly the heuristics given by Ted Fischer [1], is easy to understand and evaluate. It can be said that the backbone of our degree reduction algorithm carries some ideas from his first algorithm, but the compressing section and the cycle construction algorithms are totally different. It seems on theory that our algorithm might have some better results than Ted Fischer's in larger graphs because of compression idea. Moreover, at worst where there is no possible compression, the performance of our algorithm is equal to his one's. But surely, this condition is very rare.

Another creative paper was of Bruce Bolton, Narsingh Deo and Nishit Kumar [2]. It presents five heuristics that are really good at approximating degree constrained MST, but they are hard-coded into the algorithm and lack of flexibility. Also, they are not explained in full detail. Hence, they don't give much idea. But the value of that paper comes from the various run time comparisons of the heuristics while solving the problem. So, it is a good document to save for the testing part of our algorithm to determine a lower bound for our approximations.

There is a very interesting paper which was written by Fekete, Khutter, Klemmstein, Raghavachari and Young [4], among the ones about the degree constraint MST problem. The interest concerning this paper, comes directly from the fact that the degree constraint MST problem is attacked by a network-flow based algorithm, in that paper. And, it is claimed that this method yields a better performance than any other approach. Also, that algorithm is guaranteed to give a weight of at most the weight of MST times $(2 - \min((d(v)-2)/(d_i(v)-2)))$, where $d_i(v)$ is the initial degree of v and $d(v)$ is the degree constraint. At the final pages, the concepts are visualized by graphical examples.

Robins and Salowe [5], had written a paper about on the degree problem of MST's. But, some examination on this paper showed the fact that this paper is about something different from the degree constraint MST problem. It explains how large the maximum degree of a vertex in a MST, can be achieved. So, this paper has little to do with this project and this report in the first place. But, it gave some insight into how a problem like degree constraint MST problem, can be thought in three dimensions with its 3-D graphical representations.

The last paper of this rather brief survey is written by Khutter, Raghavachari and Young [3]. The concepts discussed in this paper, are somewhat general points about how can low degree spanning trees of small weights be achieved. The developed techniques are mainly about getting the spanning tree with low degrees and weights directly, without finding the MST first and manipulating that one. As we decided to use a technique which exactly

finds MST first and then manipulate that tree, this paper was useful at only general points.

3. Developed Algorithms And Benchmarks

3.1 Description Of The Techniques Used For Solving The Problem

In this project, the technique used to solve the degree constraint MST problem, is heuristics. Actually, two different heuristics are developed and used. As, NP-Complete problems are those for which there are no polynomial time algorithms existing yet, the usage of heuristics or some equivalent techniques is unavoidable.

This section gives a rather brief description of the solution techniques used in this project and the following section gives the pseudo-code of the algorithms used.

As the first step, corresponding MST is formed and taken as the lower bound for the goal cost. The solution technique starts with this MST at hand. When an edge that is present in the original network, but not in this MST, is added to that tree, it creates a cycle. Removing an edge from the induced cycle, we are again left with a spanning tree. This operation is defined to be a 'swap'. Such a swap may increase the degree of both the new end vertices, but it will decrease the degree of the old ones. By introducing new edges and removing some others we can adjust the degrees of vertices to a desired value. The main operation of the selection of edges to be removed and edges to be included, are determined by heuristics. There are two heuristics plugged into the main algorithm that is just described. Both heuristics say that the edge to include is the least cost edge and the edge to remove has the largest cost. As obtaining these two conditions simultaneously is impossible, we decide to take a tradeoff between the two into consideration. This tradeoff is easily obtained by taking the pair of edges p and q where $c_p - c_q$ is the minimum among all (c_p and c_q gives the costs of p and q respectively and p is the edge to be excluded and q is the edge to be included). The first heuristic starts with the nodes which violates the degree constraint and goes through a search of cycle process with the least cost. The second one is somewhat more intuitive. It says that we must just include the cheapest connection which is not present in the current MST to form a cycle. If that cycle has our violating vertex also, then we remove the most expensive edge incident to that vertex and in the cycle. If that vertex is outside the constructed cycle, then we discard that edge and look for the next least cost edge. Iterating by using these processes, we, finally, get the degree constraint MST, if one exists. If there is no such degree constraint MST, we obtain the nearest feasible MST.

