The Regenerator Location Problem

Si Chen* Ivana Ljubić† S. Raghavan‡

*College of Business and Public Affairs
Murray State University, Murray, KY 42071, USA
si.chen@murraystate.edu

†Faculty of Business, Economics and Statistics
University of Vienna, Brünnerstr. 72, 1210 Vienna, Austria
ivana.ljubic@univie.ac.at

‡Robert H. Smith School of Business & Institute for Systems Research
University of Maryland, College Park, MD 20742, USA
raghavan@umd.edu

September 5, 2007; Revisions May 14, 2008; October 16, 2008

Abstract

In this paper we introduce the regenerator location problem (RLP), which deals with a constraint on the geographical extent of transmission in optical networks. Specifically, an optical signal can only travel a maximum distance of $d_{\max}$ before its quality deteriorates to the point that it must be regenerated by installing regenerators at nodes of the network. As the cost of a regenerator is high we wish to deploy as few regenerators as possible in the network, while ensuring all nodes can communicate with each other. We show that the RLP is NP-Complete. We then devise three heuristics for the RLP. We show how to represent the RLP as a max leaf spanning tree problem (MLSTP) on a transformed graph. Using this fact we model the RLP as a Steiner arborescence problem (SAP) with a unit degree constraint on the root node. We also devise a branch-and-cut procedure to the directed cut formulation for the SAP problem. In our computational results over 740 test instances, the heuristic procedures obtained the optimal solution in 454 instances, while the branch-and-cut procedure obtained the optimal solution in 536 instances. These results indicate the quality of the heuristic solutions are quite good, and the branch-and-cut approach is viable for the optimal solution of problems with up to 100 nodes. Our approaches are also directly applicable to the MLSTP indicating that both the heuristics and branch-and-cut approach are viable options for the MLSTP.

Keywords: optical network design; Steiner arborescence problem; branch-and-cut; heuristics; greedy; maximum leaf spanning tree problem

1 Introduction

The ever-increasing usage of digital communications has inspired the development of a variety of new applications in business and consumer markets. Most recently, new services such as real-time

online gaming, voice over IP (VOIP), video sharing and mobile Internet, all add a significant amount of traffic to telecommunications networks. For instance, YouTube, the leading video-sharing site, is currently serving 100 million videos per day, with more than 65,000 videos being uploaded daily [15]. Over the past ten years telecommunications service providers have built optical networks to meet the exponentially growing demand of users. One of the key benefits of optical networks is the large capacity they provide. For example an OC-192 carrier has a speed of 9.952 gigabits per second (Gbps). Given the fact that a fiber optic cable can consist of upto thousand fibers, and modern technologies like Wave Division Multiplexing (which use multiple wavelengths to transmit signals on the same fiber thereby increasing its capacity) are frequently used this has meant virtually unlimited amounts of bandwidth can be sent over an optical network.

There has been considerable research devoted to the design of optical networks. These include research on network architectures and infrastructure, control and management, protection and survivability. In this paper we address a problem called the regenerator location problem (RLP), which deals with a fundamental problem related to the geographical extent of transmission in optical networks.

To elaborate, the strength of an optical signal deteriorates as it gets farther from the source due to transmission impairments in the fiber (attenuation, dispersion, cross-talk). In other words, the distance an optical signal may be sent without losing or falsifying the information is limited. Therefore, it is necessary to regenerate the signals periodically using regenerators. In practice, there are three forms of signal regeneration, namely 1R (reamplification), 2R (reamplification and reshaping) and 3R (reamplification, reshaping and retiming) [1, 11, 16]. They differ in the sense that each is increasingly more complex. In 1R regeneration the optical signal is simply reamplified using a pump laser [16], and is comparatively cheap. However, it can only be done a certain number of times before the quality of a signal is such that it needs to be reshaped and possibly retimed in addition to reamplification. 2R and 3R regeneration is fairly complex, and due to technological constraints is done by converting the optical signal back to an electrical signal, before converting it back to an optical signal again. Given these current technological hurdles with 2R and 3R signal regeneration the associated equipment is usually expensive and it tends to be done at nodes of a telecommunication network. Consequently, our desire is to deploy as few of these regenerators as possible, while ensuring all nodes can communicate with each other (i.e., send a signal to each

other). Our focus within this paper is solely on the installation of 3R regenerators. (We note that in practice 3R regeneration seems more prevalent than 2R regeneration, perhaps because of the fact that given the optical signal is converted to an electrical one, it is best to retime it as well to make it virtually identical to the original optical signal.)

Network design in practice is typically done in a hierarchical fashion. Consequently, by addressing the RLP at the outset of the network design process a planner will ensure that regenerators are placed at nodes of the network so that all nodes of the network may communicate without worry of physical impairments of the signal. This greatly simplifies the design process.

Mathematically, the RLP problem we consider can be described as follows. Given a network $G = \{N, F, D\}$, where N is the set of nodes, F is the set of edges, and D is the associated distance matrix of edges, and a maximum distance of $d_{\max}$ that determines how far a signal can traverse before its quality deteriorates and needs to be regenerated. Determine a minimum cardinality subset of nodes L such that for every pair of nodes in N there exists a path in G with the property that there is no subpath (i.e., a subsequence of edges on the path) with length $\geq d_{\max}$ without internal regenerators (i.e., we do not include the end point of the subpath).

Although the geographical or physical extent that a signal can travel is an important issue, it seems to have been largely ignored by the academic literature on telecommunication network design. In our literature search, we have only come across two papers [8, 14] that discuss the issue of regenerator placement within the context of a large network design problem. In this paper we focus on the RLP as a stand alone problem. We prove that, despite its simple description, it is NP-complete and devise three high-quality heuristics for it. We also show how the RLP can be modeled as a Steiner arborescence problem (SAP) with a unit degree constraint on the root node. We develop a branch-and-cut algorithm for the directed cut formulation of the SAP. This provides optimal solutions in many instances, and a lower bounding mechanism to compare the quality of the heuristics.

The rest of the paper is organized as follows. In Section 2 we prove that the RLP is NP-complete. In Section 3 we develop three heuristics for the RLP. In Section 4, we formulate RLP as a SAP with a unit degree on the root node. We then develop our branch-and-cut algorithm on this model. Section 5 discusses the weighted version of RLP. Section 6 presents the computational results from our proposed heuristics and compares them with the solutions obtained from the

branch-and-cut procedure on the SAP formulation. Section 7 discusses how our heuristics and branch-and-cut procedure can be applied to the maximum leaf spanning tree problem. Section 8 presents concluding remarks.

2 Preliminaries

In this section we prove that the RLP is NP-complete. We also describe a graph transformation procedure that greatly simplifies conceptualization of the RLP and is very useful to our subsequent development of heuristics and an exact procedure for the problem.

2.1 NP-Completeness of the RLP

Theorem 1. *The regenerator location problem is NP-complete.*

Proof. Consider the Vertex Cover Problem (VCP), which is stated as follows [6].
Instance: An undirected graph $G = (U, E)$ and a positive integer $K \leq |U|$.
Question: Is there a subset U' of U with $|U'| \leq K$ such that U' contains at least one of the two endpoints of each edge in E?

We now construct the corresponding instance of the RLP. Create a node for every $u \in U$. Create an edge between every pair of nodes in U. For every edge $e_i = \{e_i^1, e_i^2\}$ in E, create a pair of nodes v_i and w_i. Connect v_i to nodes e_i^1 and e_i^2 (the endpoints of edge e_i), and w_i to nodes e_i^1 and e_i^2. Set the length of all edges in the resulting graph to $d_{\max}$.

The question in this new graph is whether there is a feasible solution (a set of nodes L where we place regenerators) to the RLP with cardinality less than or equal to K. Observe that in the RLP

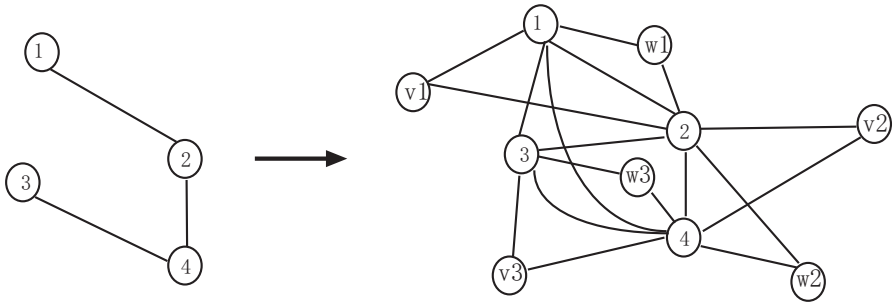


Figure 1: Transforming a vertex cover problem to a regenerator location problem.

obtained from transforming a VCP, a feasible solution need not place a regenerator at the v_i and w_i nodes. This is due to the fact that the nodes in U are fully connected in the graph corresponding to the RLP problem. Thus a feasible solution with a regenerator at a v_i or w_i node remains feasible when the regenerator is removed from that node. With this it is easy to observe that an instance of VCP has a "yes" answer if and only if the corresponding RLP has a "yes" answer. As this is a polynomial transformation, the decision version of the RLP is NP-complete. □

2.2 The Communication Graph

We now describe the graph transformation procedure that we use on the RLP. As we mention, it is very useful to our subsequent development of heuristics and an exact procedure for the problem.

