

New Primal-Dual Algorithms for Steiner Tree Problems

Vardges Melkonian

Department of Mathematics, Ohio Universtiy, Athens, Ohio 45701, USA

vardges@math.ohiou.edu

Abstract

We present new primal-dual algorithms for several network design problems. The problems considered are the generalized Steiner tree problem (GST), the directed Steiner tree problem (DST), and the set cover problem (SC) which is a subcase of DST. All our problems are NP-hard; so we are interested in approximation algorithms for them. First we give an algorithm for DST which is based on the traditional approach of designing primal-dual approximation algorithms. We show that the approximation factor of the algorithm is k , where k is the the number of terminals, in the case when the problem is restricted to quasi-bipartite graphs. We also give pathologically bad examples for the algorithm performance. To overcome the problems exposed by the bad examples, we design a new framework for primal-dual algorithms which can be applied to all of our problems. The main feature of the new approach is that, unlike the traditional primal-dual algorithms, it keeps the dual solution in the interior of the dual feasible region. The new approach allows us to avoid including too many arcs in the solution, and thus achieves a smaller-cost solution. Our computational results show that the interior-point version of the primal-dual most of the time performs better than the original primal-dual method.

(*Keywords:* Steiner tree; Integer programming, Approximation algorithm; Primal-dual algorithm)

1. Introduction

The primal-dual method is one of the main techniques for designing approximation algorithms for network design problems. In the last 10 years there has been significant progress in designing primal-dual algorithms for *undirected* network design problems [1, 5, 6, 8, 13]. *Directed* network design problems are usually much harder to solve, and little progress has

been made for these problems. In this paper, we give a primal-dual algorithm for a class of directed network design problems. We also present a new approach of designing primal-dual algorithms. This new approach is universal, and can be applied to both directed and undirected problems.

The network design problems considered in this paper can be represented by the following general integer program. Given a graph (directed or undirected) with nonnegative costs on the arcs find a minimum cost subgraph where the number of arcs entering set S is at least one for all subsets $S \in \rho$, where ρ is a collection of node subsets. Formally, given a graph $G = (V, E)$, the network design problem is the following integer program:

$$(IP) \quad \min \quad \sum_{e \in E} c_e x_e \quad (1)$$

$$\text{s.t.} \quad \sum_{e \in \delta^-(S)} x_e \geq 1, \quad \text{for each } S \in \rho, \quad (2)$$

$$x_e \in \{0, 1\}, \quad \text{for each } e \in E, \quad (3)$$

where $\delta^-(S)$ denotes the set of arcs entering S , and $\rho \subseteq 2^V$.

In the next three paragraphs we introduce our network design problems as special cases of this integer program.

In *directed Steiner tree problem* (DST), we are given a directed graph $G = (V, E)$, a root node $r \in V$, and a set of terminals $T \subseteq V$. All the other nodes are called non-terminal (or Steiner) nodes. The problem is to find the minimum cost directed tree rooted at r that contains all the terminals and any subset of the non-terminal nodes. This problem is the special case of (IP) when $\rho = \{S \subseteq V \mid r \notin S \text{ and } S \cap T \neq \emptyset\}$.

In *set cover problem* (SC), we are given a universe $U = \{u_1, \dots, u_n\}$ of n elements, a collection of subsets of U , $\Sigma = \{S_1, \dots, S_k\}$ with a nonnegative cost assigned to each of the subsets. The problem is to find a minimum cost subcollection of Σ that covers all elements of U . The following reduction from SC to DST shows that SC is a subcase of DST. For any instance of SC, an instance of DST is created the following way. Build a terminal t_i for each $u_i \in U$, a non-terminal node p_i for each subset $S_i \in \Sigma$. Have an arc $r \rightarrow p_i$ from the root to every non-terminal node with cost equal to the cost of the corresponding subset $S_i \in \Sigma$. Have an arc from a non-terminal node p_i to a terminal node t_j with cost zero if node u_j

(corresponding to t_j) belongs to subset S_i (corresponding to p_i). Easy to see that solving this instance of DST solves also the original instance of SC.

In *generalized Steiner tree problem* (GST), we are given an undirected graph $G = (V, E)$ and l pairs of vertices $(s_i, t_i), i = 1, \dots, l$; it is required to find a minimum-cost subset of edges $E' \subseteq E$ such that for all i , s_i and t_i are in the same connected component of (V, E') . The nodes to be connected are called terminals; all the other nodes are called non-terminal (or Steiner) nodes. This problem is the special case of (IP) when $\rho = \{S \subseteq V : |S \cap \{s_j, t_j\}| = 1 \text{ for some } j\}$.

Steiner tree and set cover problems occupy an important place in the theory of approximation algorithms. They also have a wide range of applications, from VLSI design to computational biology. For a survey on the applications of Steiner tree problems see [16].

All our problems are NP-hard; so we are interested in approximation algorithms for them. For SC, no polynomial time algorithm can achieve an approximation better than $O(\log n)$ unless $P=NP$ ([14]). Since DST contains SC as a special case, the same approximation hardness result applies to DST ($O(\log |T|)$ in this case). For SC, the lower bound $O(\log n)$ on the approximation factor is achieved by the greedy algorithm (Johnson [9], Lovász [10], Chvátal [4]). A simple randomized algorithm which achieves the same factor is given in Vazirani [15]. A primal-dual algorithm for SC was given by Bar-Yehuda and Even [2]. The approximation factor for this algorithm is $\max_{v \in U} |S \in \Sigma : v \in S|$. Charikar et al. [3] gave the only nontrivial approximation algorithm for DST (an algorithm for the special case of directed acyclic graphs was given before by Zelikovski [18]). Their method provides an $O(|T|^\epsilon)$ -approximation algorithm for any fixed ϵ in time $O(|V|^{1/\epsilon} |T|^{2/\epsilon})$. Wong [17] gave a primal-dual algorithm for DST; no theoretical guarantees were given for that algorithm. For GST, the best-known approximation factor is two, achieved by primal-dual algorithms (Goemans and Williamson [6], Polzin and Daneshmand [12]) and by an LP-rounding algorithm (Jain [7]).

The main goal of this paper is to give approximation algorithms for the above-mentioned Steiner tree problems. The algorithms for DST automatically extend to SC.

In this paper, we give a new primal-dual algorithm for DST. We also present a general framework for a new type of primal-dual algorithms that can be applied to all of our problems. The original primal-dual method in each iteration finds a dual solution which is on the boundary of the dual feasible region. We consider a variation of the primal-dual method which keeps the dual solution in the interior of the feasible region. We will call the new

method *interior-point primal-dual method*. The new rule allows us to avoid including too many arcs in the solution, and thus achieves a smaller-cost solution. Our computational results show that the interior-point version of the primal-dual most of the time performs better than the original primal-dual method.

The paper is structured as follows. In Section 2, we start with a general discussion about the primal-dual method, then specialize it to our problems. In Section 3, the interior-point version of the primal-dual is discussed. We speak about computational results in Section 4.

2. The Original Primal-Dual Algorithm

In this section, first we will present the generic primal-dual technique based on Goemans and Williamson [6], then we will specialize it to the Steiner problems considered in this paper.

