A GRASP heuristic for the Node Degree Constrained Minimum Spanning Tree Problem with Node Degree Costs

Christophe Duhamel, LIMOS, Université Blaise Pascal, Clermont-Ferrand, France
Mauricio Souza DEP, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Keywords: Combinatorial Optimization, network design, constrained spanning tree, metaheuristics

1. Introduction

The Node Degree Constrained Minimum Spanning Tree Problem (NDCMSTP) consists in finding a minimum spanning tree such that the degree of each node is restricted to be not greater than a given threshold. The cost function takes into account two elements: the cost of each selected edge and also the cost associated to the degree of each node. This problem makes sense in telecommunication network design where one has to take the cost of the routers into account. The communication is made through I/O modules whose connectivity, i.e. number of I/O ports is limited. Then in order to match the connectivity requirement, several modules may have to be connected together. Moreover, several modules capacities may be available and the smaller modules may have a cost per connection higher than the larger ones (due to economy of scale). Another point is that the installation cost for routers may be more expensive than the installation cost for terminals as the latter ones do not need any equipment to perform the routing. Therefore, we consider here the node degree cost function as a concave staircase cost function.

Another application for the NDCMSTP is to minimize the delay in high speed networks. In such networks, broadcasting may lead to high delays: the incoming message is copied at each router in order to be sent to each child and one copy is needed for each child. As this operation consumes a lot of time, it is of practical interest to build a communication networks taking into account the degree of each router.

This problem is NP-hard [5], even when the same degree is imposed on every node ($k$-NDCMSTP). Interestingly, the problem is polynomial on euclidean instances when $k \geq 5$.


To our knowledge, none of those approaches include a node degree cost function. In this work, we present an hybrid GRASP/VND metaheuristic to compute (hopefully) good primal solutions to the problem. We also propose a path relinking strategy to further improve the efficiency of our approach. Preliminary numerical results are reported.

2. Mathematical formulation

Let $G = (V, E)$ be a graph, where $V$ is the set of $n$ vertices and $E$ is the set of $m$ edges. To each edge $e \in E$, let $c_e > 0$ be its installation cost. To each vertex $v \in V$, let $D_v$ be the upper limit on its degree $d_v$. Without
loss of generality, it will be assumed that the restriction is the same for each vertex, that is $D_v = D$ and $F_v(d_v) = F(d_v)$. The node degree cost function $F(d)$ is a concave staircase function as explained before. Let $x_e \in \{0, 1\}$ be the decision variable associated to each edge $e \in E$. The problem can then be stated as follows:

$$
\begin{align*}
\text{Min} \quad & z = \sum_{e \in E} c_e(x_e) + \sum_{v \in V} F(d_v) \\
\text{s.t.} \quad & x_e \in \text{ST} \quad (1) \\
& \sum_{e \in E(v)} x_e = d_v \quad \forall v \in V \quad (2) \\
& d_v \leq D \quad \forall v \in V \quad (3) \\
& x_e \in \{0, 1\} \quad \forall e \in E \\
& d_v \in \mathbb{N} \quad \forall v \in V
\end{align*}
$$

$\text{ST}$ denotes the convex hull of the incidence vectors of spanning trees. Thus, the $x$ variables are restricted to define a spanning tree. Therefore, any valid formulation for the spanning tree can be used. Restriction (2) define the degree variables. Constraint (3) set an upper limit on the degree. It should be noted this model is non linear as the function involves non linear terms $F(d_v)$. Mixed integer linear formulations are available for the NDCMSTP, see for instance Gouveia and Moura [8].

3. An hybrid GRASP/VND metaheuristic

The GRASP metaheuristic has been developped by Feo and Resende [4]. It consists in repeated applications of a two-step procedure: a Construction step to produce an initial solution and an Improvement step based on a Local Search to improve it. The Construction step is usually a randomized heuristic. It builds a feasible solution $T$ by adding one element at a time. At each iteration, let $C$ be the set of all possible candidates for insertion into the partial solution $T$ and let $f$ be an evaluation function of the interest of a candidate. For each candidate $e \in C$, $f(e)$ is its value, $f$ and $\overline{f}$ are respectively the better and the worst evaluation over $C$. Then, the candidate for insertion is randomly chosen within the restricted candidate list (RCL), i.e. the list of candidates $e$ whose evaluation $f(e) \in [\overline{f}; f + \alpha(\overline{f} - f)]$. Thus, the parameter $\alpha \in [0; 1]$ controls the greediness of the selection.

```
procedure GRASP2DCMST
  set $T^* = \emptyset$
  for $i = 1, \ldots , i_{\text{max}}$ do
    $T \leftarrow \text{Construct}(\alpha)$
    $\overline{T} \leftarrow \text{Improve}(T)$
    if $F(\overline{T}) < F(T^*)$ then
      $T^* \leftarrow \overline{T}$
    end-if
  end-for
return $T^*$
end-procedure
```