Given a graph $G = \{N, F, D\}$, and a maximum distance of $d_{\max}$, apply the all pairs shortest path algorithm. Replace edge lengths by the shortest path distance. If the edge length is less than or equal to $d_{\max}$ then keep the edge, and if the edge is greater than $d_{\max}$ delete the edge. Denote this new graph as M (with node set N and edge set E). Observe then, if M is a complete graph, no regenerators are required. On the other hand every node pair that is not connected by an edge in M requires regenerators to communicate. We call such node pairs "not directly connected" or NDC node pairs. In other words M can be viewed as a communication graph, where an edge between a pair of nodes represents that they can communicate, while a node pair that is not connected by an edge requires regenerators to communicate. It suffices to consider the RLP problem on the transformed graph M and determine the minimum cardinality subset of nodes L, such that for the NDC node pairs in M there exists a path with regenerators at all internal nodes on the path. Observe the following property in the communication graph M. Suppose we place a regenerator at a node t. Then every node pair that is not directly connected in M, but that is connected to t can communicate. Consequently, after the placement of a regenerator at node t, such node pairs can be viewed as being directly connected (i.e., can communicate with each other) and the communication graph M can be updated with edges between such nodes. In this setting, our objective then is to minimize the number of nodes where regenerators are placed so that M becomes fully connected. Notice that if the communication graph M is not a connected graph, then the RLP is infeasible. Thus feasibility of the RLP can easily be checked in polynomial time. For the rest of the paper, we assume without loss of generality that M is a connected graph (and we refer to the communication

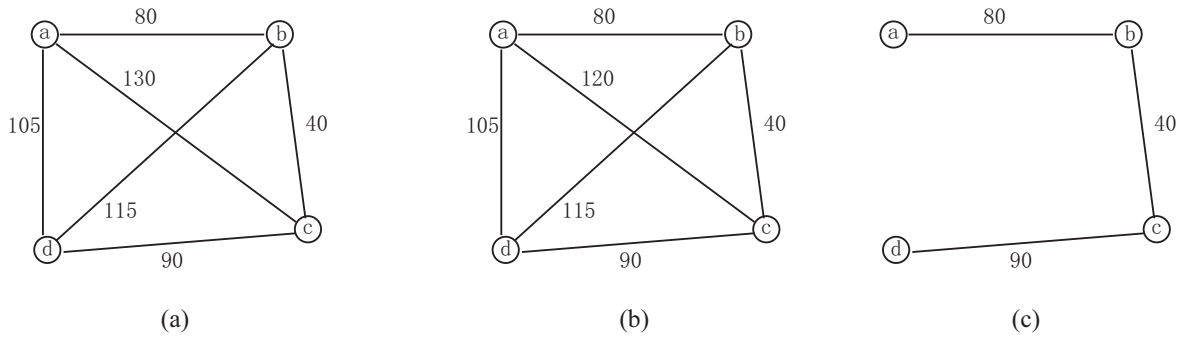


Figure 2: An example of the graph transformation procedure for a network with four nodes and $d_{\max} = 100$. (a) original graph, (b) edge lengths replaced by shortest path lengths, (c) edges longer than $d_{\max}$ are deleted.

graph by M).

Figure 2-(a) illustrates a small example of this graph transformation procedure for a network with four nodes. Here $d_{\max}$ is set to 100. The numbers besides the edges are the edge distances, which are replaced by the shortest path lengths in Figure 2-(b). Notice that edge (a, c), (a, d), and (b, d) are longer than 100, and thereby are removed in the new graph M as shown in Figure 2-(c). A possible regenerator deployment that can connect all the nodes is to add two regenerators, one at node b and the other at node c. Observe, with this, there exists a path between every pair of nodes where a signal does not travel more than $d_{\max}$ before being regenerated.

3 Heuristics for the RLP

In this section we discuss a problem reduction procedure and three heuristics, referred to as Greedy, H1 and H2 for the RLP. The problem reduction procedure (that we refer to as a preprocessing step) has two benefits. In addition to reducing the problem size it fixes regenerators in the solution.

3.1 Preprocessor

Observe that if node i in the communication graph M is only connected to one other node j, i.e., i has degree one, every feasible solution must include a regenerator deployed at node j. Once a regenerator is deployed at node j, node i can be eliminated from M. Our preprocessor repeatedly applies this idea and is described as follows.

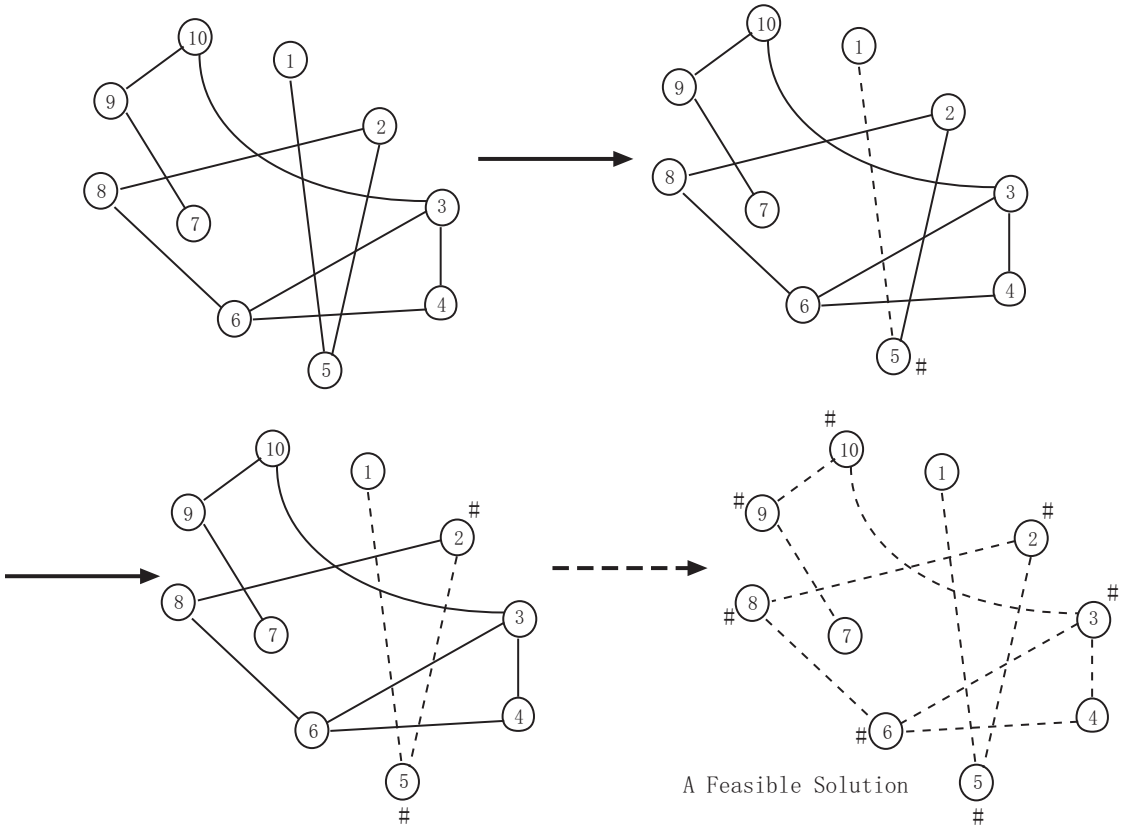


Figure 3: Preprocessor finds the optimal solution.

1. Initialization: $L = \emptyset$.

2. Scan nodes to identify a unit degree node. If none exists go to Step 4. Let n_1 denote the identified node with degree 1, and n_2 denote its neighbor. Delete n_1 from M ($M = M \setminus n_1$). If n_2 is already in L, repeat Step 2.

3. Let $\deg(n_2)$ be the degree of node n_2 in M.

- If $\deg(n_2) = 0$ and n_2 is the only node left in M, L is a feasible solution. Stop.
- If $\deg(n_2) \geq 1$, add n_2 to L. Go to Step 2.

4. Update the communication graph M, i.e., remove all the NDC node pairs that can be connected after deploying regenerators in L, and adding to M the edges associated with such node pairs.

Figure 3 gives an example where we obtain a feasible solution after applying the preprocessor. In the first step, the preprocessor detects that node 1 has degree one. Thus node 1 is eliminated

from the graph and a regenerator is added to node 5 (the node that is connected to node 1). In the resulting graph, node 5 has degree one, so the preprocessor removes node 5 and adds a regenerator to node 2. The process continues until no such node can be found. The final regenerator deployment (nodes with # are chosen regenerator locations) in this example is feasible. Observe that the preprocessor finds the optimal solution for any graph M that is a spanning tree.

Let $\bar{M} = (\bar{N}, \bar{E})$ denote the graph obtained just before Step 4 of the preprocessing algorithm is executed. Then the running time of the preprocessor procedure is $O(|N| + |\bar{L}||\bar{N}|^2)$, where $\bar{L} = L \cap \bar{N}$. This can be argued by observing that initially scanning for a node of degree 1 takes $O(|N|)$ time. In this scanning step all nodes with degree 1 can be identified and added to a list. Later, as we delete edges in the graph if any degree 1 nodes are created we add them to the list of degree 1 nodes. Updating the graph when an edge (and the associated degree 1 node) is deleted takes $O(1)$ time. Since edge deletion is always accompanied by node deletion, the total number of deletions is at most $|N|$ and this step takes $O(|N|)$ time. Step 3 of the preprocessing algorithm is executed at most once for each node and thus it takes at most $O(|N|)$ time. Thus the first 3 steps take $O(|N|)$ time. The graph that is updated in step 4 is $\bar{M}$. Updating the adjacency information for each of the neighbors of a given node in $\bar{L}$ takes $O(|\bar{N}|^2)$ time, and thus Step 4 takes $O(|\bar{L}|\bar{N}|^2)$ time.

We note that the preprocessing procedure can be generalized further (we have not implemented this generalization yet within our computational experiments). This generalization is based on the observation that if M is not node-biconnected (we will drop the prefix node and use the term biconnected to refer to a node-biconnected graph), it is necessary to place regenerators at all cutpoints [9] of M. Recall, a cutpoint of a graph is a node whose deletion increases the number of components of the graph. We can easily prove the necessity of placing regenerators at cutpoints as follows. Assume there is a solution L in which there is a cut point c of M without an associated regenerator. Assume without loss of generality that after deletion of c the graph M is disconnected into two components C_1 and C_2. Then it follows directly that all node pairs (i, j), with $i \in C_1$ and $j \in C_2$ are still NDC nodes. In other words the solution L is not feasible, which is a contradiction. We note further, that the problem can be reduced in size (i.e., decomposed into multiple smaller problems) by observing that it suffices to solve the RLP problem separately on the blocks [9] of M (recall that blocks correspond to maximal biconnected subgraphs of a graph). The solution to the

overall problem is obtained as the union of the solutions to the RLP on each block together with the regenerators installed at each of the cutpoints of M. Detecting cutpoints and determining the biconnected components of a graph (i.e., the blocks) requires $O(|N| + |E|)$ time [13]. After that we need to update the adjacency information in each of the blocks of M based on the installed regenerators which takes $O(|L||N|^2)$ time.

After the preprocessing step,[1] the RLP is defined on a graph $\bar{M} = (\bar{N}, \bar{E})$, with a set of regenerators already installed at the nodes $\bar{L} \subset \bar{N}$. We now show that this problem can be considered as a RLP on the graph $\bar{M} = (\bar{N}, \bar{E})$ with no regenerators installed at the nodes (i.e., with $\bar{L} = \phi$). In other words while solving the RLP problem on $\bar{M}$ we do not need to take into account nodes where regenerators have already been pre-installed.