3.2 Pseudo-Code Of The Algorithms Used

The collection of the algorithms used in this project, are named as 'Degree Reduction Algorithm (DRA)' as a whole, after the job they mainly do. DRA consists of two parts. The first part is used for just finding the MST of the given network, which can be easily done by one of the well known algorithms given by Prim, Kruskal or Sollin. All of these algorithms can be found in nearly every textbook on data structures. The second part of the algorithm, is just the implementation of the techniques described in the previous section.

Now, suppose that we are given the graph $G=(V,E)$ with a degree constraint of K . DRA goes like the following :

- a) Finding the MST of G by using the algorithm by Prim : As the Prim's algorithm is the fastest and also requires the least space, it is the obvious choice. The pseudo-code for the Prim's algorithm is given below :

- (1) Start with one node in the tree and all other nodes out of tree.
- (2) If there are still out of tree nodes go to (3). If not go to (5).
- (3) Find the out of tree node which is nearest to the tree. Bring that node into the tree and record the edge which connects it into the tree.
- (4) Go to (2).
- (5) Return the generated MST.

- b) Degree Adjustment Phase With Heuristic 1 : Note that, only one of the (b) or (c) parts of the algorithm works in a run, not both.

- (1) For each vertex V , check whether degree of V exceeds K or not and collect all the vertices that has degrees larger than K into TO_ADJUST set.
- (2) Remove one vertex V_1 from TO_ADJUST set. If there is no such vertex to remove then jump to (11).
- (3) Collect all the vertices that are adjacent to V_1 and put them into $CHILDREN$ set.
- (4) For each vertex V_2 in $CHILDREN$ set, determine a cycle that is beginning with (V_1, V_2) and generated by including an edge E that is in the original graph but not in the current MST. If there is no such cycle that is not considered, go to (6).
- (5) Assign E the value of $C(E) - C(V_1, V_2)$ where C gives the cost. Go to (4).
- (6) Include E with the least cost value associated and exclude the (V_1, V_2) . If there is no such edge E go to (10).
- (7) $DEGREE(V_1)$ and $DEGREE(V_2)$ decreases by 1.
- (8) If $DEGREE(V_1) \leq K$ then go to (9), else go to (3) to iterate once more.

- (9) Go to (2)
 - (10) Print output 'No More Improvements For V_1 ' and go to (2)
 - (11) Return the found MST.
- c) Degree Adjustment Phase With Heuristic 2 : Note that, only one of the (b) or (c) parts of the algorithm works in a run, not both.
- (1) For each vertex V , check whether degree of V exceeds K or not and collect all the vertices that has degrees larger than K into TO_ADJUST set.
 - (2) Remove one vertex V_1 from TO_ADJUST set. If there is no such vertex to remove then jump to (10)
 - (3) Find the least cost edge E that is in the network, but not in the current MST and include it into the MST to get cycle CYC. If there is no such edge jump to (9). Make $C(E) = \text{Infinity}$ where C gives the cost of edges.
 - (4) Determine vertices V_2 and V_3 which are two children of V_1 and both are on the cycle CYC.
 - (5) Remove the edge which has the larger cost among (V_1, V_2) and (V_1, V_3)
 - (6) $\text{DEGREE}(V_1)$ and $\text{DEGREE}(V_2)$ or $\text{DEGREE}(V_3)$ decreases by 1.
 - (7) If $\text{DEGREE}(V_1) \leq K$ then go to (8), else go to (3) to iterate once more.
 - (8) Restore the cost matrix C for the network and go to (2)
 - (9) Print output 'No More Improvements For V_1 ' and go to (2).
 - (10) Return the found MST.

3.3 Evaluating The Quality Of Our Solutions

The test graphs are mainly generated randomly by using the internal random number generator of MATLAB which is the main tool used in developing this project. A better choice can be to work with a biased random number generator, but, MATLAB has no tool like that and defining it in the code is a hard job in a system like MATLAB. The further versions, if any is done, may contain such a generator.

Degree Reduction Algorithm, runs on these test graphs and produces both the MST's and the degree constraint MST's with the total costs associated with each other. These results are examined in three categories. In the first one, the expression which is given as (cost of degree constrained MST / cost of MST), is plotted as a function of the number of nodes. This style of testing the value of solutions is taken from Boldon, Deo and Kumar [2] with very little modifications. The expression mentioned is a good benchmark which shows the gap between the degree constraint MST and normal MST in a very instructive manner. Of course, this value will be over one and how much it exceeds one,

gives just the quality of our solution. The second category in which the results are examined, also is a function between the mentioned quality expression and the degree limit (d) value. In this way, we can easily see whether our algorithm succeeds and how good it does in hard problems like (d=2). The third consideration is the execution time of the algorithms when the number of nodes changes. This is a classical performance measure and gives an idea about the complexity of the algorithms.