2.1 General Primal-Dual Technique

Consider the LP relaxation of (IP) :

$$(LP) \quad \min \sum_{e \in E} c_e x_e \quad (4)$$

$$\text{s.t.} \quad \sum_{e \in \delta^-(S)} x_e \geq 1, \quad \text{for each } S \in \rho, \quad (5)$$

$$x_e \geq 0, \quad \text{for each } e \in E. \quad (6)$$

and its dual linear program:

$$(DP) \quad \max \sum_{S \in \rho} y_S \quad (7)$$

$$\text{s.t.} \quad \sum_{S \in \rho: e \in \delta^-(S)} y_S \leq c_e, \quad \text{for each } e \in E, \quad (8)$$

$$y_S \geq 0, \quad \text{for each } S \in \rho. \quad (9)$$

In the primal-dual method for approximation algorithms, an approximate solution to (IP) and a feasible solution to the dual of LP relaxation are constructed simultaneously; the performance guarantee is proved by comparing the values of both solutions.

Consider the complementary slackness conditions for (LP) and its dual (DP) . There are *primal complementary slackness conditions* corresponding to primal variables:

$$x_e > 0 \Rightarrow \sum_{S \in \rho: e \in \delta^-(S)} y_S = c_e \quad (10)$$

and there are *dual complementary slackness conditions* corresponding to dual variables:

$$y_S > 0 \Rightarrow \sum_{e \in \delta^-(S)} x_e = 1 \quad (11)$$

The general outline of primal-dual method for approximation algorithms is given in Algorithm 2.1.

Algorithm 2.1 General Primal-Dual Method for Approximation Algorithms

Initialize: dual $y \leftarrow 0$; infeasible primal A
while there does not exist a feasible integral solution obeying primal complementary slackness conditions **do**
 Get direction of increase for dual (moving towards feasibility of primal)
end while
Return feasible integral solution x obeying primal complementary slackness

The primal-dual method for approximation algorithms differs from the classical primal-dual method in that the dual complementary slackness conditions are not enforced. We cannot hope to have the dual conditions satisfied for NP-hard problems: enforcing both primal and dual complementary slackness conditions would either lead to an optimal solution or not be able to find a solution. Though the dual complementary slackness conditions are not enforced, a main idea of the primal-dual approximation algorithm is to keep those conditions as little violated as possible.

Now we go into detail to see how the ideas of Algorithm 2.1 are implemented for network design problems.

The primal-dual is an iterative algorithm. It maintains a dual solution that is initially zero and an infeasible solution $A \subseteq E$ for (IP) that is initially empty. In each iteration the algorithm adds an arc to current A until A gets feasible for (IP) . The choice of the arc is based on the following idea. If the current A is infeasible for (IP) , then inequality (2) is not satisfied for some subsets of V . There is a dual variable y_S corresponding to each of those subsets. The idea is to choose a set of subsets violating (2), call it $Violated(A)$, and increase the dual values of all $S \in Violated(A)$ uniformly until a dual constraint gets tight. The

arc corresponding to the dual constraint is added to A (ties are broken arbitrarily). Note that this choice of the arc ensures that the dual solution remains feasible and that primal complementary slackness conditions are satisfied.

The above algorithm has the shortcoming that though at any particular iteration the edge added to A was needed for feasibility, by the time the algorithm terminates it may no longer be necessary. These unnecessary edges increase the cost of A . To solve this problem, the refined algorithm does reverse deletion of the redundant edges in stage 2 (after getting an original feasible A). The summary of the primal-dual algorithm for general network design problems is given in Algorithm 2.2 ([6]).

Algorithm 2.2 Primal-Dual (Original Version)

```

Initialize:  $y \leftarrow 0$ ,  $A \leftarrow \emptyset$ ,  $l \leftarrow 0$  ( $l$  is a counter)
while  $A$  is not feasible do
     $l \leftarrow l + 1$ 
     $\mathcal{V} \leftarrow \text{Violated}(A)$  (a subroutine returning a set of violated subsets of  $V$ )
    Increase  $y_S$  uniformly for all  $S \in \mathcal{V}$  until  $\exists e_l \notin A$  such that  $\sum_{S \in \rho: e_l \in \delta^-(S)} y_S = c_{e_l}$ 
     $A \leftarrow A \cup \{e_l\}$ 
end while
for  $j \leftarrow l$  down to 1 do
    if  $A - \{e_j\}$  is still feasible then
         $A \leftarrow A - \{e_j\}$ 
    end if
end for
Return  $A$ 

```

To analyze the performance guarantee of this algorithm, we need the following definition.

Definition 1 A set $B \subseteq E$ is said to be a minimal augmentation of a set $A \subseteq E$ if:

- $A \cup B$ is feasible, and
- for any $e \in B$, $A \cup B - \{e\}$ is not feasible.

Let A_f be the output of the algorithm. Note that if A is the infeasible set in some iteration, then $A_f - A$ is a minimal augmentation of A .

Earlier we mentioned that the dual complementary slackness conditions (11) are not enforced in primal-dual approximation algorithms but they are kept violated as little as possible to achieve a good approximation guarantee. What are those conditions in terms of this particular algorithm? Note that $y_S > 0$ means that subset S was in $\text{Violated}(A)$ in

some iteration. Now $\sum_{e \in \delta^-(S)} x_e$ is the number of $\delta^-(S)$ -arcs in the minimal augmentation $A_f - A$ of A (this follows from *reverse* deletion). Thus, condition (11) says that for any $S \in \text{Violated}(A)$ we should have $|\delta^-(S) \cap (A_f - A)| = 1$. We cannot hope to have this condition satisfied for NP-hard network design problems. But if, on average, $|\delta^-(S) \cap (A_f - A)|$ is not a big number for $S \in \text{Violated}(A)$, that is, if the dual complementary slackness conditions on average are not violated much (for any solution), then it is possible to give a good approximation guarantee for the primal-dual. Formally, this result is stated in the next theorem which is the main analysis tool for primal-dual approximation algorithms and is due Goemans and Williamson [6]:

Theorem 1 *Suppose for any infeasible A and any minimal augmentation B of A ,*

$$\sum_{S \in \text{Violated}(A)} |\delta^-(S) \cap B| \leq \alpha |\text{Violated}(A)|. \quad (12)$$

Then

$$\sum_{e \in A_f} c_e \leq \alpha \cdot c(y) \leq \alpha \cdot \text{OPT}, \quad (13)$$

where $c(y)$ is the cost of the dual solution at the end of the algorithm, and OPT is the optimal value of (IP).

Theorem 1 implies that the primal-dual is an α -approximation algorithm. The question is how to achieve a good (small) α . This is normally achieved by choosing $\text{Violated}(A)$ properly. In the next two subsections we specify how $\text{Violated}(A)$ is chosen for GST and DST.