Lemma 1. *Suppose we consider the RLP on $\bar{M} = (\bar{N}, \bar{E})$ with a set of regenerators already installed at the node set $\bar{L}$. A minimal solution for the RLP (i.e., one in which removal of any of the regenerators will make the solution infeasible) on $\bar{M}$ that ignores the pre-installed regenerators (i.e., that assumes $\bar{L} = \phi$) will not install regenerators at nodes in $\bar{L}$.*

Proof. Since the graph $\bar{M}$ was updated in the preprocessing based on the installation of regenerators at $\bar{L}$, removal of a regenerator installed at a node in $\bar{L}$ does not make the problem infeasible (alternatively installing a regenerator at a node in $\bar{L}$ does not reduce the number of NDC node pairs). In other words in the graph $\bar{M}$, for any node $i \in \bar{L}$, all of i's neighbors are adjacent to each other. Thus installing a regenerator at i does not reduce the number of NDC node pairs. □

Consequently, in our heuristic procedures we ignore the set of pre-installed regenerators in the reduced graph obtained after the preprocessing step. Also, for convenience, from hereon we will refer to the graph obtained after the preprocessing as M (instead of $\bar{M}$).

3.2 Greedy Heuristic

We now describe our first heuristic that employs a simple "greedy" strategy. The greedy heuristic considers the communication graph M and tries to find the node which can eliminate the greatest

[1]The notation here assumes we applied the simpler preprocessing procedure (since this is what we did in our computational experiments). However, the argument that follows applies to any graph $\bar{M}$ that has been updated based on having a set of regenerators already installed at the node set $\bar{L}$. Thus it applies to each block of M, after the blocks have been updated based on pre-installed regenerators.

number of NDC node pairs in M if a regenerator is deployed at its location. It adds a regenerator to this node, updates the graph M, and repeats (i.e., again looks for the node which can eliminate the greatest number of NDC node pairs in M if a regenerator is deployed at its location) until a feasible solution is reached.

We now discuss the running time of the greedy heuristic. In each iteration, the heuristic calculates for each node the number of NDC node pairs ($nr(i)$) it can reduce if a regenerator is added at its location. It takes $O(|N|^2 - |E|)$ time (the number of NDC node pairs) to find ($nr(i)$) for one node, and thus $O(|N|^3 - |N||E|)$ time for all the nodes in the graph. It also takes $O(|N|)$ comparisons to find the node with the largest value of $nr(i)$. Once this node is identified, we add a regenerator to its location, update M (which takes $O(|N|^2)$ time), and check to see if the solution is feasible (this step takes $O(|N|)$ time since it needs to check whether M is fully connected after the graph update). Since we can at most add N regenerators, the running time of the greedy heuristic is $O(|N|^4 - |N|^2|E|)$ or $O(|N|^4)$. Note that this is a worst-case running time bound and the actual running time is much faster.

3.3 Heuristic H1

Our next heuristic that we refer to as H1 is based on the following observation that states that a minimal solution for the RLP can be represented as a spanning tree that has regenerators at all internal nodes of the spanning tree.

Lemma 2. *Any minimal solution for the RLP on M can be represented as a spanning tree with regenerators at all internal nodes of the tree.*

Proof. Let L be an arbitrary minimal solution for the RLP on M. Observe that the node-induced subgraph of M on the node set L (i.e., the graph constructed on the nodes of L together with any edges in M that have both endpoints in L) must be connected. Otherwise, the solution L is infeasible. Consequently, we can construct a spanning tree on the node-induced subgraph of M on the node set L.

We now argue how the minimal solution L can be represented as a spanning tree with regenerators at all internal nodes of the tree. Let T be a spanning tree on the node-induced subgraph of M on the node set L. Recall, since L is a minimal solution, deletion of any node in L will result in

at least one node pair that can no longer communicate with each other. This means for each node $i \in L$, there is at least one node $a(i)$ in $N \backslash L$ that is adjacent to i in M and is not adjacent to any of the nodes in $L \backslash i$. We expand T by adding for each $i \in L$ the node $a(i)$ together with the edge from $a(i)$ to i. Observe that T remains a tree (that is a subgraph of M), but now all the nodes in L are internal nodes of the tree. For the nodes of M that are not yet in T observe that each one of them must have an edge to at least one of the nodes in L. Thus, we add each one of these nodes in $N \backslash \{T \cup_{i \in L} a(i)\}$ to T together with an edge to exactly one of the nodes in L that it has an edge to in M. Observe T is a spanning tree on M with internal nodes L. □

Lemma 2 has some very interesting implications. It suggests that a solution to the RLP can be found by finding the spanning tree T on M that has the fewest internal nodes, or the maximum number of leaf nodes. This corresponds to the maximum leaf spanning tree problem (MLSTP) which is known to be NP-complete [6].

Observe that a feasible solution to the RLP can be obtained by constructing a spanning tree on M and installing regenerators at all internal nodes of the tree. Thus, any heuristic that constructs a spanning tree solution will generate feasible solutions to the RLP. Our heuristic H1 aims to find a spanning tree on M with the maximum number of leaf nodes (i.e., with the fewest number of internal nodes). It uses a subroutine called TREE(i) that is essentially a graph search algorithm. This algorithm aims to identify high degree nodes as candidates for internal nodes in the spanning tree.

Heuristic H1 has the following steps.

1. Initialization. $S = L = \emptyset$, $C_i = \emptyset$, pred$[i] = -1$ and visit(i)= **false** $\forall i \in N$.

2. Find the node i that has the lowest degree.

3. Call $L =$ TREE(i).

The pseudo-code for TREE(i) is provided in Figure 4. The set S denotes the set of the nodes that are in the spanning tree under construction. $E(S)$ denotes the set of edges that have both endpoints in S. L denotes the current set of regenerator locations. C_i denotes the nodes adjacent to node i that are not in S. The predecessor index pred$[i]$ obtained at the conclusion of the algorithm defines the spanning tree. The intuition behind TREE(i) is that if we add a regenerator to a node

```
TREE(i)
{
   visit(i) = true;
   For each neighbor j of i that is not in S, add it to the list C_i and set pred[j] = i;
   If S = ∅, let S = S ∪ i; Otherwise let S = S ∪ i ∪ C_i;
   While not all the nodes in C_i are visited
   {
      Among all the unvisited nodes in C_i find the node c that has the largest degree,
      D_c, in the graph {N, E\E{S}};
      If D_c > 0
      {
         L = L ∪ c;
         L = TREE(c);
      }
      Else, return L;
   }
   Return L;
}
```

Figure 4: Steps of TREE(i).

with the largest degree in $E\backslash E(S)$ (i.e., choose it as a non-leaf node), there is a better chance that we can connect more NDC node pairs and have fewer internal nodes in the spanning tree under construction.

We illustrate H1 using a small example shown in Figure 5. In the graph, node 1 has the lowest degree, thus H1 starts by calling TREE(1). We have $C_1 = \{2, 8\}$, pred[2] = pred[8] = 1, $S = \{1\}$ and $E\{S\} = \emptyset$. The degrees of node 2 and node 8 in $E\backslash E(S)$ are both three (we call the degree of nodes in $E\backslash E(S)$ the revised degree). Breaking ties arbitrarily, suppose H1 chooses node 2, then it calls TREE(2). We have $C_2 = \{7, 8\}$, pred[7] = pred[8] = 2, and $S = \{1, 2, 8, 7\}$. Since the revised degree of node 7 is four, which is greater than the revised degree of node 8, H1 selects node 7 and calls TREE(7). We have $C_7 = \{3, 6, 4, 5\}$, pred[3] = pred[6] = pred[4] = pred[5] = 7 and $S = \{1, 2, 8, 7, 3, 6, 4, 5\}$. The revised degree of every element of C_7 is zero. Thus H1 stops and returns $L = \{2, 7\}$.

We now show that the running time of H1 is $O(|N| + |E|)$. It takes $O(|N|)$ comparisons to find the node with the lowest degree initially. H1 then calls the TREE subroutine. TREE(i) takes $O(\alpha_i)$

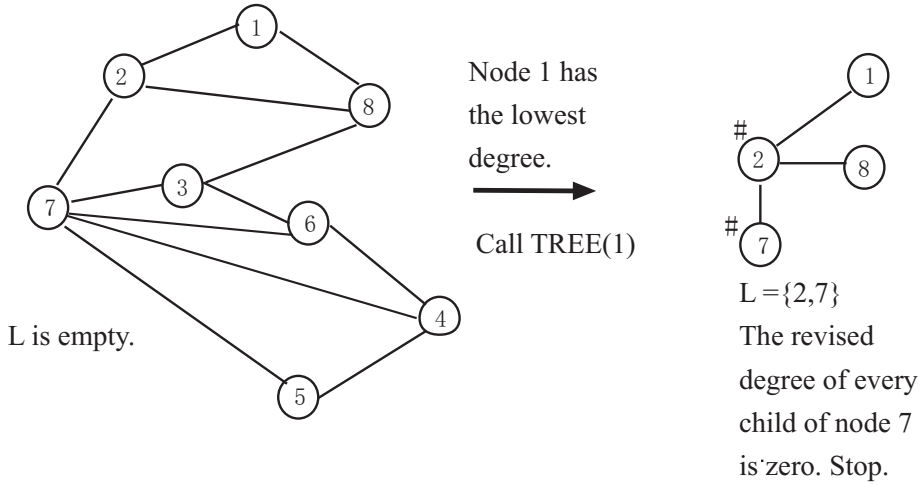


Figure 5: Application of heuristic H1 to a small example.

time (α_i is the revised degree of node i) to check all the neighbors of node i and adds those that are not in either S or L to set C_i. It then takes $O(|C_i|)$ time to obtain the revised degrees of the nodes in C_i. (This can be done by keeping track of the revised degrees during the course of the algorithm. Every time a node i is added to S one needs to only update the revised degrees of its neighbors that are not in the tree.) It also takes $O(|C_i|)$ comparisons to find the node c with the largest revised degree. The subroutine continues recursively by calling TREE(c). Each run of the TREE subroutine thus takes $O(\deg(i))$ time (since α_i and C_i are bounded above by the degree of node i, i.e., $\deg(i)$). The TREE subroutine is called at most $|N|$ times, once for each node in the graph. Thus the total running time for the TREE subroutine is bounded by $O(\sum_{i=1}^{N}\deg(i)) = O(|E|)$. Thus, the running time of H1 is $O(|N| + |E|)$.