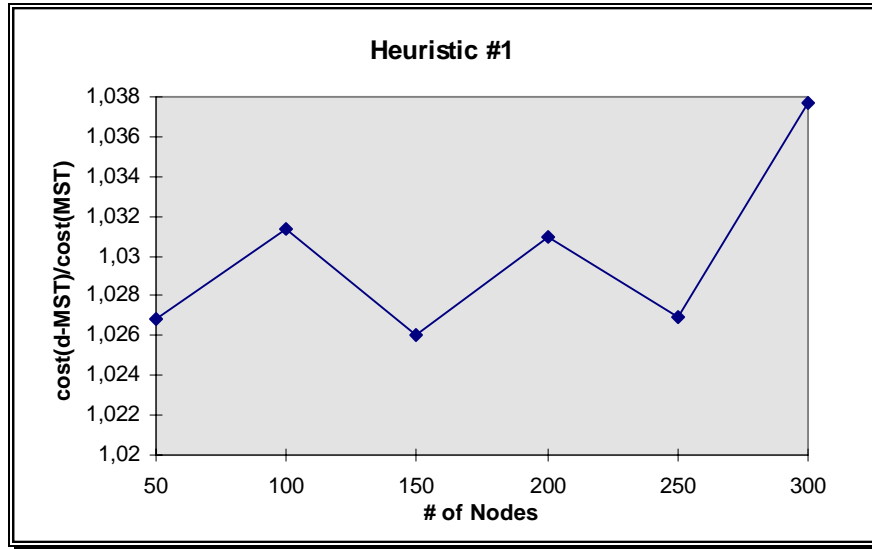
The collection of these three categories will make it a lot clear whether the solutions found by our algorithms, are satisfactory at a certain level or not.

3.4 Test Problems And Results

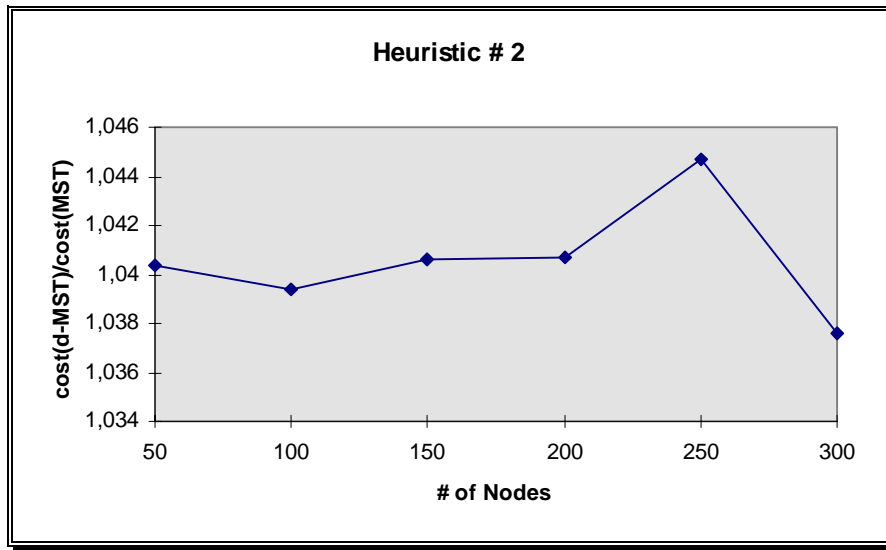
Bolton, Deo and Kumar [2] proposed a bunch of good techniques in their paper. This collection consists of testing the algorithms according to mainly three categories, as mentioned in detail before. In the current section, the actual test values obtained for various cases, are presented in the tables and illustrated with charts. These tables and charts had been directly compared to the ones given in the paper of Bolton, Deo and Kumar and this comparison had shown that the algorithms proposed by us and the ones proposed by them produce very close results. This fact proves the quality of our solutions, in the first place. But, there are some more points to consider in this comparison. The first of these is that the number of nodes that Bolton, Deo and Kumar used in test cases, is highly greater than ours. But this is not a problem that is caused by our algorithms. That problem is mainly because of MATLAB's memory allocation for large matrices. As you may notice, our program uses a lot of large matrices and these consumes the memory considerably. Note also that Bolton, Deo and Kumar had run their algorithms on a special computer architecture which is designed just for these purposes and which uses parallel processing techniques. Surely, a PC (though it has a Pentium 100 chip) can not be the same with such a special purpose computer. So, the result produced by each of these are different, but the charts show that they have the same dependencies. Moreover, this point is also same for the $((\text{cost of degree constraint MST})/(\text{cost of MST}))$ expression. Bolton, Deo and Kumar's results have much higher values than ours. Again, this is the consequence of the higher number of nodes used in the test.