2.2 Primal-Dual Algorithm for Generalized Steiner Tree Problem

Recall that in GST we are given an undirected network (V, E) and l pairs of vertices $(s_i, t_i), i = 1, \dots, l$; it is required to find a minimum-cost subset of edges $E' \subseteq E$ such that for all i , s_i and t_i are in the same connected component of (V, E') . When applying the primal-dual to this problem, $\text{Violated}(A)$ is chosen as $\{S \subseteq V : S \text{ is a connected component of } (V, A) \text{ s.t. } |S \cap \{s_j, t_j\}| = 1 \text{ for some } j\}$. The example of Figure 1 illustrates the choice of $\text{Violated}(A)$. Agrawal, Klein, Ravi [1] and Goemans, Williamson [6] showed that this choice of $\text{Violated}(A)$ allows to achieve a factor of $\alpha = 2$ in Theorem 1. Thus, the primal-dual for GST is a 2-approximation algorithm.

We will consider the interior-point modification of this algorithm in Section 3.

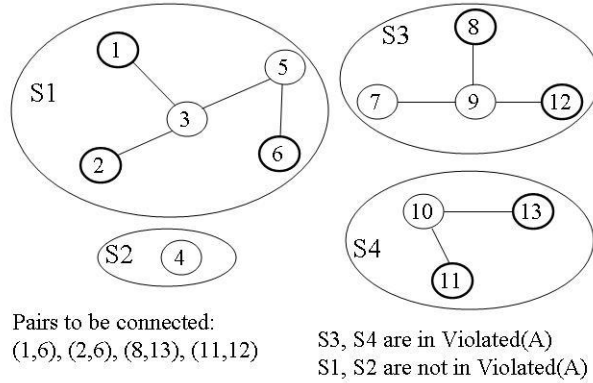


Figure 1: Choosing $Violated(A)$ for Generalized Steiner Tree Problem

2.3 Primal-Dual Algorithm for Directed Steiner Tree Problem

In this section we give those details of the algorithm that are specific to DST.

For a current infeasible set A , the graph can be partitioned into a set of maximal strongly connected components. We are particularly interested in some of those components.

Definition 2 A maximal strongly connected component S of (V, A) is called a leaf-component if

- ◇ S contains a terminal;
- ◇ S is not reachable from the root or from a terminal which is not in S .

We will say that a non-terminal node v dangles from a leaf-component S if $v \notin S$ and S is reachable from v . A subset which is the union of a leaf-component S and all the non-terminal nodes which dangle from S is called a violated leaf-subset.

Our version of primal-dual algorithm for the directed Steiner tree problem chooses the collection of all violated leaf-subsets as $Violated(A)$. Figure 2 gives an example of choosing $Violated(A)$.

What are the theoretical guarantees for this algorithm? The primal-dual algorithm by Bar-Yehuda and Even [2] for SC (the special case of DST) achieves an approximation factor of $\max_{v \in U} |S \in \Sigma : v \in S|$ which is equivalent to $\max_{v \in T} indegree(v)$ for DST. $|T|$ and $\max_{v \in T} indegree(v)$ would be conjectures for approximation factors for our algorithm too. However, the example of Figure 3 shows that neither of these factors can be achieved using the proof technique of Theorem 1. In this example, $|Violated(A)| = 2$ and $\sum_{S \in Violated(A)} |\delta^-(S) \cap$

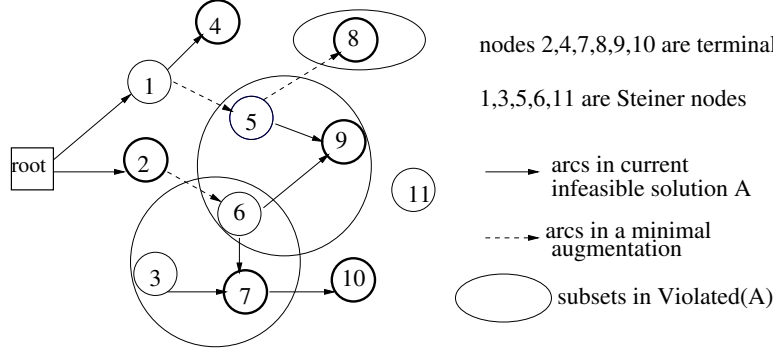


Figure 2: Choosing $Violated(A)$ for Directed Steiner Tree Problem

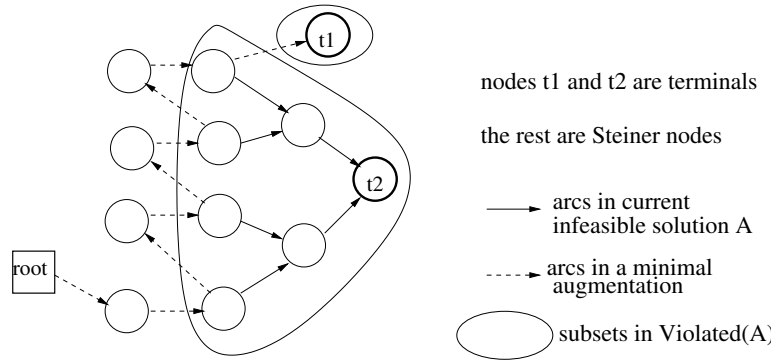


Figure 3: Bad Case Example for the Proof Technique

$|B| = 4 + 1 = 5$, giving ratio of 2.5 (recall that B is a minimal augmentation of A). This is greater than both $|T| = 2$ and $\max_{v \in T} \text{indegree}(v) = 2$.

The problem with this example is that there are too many arcs connecting pairs of non-terminal nodes. Rajagopalan and Vazirani [13] considered the restriction of undirected Steiner tree problem to quasi-bipartite graphs to overcome this problem. We will show that our algorithm achieves an approximation factor of $|T|$ when restricted to quasi-bipartite graphs. Recall that the best hope for the theoretical guarantee is $O(\log |T|)$. We also note that SC remains a special case of DST when restricted to quasi-bipartite graphs.

Definition 3 *A graph (directed or undirected) is called quasi-bipartite if it does not have edges connecting pairs of non-terminal nodes.*

Lemma 1 *For any directed Steiner tree in quasi-bipartite graphs, the number of non-terminal nodes is not more than the number of terminals.*

Proof: In any directed Steiner tree for quasi-bipartite graphs, a non-terminal node must have at least one terminal node as a direct descendent since it cannot have any non-terminal nodes as a direct descendent. In the directed Steiner tree there is a unique path to the terminal node so each terminal node is the direct descendent of at most one non-terminal node. Thus, the number of non-terminal nodes used in the directed Steiner tree is not more than the number of terminals. \square

Theorem 2 *Suppose the primal-dual algorithm is applied to DST restricted to quasi-bipartite graphs. Then for any infeasible A and any minimal augmentation B of A ,*

$$\sum_{S \in \text{Violated}(A)} |\delta^-(S) \cap B| \leq |T| \cdot |\text{Violated}(A)|.$$

Proof: We will show that for any $S \in \text{Violated}(A)$, $|\delta^-(S) \cap B| \leq |T|$. When $|\delta^-(S) \cap B| = 1$, it is trivial. So assume that $|\delta^-(S) \cap B| \geq 2$. Recall that S is a union of a leaf-component and the non-terminal nodes dangling from it. Any B -arc (an arc in B) entering S connects the leaf-component of S and all the nodes which are reachable from the leaf-component to the root. Thus, we can have $|\delta^-(S) \cap B| \geq 2$ only if all the B -arcs enter S through non-terminal nodes dangling from the leaf-component. Otherwise, the B -arcs entering the leaf-component would be redundant in the minimal augmentation. On the other hand, at most one B -arc can enter any node in the minimal augmentation. Thus, the B -arcs enter *distinct* non-terminal nodes of S . Denote the set of those non-terminal nodes by C . Then $|\delta^-(S) \cap B| = |C|$.