3.4 Heuristic H2

Heuristic H2 is a hybrid between the greedy algorithm and H1. In each iterative step it identifies the node with the lowest degree. It then examines the neighbors of this node and selects as a candidate for regenerator placement the neighbor with the highest degree. It then updates the graph M, and repeats the iterative step. It is similar to H1 in the sense that it starts off by identifying the lowest degree node and selecting its neighbor with the highest degree as the node to place a regenerator at. It is similar to greedy in the sense that it updates the graph after this step, and then repeats this basic iterative step. H2 has the following steps.

```
Node(i)
{
    Among all the nodes adjacent to node i that are not in L
    find the node c that has the largest degree;
    Return c;
}
```

Figure 6: Steps of Node(i).

1. $L = \phi$.

2. Find the node i that has the lowest degree. If the degree of node i is equal to $|N| - 1$, the solution is feasible, go to Step 4.

3. Call $u =$ Node(i). $L = L \cup u$, update the graph and go to Step 2.

4. Return L.

We provide the pseudo-code for Node(i) in Figure 6.

We now show that the running time of H2 is $O(|N|^3)$. H2 first takes $O(|N|)$ to find the node i with the lowest degree. It then calls Node(i) which spends $O(\deg(i))$ time checking the neighbors of i and finding the node c which has the largest degree and is not yet in L. H2 adds a regenerator at node c and spends $O(|N|^2)$ time updating the adjacency lists. Thus each iteration of H2 takes $O(|N|^2)$ time. Since we can at most add $|N|$ regenerators, Node(i) is called for at most $|N|$ times. Thus, the running time for H2 is bounded by $O(|N|^3)$.

3.5 Post-optimizer

We now discuss a post-optimizer step that we apply to the solutions output by all three of our heuristics in order to try and improve upon them. It consists of a subroutine that we call **2-for-1**. **2-for-1** tries to replace two regenerator locations in the current solution with a single regenerator location (that is not currently a regenerator location). When **2-for-1** results in a feasible solution, its application reduces the number of regenerators in the solution by 1. Figure 7 illustrates an example where **2-for-1** improves the solution. The initial solution has three regenerators deployed at nodes 5, node 8 and node 10, respectively. Observe if we remove any of the regenerators the

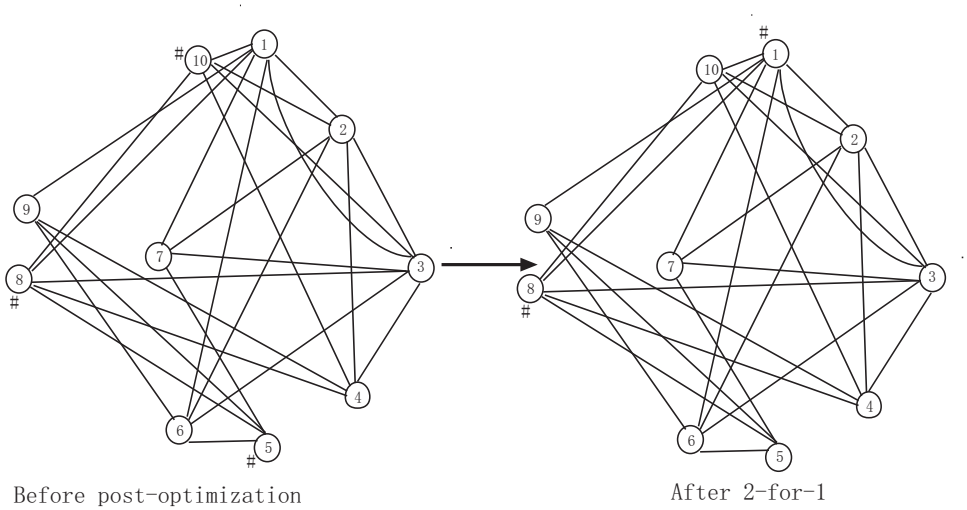


Figure 7: Illustration of **2-for-1**. Replacing nodes 5 and 10 with node 1 results in a feasible solution.

problem is no longer feasible. However, **2-for-1** replaces the two regenerators at node 5 and node 10, by one regenerator at node 1. The resulting solution has two regenerators (at node 1 and node 8) instead of three.

Figure 8 describes how a single **2-for-1** move can be implemented. For a given graph M, regenerator locations L, and a pair of regenerator locations i and j ($i, j \in L$), it tries to find a new location k, $k \notin L$, that can replace i and j. In the notation, for each node $i \in N$, we denote with $\mathcal{N}(i) = \{k \mid (k, i) \in E\}$ the set of its neighbors in M.

To understand why the implementation in Figure 8 works, consider a spanning tree T_L of M such that all its inner nodes are in L. We note that **2-for-1** moves are only interesting in the case that $|L| \geq 3$. When $|L| = 2$ and we search for an improvement, we only need to check if M contains a star, which can be done in constant time if nodes in M are sorted in decreasing order with respect to their size of adjacency lists. Nodes $k \in \mathcal{N}(i) \cup \mathcal{N}(j) \setminus L$ are leaves in T_L. Some, but not necessarily all of them, are connected to nodes i or j in T_L. Set *ToConnect* consists of those leaves whose adjacency lists intersected with L only consists of nodes i or j. When deleting i and j from L, we need to construct a new spanning tree T_N of M such that i and j become leaves and all the leaves from *ToConnect* get connected through a new common regenerator. Therefore, the

```
2-for-1(M, L, i, j)
{
    NewRegen = N \ L;
    ToConnect = ∅;
    For all k ∈ N(i) ∪ N(j) \ L do:
    {
        If N(k) ∩ (L \ {i, j}) = ∅ then {
            NewRegen = NewRegen ∩ N(k);
            ToConnect = ToConnect + k;
            If NewRegen = ∅ then STOP. No feasible move found.
        }
    }
    Place a regenerator randomly at one node from NewRegen.
}
```

Figure 8: A single **2-for-1** move. Set *NewRegen* contains all possible positions where a new regenerator can be placed.

only feasible **2-for-1** moves[2] for selected i and j are those for which

$$\bigcap_{k \in ToConnect} \mathcal{N}(k) \setminus L \neq \emptyset.$$

Consequently, the time complexity of a single **2-for-1** move is $O(|N|^2)$ in the worst case. Noting that the size of the **2-for-1** neighborhood is $O(|L|^2)$ indicates that the application of 2-for-1 takes $O((|L||N|)^2)$ or $O(|N|^4)$ time.

If we were to apply **2-for-1** repeatedly until there is no further improvement in the solution, there would be at most $|L| - 1$ applications (as the number of regenerators decreases by 1 in each successful step or there are no improving moves in the neighborhood) resulting in $O(|N|^5)$ time. In our computational experiments, we applied **2-for-1** until there was no further improvement in our solution.

[2]Note that if $ToConnect = \emptyset$, a **2-for-1** move can also lead to an **2-for-0** improved solution in which 2 regenerator locations are simply deleted from L. However, such moves do not occur in our solutions, as the solutions produced by the heuristics are minimal.

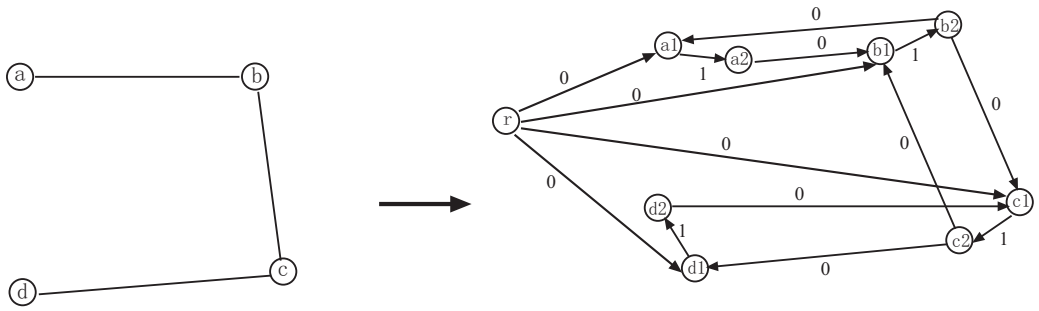


Figure 9: The transformed graph H from M.

4 Branch-and-Cut for the RLP

In this section, we show how the RLP can be viewed as a SAP with a unit degree constraint on the root node. We then develop a branch-and-cut procedure for the RLP.

4.1 Formulating the RLP as a Steiner Arborescence Problem with a Unit Degree Constraint

We first transform M to a new directed graph H as follows.

- For every node i in M create two nodes i_1 and i_2 in H, and add an arc $< i_1, i_2 >$ in H with cost $c_{i_1 i_2} = 1$. Denote the set of nodes with index 1 in H as V_1, the set of nodes with index 2 in H as V_2, and the set of all nodes in H as $V = V_1 \cup V_2$. Denote the set of unit cost arcs added as A_1.

- For every edge (i, j) in M, add two arcs $< i_2, j_1 >$ and $< j_2, i_1 >$ to H with costs $c_{i_2 j_1} = c_{j_2 i_1} = 0$. Denote these arcs by A_2.

- Create a dummy root node r, add an arc from r to each $i_1 \in V_1$ with cost $c_{r i_1} = 0$. Denote these arcs by A_r.

Figure 9 illustrates the transformation for the example in Figure 2-(c).

Let A denote the arcs in H, i.e. $A = A_1 \cup A_2 \cup A_r$. On the directed graph $H = (V \cup \{r\}, A)$, we wish to find the minimum cost Steiner arborescence rooted at node r that spans the terminal nodes V_1, and that has a unit degree at the root node. We refer to this unit degree constrained

SAP as the unit degree SAP. We claim that the solution to the unit degree SAP correctly models the RLP.

Lemma 3. *The solution to the unit degree SAP on H solves the RLP on M.*

Proof. When we impose a unit degree on the root node it ensures that the graph obtained when node r is deleted from the solution is connected. Observe that in a solution to the unit degree SAP on H at most one of the arcs $< i_2, j_1 >$ and $< j_2, i_1 >$ will be in the solution (otherwise we will have a directed cycle). Thus a feasible solution (i.e., an arborescence R with terminal nodes V_1 and unit degree at the root r) for the unit degree SAP on H, can be obtained by including in T the edges on M that correspond to the arcs in $R \cap A_2$.