The following is the tables and charts that gives the test results of our algorithms supported by our two heuristics :

Heuristic #1				
# of Nodes	d-MST cost	MST cost	(d-MST cost)/(MST cost)	
50	1875	1826	1,026834611	
100	1973	1913	1,031364349	
150	2210	2154	1,025998143	
200	2627	2548	1,03100471	
250	2703	2632	1,026975684	
300	3059	2948	1,037652646	

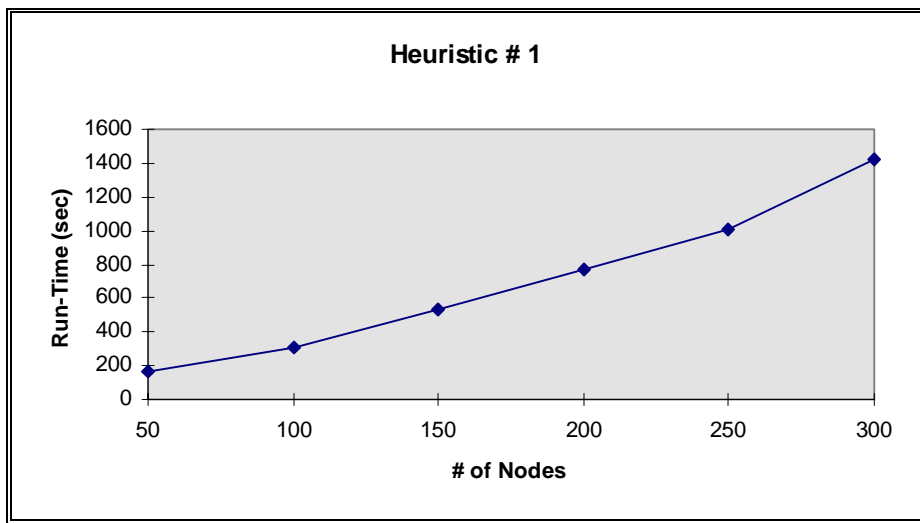


Heuristic #2				
# of Nodes	d-MST cost	MST cost	(d-MST cost)/(MST cost)	
50	2011	1933	1,040351785	
100	2139	2058	1,039358601	
150	2307	2217	1,040595399	
200	2430	2335	1,040685225	
250	2734	2617	1,044707681	
300	2951	2844	1,037623066	