Suppose dSt is a directed Steiner tree obtained by deleting some of the arcs in $A \cup B$. No B -arc should be deleted since they are in the minimal augmentation. Thus, all non-terminal nodes of C are also in dSt . Then, based on Lemma 1, $|T| \geq |C| = |\delta^-(S) \cap B|$. \square

We note that our algorithm has the same performance guarantee as the algorithm of Charikar et al. [3] when $\epsilon = 1$. Our performance guarantee is only for quasi-bipartite graphs while the guarantee in [3] is more general. On the other hand, the advantage of our algorithm is that it is primal-dual and more practical. Finally we note that Wong [17] gave another primal-dual algorithm for DST. The main difference from our algorithm is that the dual value of only one violated subset is increased in each iteration. No theoretical guarantees were given for Wong's algorithm. In Section 4, we compare the performances of our and Wong's algorithms computationally.

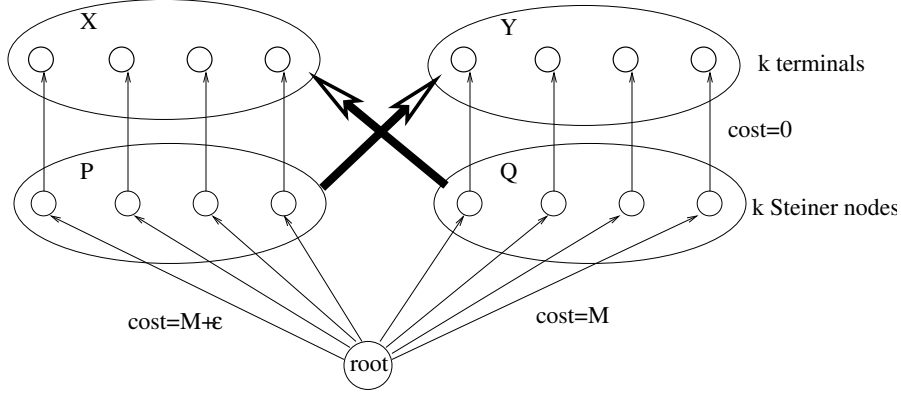


Figure 4: Bad Case Example for the Algorithm Performance

2.4 Bad Case Example

Theorem 2 gives an approximation factor of $|T|$ for our primal-dual algorithm for DST. In this section, we give an example which shows that we cannot hope to improve this factor substantially using a different proof technique.

Consider the instance of DST given in Figure 4. In this example, there are two groups of terminals, $X = \{x_1, \dots, x_{k/2}\}$ and $Y = \{y_1, \dots, y_{k/2}\}$, and two groups of non-terminal nodes: $P = \{p_1, \dots, p_{k/2}\}$ and $Q = \{q_1, \dots, q_{k/2}\}$. There are arcs $p_i \rightarrow x_i$ and $q_i \rightarrow y_i$ for each $i = 1, \dots, k/2$. There are also arcs $p \rightarrow y$ and $q \rightarrow x$ for any $p \in P$, $y \in Y$, $q \in Q$, $x \in X$; that is, the two thick arcs in the figure stand for complete bipartite subgraphs between P and Y , Q and X . The costs of any arc from a non-terminal node to a terminal is zero. There are arcs from the root to any $p \in P$ with cost $M + \epsilon$ and to any $q \in Q$ with cost M , where M is a very large positive number and ϵ is a very small positive number. (Note that this is also an instance of the special case of SC).

When the primal-dual algorithm is applied to this problem instance, it first includes all zero-cost arcs in the solution. Then, in the result of increasing the dual values, all the arcs with cost M will get saturated simultaneously. Consequently, we have to include all those arcs in the solution in the last $k/2$ iterations of the algorithm. The cost of the primal-dual output is $\frac{k}{2} \cdot M$. On the other hand, an optimal solution consists of only one arc with cost M and only one arc with cost $M + \epsilon$, with total cost of $2M + \epsilon$. Thus, the approximation ratio for this instance is $O(k)$.

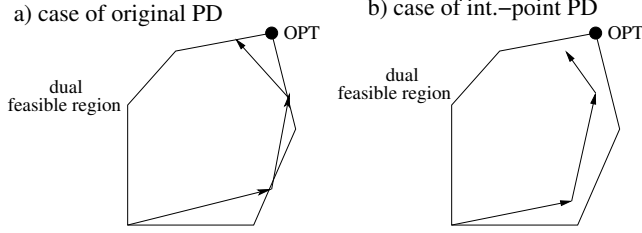


Figure 5: Increases in Dual Solution

3. The Interior-Point Version of the Primal-Dual Method

3.1 Intuition Behind the Algorithm

In the bad case example of Subsection 2.4, the output of the primal-dual algorithm is $O(k)$ times as expensive as the optimal solution. The reason is that the primal-dual solution has too many expensive arcs ($k/2$ versus only 2 in the optimal solution). We have to include all those arcs in the solution in the last $k/2$ iterations of the algorithm because they get saturated simultaneously. How to overcome this problem?

Consider the case when the number of terminals is $k \geq 6$. Let l be the iteration index when the $k/2$ arcs with cost M get saturated. What if we actually do not saturate the arcs? That is, in iteration l , (i) we still include an arc with cost M in the solution, but (ii) we increase the dual values until the left-hand sides of the dual constraints are at λM where λ is slightly less than one. What will happen in iteration $l + 1$? The slack in a dual constraint of an M -cost arc is $(1 - \lambda)M$, and there is only one active dual variable in the left-hand side. On the other hand, the slack in a dual constraint of an $M + \epsilon$ -cost arc is $(1 - \lambda)M + \epsilon$ but there are $k/2 - 1$ active dual variables in the left-hand side. If we choose $\lambda < 1 - \frac{2\epsilon}{(k-4)M}$ then it is easy to check that

$$\frac{(1-\lambda)M}{1} > \frac{(1-\lambda)M+\epsilon}{k/2-1}$$

and thus, in iteration $l + 1$ an arc with cost $M + \epsilon$ will get saturated first. This will lead to a feasible solution with one M -cost arc and one $M + \epsilon$ -cost arc which is an optimal solution.

Here is a higher level discussion of the idea we suggested in the previous paragraph. In each iteration, the generic primal-dual algorithm moves from one boundary point of the dual feasible region to another (see Figure 5(a)). In our modified version of the primal-dual we suggest to keep the dual solution in the interior of the dual feasible region (see Figure 5(b)). We stop increasing dual values once the dual solution is getting close to the boundary. What

is the benefit of this strategy? As we saw in the example, moving all the way to the boundary might result in getting stuck with a really bad solution; while staying in the interior allows us to avoid it.

Our experience of working with network design problems shows that this kind of bad examples arise in many situations (see Melkonian and Tardos [11]). This phenomenon is especially common in the situations when the cost range of a big subset of arcs is narrow.

