最小コスト木状被覆問題の2倍近似アルゴリズム

藤戸 敏弘

豊橋技術科学大学 情報工学系 計算機大講座
〒 441-8580 愛知県豊橋市天伯町雲雀ヶ丘 1-1

概要：辺コストをもつ連結グラフ G において、その頂点集合が G の頂点被覆を成す木を木状被覆、そのような木の中からコスト最小のものを計算する問題を木状被覆問題という。*NP* 困難である同問題に対し、辺コストが一定の場合、線形時間2倍近似アルゴリズムが従来から知られていたが、任意コストの場合、3倍近似アルゴリズムがベストで、しかもそれらは（多項式時間とはいえ）非効率的である。本稿では、最小スパンニング木の葉を刈ることで、高速な2倍近似アルゴリズムが得られることを示す。

How to trim an MST: A 2-approximation algorithm for minimum cost tree cover

Toshihiro Fujito
Department of Information & Computer Sciences, Toyohashi University of Technology
Tempaku, Toyohashi 441-8580 Japan

Abstract : The *minimum cost tree cover* problem is to compute a minimum cost tree T in a given connected graph G with costs on the edges, such that the vertices of T form a vertex cover for G. The problem is supposed to arise in applications of vertex cover and edge dominating set when connectivity is additionally required in solutions. Whereas a linear-time 2-approximation algorithm for the unweighted case has been known for quite a while, the best approximation ratio known for the weighted case is 3. Moreover, the known 3-approximation algorithm for such case is far from practical in its efficiency.
In this paper we present a fast, purely combinatorial 2-approximation algorithm for the minimum cost tree cover problem. It constructs a good approximate solution by trimming some leaves within a minimum spanning tree (MST), and to determine which leaves to trim, it uses both of the primal-dual schema and the local ratio technique in an interlaced fashion.

1 Introduction

In an undirected graph $G = (V, E)$ a set C of vertices is a *vertex cover* if every edge in G has at least one of its end-vertices in C, whereas an edge set D is an *edge dominating set* if every edge not in D is adjacent to some edge in D. A tree $T \subseteq E$ in a connected graph G is called a *tree cover* if it is an edge dominating set for G. Or equivalently, it is a tree such that the set of vertices induced by T is a vertex cover for G. The *minimum cost tree cover* problem is to compute a tree cover of minimum total cost in a given connected graph $G = (V, E)$ with a nonnegative cost l_e on each edge $e \in E$. The problem is clearly *NP*-hard even in the unweighted case since it then becomes equivalent to the *connected vertex cover* problem, which in fact is known to be as hard (to approximate) as the vertex cover problem [11]. In fact, while it is possible to approximate minimum vertex cover to within a factor slightly better than 2 [13, 3, 16, 12], doing so within any factor smaller than $10\sqrt{5} - 21 \approx 1.36067$ is *NP*-hard [6].

The tree cover problem was introduced by Arkin, Halldórsson, and Hassin [1], and they were partially motivated by closely related problems of locating tree-shaped facilities on a graph such that all the vertices are dominated by chosen facilities. They presented a 2-approximation algorithm for the unweighted version, as well as a 3.55-approximation algorithm for the case of general costs. In fact a simpler 2-approximation algorithm appeared earlier for the unweighted case, due to Savage [18], although it was designed for vertex cover and not intended for connected vertex cover. A better approximation algorithm was later developed for minimum weight tree cover by Könemann et al. [15] and independently by Fujito[8], lowering the approximation ratio down to 3, and it is currently the best bound for the problem. Thus, whereas vertex cover, edge dominating set [9, 17], and many problems closely related to them are known to be approximable to within a factor of 2, regardless of associated costs, it is not the case for tree cover. Even worse, the algorithms of [15] and [8] are far from practical in their efficiency; either one requires to solve optimally an LP of huge size (see (P) in Sect. 1.1), and to do so, it inevitably resorts to calling the ellipsoid method as their subroutine.

In this paper we present a fast, purely combinatorial 2-approximation algorithm for the minimum cost tree cover problem. All the previous algorithms for general costs [1, 15, 8] are in the similar style of computing a vertex cover C first, and then connecting all the vertices in C by a Steiner tree. Our algorithm in contrast is designed based on a hunch that a good approximate solution can be always found in the vicinity of a minimum spanning tree (MST).[1]

1.1 Bidirected Formulation

An instance of the minimum cost tree cover problem consists of an undirected graph $G = (V, E)$ and nonnegative costs l_e for all edges $e \in E$. Let $\vec{G} = (V, \vec{E})$ denote the directed graph obtained by replacing every edge $e = \{u, v\}$ of G by two anti-parallel arcs, (u, v) and (v, u), each having the same cost $c(\{u, v\})$ as the original edge e. Pick one vertex in V as the *root*, and suppose $\vec{T}' \subseteq \vec{E}$ is a *branching* (or a directed tree) rooted at r. It is assumed throughout that the arcs in a branching are always directed away from the root to a leaf. (Note: we will often use $\vec{T}$ and T interchangeably, to denote a branching and an undirected tree, respectively, with a root in common). In the *bidirected formulation* of the tree cover problem, one seeks for a minimum cost branching $\vec{T}'$ rooted at r in $\vec{G}$ such that T' is a tree cover rooted at r in G. We call either of such a branching or an undirected tree an *r-tree cover* for $\vec{G}$ (or G).