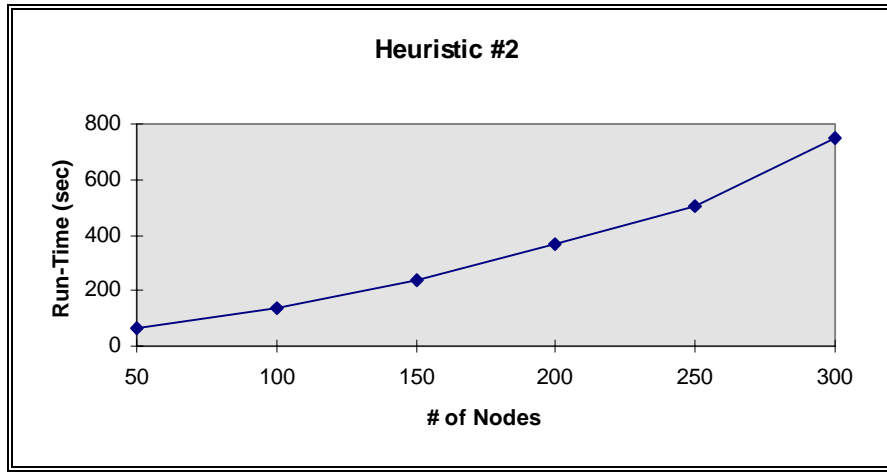


Heuristic #1

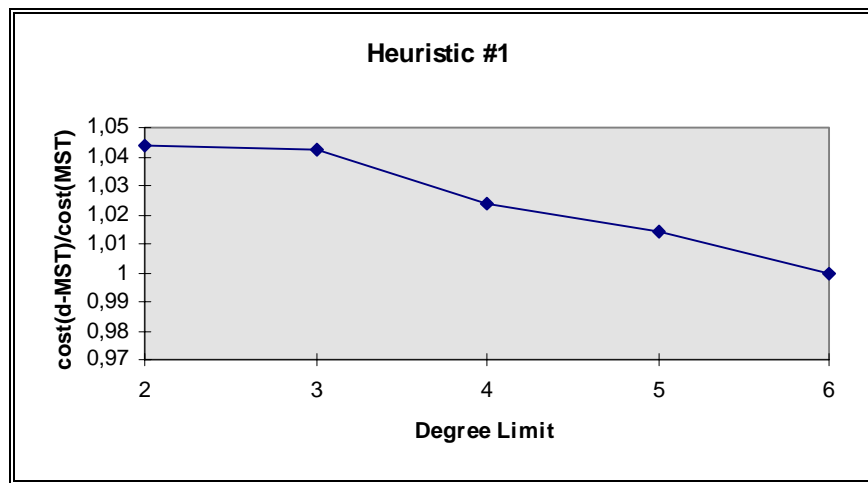
# of Nodes	Run-Time(secs)
50	165
100	305
150	532
200	765
250	1013
300	1420



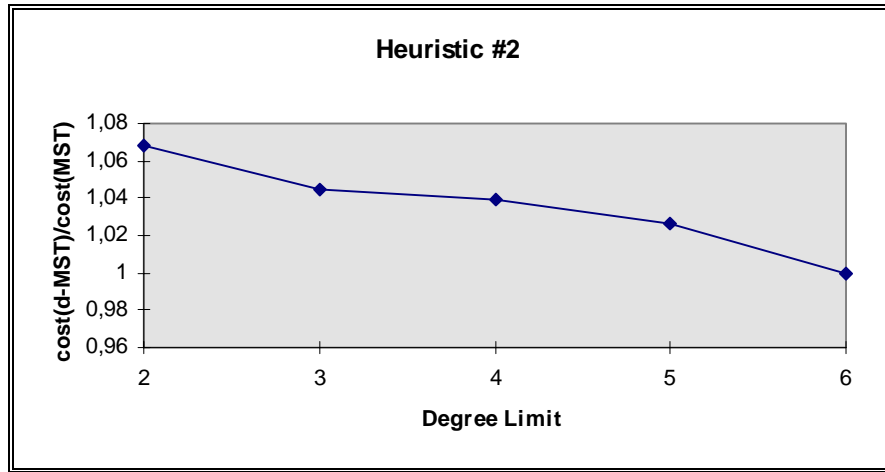
Heuristic #2	
# of Nodes	Run-Time(secs)
50	63
100	140
150	235
200	367
250	503
300	752



Heuristic #1				
Degree Limit	d-MST cost	MST cost	(d-MST cost)/(MST cost)	
2	2083	1995	1,044110276	
3	1907	1830	1,042076503	
4	1985	1939	1,023723569	
5	1905	1879	1,013837147	
6	1782	1782	1	



Heuristic #2				
Degree Limit	d-MST cost	MST cost	(d-MST cost)/(MST cost)	
2	1855	1737	1,067933218	
3	2017	1930	1,04507772	
4	1876	1805	1,03933518	
5	1794	1748	1,026315789	
6	1860	1860	1	



4. Conclusion

Degree constrained MST problem is an interesting item in the NP-Complete problem set. The definition of it is very clear and so simple that it is understood for sure just in first sight. But, this does not make the problem easier at all. Degree constraint MST problem is NP-Complete and this means that there is no polynomial algorithm existing for that problem. Inherent exponential complexity of the problem, can make the developed algorithms useless. To avoid this, one particular solution is to use heuristics and this is mainly what is done in this project.

Here, two heuristics are proposed to attack the degree constrained MST problem. These techniques are both implemented and incorporated into a random network generator - MST finder component. This component generated networks and heuristic algorithms are run on these test cases. The results are presented with this report supported by illustrative charts.

In the coding phase, MATLAB is used. But this is not the best choice at all. It has many restrictions including memory, speed etc. Although MATLAB caused some difficulties during the testing as mentioned, it made the implementation phase easier by its strong matrix operations. Moreover, as you may noticed, the code of the project is not so long. This is another advantage that programming in MATLAB brings. On the other hand, it may be somewhat more instructive to code the given algorithms in another language and compare with the MATLAB code.