Note that in this modified version of the primal-dual algorithm, the primal as well as the dual complementary slackness conditions are violated, while the primal conditions were enforced in the original version.

3.2 The Interior-Point Primal-Dual Algorithm

In this section we summarize the ideas discussed above to give a new general framework for primal-dual approximation algorithms. This framework can be used for any network design problem which is a special case of (IP). The new algorithm framework is given in Algorithm 3.1.

Algorithm 3.1 Interior-Point-Primal-Dual(λ)

Initialize: $y \leftarrow 0$, $A \leftarrow \emptyset$

while A is not feasible **do**

- $\mathcal{V} \leftarrow \text{Violated}(A)$ (subroutine returning several violated sets);
- Let e be the arc that would get saturated first if y_S were increased uniformly for all $S \in \mathcal{V}$;
- $A \leftarrow A \cup \{e\}$;
- Increase y_S uniformly for all $S \in \mathcal{V}$ until arc e gets λ -saturated (where $0 < \lambda < 1$).

end while

Delete redundant arcs (in reverse order).

Return A .

Note that the criterion of including an arc in the solution is the same as in Algorithm 2.2. The difference is that we do not saturate that arc; we stop increasing the dual values when the left-hand side of the corresponding dual constraint is λ times the cost of the arc. (If the arc was already λ -saturated at the beginning of the iteration, then we do not touch the dual values at all).

3.3 Theoretical Guarantee for the Interior-Point Primal-Dual Algorithm

The following theorem, which gives theoretical guarantees for Algorithm 3.1, is analogous to Theorem 1.

Theorem 3 *Suppose for any infeasible A and any minimal augmentation B of A ,*

$$\sum_{S \in \text{Violated}(A)} |\delta^-(S) \cap B| \leq \alpha |\text{Violated}(A)|. \quad (14)$$

Then

$$\sum_{e \in A_f} c_e \leq \frac{\alpha}{\lambda} \cdot c(y) \leq \frac{\alpha}{\lambda} \cdot \text{OPT}, \quad (15)$$

where A_f is the output of the algorithm, $c(y)$ is the cost of the dual solution at the end of the algorithm and OPT is the optimal value of (IP).

Proof: The second inequality of (15) follows from weak duality: the objective value of any feasible dual solution is no more than the objective value of any feasible primal solution. Let's show the first part of (15) by evaluating $\sum_{e \in A_f} c_e$ and $c(y)$.

For $\sum_{e \in A_f} c_e$ we have

$$\lambda \cdot \sum_{e \in A_f} c_e \leq \sum_{e \in A_f} \sum_{S \in \rho: e \in \delta^-(S)} y_S = \sum_{S \in \rho} |A_f \cap \delta^-(S)| y_S \quad (16)$$

In (16), (i) the inequality is true because the algorithm makes constraint (2) of (IP) λ -tight for all arcs of A_f ; (ii) the equality is true because each y_S is counted once for each $e \in A_f$ that is also in $\delta^-(S)$.

$\sum_{S \in \rho} |A_f \cap \delta^-(S)| y_S$ is zero at the beginning of the algorithm, and in iteration l it is increased by $\sum_{S \in \text{Violated}(A)} |\delta^-(S) \cap B| \cdot \epsilon_l$ where $B = A_f - A$ and ϵ_l is the increase of the dual variables corresponding to violated subsets in iteration l . Note that $A_f - A$ is a minimal augmentation of A . On the other hand, $c(y)$ is zero at the beginning of the algorithm and in iteration l it is increased by $|\text{Violated}(A)| \cdot \epsilon_l$. So based on (14),

$$\sum_{S \in \rho} |A_f \cap \delta^-(S)| y_S \leq \alpha \cdot c(y). \quad (17)$$

(16) and (17) imply that $\sum_{e \in A_f} c_e \leq \frac{\alpha}{\lambda} \cdot c(y)$. \square

The approximation factor obtained in Theorem 3 is $1/\lambda$ times worse than the factor obtained in Theorem 1. The reason is that we do not saturate the arcs and thus the dual

solution is smaller. But our conjecture is that by having a slightly smaller dual solution, we can achieve substantial savings in the cost of the primal solution (as in the example of Figure 4). Our computational results mostly support this conjecture, particularly for GST and SC. But we do not know how to modify the proof technique of Theorem 3 to take into account this improvement in the primal solution.

4. Computational Results

In order to evaluate the relative performance of our algorithms, we have implemented several versions of them. We have used *C* as our implementation environment. The integer programs were solved with CPLEX 8.1. We used a modified version of the mipex2.c file of CPLEX to call the MIP solver within our main *C* code.

Below we discuss the sets of problems on which the algorithms were tested. Then we analyze the computational results.

4.1 Problem Instances

We have generated random instances to test the algorithms. Several configurations of problem sets were created by changing the problem sizes, number of terminals and graph densities.

We distinguish two sizes of networks:

1) Instances for which we *could* find optimal solutions. We will refer to these networks as “*small*” networks. Our instances of small networks have 20 nodes. The basis for comparison of the algorithms in this case is naturally the optimal cost. It will be denoted **OPT** in the further discussion.

2) Instances for which we *could not* find an optimal solution. We will refer to these networks as “*large*” networks. The number of nodes in our large networks is 100. What is the basis of comparison in this case? Consider the cost axis of Figure 6. Here D_f and P_f

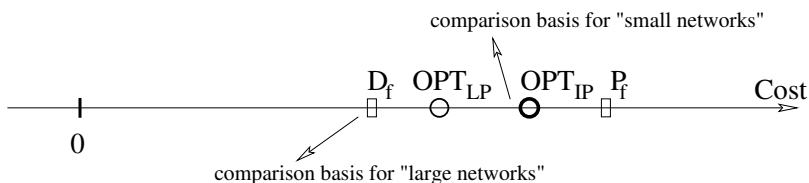


Figure 6: Cost axis

are the costs of the dual and primal outputs of the primal-dual algorithm. OPT_{IP} is the

optimal cost of (IP), and OPT_{LP} is the optimal cost of the LP-relaxation of (IP). Since we always have $D_f \leq OPT_{LP} \leq OPT_{IP} \leq P_f$, D_f is a lower bound for OPT_{IP} , and we will use D_f as a basis for comparison of the algorithms for large networks. Note that OPT_{LP} is an even better lower bound, but we could not use it as a comparison basis because CPLEX had memory problems when setting up LPs for large problems. When we test several primal-dual algorithms for the same problem, we take the largest dual output cost as lower bound. The lower bound is denoted **LB** in the further discussion.

100 random instances were generated for each problem configuration. For each configuration, the following results are reported: the mean, standard deviation, and maximum of the ratio of the performance of the algorithms to OPT and/or LB. For small networks, we also report the percentage of instances for which optimal solution was returned by the algorithms.

Further discussion on creating the instances is given separately for each of our three problems, GST, DST, and SC, in the next subsection.