A set $S \subseteq V - \{r\}$ is called *dependent* if S induces at least one edge in G, and let $\mathcal{D}$ denote the family of such dependent sets. Let $\delta^-(S)$ denote the set of arcs with heads in S and tails out of S (when needed, we use $\delta^-_H()$ to specify that only the arcs of graph H are considered). We call the arc set $\delta^-(S)$ in $\vec{G}$ an *r-edge cut* if $S \subseteq V - \{r\}$ is a dependent set. By using a max-flow/min-cut argument, one can see that the bidirected formulation of the minimum cost tree cover problem can be modeled by the integer program:

$$\min\{l^T x \mid x \in \{0,1\}^{\vec{E}}, x(\delta^-(S)) \geq 1, \forall S \in \mathcal{D}\},$$

where $x(\vec{F}) = \sum_{a \in \vec{F}} x_a$ for $\vec{F} \subseteq \vec{E}$, as an r-tree cover must pick at least one arc from every r-edge cut. Replacing the integrality constraints by $x \geq 0$, we have the LP relaxation of form:

$$\text{(P)} \quad \begin{array}{lll} \min & \sum_{a \in \vec{E}} l_a x_a & \\ \text{subject to:} & x(\delta^-(S)) \geq 1 & \forall S \in \mathcal{D} \\ & x_a \geq 0 & \forall a \in \vec{E} \end{array}$$

Unlike the algorithms of [8, 15], our algorithm also makes good use of the LP dual of (P):

$$\text{(D)} \quad \begin{array}{lll} \max & \sum_{S \in \mathcal{D}} y_S & \\ \text{subject to:} & \sum_{S \in \mathcal{D}: a \in \delta^-(S)} y_S \leq l_a & \forall a \in \vec{E} \\ & y_S \geq 0 & \forall S \in \mathcal{D} \end{array}$$

At this point one may notice that the bidirected minimum cost tree cover problem has some similarity with another well-known combinatorial optimization problem, no matter how superficial it might be. In a directed graph $D = (V, A)$ with $r \in V$ an *r-arborescence* $A' \subseteq A$ is a spanning tree of the underlying undirected graph of D such that each vertex of D other than r is entered by exactly one arc of A' (and no arc enters r). An arc set $C \subseteq A$ is called an *r-cut* if $C = \delta^-(U)$ for some nonempty $U \subseteq V - \{r\}$. The *shortest r-arborescence problem* is to, given D, r, and nonnegative costs l_a for all the arcs $a \in A$, compute an r-arborescence of minimum cost.

Suppose now that the set of constraints in (P) concerning all the r-edge cuts is enlarged such that it consists of $x(\delta^-(S)) \geq 1$ for all nonempty $S \subseteq V - \{r\}$; that is, replace $\mathcal{D}$ by $\mathcal{D}' = \{S \subseteq V - \{r\} \mid S \neq \emptyset\}$, and denote it (P'). It was shown by Edmonds that the shortest r-arborescence problem can be formulated *exactly* by (P') [7]. Likewise, replace $\mathcal{D}$ by $\mathcal{D}'$ in (D), and call it (D'). Then, (D'), which is the LP dual of (P'), formulates the problem of maximum (fractional) *r-cut packing*, and it was shown by Fulkerson that (D') has integer optimum solutions if l is integral [10] (thus, there exist an r-arborescence and an integral r-cut packing of the same cost). Recall now that original (P) and (D) are actually based on graphs in bidirected forms of undirected graphs, and not on arbitrary digraphs, and if graphs in (P') are also restricted as such, the problem formulated by (P') reduces to the one on undirected graphs, namely, the *minimum spanning tree problem*. It is this observation that has motivated us to investigate the possibility of whether an MST, as an integer optimal solutions in (P'), could give us a lead when they are cast in (P) (or an r-cut packing when cast in (D)).

[1]Interestingly, it was already tried by Arkin et al. [1] to use (a modification of) the Prim's or Kruskal's algorithm for MST problem, and either of them was found not to perform well.

1.2 Primal-Dual Schema vs. Local Ratio Technique

Among various methods for design and analysis of approximation algorithms, the ***primal-dual schema*** and the ***local ratio technique*** have been popular and applied to a wide range of problems. While it is often possible to interpret algorithms in one framework within the other [3, 5, 2], and moreover, these two methods have been shown essentially equivalent [4], yet it could be of great use to have both of them at our disposal, as they can provide different lines of approaches to a problem of concern.