Searching solutions for an inherently exponential problem, is rather a hard task to manage. But attacking to it with some heuristic techniques, gives a lot of ideas about the structure of such network problems and how reasonable solutions can be achieved. The techniques which are presented in this project, also gives somewhat reasonable answers but not the optimal ones. The further effort may be concentrated to optimize the algorithms to obtain results faster and closer to the optimality.

5. References

- [1] T. Fischer. Optimizing the degree of minimum spanning trees. Dept. of Computer Science, Cornell University, April 1993.
- [2] B. Boldon, N. Deo and N. Kumar. Minimum-weight degree-constrained spanning tree problem : Heuristics and Implementation on an SIMD Parallel Machine. Tech. Rep. CS-TR-95-02. Dept. of Computer Science, University of Central Florida.
- [3] S. Khuller, B. Raghavachari and N. Young. Low degree spanning trees of small weight. May 1994.
- [4] S. P. Fekete, M. Klemmstein, S. Khuller, B. Raghavachari and N. Young. A network-flow technique for finding low-weight bounded-degree spanning trees.
- [5] G. Robins and J.S. Salowe. On the maximum degree of minimum spanning trees. Tech. Rep. CS-93-22. May 12, 1993.
- [6] M. R. Garey and D. S. Johnson. Computers and Intractability : A Guide to the theory of NP-Completeness. Freeman, San Francisco, (1979).

6. Appendix

This section gives the code written as a M-file of MATLAB, below.

```
% CmpE 582 - Main Body

rand('seed',sum(100*clock));
generate;
mst;
d_const;
quit;
```

```
% CmpE 582 - Network Generator

% Get the system parameters
n = input('Enter the number of nodes (n) : ');
if (n <= 0)
    disp('Incorrect Value ...');
    break
end;
r = input('Enter the range of coordinate system (r) : ');
if r <= 0
    disp('Incorrect Value ...');
    break
end;

disp(' ');
disp('1. Heuristic #1 ');
disp('2. Heuristic #2 ');
choice = input('Enter your choice : ');
if ((choice ~= 1) & (choice ~=2))
    disp('Incorrect Value ...');
    break
end;

% Create a random network
x_y = [rand(n,1) rand(n,1)] * r;
for i = 1 : n
    for j = 1 : n
        cost(i,j) = 0;
    end
    for j = i+1 : n
        cost(i,j) = sqrt((x_y(i,1) - x_y(j,1))^2 + ...
            (x_y(i,2) - x_y(j,2))^2);
    end
end
cost = cost + cost';
```

```

for i = 1 : n
    for j = 1 : n
        adj(i,j) = 0;
    end
    for j = i+1 : n
        adj(i,j) = round(rand(1));
    end
end
adj = adj + adj';
for i = 1 : n
    one_one = 0;
    for j = 1 : n
        if (adj(i,j)==1)
            one_one = 1;
            break
        end
    end
    if (one_one ==0)
        to_one = round(rand(1)*n);
        adj(i,to_one) = 1;
        adj(to_one,i) = 1;
    end
end

% Put the created network onto the screen
figure;
title('Created Network');
xlabel('X');
ylabel('Y');
hold;
for i = 1 : n
    plot(x_y(i,1),x_y(i,2),'wo');
end
for i = 1 : n
    for j = i+1 : n
        if (adj(i,j) == 1)
            line_x = [x_y(i,1); x_y(j,1)];
            line_y = [x_y(i,2); x_y(j,2)];
            line(line_x,line_y);
        end
    end
end
set(gcf,'DefaultTextColor','c');
my_axis = axis;
for i = 1 : n
    text(x_y(i,1),x_y(i,2) - round(my_axis(1,2)/35),...
        [num2str(i)]);
end
set(gcf,'DefaultTextColor','w');
hold off;