Notice that when a terminal node i_1 is a leaf node in the Steiner arborescence R, no cost is incurred on arc $< i_1, i_2 >$, and i is a leaf node in the corresponding tree T on M. On the other hand when a terminal node i_1 is an internal node of the Steiner arborescence R, the only way out of i_1 is on the arc $< i_1, i_2 >$ with a unit cost. Consequently, when terminal node i_1 is an internal node of the Steiner arborescence a unit cost is incurred, and i is an internal node in the corresponding tree T on M.

Since we wish to minimize the cost of the Steiner arborescence, the optimal solution maximizes the number of leaf nodes in the arborescence among the nodes in V_1. Recalling that the RLP can be viewed as a MLSTP on M completes the argument. □

We note that rather than introducing the degree constraint on the root node explicitly in our formulation, it is also possible to remove the constraint and set a large cost on the arcs from the root node. A consequence of Lemma 2 is that the maximum number of regenerators in a solution is $|N| - 2$ (because the maximum number of internal nodes in a tree is $|N| - 2$). Consequently, it is sufficient to set the cost of the arcs in A_r to $|N|$. We found that imposing the degree constraint explicitly was beneficial in terms of computational speed. Consequently, we explicitly include a unit degree constraint on the root node.

We now describe our formulation for the Steiner arborescence problem. To model an arborescence $R = (V_R \cup \{r\}, A_R)$, $V_R \subset V, A_R \subset A$, we use the following binary variables:

$$x_{ij} = \begin{cases} 1, & \text{if } < i,j > \in A_R \\ 0, & \text{otherwise} \end{cases} \quad \forall < i,j > \in A$$

$$y_k = \begin{cases} 1, & \text{if } k \in V_R \\ 0, & \text{otherwise} \end{cases} \quad \forall k \in V_2$$

For convenience we introduce the following notation: A set of vertices $S \subseteq V$ and its complement $\overline{S} = (V \cup \{r\}) \setminus S$ induce two directed cuts: $\delta^+(S) = \{< i,j > \mid \ i \in S, j \in \overline{S}\}$ and $\delta^-(S) = \{< i,j > \mid \ i \in \overline{S}, j \in S\}$. We also write $x(A') = \sum_{<i,j> \in A'} x_{ij}$ for any subset of arcs $A' \subset A$. The corresponding integer linear programming model then reads as follows (see also [3]):

$$\min \quad \sum_{<i,j> \in A} c_{ij} x_{ij} \tag{1}$$

$$x(\delta^-(\{k\})) = 1 \qquad \forall k \in V_1 \tag{2}$$

$$x(\delta^-(\{k\})) = y_k \qquad \forall k \in V_2 \tag{3}$$

$$x(\delta^-(S)) \geq 1 \qquad \forall S \subseteq V,\ S \cap V_1 \neq \emptyset,\ |S| \geq 2 \tag{4}$$

$$x(\delta^-(S)) \geq y_k \qquad \forall S \subseteq V,\ k \in S \cap V_2,\ |S| \geq 2 \tag{5}$$

$$x(\delta^+(\{r\})) = 1 \tag{6}$$

$$x_{ij}, y_k \in \{0,1\} \qquad \forall < i,j > \in A,\ k \in V_2 \tag{7}$$

Indegree requirements are expressed by inequalities (2) and (3). The *cut constraints* (4) and (5) ensure the connectivity of the solution: for each terminal node $k \in V_1$, there must be a directed path from the root to k. The same holds for each Steiner node $k \in V_2$ with indegree 1.

4.2 Branch-and-Cut Algorithm

Our branch-and-cut algorithm is implemented using the commercial package CPLEX 10.0. The general framework follows the description given in Ljubić et al. [10]. Therefore we only mention the basic concepts that make our method computationally efficient:

Initialization In the initialization phase, we explicitly add the constraints

$$x_{i_2 j_1} + x_{j_2 i_1} \leq 1, \quad \forall i_1, j_1 \in V_1, i_2, j_2 \in V_2, < i_2, j_1 >, < j_2, i_1 > \in A. \tag{8}$$

Note that these inequalities do not strengthen the formulation, because they are a special case of the *generalized subtour elimination constraints* (GSECs) for the tours given by arcs $\{< i_1, i_2 >, < i_2, j_1 >, < j_1, j_2 >, < j_2, i_1 >\}$

$$x_{i_1 i_2} + x_{i_2 j_1} + x_{j_1 j_2} + x_{j_2 i_1} \leq 1 + y_{i_2} + y_{j2}.$$

The equivalence of the GSEC model and the directed cut model for the SAP is well-known. See Goemans and Myung [7] or Chimani et al. [2], for example. Observe that by construction of the graph H and by indegree inequalities (3), $x_{j_1 j_2} = y_{j_2}$ and $x_{i_1 i_2} = y_{i_2}$, and thus inequality (8) follows immediately. Although a GSEC based model is equivalent to the directed cut model, introducing some GSECs, as in (8), has the benefit that they may significantly reduce the runtime needed for the separation of the corresponding cut inequalities. This is what we found in our preliminary computational experiments.

Asymmetry Constraints Symmetry is a significant problem with the SAP representation of the RLP. Consider the optimal solution L (or any minimal solution L) to the RLP. As we argued in the proof of Lemma 2 *any spanning tree* T on M with internal node set L represents this solution. In other words, there can be an exponential number (in $|L|$) of optimal solutions. Further, with the introduction of a dummy root node in the SAP model, for every node $i \in L$ (i.e., with $x_{i_1 i_2} = 1$ in the SAP model) we can choose any one of the arcs from the root node to the node set L to be in the solution (more precisely we can choose any one of the arcs (r, i_1), with $x_{i_1 i_2} = 1$ in the SAP model).

To break symmetries induced by introducing the dummy root node, we impose the following *asymmetry constraints*:

$$x_{rk} \leq 1 - x_{i_1 i_2} \quad \forall i_1 < k, \quad i_1, k \in V_1 \tag{9}$$

These inequalities ensure that, among all inner nodes of the solution (i.e. those such that $x_{i_1 i_2} = 1$), the one with the smallest index is chosen to be adjacent to the root.

We found that introduction of these asymmetry constraints significantly improved the running time of the branch-and-cut procedure.

Separation Separation of the cut inequalities is based on the calculation of the maximum flow in the support graph. Instead of adding single cuts, we include bunches of them at once. These cuts are known as *nested, back* and *minimum-cardinality cuts*. Specifically, we first detect a violated (r, i)-cut C_1 for a node $i \in V$ and insert it into the LP. Nested cuts are based on iteratively adding further violated constraints induced by the minimum (r, i)-cut in the support graph in which all the capacities of the C_1 cut are set to one. In one iteration we add at most 100 such constraints. Back cuts rely on finding the reversal flow between nodes r and i, thus adding the cuts closer to node i first. They are also nested in our implementation. Finally, minimum cardinality cuts add a small ϵ value to all capacities of the support graph, so that in case of a disconnected graph, those cuts with the smallest cardinality are preferred.

Primal Heuristic To obtain upper bounds, we run the minimum spanning tree algorithm on the graph M with edge weights obtained as $w_{ij} = 1 - (x_{i_2 j_1} + x_{j_2 i_1})$, where $< j_2, i_1 >$ and $< i_2, j_1 >$ are the corresponding arcs obtained by transforming the edge (i, j) into H. The heuristic runs in $O(|N| \log |N| + |E|)$ time, and is applied in every node of the branch-and-bound tree. We found that the application of the primal heuristic was not of much benefit for problems that were solved rapidly (within a few minutes) by the branch-and-cut algorithm. However, for instances where the branch-and-cut procedure took a long time (i.e., instances that ran for an hour) we found that the application of the primal heuristic provided significant benefits in terms of speedup.

5 Weighted RLP

We now discuss the weighted version of the RLP problem that we refer to as the WRLP. The WRLP is motivated by recognizing that in many situations the cost of installing regenerators at different nodes of a network may vary due to real estate costs. In particular, installing and maintaining a regenerator at a dense urban area may be much more expensive than in a small town (or a rural area).

Mathematically the WRLP is identical to the RLP, except that the objective is to minimize the weighted sum of nodes selected as regenerator locations. If we let w_i denote the weight of node i, the objective of the WRLP is to minimize $\sum_{i \in L} w_i$, while the objective of RLP is to minimize $\sum_{i \in L} 1 = |L|$. To address the WRLP, we only need to make minor modifications to our heuristics and the cut formulation to the unit degree SAP. Note that the construction of the graph M and the preprocessing step does not depend on weights and thus remains unchanged for the WRLP.

In the greedy heuristic for WRLP, instead of identifying the node which can eliminate the greatest number of NDC node pairs when a regenerator is placed at its location (as in RLP), we find the node that maximizes the number of NDC node pairs eliminated divided by the weight of the node. In other words we find the node that maximizes the number of NDC node pairs eliminated per unit cost. This idea carries on to heuristics H1 and H2. Recall, in RLP, TREE(i) in H1 installs a regenerator at the node (not yet in L) that is adjacent to node i and has the greatest non-zero revised degree. For WRLP we simply divide the revised degree of a node by its weight and call that the weighted revised degree. Then, in the WRLP, TREE(i) in H1 installs a regenerator at the node (not yet in L) that is adjacent to node i and has the greatest non-zero weighted revised degree. Accounting for weights in the heuristic H2 is similar to H1. Specifically, each time Node(i) is called the heuristic looks for the node (not yet in L) that is adjacent to node i and has the highest value of degree divided by weight.

We note that the running times of the heuristics remain unchanged with these minor modifications to account for node weight. Further, it is straightforward to incorporate weights into the post-optimizer, and its running time remains unchanged.

The model for the WRLP as a unit degree SAP is identical except for changes in some of the arc weights. Specifically, we set c_{i_1,i_2} (the cost of arc $< i_1, i_2 >$) to w_i instead of one as in the case

of the RLP. Recall that using arc $< i_1, i_2 >$ in a SAP solution adds to the solution value the cost of arc $< i_1, i_2 >$ and is equivalent to adding a regenerator at node i on the original graph.