Figure 1: GRASP metaheuristic

For the NDCMSTP, the following constructive heuristic is proposed: first, feasible degrees are assigned to the nodes, then a solution is build according to those degrees. To assign the degrees, let $c_k^v$ be the cost of the $k^{th}$ shortest edge incident to $v$. Given a node degree $d_v$, an underestimation of the node $v$ total cost is
$F(d_v) + \sum_{i=1}^{d_v} c_i$. Thus, the node $v$ expansion cost can be approximated as $g(v) = F(d_v+1) - F(d_v) + c_{d_v+1}^v$. The procedure to assign degrees to nodes is presented in figure 2.

```
procedure Construct_1
    set $d_v = 1$, $\forall v \in V$
    repeat
        $a = \min \{g(v), v \in V\}$
        $b = \max \{g(v), v \in V\}$
        $RCL \leftarrow \{v \in V \mid a \leq g(v) \leq a + \alpha (b-a)\}$
        randomly select $v \in RCL$
        set $d_v \leftarrow d_v + 1$
        until $\sum_{v \in V} d_v = 2(n-1)$
end-procedure
```

Figure 2: degree assignment

Building a tree when the node degrees are fixed is a NP-hard problem. Thus, once the degrees have been assigned, a compatible tree $T$ will be built using a Kruskal-like heuristic. Let $e_v$ be vertex $v$ free degree. Let $D_0 = \{v \in V \mid e_v = 0\}$, $D_1 = \{v \in V \mid e_v = 1\}$ and $D_2 = \{v \in V \mid e_v > 1\}$ be respectively the set of fixed vertices, the set of leaves and the set of non-leaves in the partial tree $T$. At each iteration, an edge is inserted into $T$ to fix one vertex of $D_1$ and the sets are updated. The procedure is presented in figure 3.

```
procedure Construct_2
    set $e_v = d_v$, $\forall v \in V$
    set $D_0 = \emptyset$, $D_1$, $D_2$
    repeat
        $a = \min \{c_{ij}, i \in D_1, j \in D_2\}$
        $b = \max \{c_{ij}, i \in D_1, j \in D_2\}$
        $RCL \leftarrow \{(i,j) \in D_1 \times D_2 \mid a \leq c_{ij} \leq a + \alpha (b-a)\}$
        randomly select $(i,j) \in RCL$
        update $e_i$, $e_j$, $D_0$, $D_1$ and $D_2$
        until $D_2 = \emptyset$
    end-procedure
```

Figure 3: tree building

At the end of the Construction procedure, a feasible tree has been built, provided the graph $G$ is complete. Otherwise, the procedure may end up without being able to connect all the vertices. Therefore, $G$ is transformed into a complete graph by adding artificial edges (the missing ones) with an arbitrarily high cost. This way, the procedure will construct a degree-compatible tree, even if it is infeasible with respect to the edges in the initial graph. No restoration procedure need to be used: the Local Search will try to remove as many artificial edges as possible, since they are too expensive.

The Variable Neighbourhood Descent (VND) is used as Local Search for the Improvement step. The VND was introduced by Mladenović and Hansen [10] and is designed to be the Local Search of the Variable Neighbourhood Search (VNS) metaheuristic. It consists in defining a set $\mathcal{N} = \mathcal{N}_1, \ldots, \mathcal{N}_K$ of $K$ neighbourhood structures. Given a solution $s \in \mathcal{S}$, the neighbourhood $\mathcal{N}_k(s)$ is defined as $\mathcal{N}_k(s) = \{s' \in \mathcal{S} \mid d(s',s' = k)\}$, where $d(\cdot,\cdot)$ is a user-defined distance (the $||.||_1$ norm for instance). At each iteration, let $s \in \mathcal{S}$ be the current solution and $\mathcal{N}_k$ be the active neighbourhood structure. The best neighbour $s' \in \mathcal{N}_k(s)$ is selected and the state is updated as shown in figure 4. The VND stops when the current neighbourhood is $\mathcal{N}_K(s)$ and when it contains no improving solution.