```

```
% CmpE 582 Project - MST Calculation by Prim's Algorithm

link_costs = cost .* adj;
for i = 1 : n
    for j = 1 : n
        if (link_costs(i,j) == 0)
            link_costs(i,j) = Inf;
        end
    end
end

which = round(rand(1)*n);
mst = [];
nodes = [];
mst_ptr = 0;
node_ptr = 0;
while (mst_ptr < n-1)
    [not_used,cur_ind] = min(link_costs(which,:));
    link_costs(which,cur_ind) = Inf;
    link_costs(cur_ind,which) = Inf;
    if ((find(nodes==which)==[]) | ...
        (find(nodes==cur_ind)==[]))
        mst_ptr = mst_ptr + 1;
        mst(mst_ptr,1) = which;
        mst(mst_ptr,2) = cur_ind;
    end
    if (find(nodes==which)==[])
        node_ptr = node_ptr + 1;
        nodes(node_ptr) = which;
    end
    if (find(nodes==cur_ind)==[])
        node_ptr = node_ptr + 1;
        nodes(node_ptr) = cur_ind;
    end
    cur_min = Inf;
    for i = 1 : node_ptr
        if (cur_min >= min(link_costs(nodes(i),:)))
            cur_min = min(link_costs(nodes(i),:));
            which = nodes(i);
        end
    end
end
end

printmst;
```

```

% CmpE 582 - Put the MST found onto the screen

figure;
total_cost = 0;
for i = 1 : mst_ptr
    total_cost = total_cost + cost(mst(i,1),mst(i,2));
end
title(['Found MST - Cost : ',num2str(total_cost)]);
xlabel('X');
ylabel('Y');
hold;
for i = 1 : n
    plot(x_y(i,1),x_y(i,2),'wo');
end
for i = 1 : mst_ptr
    line_x = [x_y(mst(i,1),1); x_y(mst(i,2),1)];
    line_y = [x_y(mst(i,1),2); x_y(mst(i,2),2)];
    line(line_x,line_y);
end
set(gcf,'DefaultTextColor','c');
my_axis = axis;
for i = 1 : n
    text(x_y(i,1),x_y(i,2)-...
        round(my_axis(1,2)/35),[num2str(i)]);
end
set(gcf,'DefaultTextColor','w');
hold off;

```

```

% CmpE 582 - Degree Constraint MST by Heuristics

d = input('Enter the max degree of nodes (d) : ');
if (d <= 0)
    disp('Incorrect Value ...');
    break
end;

for i=1 : n
    [R C] = size(find(mst==i));
    degrees(i) = R*C;
end

for i=1 : n
    for j=1 : n
        mst_adj(i,j) = 0;
    end
end
for i=1 : mst_ptr
    mst_adj(mst(i,1),mst(i,2)) = 1;
end
mst_adj = mst_adj + mst_adj';
to_adjust = find(degrees > d);
adj_counter = length(to_adjust);

```

```

while (adj_counter > 0)
  while (adj_counter > 0)
    if (degrees(to_adjust(adj_counter)) <= d)
      adj_counter = adj_counter - 1;
    else
      break;
    end
  end
  if (adj_counter == 0)
    break;
  end
  net_cnt = 0;
  net_link_i = [];
  net_link_j = [];
  net_costs = [];
  net_par_next = [];
  l_cnt = 0;
  looked = [];
  old_parent = to_adjust(adj_counter);
  bfs_nodes = [];
  bfs_nodes = to_adjust(adj_counter);
  if (choice == 1)
    bfs;
  end
  if (choice == 2)
    hero_bfs;
  end
  [V I] = min(net_costs);
  for i = 1 : mst_ptr
    if ((mst(i,1) == to_adjust(adj_counter))...
        & (mst(i,2) == net_par_next(I)) | ...
        ((mst(i,2) == to_adjust(adj_counter))...
        & (mst(i,1) == net_par_next(I))))
      mst_adj(mst(i,1),mst(i,2)) = 0;
      mst_adj(mst(i,2),mst(i,1)) = 0;
      mst_adj(net_link_i(I),net_link_j(I)) = 1;
      mst_adj(net_link_j(I),net_link_i(I)) = 1;
      mst(i,1) = net_link_i(I);
      mst(i,2) = net_link_j(I);
      degrees(to_adjust(adj_counter)) = ...
          degrees(to_adjust(adj_counter)) - 1;
      degrees(net_par_next(I)) = degrees(net_par_next(I)) - 1;
      degrees(net_link_i(I)) = degrees(net_link_i(I)) + 1;
      degrees(net_link_j(I)) = degrees(net_link_j(I)) + 1;
      degrees
      break;
    end
  end
  printmst;
  count_dec = 0;
  if (length(net_costs)==0)
    disp(['No More Improvements For ',...
          num2str(to_adjust(adj_counter)),'...']);
    count_dec = 1;
  end
  if ((degrees(to_adjust(adj_counter)) <= d) | ... (count_dec
      == 1))