6 Computational Results

We now report on several computational experiments with our three heuristics for the RLP and the WRLP. We also report on our experience with the branch-and-cut approach for the unit degree SAP model. We coded our heuristics in Visual C++ 6.0, and our branch-and-cut using C++ under Linux and ILOG CPLEX 10.0. We conducted all runs on a Pentium D, 3.0 GHz machine with 2GB RAM. We first report on our test problems before we discuss our results.

6.1 Problem Generation and Characteristics

We generated problems on three types of networks that we refer to as (i) randomly generated networks (i.e., we generate the graph M directly), (ii) networks with random distances, and (iii) Euclidean networks. These different network types are designed to capture the attributes of different types of networks, and to understand the behavior of the heuristics and the branch-and-cut procedure on them. We now describe them in detail.

Randomly Generated Networks: For these problem types we directly generate the transformed graph M (i.e., we do not generate any distances but instead generate the graph M). A node pair is in the set of NDC node pairs if there is no edge connecting them. Here, an instance is generated according to two parameters, the first one n controls the number of nodes (problem size) and the other one $p \in [0, 1]$ that is the percentage of NDC node pairs. To ensure feasibility, we first construct an arbitrary spanning tree over the n nodes. We then randomly generate $\lfloor (1-p) \times (\frac{n(n-1)}{2} - (n-1)) \rfloor$ edges and add them to the spanning tree generated previously. In the resulting graph, there are $\lceil p \times (\frac{n(n-1)}{2} - (n-1)) \rceil$ NDC node pairs. Clearly, the larger n and p are the more NDC node pairs there are in the graph.

Networks with Random Distances: For these instances we first generate a network with edge lengths, and then compute the graph M that is obtained from the generated problem. Here, an instance is generated according to four parameters: n, q, a and b. Parameter n controls the size of

the instance. Parameters q, a and b affect the number of NDC node pairs. Specifically, $q \in [0,1]$ is the percentage of edges with distances greater than $d_{\max}$, while a and b define the uniform distribution $[ad_{\max}, bd_{\max}]$ from which we randomly choose the edge lengths for the rest of the edges ($0 < a \leq b \leq 1$). To ensure feasibility, we first construct an arbitrary spanning tree over the n nodes. The length of the edges on the tree are drawn uniformly in $[ad_{\max}, bd_{\max}]$. Next, we randomly add $\lfloor (1-q)(n-1)n/2 - (n-1) \rfloor$ more edges to the tree with edge length uniformly distributed in $[ad_{\max}, bd_{\max}]$. In the resulting graph, there are $\lceil q(n-1)n/2 \rceil$ node pairs that do not have a direct connection between them. These node pairs represent the q percentage edges that are longer than $d_{\max}$. We now construct the graph M from the generated problem. Observe that for a fixed $d_{\max}$, a and b, we would expect that the larger the value of q the greater the number of NDC node pairs in M.

Euclidean Networks: For these problems, nodes are randomly located on a 100×100 square grid. Euclidean distances are used as edge lengths. Here, an instance is generated according to two parameters: n and $d_{\max}$. Parameter n controls the size of the instance. Parameter $d_{\max}$ affects the number of NDC node pairs. To ensure feasibility, we first construct an arbitrary spanning tree over the nodes with the lengths of the edges in the tree a uniformly distributed in $[1, d_{\max}]$ (these are the only edges that do not have Euclidean edge lengths). For the remaining edges we use the Euclidean distances as the edge lengths. We now construct the graph M from the generated problem (observe that generating M is easy on Euclidean networks; we simply remove those edges that are strictly greater than $d_{\max}$). We note that we would expect that the smaller the value of $d_{\max}$ the greater the number of NDC node pairs in M.

6.2 Data Sets

We generated three different problem sets that we refer to as Set1, Set2, and Set3.

In Set1 we generated problems with upto 100 nodes on all three types of networks (randomly generated networks, networks with random distances, and Euclidean networks). For each of the three types of networks we generated 40, 60, 80, and 100 node problems.

For randomly generated networks we use five different values of p—10%, 30%, 50%, 70% and 90%. For each n and p combination we generated 10 instances. Thus, there are a total of 200

instances of randomly generated networks in Set1.

For networks with random distances, we use two different values of q—80% and 90%. $d_{\max}$ is fixed at 100 (actually the value of $d_{\max}$ is immaterial), while $a = 0.2$ and $b = 1$. For each n and q combination we generate 10 instances. Thus, there are a total of 80 instances of networks with random distances.

For Euclidean networks we use different values of $d_{\max}$. For the 40 and 60 node instances we use three values of $d_{\max}$—30, 40 and 50. For the 80 and 100 node instances we use four values of $d_{\max}$—20, 30, 40 and 50. For each n and $d_{\max}$ combination we generate 10 instances. Thus, there are a total of 140 instances of Euclidean networks.

Problem Set2, creates test instances for the WRLP. In our experiments on the RLP (i.e., Set1) we found that the characteristics that most influence the ability to solve the RLP are the number of nodes and the number of NDC node pairs. Consequently, in Set2, we focused our attention only on creating instances on randomly generated networks. In fact, we use the very same instances from Set1 and modify them by assigning an integer weight uniformly in $[2, 4]$.[3] Thus like Set1, there are a total of 200 WRLP instances.

The purpose of problem Set3 is to understand the behavior of the heuristics and branch-and-cut procedure as the size of the problems increase. Since the randomly generated networks were the hardest to solve, we only generated these types of test instances. We tested out instances with 200 to 500 nodes. Also, due to the fact that the running times of the greedy heuristic, and H1 were significantly greater than H2, and exceeded an hour for the larger instances, we only report on the experience with H2 on this set.

6.3 Results

We now discuss the computational results. For each instance, we compare between the heuristic solutions and the lower bound obtained by the branch-and-cut algorithm.

[3] We chose integer weights 2, 3, and 4 to represent three levels of costs for installing regenerators (viz. low, medium, and high). Based on our discussions with our contacts in the telecommunications industry the costs from the least expensive site to the most expensive site can vary by a factor of 2.

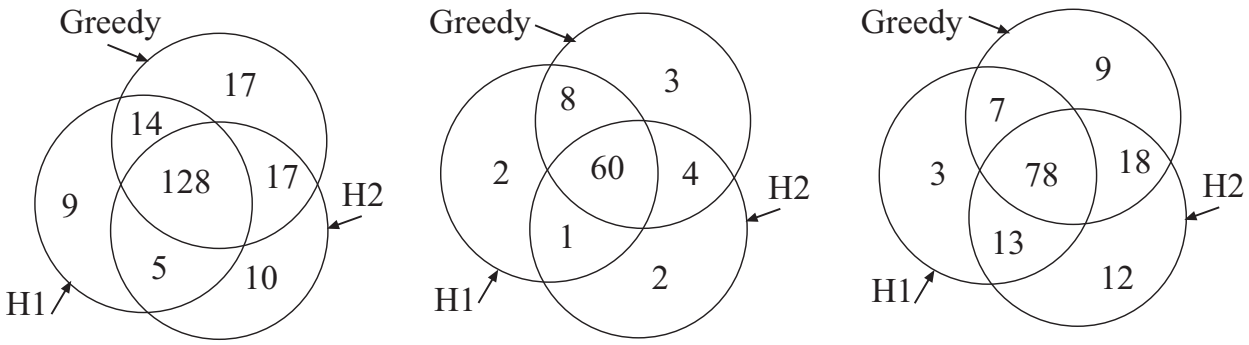
6.3.1 Set1

Tables 1, 2, and 3 summarize the computational results for Set1 over the three types of networks, respectively. Each row of these three tables aggregates the results from 10 instances with the same parameter settings. For example, in Table 1 the parameters settings are specified in the first two columns "n" and "p". Columns "Greedy", "H1", and "H2", report the computational results for the three heuristics, respectively. In particular, "NR" is the average number of regenerators obtained by the heuristic, "RT" records the average running time in seconds, "#Opt" records the number of optimal solutions obtained by the heuristic, and "#Imp" records the number of times the post processor improved the solution obtained by the heuristic. We also report the lower bound obtained from the branch-and-cut procedure for the unit degree SAP formulation. A time limit of one hour is set for the branch-and-cut procedure.

Table 1 reports computational results for the randomly generated networks. Of the 200 instances the branch-and-cut procedure finds the optimal solution in 165 instances. Of these 165 instances, the heuristics find the optimal solution in 130 instances. Overall, taking the best heuristic solution, the heuristics find the optimal solution in 132 instances.[4] Comparing between the three heuristics we find that the greedy heuristic provides the best solution in 176 instances, finds the optimal solution in 116 instances, and the post optimizer was useful (i.e., found an improvement to the greedy solution) in 88 instances. H1 provides the best solution in 156 instances, finds the optimal solution in 104 instances, and the post optimizer was useful in 114 instances. H2 provides the best solution in 160 instances, finds the optimal solution in 106 instances, and the post optimizer was useful in 111 instances. Figure 10 provides a more detailed comparison between the three heuristics on problem Set1. It is evident from Figure 10 the greedy heuristic solely provides the best heuristic solution in 17 instances, while H1 solely provides the best heuristic solution in 9 instances, and H2 solely provides the best heuristic solution in 10 instances.

Table 2 reports computational results for networks with randomly generated distances. Of the 80 instances the branch-and-cut procedure finds the optimal solution in all instances. On the other hand, of these 80 instances, the heuristics find the optimal solution in 70 instances. Comparing between the three heuristics we find that the greedy heuristic provides the best solution in 75

[4]When the branch-and-cut procedure is terminated we round up the fractional lower bound. If the heuristic solution equals this lower bound we can assert that the heuristic has found the optimal solution.



(a) Random Networks (b) Networks with Random Distances (c) Euclidean Networks

Figure 10: Venn diagram illustrating the number of times each heuristic obtained the best solution on Set1.

instances, finds the optimal solution in 68 instances, and the post optimizer was useful in 35 instances. H1 provides the best solution in 71 instances, finds the optimal solution in 64 instances, and the post optimizer was useful in 46 instances. H2 provides the best solution in 67 instances, finds the optimal solution in 60 instances, and the post optimizer was useful in 23 instances.