For the NDCMSTP, the neighbourhood structures are chosen to rely on $k$-edge exchange moves: $k$ new edges are inserted into the tree $T$ while $k$ others are removed, producing a new tree $T'$. Thus, $T$ and $T'$ differ from $2k$ edges. To take the node degree into account, three sets are defined according to the current tree $T$: $E_0 = \{(u,v) \in E \mid T \mid d_u < D, \ d_v < D\}$, $E_3 = \{(u,v) \in E \mid T \mid d_u = D, \ d_v < D\}$ and

3
procedure VND
set \( k \leftarrow 1 \)
set \( s \leftarrow s_0 \)
repeat
\( s' \leftarrow N_k(s) \)
if \( f(s') < f(s) \) then
\( s \leftarrow s' \)
\( k \leftarrow 1 \)
else
\( k \leftarrow k + 1 \)
end-if
until \( k \leq K \)
return \( s \)
end-procedure

Figure 4: VND local search

\[ E_2 = \{(u, v) \in E \setminus T \mid d_u = D, d_v = D\} \] are respectively the sets of edges whose insertion would violate 0, 1 and both extremities. Only neighbourhood structures \( N_1 \) and \( N_2 \) are used.

In order to improve the quality of the solution found at the end of each GRASP iteration, a path relinking strategy [7] is used. A pool \( P \) of elite solutions is created, initially \( P = \emptyset \). Let \( s_1 \) be the local optimum returned by the VND. An elite solution \( s_2 \in P \) is randomly chosen and moves of the \( N_1 \) structure are performed to transform \( s_1 \) into \( s_2 \). Those moves are biased to favour the inclusion of elements belonging in \( s_2 \) and not in \( s_1 \). By doing so, the idea is to try to identify interesting solutions on the path from \( s_1 \) towards \( s_2 \). Let \( s_3 \neq s_1, s_2 \) be the best solution found in the path. Then, both \( s_1 \) and \( s_3 \) are candidates for insertion into \( P \). Each solution is inserted if it is better than the worst solution of \( P \) and if does not already belong to it.

4. Preliminary numerical results

The heuristic has been tested on a set of randomly generated instances. The number of nodes has been set to 25, 50 and 100 nodes. For each size, three densities (25%, 50% and 100% the density of a complete graph) have been used. Three values for the maximum node degree have been used, \( D \in \{4, 5, 6\} \). The cost \( c_e \) of each edge \( e \in E \) is randomly chosen in the interval \([1; 100]\) and the staircase cost function is as follows: \( F = [0 \ 30 \ 30 \ 40 \ 40 \ 40] \). The program has been coded in C and compiled with gcc version 4.0.1. The experiments are run on a pentium IV processor at 800MHz with 1Gb memory and running linux 2.6.12. For each instance, 1000 iterations of GRASP are run and at each iteration \( \alpha \) is randomly chosen in \( \{1/10, 2/10, \ldots, 9/10, 1\} \). The pool size has been set to 20.

The results are shown on table 1. For each instance, the optimal value \( z_{IP} \) is presented when available. For the instance \((50, 1225, 6)\), the value 2332 is only an upper bound on the optimal value. The columns \( z_{GRASP} \) and \( z_{PR} \) show the value computed by GRASP and by GRASP using the path relinking strategy. The columns “feas_C” and “feas_V” report the percent feasibility of the solution computed by the Construction step and then optimized by the VND. The CPU time is in seconds. It is only reported for the GRASP with path relinking as it is only slightly higher than the time for GRASP alone.