4.2 Results of Experiments and Their Analysis

4.2.1 Generalized Steiner Tree Problem

Two algorithms were tested for this problem: the original primal-dual algorithm of Goemans and Williamson (see Subsection 2.2), and its interior-point modification (the general framework is given in Subsection 3.2). We will refer to these algorithms as $PDorig$ and $PDint(\lambda)$ correspondingly where λ is the saturation level. $PDint(\lambda)$ was tested for several values of λ . The rule of choosing $Violated(A)$ is the same for both algorithms (and is described in Section 2.3).

Note that an alternative way to formulate the GST is to specify which node sets should be in the same connected component of the solution. The connectivity requirements of nodes define an equivalence relation on V , which partitions V into to-be-connected node sets (in the example of Figure 1, the node sets are $\{1, 2, 6\}$, $\{8, 13\}$, $\{11, 12\}$). We have created our instances by specifying the to-be-connected node sets.

The computational results for small networks are summarized in Table 1. The number of to-be-connected components is 3. The number of non-terminal nodes is about half of $|V|$. The graph density is 50% for small networks. Note that for small networks we have also computed the average deviations from lower bounds to make the numbers for large networks more meaningful.

Table 1: Average deviations from OPT and LB for small networks

	PDorig	PDint(.2)	PDint(.4)	PDint(.6)	PDint(.8)
mean(from OPT)	1.014	1.019	1.014	1.009	1.01
mean(from LB)	1.62	1.63	1.62	1.61	1.62
st.dev.(from OPT)	0.03	0.03	0.02	0.02	0.02
max(from OPT)	1.1	1.08	1.08	1.08	1.08
perc. of opt. sol.	75%	55%	60%	75%	70%

For large networks, the algorithms were tested on several configurations of instances to see how the graph densities and ratios $|terminal\ nodes|/|V|$ affect the numerical behavior of the algorithms. The number of nodes is 100; the number of to-be-connected components is 12. Table 2 reports the results for three different $|terminal\ nodes|/|V|$ ratios: 1:2, 1:3, 1:4; here the graph density is fixed at 50%. Table 3 reports the results for three different graph densities: 50%, 75%, 100%; the $|terminal\ nodes|/|V|$ ratio is fixed at 1:2. Table 4 reports the running times of the algorithms for different problem sizes.

Table 2: Average deviations from LB for different $|terminal\ nodes|/|V|$ ratios

$ terminal\ nodes / V $		PDorig	PDint(.2)	PDint(.4)	PDint(.6)	PDint(.8)
1:2	mean	1.845	1.845	1.841	1.84	1.842
	st.dev.	0.03	0.04	0.04	0.04	0.03
	max	1.91	1.93	1.92	1.9	1.9
1:3	mean	1.81	1.81	1.8	1.8	1.81
	st.dev.	0.04	0.06	0.05	0.04	0.04
	max	1.92	1.95	1.91	1.9	1.92
1:4	mean	1.79	1.79	1.78	1.78	1.78
	st.dev.	0.05	0.05	0.05	0.05	0.05
	max	1.88	1.93	1.91	1.88	1.88

Table 3: Average deviations from LB for different graph densities

graph density		PDorig	PDint(.2)	PDint(.4)	PDint(.6)	PDint(.8)
100%	mean	1.85	1.85	1.85	1.85	1.85
	st.dev.	0.03	0.04	0.04	0.03	0.03
	max	1.92	1.93	1.92	1.93	1.92
75%	mean	1.85	1.84	1.84	1.84	1.84
	st.dev.	0.04	0.04	0.04	0.04	0.04
	max	1.92	1.93	1.93	1.92	1.92
50%	mean	1.845	1.845	1.841	1.84	1.842
	st.dev.	0.03	0.04	0.04	0.04	0.03
	max	1.91	1.93	1.92	1.9	1.9

Table 4: Average running times (in seconds)

$ V $	PDorig	PDint
100	0.05	0.08
200	0.45	0.7
300	1.4	2.6
400	3	5.3

The results are analyzed below:

- $PDint(\lambda)$ performs best for saturation levels close to $\lambda = 0.5$. For larger λ 's, $PDint(\lambda)$ naturally performs like $PDorig$.
- $PDint(0.6)$ outperforms $PDorig$ for both small and large instances. For small networks, on average $PDint(0.6)$ is 0.89% off the optimum while $PDorig$ is 1.38% off the optimum. Note that when comparing to the lower bounds, the average deviations are 61% and 62% for $PDint(0.6)$ and $PDorig$ correspondingly. Thus, though for large instances the improvement of $PDint(0.6)$ over $PDorig$ seems to be insignificant (about 0.5 to 1%) when comparing to the lower bounds, the significance could be much larger if we had the deviations from optimal values.
- The relationship between λ and the deviations from OPT (LB) seems to be smooth.
- Changes in graph density do not seem to have significant effect on the performance of the algorithms. The algorithms perform better when the percentage of terminal nodes is decreased.
- Another interesting observation is that the average deviations from lower bounds are pretty big, getting to 85% for large networks. Note also that the standard deviations are relatively small for such big average deviations. Recall from Subsection 2.2 that the gap between the primal and dual outputs of the original primal-dual algorithm can be at most 100%, so it is getting close to the limit.
- The algorithms are very fast even for big networks. We note that changes in graph density and $|terminal\ nodes|/|V|$ do not have any significant effect on running times.

4.2.2 Directed Steiner Tree Problem

Three algorithms were tested for this problem: our primal-dual algorithm given in Subsection 2.3 ($PDorig$), the interior-point version of the same algorithm ($PDint(\lambda)$), and Wong's primal-dual algorithm [17] ($PDone$). We do not know about any implementations of the algorithms by Charikar et al. [3] and Zelikovski [18]. Recall that Wong's algorithm, unlike ours, increases only one dual variable in each iteration. For $PDint(\lambda)$, the rule of choosing $Violated(A)$ is the same as for $PDorig$. The instances for DST were created as acyclic graphs with three or four levels of nodes.

The computational results for small networks are summarized in Table 5. The number of non-terminal nodes is about half of $|V|$. The density of the graph is 100% for small networks.

Table 5: Average deviations from OPT for small networks

	PDorig	PDint(.5)	PDint(.6)	PDint(.7)	PDint(.8)	PDint(.9)	PDone
mean	1.006	1.01	1.009	1.01	1.009	1.008	1.012
st.dev.	0.02	0.02	0.02	0.02	0.02	0.02	0.03
max	1.095	1.09	1.085	1.095	1.095	1.095	1.13
perc. of opt. sol.	84%	71%	73%	69%	74%	79%	79%

For large networks, the algorithms were tested on several configurations of instances to see how the graph densities and ratios $|terminals|/|V|$ affect the numerical behavior of the algorithms. The number of nodes is 100. Table 6 reports the results for three different $|terminals|/|V|$ ratios: 1:2, 1:3, 1:4; here the graph density is fixed at 100%. Table 7 reports the results for three different graph densities: 50%, 75%, 100%; the $|terminals|/|V|$ ratio is fixed at 1:2.