Table 3 reports computational results for the Euclidean networks. Of the 140 instances the branch-and-cut procedure finds the optimal solution in 112 instances. Of these 112 instances, the heuristics find the optimal solution in 99 instances. Comparing between the three heuristics we find that the greedy heuristic provides the best solution in 112 instances, finds the optimal solution in 80 instances, and the post optimizer was useful in 76 instances. H1 provides the best solution in 101 instances, finds the optimal solution in 75 instances, and the post optimizer was useful in 90 instances. H2 provides the best solution in 121 instances, finds the optimal solution in 87 instances, and the post optimizer was useful in 34 instances.

From Figure 10 one notices that there are always instances where each one of the heuristics solely provides the best solution. From Tables 1 and 2, it appears that for random networks, and networks with random distances, the greedy heuristic is marginally better than H1 and H2 in terms of the number of times it finds the best solution. On the other hand, from Table 3, for Euclidean networks it appears that H2 is marginally better than H1 and the greedy heuristic in terms of the number of times it finds the best solution. We note that the running time of the greedy heuristic is significantly greater than H1 and H2.

		Greedy				H1				H2				Unit degree SAP		
n	p	NR	RT	#Opt	#Imp	NR	RT	#Opt	#Imp	NR	RT	#Opt	#Imp	LB	RT	#Opt
40	10	1.4	0.0	10	5	1.4	0.0	10	5	1.4	0.1	10	6	1.4	0.8	10
	30	2.0	0.2	10	10	2.0	0.1	10	5	2.0	0.0	10	6	2	3.1	10
	50	3.0	0.1	10	5	3.1	0.3	9	5	3.3	0.1	7	4	3	4.6	10
	70	4.6	0.7	7	4	4.7	0.5	6	7	4.6	0.2	7	6	4.3	18.9	10
	90	9.5	1.7	5	3	9.7	1.3	2	9	9.8	0.5	2	7	8.9	3.5	10
60	10	1.9	0.3	10	1	1.9	0.3	10	1	1.9	0.0	10	2	1.9	7.0	10
	30	2.1	0.8	9	8	2.1	0.3	9	8	2.2	0.2	8	6	2	8.2	10
	50	3.2	1.9	8	8	3.3	1.1	7	7	3.4	0.4	6	7	3	48.9	10
	70	5.3	4.3	5	4	5.4	3.9	4	6	5.3	0.7	5	6	4.8	252.1	10
	90	10.8	15.8	4	6	11.2	18.0	2	10	11.2	2.5	2	4	10.2	37.0	10
80	10	1.9	1.3	10	2	1.9	0.7	10	1	1.9	0.1	10	4	1.9	19.3	10
	30	2.9	3.9	4	1	2.8	2.0	5	2	2.8	0.3	5	5	2.3	201.9	10
	50	3.8	9.1	2	2	4.0	5.6	0	0	3.7	0.9	3	6	2.9	1834.6	6
	70	6.0	19.4	0	3	5.9	17.3	1	9	5.8	3.0	2	6	4.9	985.5	9
	90	11.9	88.9	3	7	12.5	70.7	1	10	12.8	13.0	0	7	10.7	1543.0	7
100	10	2.0	3.6	10	2	2.0	1.8	10	0	2.0	0.5	10	5	2	124.5	10
	30	2.9	9.6	8	2	2.9	7.1	8	1	2.8	1.4	9	8	2.7	1024.9	10
	50	4.0	21.1	0	3	4.0	13.6	0	10	4.0	2.6	0	6	2.4	3439.1	1
	70	6.0	49.0	1	5	6.4	56.0	0	8	6.1	8.2	0	4	4.2	3211.8	2
	90	13.4	286.4	0	7	13.7	295.0	0	10	13.3	42.9	0	6	10	3600.0	0

Table 1: Computational results for randomly generated networks in Set1.

		Greedy				H1				H2				Unit degree SAP		
n	q	NR	RT	#Opt	#Imp	NR	RT	#Opt	#Imp	NR	RT	#Opt	#Imp	LB	RT	#Opt
40	80	2.6	0.4	8	3	2.5	0.1	9	2	2.7	0.1	7	1	2.4	0.7	10
	90	7.1	1.3	9	3	7.3	0.7	7	7	7.1	1.0	9	5	7	0.6	10
60	80	1.8	0.2	10	4	1.8	0.3	10	3	1.9	0.5	9	0	1.8	6.1	10
	90	5.6	4.9	8	5	5.8	4.6	6	9	6.1	3.6	4	2	5.4	3.3	10
80	80	1.6	1.6	10	5	1.6	0.4	10	4	1.6	0.3	10	3	1.6	23.1	10
	90	5.1	14.9	6	2	5.2	14.5	5	8	5.2	10.9	5	4	4.7	39.0	10
100	80	1.0	1.8	10	8	1.0	0.6	10	7	1.0	0.3	10	7	1	3.3	10
	90	4.2	31.6	7	5	4.3	19.3	7	6	4.3	18.6	6	1	3.9	122.0	10

Table 2: Computational results for networks with random distances in Set1.

		Greedy				H1				H2				Unit degree SAP		
n	$d_{\max}$	NR	RT	#Opt	#Imp	NR	RT	#Opt	#Imp	NR	RT	#Opt	#Imp	LB	RT	#Opt
40	30	7.9	1.0	9	4	8.0	0.8	8	5	7.9	0.7	9	2	7.8	4.9	10
	40	4.5	0.3	9	5	4.5	0.3	9	5	4.6	0.4	8	5	4.4	4.2	10
	50	2.8	0.1	10	6	3.0	0.1	8	2	3.1	0.1	7	1	2.8	0.7	10
60	30	8.0	8.9	4	4	7.8	4.7	5	7	7.6	4.6	6	2	7.2	366.0	10
	40	4.9	3.3	5	5	4.6	2.0	8	6	4.7	1.2	7	0	4.4	27.8	10
	50	2.9	1.4	10	5	3.1	0.8	8	5	2.9	0.4	10	1	2.9	3.8	10
80	20	15.5	86.0	1	5	16.5	55.1	0	10	16.0	74.1	0	7	13.5	2842.3	3
	30	7.9	30.7	3	5	7.5	19.4	5	8	7.3	14.8	7	4	6.8	1605.9	7
	40	4.7	9.8	5	5	4.7	7.3	5	6	4.5	3.9	7	1	4.2	135.9	10
	50	2.9	4.8	9	6	3.0	2.5	8	5	2.8	1.9	10	3	2.8	10.3	10
100	20	16.8	358.3	0	7	16.7	196.9	0	10	16.7	231.0	0	5	12.1	3600.0	0
	30	7.5	49.5	1	6	8.1	56.0	0	7	7.4	38.9	1	1	6.1	3009.1	2
	40	4.7	30.0	5	7	4.7	21.9	5	10	4.5	15.0	7	2	4.2	309.3	10
	50	3.1	14.8	9	6	3.4	7.1	6	4	3.2	4.9	8	0	3.0	65.3	10

Table 3: Computational results for Euclidean networks in Set1.

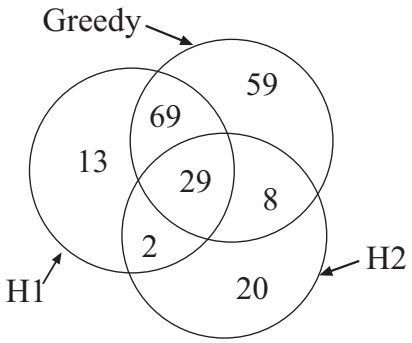


Figure 11: Venn diagram illustrating the number of times each heuristic obtained the best solution on weighted instances (Set2).

From Tables 1, 2, and 3 it is evident that as the number of nodes n increases the problem takes longer to solve. Also, when the number of nodes is fixed and as (i) p increases, (ii) q increases, and (iii) $d_{\max}$ decreases, the computation times for the heuristics increase. This is due to the fact that as p increases, q increases, and $d_{\max}$ decreases, the number of NDC node pairs increases. From these observations, it appears that regardless of the underlying network type the difficulty of an instance is largely determined by the number of NDC node pairs. Given that the randomly generated networks (i.e., where we directly generate the transformed graph M) directly control for the number of NDC node pairs, we conduct our next two sets of experiments (i.e., for Set2 and Set3) on them.

6.3.2 Set2

Table 4 reports computational results for Set2. Since these instances are weighted, the column "Obj" (which replaces "NR" Set1) represents the average objective function obtained by the heuristic. Of the 200 instances the branch-and-cut procedure finds the optimal solution in 159 instances. Of these 159 instances, the heuristics find the optimal solution in 117 instances. Overall, taking the best heuristic solution, the heuristics find the optimal solution in 118 instances. Comparing between the three heuristics we find that the greedy heuristic provides the best solution in 165 instances, finds the optimal solution in 108 instances, and the post optimizer was useful in 81 instances. H1 provides the best solution in 113 instances, finds the optimal solution in 78 instances, and the post optimizer was useful in 162 instances. H2 provides the best solution in 59 instances,

finds the optimal solution in 38 instances, and the post optimizer was useful in 157 instances. Figure 11 provides a more detailed comparison between the three heuristics on problem Set2. It shows that in 59 instances the greedy heuristic was solely the best heuristic, while H1 was solely the best heuristic in 13 instances, and H2 was solely the best heuristic in 20 instances. From these results it is clear that there are always instances where each one of the heuristics solely provides the best solution. However, the greedy heuristic significantly outperforms H1 and H2, on these weighted instances, in terms of the number of times it finds the best solution. We note as in Set1, the running time of the greedy algorithm is significantly larger than H1 and H2.

6.3.3 Set3

Our computational experiments indicated that the branch-and-cut algorithm is able to solve a large number of instances to optimality. Consequently, our objective with Set3 is to observe the behavior of the heuristics and branch-and-cut algorithm on large scale instances. Set3 contains large-scale instances that range from 200 to 500 nodes. For these problems the branch-and-cut algorithm for the unit degree SAP is again provided a time limit of one hour. We found that as the problem size and p increased the running time of the heuristics significantly increased. In particular, the running time for the greedy heuristic and H1 grew quite rapidly to the extent that some instances were taking over two hours to run for 300, 400, and 500 node problems. Consequently, in these large-scale instances we only report on our experience with H2, which runs significantly faster than H1 and the greedy heuristic on these large-scale instances. We also found that for larger values of p, as the number of nodes increases, the number of NDC pairs increases to the point that even the running time of H2 is quite significant. Therefore, in the computational experiments for Set3, as the number of nodes increase we do not conduct experiments for the larger values of p.