For small instances, GRASP alone finds the optimal solution on the most sparse graphs. For the other densities, it stays within 1% of the optimal values. Using the path relinking strategy helps finding nearly all the optimal values. For medium instances, GRASP still performs well, especially when the path relinking is used. The values are still within 1% of the optimal values. For the biggest instances \((n = 100)\), no optimal values could be computed using exact methods. However, one can note that the path relinking strategy is quite effective as it always improves the value of the best solution found. As the number of nodes and the density grow, the CPU time quickly increases. This probably means a better stopping criterion (than stopping
after 1000 iterations) should be used on GRASP. One interesting fact is that the limit $D$ has a positive impact on the running time: as $D$ increases, the CPU decreases. An explanation is that the problem becomes less constrained and then more moves are available, which is known to help methods like the VND. One can also note that the feasibility of the solution computed by the Construction procedure mostly depends on the density of the instance. For the smallest density ($m = 0.25n^2$), the solution is nearly never feasible in the original graph. For complete graphs ($m = n^2$), the solution is feasible 60% of the time. After the VND, the solution has always been found to be feasible.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$D$</th>
<th>$z_{IP}$</th>
<th>$f_{feas,C}$</th>
<th>$z_{GRASP}$</th>
<th>$f_{feas,V}$</th>
<th>$z_{PR}$</th>
<th>cpu(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>75</td>
<td>4</td>
<td>1665</td>
<td>0.000</td>
<td>1665</td>
<td>1.00</td>
<td>1665</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1623</td>
<td>0.000</td>
<td>1623</td>
<td>1.00</td>
<td>1623</td>
<td>0.50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1589</td>
<td>0.002</td>
<td>1589</td>
<td>1.00</td>
<td>1589</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>4</td>
<td>1372</td>
<td>0.074</td>
<td>1373</td>
<td>1.00</td>
<td>1372</td>
<td>1.06</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1311</td>
<td>0.069</td>
<td>1316</td>
<td>1.00</td>
<td>1311</td>
<td>0.87</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1282</td>
<td>0.050</td>
<td>1286</td>
<td>1.00</td>
<td>1284</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>4</td>
<td>1476</td>
<td>0.745</td>
<td>1477</td>
<td>1.00</td>
<td>1476</td>
<td>2.39</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1429</td>
<td>0.685</td>
<td>1432</td>
<td>1.00</td>
<td>1429</td>
<td>2.18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1403</td>
<td>0.619</td>
<td>1413</td>
<td>1.00</td>
<td>1409</td>
<td>2.08</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>300</td>
<td>4</td>
<td>3336</td>
<td>0.001</td>
<td>3337</td>
<td>1.00</td>
<td>3336</td>
<td>9.47</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3191</td>
<td>0.000</td>
<td>3191</td>
<td>1.00</td>
<td>3191</td>
<td>8.93</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3109</td>
<td>0.000</td>
<td>3113</td>
<td>1.00</td>
<td>3113</td>
<td>8.35</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>4</td>
<td>3063</td>
<td>0.039</td>
<td>3065</td>
<td>1.00</td>
<td>3064</td>
<td>17.66</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2948</td>
<td>0.030</td>
<td>2953</td>
<td>1.00</td>
<td>2948</td>
<td>16.59</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2900</td>
<td>0.038</td>
<td>2900</td>
<td>1.00</td>
<td>2900</td>
<td>14.17</td>
<td></td>
</tr>
<tr>
<td>1225</td>
<td>4</td>
<td>2498</td>
<td>0.645</td>
<td>2504</td>
<td>1.00</td>
<td>2500</td>
<td>39.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2380</td>
<td>0.578</td>
<td>2386</td>
<td>1.00</td>
<td>2383</td>
<td>34.53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2332</td>
<td>0.605</td>
<td>2339</td>
<td>1.00</td>
<td>2336</td>
<td>29.44</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1250</td>
<td>4</td>
<td>0.000</td>
<td>6316</td>
<td>1.00</td>
<td>6283</td>
<td>86.11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.000</td>
<td>6093</td>
<td>1.00</td>
<td>6068</td>
<td>76.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.000</td>
<td>5973</td>
<td>1.00</td>
<td>5951</td>
<td>73.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2500</td>
<td>4</td>
<td>0.067</td>
<td>5456</td>
<td>1.00</td>
<td>5410</td>
<td>186.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.068</td>
<td>5225</td>
<td>1.00</td>
<td>5195</td>
<td>173.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.049</td>
<td>5126</td>
<td>1.00</td>
<td>5094</td>
<td>158.46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4950</td>
<td>4</td>
<td>0.768</td>
<td>5503</td>
<td>1.00</td>
<td>5475</td>
<td>580.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.704</td>
<td>5278</td>
<td>1.00</td>
<td>5215</td>
<td>525.80</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.668</td>
<td>5125</td>
<td>1.00</td>
<td>5074</td>
<td>359.23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: results for small, medium and big instances

References