Table 6: Average deviations from LB for different $|terminals|/|V|$ ratios

$ term./ V $		PDorig	PDint(.5)	PDint(.6)	PDint(.7)	PDint(.8)	PDint(.9)	PDone
1:2	mean	1.064	1.065	1.065	1.062	1.064	1.062	1.07
	st.dev.	0.04	0.03	0.03	0.03	0.03	0.04	0.04
	max	1.15	1.15	1.15	1.15	1.17	1.15	1.19
1:3	mean	1.07	1.07	1.07	1.069	1.069	1.069	1.08
	st.dev.	0.05	0.04	0.04	0.04	0.04	0.05	0.05
	max	1.2	1.19	1.16	1.18	1.18	1.18	1.29
1:4	mean	1.06	1.063	1.064	1.061	1.056	1.06	1.055
	st.dev.	0.04	0.04	0.04	0.04	0.04	0.05	0.05
	max	1.2	1.2	1.2	1.2	1.2	1.24	1.25

Table 7: Average deviations from LB for different graph densities

density		PDorig	PDint(.5)	PDint(.6)	PDint(.7)	PDint(.8)	PDint(.9)	PDone
100%	mean	1.064	1.065	1.065	1.062	1.064	1.062	1.07
	st.dev.	0.04	0.03	0.03	0.03	0.03	0.04	0.04
	max	1.15	1.15	1.15	1.15	1.17	1.15	1.19
75%	mean	1.058	1.059	1.055	1.057	1.055	1.057	1.062
	st.dev.	0.03	0.03	0.03	0.03	0.03	0.03	0.04
	max	1.15	1.12	1.13	1.13	1.13	1.15	1.18
50%	mean	1.05	1.055	1.054	1.051	1.05	1.05	1.05
	st.dev.	0.03	0.03	0.03	0.03	0.03	0.03	0.03
	max	1.11	1.14	1.15	1.11	1.11	1.11	1.13

Table 8 reports the running times of the algorithms for different problem sizes. Table 9 reports the running times for different $|terminals|/|V|$ ratios. Table 10 reports the running times for different graph densities. $|V| = 200$ for the instances in the last two tables.

Table 8: Summary of running times for different problem sizes (in seconds)

$ V $	PDorig	PDint	PDone
100	1	0	0
200	6.8	3.8	0.6
300	36	20.3	1.3
400	120	65	3

Table 9: Summary of running times for different $|terminals|/|V|$ ratios (in seconds)

$ terminals / V $	PDorig	PDint	PDone
1:2	6.8	3.8	0.6
1:3	6.3	3	0
1:4	4.3	2.3	0.7

Table 10: Summary of running times for graph densities (in seconds)

graph density	PDorig	PDint	PDone
100%	6.8	3.8	0.6
80%	4.3	2.7	0.3
60%	3.3	2.3	0.3
40%	2.3	1.3	0

The results are analyzed below:

- *PDorig* has the best performance for small networks. For large networks, *PDint*(λ) slightly improves the performance of *PDorig* for appropriate values of λ . It should be noted that *PDorig* outputs are so close to the optimal solutions that there is not much room for improvement.
- *PDint*(λ) performs best for saturation levels around $\lambda = 0.8$. We did not include the results for $\lambda < 0.5$ since the performance of *PDint*(λ) was getting consistently worse for those values of λ . No significant zigzags are noticed in the relationship between λ and the deviations from *OPT* (*LB*).
- Changes in $|terminals|/|V|$ ratio do not seem to have significant effect on the performance of the algorithms. The algorithms perform slightly better for graphs with lower densities. One possible explanation is that the number of feasible solutions is less in low-density graphs; thus the chances are less that the algorithms will return suboptimal solutions.
- *PDone* has the best running time. The reason is that finding *Violated*(*A*) takes much less time for *PDone* than for the other two algorithms. Still, *PDorig* and *PDint*(λ) are reasonably fast: the running time for 400 nodes is under 2 minutes.

- The algorithms are faster for smaller $|terminals|/|V|$ ratios. A possible reason could be the following. For smaller ratios, the number of terminals is less, thus it might take less number of iterations to connect them.
- The algorithms are faster for graphs with lower densities. The reason could be that less number of arcs requires less number of operations at all stages of the algorithms.

4.2.3 Set Cover Problem

Four algorithms were tested for this problem: the specialization of our primal-dual algorithm given in Subsection 2.3 to this problem ($PDorig$), the interior-point version of the same algorithm ($PDint(\lambda)$), the primal-dual algorithm of Bar-Yehuda and Even [2] ($PDone$), and the greedy algorithm. Like Wong’s algorithm, $PDone$ increases only one dual variable in each iteration. For $PDint(\lambda)$, the rule of choosing $Violated(A)$ is the same as for $PDorig$.

The computational results for small networks are summarized in Table 11. The number of elements in the universe U is equal to the number of subsets in Σ . The graph density is 50% for small networks.

Table 11: Average deviations from OPT for small networks

	PDorig	PDint(.3)	PDint(.5)	PDint(.7)	PDint(.9)	PDone	Greedy
mean	1.12	1.067	1.047	1.052	1.05	1.26	1.067
st.dev.	0.2	0.1	0.1	0.1	0.1	0.3	0.1
max	1.73	1.4	1.4	1.4	1.4	1.77	1.4
perc. of opt. sol.	55%	65%	70%	65%	70%	35%	65%

For large networks, the algorithms were tested on several configurations of instances to see how the graph densities and ratios $|U|/|\Sigma|$ affect the numerical behavior of the algorithms. The number of nodes is 100. Table 12 reports the results for three different $|U|/|\Sigma|$ ratios: 1:1, 1:3, 1:5; here the graph density is fixed at 50%. Table 13 reports the results for three different graph densities: 50%, 35%, 20%; the $|U|/|\Sigma|$ ratio is fixed at 1:1.

Table 12: Average deviations from LB for different $|U|/|\Sigma|$ ratios

$ U / \Sigma $		PDorig	PDint(.2)	PDint(.4)	PDint(.6)	PDint(.8)	PDone	Greedy
1:1	mean	2.2	2	2.02	2.07	2.1	2.71	1.98
	st.dev.	0.26	0.2	0.18	0.19	0.2	0.39	0.21
	max	2.81	2.29	2.27	2.61	2.61	3.36	2.29
1:3	mean	1.83	1.73	1.73	1.8	1.84	2.3	1.72
	st.dev.	0.21	0.16	0.16	0.14	0.19	0.39	0.15
	max	2.39	1.93	1.94	2.18	2.39	2.88	1.89
1:5	mean	1.59	1.51	1.51	1.55	1.6	2.07	1.5
	st.dev.	0.27	0.2	0.2	0.25	0.27	0.38	0.2
	max	2.22	1.88	1.88	2.22	2.22	2.69	1.88

Table 13: Average deviations from LB for different graph densities

density		PDorig	PDint(.2)	PDint(.4)	PDint(.6)	PDint(.8)	PDone	Greedy
50%	mean	2.2	2	2.02	2.07	2.1	2.71	1.98
	st.dev.	0.26	0.2	0.18	0.19	0.2	0.39	0.21
	max	2.81	2.29	2.27	2.61	2.61	3.36	2.29
35%	mean	2.04	1.87	1.88	1.91	2.02	2.89	1.84
	st.dev.	0.21	0.18	0.2	0.18	0.23	0.53	0.15
	max	2.63	2.32	2.35	2.37	2.63	3.82	2.08
20%	mean	1.77	1.72	1.72	1.72	1.78	2.53	1.68
	st.dev.	0.16	0.16	0.17	0.16	0.18	0.38	0.13
	max	2.04	2.13	2.13	2.08	2.08	3.19	2.03

Table 14 reports the running times of the algorithms for different problem sizes. Table 15 reports the running times for different $|U|/|\Sigma|$ ratios. Table 16 reports the running times for different graph densities. $|V| = 200$ for the instances in the last two tables.