		Greedy				H1				H2				Unit degree SAP		
n	p	Obj	RT	#Opt	#Imp	Obj	RT	#Opt	#Imp	Obj	RT	#Opt	#Imp	LB	RT	#Opt
40	10	3.1	0.1	10	3	3.2	0.1	9	6	5.2	0.0	2	4	3.1	1.0	10
	30	4.4	0.1	9	3	4.8	0.1	5	9	5.1	0.0	3	7	4.3	2.5	10
	50	6.8	0.3	7	4	7.2	0.1	4	9	9.4	0.0	1	7	6.4	5.7	10
	70	10.6	0.4	7	5	12.8	0.2	0	10	13.3	0.4	1	6	10.1	4.1	10
	90	24.8	2.1	2	10	26.3	1.0	1	10	27.4	1.1	0	9	22.6	2.3	10
60	10	3.9	0.3	10	1	3.9	0.1	10	6	5.0	0.1	3	6	3.9	10.4	10
	30	4.6	0.6	9	2	5.4	0.5	3	7	6.7	0.2	1	7	4.5	27.9	10
	50	6.7	1.0	8	4	7.5	0.6	4	7	8.8	0.7	2	9	6.5	208.6	10
	70	12.7	3.7	2	7	13.8	1.5	0	8	14.8	1.6	2	7	11.3	208.5	10
	90	30.7	23.0	0	10	32.0	10.0	1	10	30.4	8.2	5	7	27.1	33.3	10
80	10	3.9	1.1	10	1	3.9	0.2	10	8	5.6	0.5	3	7	3.9	42.1	10
	30	5.6	2.8	8	0	5.7	1.4	8	7	7.5	0.9	3	9	5.3	413.5	10
	50	8.0	5.5	1	0	8.9	3.1	1	7	9.3	2.5	2	9	6.8	2302.7	5
	70	12.7	13.1	1	6	14.2	7.3	1	10	16.9	7.4	0	7	10.8	2586.9	4
	90	30.2	106.8	2	10	30.8	45.7	2	10	29.0	44.0	2	10	27.3	1702.0	6
100	10	4.0	3.0	10	0	4.0	1.4	10	4	4.8	1.2	5	9	4.0	200.6	10
	30	5.9	7.8	8	1	5.9	3.5	8	6	7.1	2.8	3	10	5.7	1757.8	8
	50	8.6	14.4	3	2	9.3	8.1	1	8	11.3	7.9	0	9	7.1	2883.9	4
	70	13.5	38.3	0	3	15.2	23.0	0	10	17.2	24.7	0	8	9.7	3600.4	0
	90	32.2	268.0	1	9	30.4	158.3	0	10	31.1	137.7	0	10	26.6	3148.5	2

Table 4: Computational results for weighted instances (Set2).

		H2				Unit degree SAP		
n	p	NR	RT	#Opt	#Imp	LB	RT	#OPT
200	10	2.0	4.7	10	6	2	331.6	10
	30	3.0	22.9	0	10	2	3600.0	0
	50	4.9	65.2	0	5	2	3600.0	0
	70	7.4	212.8	0	4	4	3600.0	0
300	10	2.0	24.6	10	9	2	1765.9	10
	30	3.7	140.9	0	6	2	3600.0	0
	50	4.9	459.4	0	8	2	3600.0	0
400	10	2.0	101.7	10	9	2	3600.0	0
	30	3.9	514.5	0	6	2	3600.0	0
	50	5.5	1611.3	0	6	2	3600.0	0
500	10	2.0	210.5	5	9	1.5	3600.0	0
	30	4.0	1221.7	0	5	2	3600.0	0

Table 5: Computational results for large scale instances (Set3).

Table 5 presents the results for the large-scale instances. The branch-and-cut algorithm is able to solve 200 and 300 node instances with $p = 10\%$ within one hour. For all other problem sets it is only able to provide a trivial lower bound of 1 or 2. In fact in five cases (for $n = 500$, $p = 10\%$) the branch-and-cut algorithm is unable to solve the LP relaxation at the root node in less than an hour. Furthermore, in 36 cases (all 20 instances for the 500 node problems and 16 instances for the 400 node problems) the branch-and-cut algorithm did not proceed beyond the root node within the one hour time limit. To get a better understanding of the difficulty of large-scale problems we ran the 500 node problems with a time limit of 3 hours. With this increased time limit the branch-and-cut algorithm solves the LP relaxation at the root node in all 20 instances. However, when $p = 10\%$ it only proceeds to the branching stage in 2 of the 10 instances, and when $p = 30\%$ it only proceeds to the branching stage in 7 of the 10 instances. Furthermore, the bounds produced did not improve beyond the trivial lower bound.

7 On Solving the Maximum Leaf Spanning Tree Problem

We would like to emphasize that our heuristics and branch-and-cut procedure can be applied to solve the maximum leaf spanning tree problem. Notice that our M graphs can be viewed as MLSTP instances. Consequently, we believe our work represents the first computational attempt in the literature to solve the MLSTP by means of integer programming approaches. Instead of

maximizing the number of leaves, we minimize the number of inner nodes in the tree, showing thereby that the problem can be reformulated as the minimum Steiner arborescence problem with a unit-degree constraint at the root node.

The only exact approach to the MLSTP so far, given by Fujie [4], is a combinatorial branch-and-bound algorithm based on computation of minimum spanning trees for getting upper bounds. Fujie [5] also gives two integer linear programming formulations for the MLSTP and studies their facial structure, without giving any insight as to how useful they might be in practice.

Our heuristics and branch-and-cut procedure would also apply to a weighted version of the maximum leaf spanning tree problem where each of the nodes has a weight w_i, and we wish to denote the total weight of the leaf nodes in the spanning tree. It is easy to see that maximizing the total weight of the leaf nodes in a tree is equivalent to minimizing the total weight of the internal nodes of a tree. We note that our computational experiments on Set2 may be viewed as experiments on weighted MLSTP instances. Interestingly, it appears that the weighted MLSTP is not well-studied in the literature. We were only able to find two references to the weighted MLSTP in the literature. The first by Fujie [4] where this problem is introduced but no computational experiments are reported for it. More recently Czaderna and Wala [12] describe a genetic local search algorithm for the Weighted MLSTP. However, no comparisons are made to lower bounds or optimal solutions, thus it is hard to assess the quality of their solutions.

8 Conclusions

In this paper, we introduced and addressed the regenerator location problem that arises in optical networks. We showed that the RLP is NP-complete. We then developed a graph transformation procedure that simplifies conceptualization of the problem. We showed that on this transformed graph the RLP is equivalent to the maximum leaf spanning tree problem (which is a well-studied problem in the approximation algorithms community). We then developed a problem reduction procedure, three heuristics, and a post-optimizer (that tries to reduce the cost of the solution by replacing two regenerator locations by one) for the RLP. We showed that the regenerator location problem can be modeled as a unit degree SAP (which is an interesting observation in its own right as it implies that the MLSTP can be modeled as a unit degree SAP), and used this observation to adapt

a branch-and-cut algorithm originally developed for the prize collecting Steiner tree problem [10] to the unit degree SAP (thereby allowing us to find exact solutions and lower bounds to the RLP).

Our computational results on 740 test instances indicated the quality of the heuristic solutions are quite good, and that the branch-and-cut procedure is a viable approach for problems with up to 100 nodes (and a few 200 and 300 node problems). Over the 740 test instances the heuristic procedures obtained the optimal solution in 454 instances while the branch-and-cut procedure solved 536 instances. For large-scale problems the branch-and-cut algorithm was unable to generate better lower bounds than the trivial lower bounds. This indicates that the heuristics appear to be the only viable approach for the larger problems. In particular, for large-scale problems, H2 seems the most promising heuristic in terms of running time.

Examining the results where the branch-and-cut algorithm found the optimal solution, we find that on Set1 the heuristics obtained the optimal solution in 299 out of 357 instances. In the 58 instances where the heuristics did not find the optimal solution, the best heuristic solution was one unit greater than the optimal solution in 57 instances and two units greater in 1 instance. Examining the results where the branch-and-cut algorithm finds the optimal solution on Set2, we found that the heuristics ontained the optimal solution in 117 out of 159 instances. In the 42 instances where the heuristics did not find the optimal solution, the best heuristic solution was between one and five units greater than the optimal solution. This suggests that there is scope for devising better heuristic procedures for the WRLP, which is part of our current research.

References

[1] M.S. Borella, J.P. Jue, D. Banerjee, B. Ramamurthy, and B. Mukherjee. Optical components for WDM lightwave networks. In *Proc IEEE*, volume 85, 1997.

[2] M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel. Obtaining optimal k-cardinality trees fast. In *2008 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 27–36. SIAM, 2008.

[3] M. Fischetti. Facets of two Steiner arborescence polyhedra. *Mathematical Programming*, 51:401–419, 1991.

[4] T. Fujie. An exact algorithm for the maximum leaf spanning tree problem. *Computers and Operations Research*, 30(13):1931–1944, 2003.

[5] T. Fujie. The maximum-leaf spanning tree problem: Formulations and facets. *Networks*, 43(4):212–223, 2004.

[6] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-Completeness*. Freeman,NY, 1979.

[7] M.X. Goemans and Y. Myung. A catalog of Steiner tree formulations. *Networks*, 23:19–28, 1993.

[8] L. Gouveia, P. Patricio, A. Sousa, and R. Valadas. MPLS over WDM network design with packet level QoS constraints based on ILP models. In *Proc IEEE Infocom*, volume 1, pages 576–586, 2003.

[9] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1994.

[10] I. Ljubić, R. Weiskircher, U. Pferschy, G.W. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Math Prog*, 105:427–449, 2006.

[11] B. Mukherjee. WDM optical communication networks: progress and challenges. *IEEE J Selected Areas Commun*, 18:1810–1824, 2000.

[12] K. Wala P. Czaderna. A genetic local search algorithm for weighted maximum leaf spanning tree problem. In *23rd IFIP TC 7 Conference on System Modelling and Optimization*, 2007.

[13] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[14] E. Yetginer and E. Karasan. Regenerator placement and traffic engineering with restoration in GMPLS networks. *Photonic Network Commun*, 6:139–149, 2003.

[15] YouTube. $http://www.youtube.com/t/factsheet$, Oct 2006.

[16] A. Zymolka. Untersuchung eines Tourenplanungsproblems. Master thesis, Mathematics, Philipps-University Marburg, 1999. Germany.