```

```

    adj_counter = adj_counter - 1;
end
to_adjust
adj_counter
pause
end

```

```

% CmpE 582 - Breath First Search First Heuristic

```

```

very_first = 1;
cnt = length(bfs_nodes);
while (cnt > 0)
    cur_node = bfs_nodes(cnt);
    cnt = cnt - 1;
    l_cnt = l_cnt + 1;
    looked(l_cnt) = cur_node;
    children = find(mst_adj(cur_node, :)==1);
    if (mst_adj(old_parent, cur_node)==1)
        very_first = 0;
        parent_next = cur_node;
        gain = cost(old_parent, cur_node);
    end
    for i = 1 : length(children)
        if (find(looked==children(i))==[])
            cnt = cnt + 1;
            bfs_nodes(cnt) = children(i);
            grand_children = find(adj(children(i), :)==1);
            for j = 1 : length(grand_children)
                if ((mst_adj(children(i), grand_children(j))~=1) & ...
                    (degrees(grand_children(j))+1 <=d) &...
                    (degrees(children(i))+1 <=d))
                    ok = 1;
                    ch_cycle;
                    if (ok == 1)
                        if (very_first == 1)
                            parent_next = children(i);
                            gain = cost(old_parent, children(i));
                        end
                        net_cnt = net_cnt + 1;
                        net_costs(net_cnt) =...
                            cost(children(i), grand_children(j)) - gain;
                        net_link_i(net_cnt) = children(i);
                        net_link_j(net_cnt) = grand_children(j);
                        net_par_next(net_cnt) = parent_next;
                    end
                end
            end
        end
    end
end
end
end
end
end
end

```

```

% CmpE 582 - Breath First Search Second Heuristic

link_costs = cost .* adj;
[zero_x zero_y] = find(link_costs==0);
for i = 1 : length(zero_x)
    link_costs(zero_x(i),zero_y(i)) = Inf;
    link_costs(zero_y(i),zero_x(i)) = Inf;
end
for i = 1 : mst_ptr
    link_costs(mst(i,1),mst(i,2)) = Inf;
    link_costs(mst(i,2),mst(i,1)) = Inf;
end

while (1)
    [temp_min_val miny] = min(link_costs);
    [next_min_val minx] = min(temp_min_val);
    next_min_val
    if (next_min_val==Inf)
        break;
    end
    children(1) = minx
    grand_children(1) = miny(minx)
    link_costs(children(1),grand_children(1)) = Inf;
    link_costs(grand_children(1),children(1)) = Inf;
    old_parent
    i=1;
    j=1;
    if ((degrees(grand_children(j))+1 <=d) & ...
        (degrees(children(i))+1 <=d))
        ok = 1;
        ch_cycle;
        if (ok == 1)
            temp = children(1);
            children(1) = -1;
            ch_cycle;
            children(1) = temp;
            temp = grand_children(1);
            grand_children(1) = -1;
            ch_cycle;
            grand_children(1) = temp;
            old_par_two = twice;
            if (cost(old_parent,once) >= cost(old_parent,twice))
                old_par_two = once;
            end
            net_cnt = 1;
            net_costs(net_cnt) = ...
                cost(children(i),grand_children(j)) - ...
                cost(old_parent,old_par_two);
            net_link_i(net_cnt) = children(i);
            net_link_j(net_cnt) = grand_children(j);
            net_par_next(net_cnt) = old_par_two;
            break;
        end
    end
end
end

```

```

% CmpE 582 - Check Cycle

children_cycle = find(mst_adj(old_parent,:) == 1);
sp = 0;
for cyc = 1 : length(children_cycle)
    found1 = 0;
    found2 = 0;
    sp = sp + 1;
    stack = [];
    cyc_noted = old_parent;
    cyc_n = 1;
    stack(sp) = children_cycle(cyc);
    while (sp > 0)
        a_node = stack(sp);
        cyc_n = cyc_n + 1;
        cyc_noted(cyc_n) = a_node;
        sp = sp - 1;
        if (a_node == children(i))
            found1 = 1;
            once = children_cycle(cyc);
        end
        if (a_node == grand_children(j))
            found2 = 1;
            twice = children_cycle(cyc);
        end
        if ((found1 == 1) & (found2 == 1))
            break;
        end
        in_children = find(mst_adj(a_node,:) == 1);
        for in_ch = 1 : length(in_children)
            if (find(cyc_noted == in_children(in_ch)) == [])
                sp = sp + 1;
                stack(sp) = in_children(in_ch);
            end
        end
    end
    if (((found1 == 1) & (found2 == 0)) | ...
        ((found1 == 0) & (found2 == 1)))
        break
    end
    if ((found1 == 1) & (found2 == 1))
        ok = 0;
        break;
    end
end
end

```