Table 14: Summary of running times for different problem sizes (in seconds)

$ V $	PDorig	PDint	Greedy	PDone
100	0.6	0.2	0	0
200	10.3	6.3	0	1.3
300	74	41	0	8
400	296	161	0	29

Table 15: Summary of running times for different $|U|/|\Sigma|$ ratios (in seconds)

$ U / \Sigma $	PDorig	PDint	Greedy	PDone
1:1	33	18	0	3
1:3	10.3	6.3	0	1.3
1:5	5	3	0	1

Table 16: Summary of running times for different graph densities (in seconds)

graph density	PDorig	PDint	Greedy	PDone
50%	20	11	0	2
35%	10.3	6.3	0	1.3
20%	4	3	0	1

The results are analyzed below:

- For small instances, $PDint(\lambda)$ outperforms the other algorithms. $PDint(\lambda)$ performs best for saturation levels around $\lambda = 0.6$. $PDint(\lambda)$ returns optimal solutions 65-70% of time while $PDorig$ only 55% of time. $PDint(\lambda)$ also improves the worst-case scenario over $PDorig$: 73% off the optimum for $PDorig$ vs. only 40% off the optimum for $PDint(\lambda)$. Recall that our pathologically bad example (Figure 4) was also for SC. Thus, the computational results support the conjecture that staying in the interior of the dual feasible region might help the primal-dual algorithm to avoid suboptimal solutions.

- For large networks, the greedy algorithm has the best performance. $PDint(\lambda)$ is the runner-up. No significant zigzags are noticed in the relationship between λ and the deviations from LB (OPT). $PDint(\lambda)$ performs better for small λ 's, getting close to greedy's performance for $\lambda = 0.2$.
- The performance of all four algorithms is getting better for smaller $|U|/|\Sigma|$ ratios. A possible explanation could be the following. For smaller ratios, the number of subsets in Σ , and thus the number of feasible solutions is relatively big. Therefore, even if a "wrong" subset was included in the solution at the beginning of an algorithm, it is possible to compensate for that by choosing a right combination of subsets later in the algorithm. While if the $|U|/|\Sigma|$ ratio is small, then relatively few subsets are available. If a "wrong" subset was included in the solution at some stage of an algorithm, it might not be possible to fix the situation later.
- The performance of the algorithms is getting better for smaller graph densities. One way to explain that is the following. Since the number of arcs is relatively small, the chances are less that a "wrong" arc will be included in the solution.
- Greedy has the best running time. The primal-dual algorithms are faster for smaller $|U|/|\Sigma|$ ratios and graph densities. Possible reasons could be the same as those given for DST.
- An interesting observation is that all the primal-dual algorithms perform much better for DST than for its special case of SC. Recall that our pathologically bad example (Figure 4) was also for SC. A natural question arises here. Is it the special case of SC that makes DST hard to solve? Or are there other hard special cases too?

5. Future Directions

Here are some interesting open problems:

- *More detailed analysis of the interior-point primal-dual method*. A more careful analysis could improve the theoretical guarantee given in Theorem 3. An important question is how to choose λ for a given problem. It is possible that instead of fixing λ for the whole execution of the algorithm, a different λ should be chosen in each iteration.

- *Extending the interior-point primal-dual method to other problems.* The general framework of the new approach could be extended to other network design problems. It would be interesting to see whether, theoretically or computationally, it can improve the performance of original primal-dual algorithms.

- *Improving the approximation factor for DST.* $O(\log |T|)$ -approximation would be a breakthrough.

- *Relationship between DST and SC.* As mentioned at the end of Section 4, SC seems to be the only special case of DST when the primal-dual algorithms do not perform well computationally. This observation rises the following interesting question: is it possible to improve the theoretical guarantee of the primal-dual algorithm for DST if it is restricted to a special case which does not include SC?

Acknowledgments

The author would like to thank the anonymous referees for their helpful comments and suggestions.

References

- [1] A. Agrawal, P. Klein, R. Ravi, When trees collide: an approximation algorithm for generalized Steiner tree problems on networks, *SIAM Journal on Computing* 24 (1995) 440–456.
- [2] R. Bar-Yehuda, S. Even, A linear time approximation algorithm for the weighted vertex cover problem, *Journal of Algorithms* 2 (1981) 198–203.
- [3] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, M. Li, Approximation algorithms for directed Steiner problems, *Journal of Algorithms* 33(1) (1999) 73–91.
- [4] V. Chvátal, A greedy heuristic for the set covering problem, *Mathematics of Operations Research* 4 (1979) 233–235.
- [5] M. X. Goemans, A. Goldberg, S. Plotkin, D. Shmoys, E. Tardos, D. P. Williamson, Improved approximations algorithms for network design problems, In *Proceedings of the 5th Annual Symposium on Discrete Algorithms*, 1994, pp. 223–232.

- [6] M. X. Goemans, D. P. Williamson, A general approximation technique for constrained forest problem, *SIAM Journal on Computing*, 24 (1995) 296–317.
- [7] K. Jain, A factor 2 approximation algorithm for the generalized Steiner network problem, *Combinatorica*, 1 (2001) 39–60.
- [8] K. Jain, V. Vazirani, Primal-dual approximation algorithms for metric facility location and k -median problems using the primal-dual schema and Lagrangean relaxation, *Journal of ACM*, 48 (2001) 274–296.
- [9] D. S. Johnson, Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences*, 9 (1974) 256–278.
- [10] L. Lovász, On the ratio of optimal integral and fractional covers, *Discrete Mathematics*, 13 (1975) 383–390.
- [11] V. Melkonian, E. Tardos, Primal-dual-based algorithms for a directed network design problem, *INFORMS Journal on Computing*, 17(2) (2005) 159–174.
- [12] T. Polzin, S.V. Daneshmand, Primal-dual approaches to the Steiner problem, *Electronic Colloquium on Computational Complexity (ECCC)*, 7(47) (2000), <ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/2000/TR00-047/index.html>.
- [13] S. Rajagopalan, V. Vazirani, On the bidirected cut relaxation for the metric steiner tree problem, In *Proceedings of 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 742–751.
- [14] R. Raz, S. Safra, A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP, In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, 1997, pp. 475–484.
- [15] V. Vazirani, *Approximation algorithms*, Springer-Verlag, Berlin, 2001.
- [16] P. Winter, Steiner problem in networks: a survey, *Networks* 17 (1987) 129–167.
- [17] R. Wong, A dual ascent approach for Steiner tree problems on a directed graph, *Mathematical Programming* 28 (1984) 271–287.
- [18] A. Zelikovski, A series of approximation algorithms for the acyclic directed Steiner tree problem, *Algorithmica* 18 (1997) 99–110.