Certainly, the primal-dual method could be helpful in approximating the tree cover problem. Consider, for instance, the Savage's 2-approximation algorithm for unweighted tree cover, which simply returns the tree $T_{\rm tc}$ remaining after all the leaves are trimmed from a depth-first-search (DFS) spanning tree T [18]. The directed version $\vec{T}_{\rm tc}$ of $T_{\rm tc}$ rooted at r is clearly feasible to (P). To estimate its cost $|T_{\rm tc}|$, let M be a matching on T such that all the internal nodes of T but r are matched by M (Note: it is easy to find such a matching). Since M is a matching, r-edge cuts, $\delta^-(e)$ and $\delta^-(e')$, are disjoint for any two different edges e and e' of M. Hence, y with $y_e = 1$ for each $e \in M$ and $y_D = 0$ for all the other dependent sets D, is feasible to (D). To show that $|T_{\rm tc}|$ is a factor of at most 2 away from the optimum, we need only to verify that $|\vec{T}_{\rm tc}| \le 2|M|$, by simple combinatorial arguments, for then, $|\vec{T}_{\rm tc}| \le 2\sum y_D \le 2$(optimal value of (P)).

It does not look so easy, however, to find a way to go from here to the case of arbitrary costs, under guidance of the known primal-dual schema only, and it was not until introducing the local ratio technique on top of it that we could find one. One basic scenario in the paradigm of local ratio technique is to "decompose" a cost function w defined on a problem instance I into many "slices" of cost functions $w_0, w_1, \ldots, w_{k-1}$, such that $w = \sum_i w_i$ and $w_i \ge 0, \forall i$. It is expected that an easily computable solution such as a minimally feasible solution, is a good enough approximation to the optimal one under each of w_i's, and if so, putting all such solutions together would yield a good approximation in the original instance.

A brief overview of our algorithm can be stated now as follows. It first decomposes (G, c) into uniformly costed instances of $(G_0, c_0), (G_1, c_1), \ldots, (G_{k-1}, c_{k-1})$, and it does so according to the costs of edges in an MST T. The algorithm next employs the primal-dual schema on each slice of (G_i, c_i)'s, and sets up a dual solution y^i for each of them. Finally, it determines which leaves of T to be removed using these y^i's. So in our algorithm, both of the primal-dual schema and the local ratio technique are used in an interlaced fashion. As mentioned earlier, quite a number of approximation algorithms have been developed so far using either of these two methods, yet to the best of our knowledge, no algorithm has been designed based on both.

2 Algorithm

Let $S \subseteq V$ be the set of "special" nodes, and M be a matching on a spanning tree T rooted at r. We say M is *dense* if

- r and all the special nodes are left unmatched by M, and
- every internal node ($\neq r$) of T with none of its children special is matched by M.

(Note: it does not matter for M to be dense whether any internal node having a special child or any leaf is matched by M or not). A dense matching $M \subseteq T$ can be efficiently computed by a DFS-like procedure (see Fig. 1).

Let T denote any MST in G. Suppose that T consists of edges with k different costs, $w_0, w_1, \ldots, w_{k-1}$, $(k \le n-1)$ such that $w_0 < w_1 < \cdots < w_{k-1}$. Let $\Delta_0 = w_0$ and $\Delta_i = w_i - w_{i-1}$ for $1 \le i \le k-1$ (so, $\Delta_i > 0, \forall i$). In the following algorithm a sequence of trees, $T_1, T_2, \ldots, T_{k-1}$, and a sequence of graphs, $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_{k-1} = (V_{k-1}, E_{k-1})$, will be generated from $T_0 = T$ and the original graph $G_0 = G$, respectively, in such a way that T_{i+1} (G_{i+1}) is the one obtained from T_i (G_i, resp.) by contracting all the edges of cost w_i in T_i. Such contractions might introduce parallel edges and/or self-loops in G_i's (but not in T_i's), and we may keep all the parallel edges but none of the self-loops.

When tree edges are contracted, the set of vertices connected together by these edges is replaced by a single new vertex (and it becomes a new root labeled r if r is among those merged into one), and such vertices in G_i's are called *s-nodes* (for *special* nodes). Clearly, any s-node u in any G_i corresponds naturally to some set S of vertices, all of them connected together by contracted edges, in original G. Let $D(u) \subseteq V$ denote the set of vertices merged into an s-node u. Then, $\delta_{G_i}(u) = \delta_G(D(u))$ for any s-node $u \in V_i$, and hence, $\delta^-_{\vec{G}_i}(u)$ coincides with the r-edge cut $\delta^-_{\vec{G}}(D(u))$ (for dependent $D(u)$) if $u \neq r$.

Given $G = (V, E)$ and $r \in V$, the algorithm TC computes a tree cover rooted at r (see Fig. 2). Starting with $G_0 = G$ and $T_0 =$ any MST T in G, it computes a sequence of graphs, $G_1, \ldots, G_{k-1}$, and

```
Initialize $M = \emptyset$, and mark root $r$ and all the special nodes "matched".
Call DFS-MATCH($r$).

DFS-MATCH($u$)
If $u$ is a leaf then return
If $u$ is unmatched and has an unmatched child $v$ then
    Pick $e = \{u, v\}$ and add it to $M$ by setting $M \leftarrow M \cup \{e\}$.
    Mark both $u$ and $v$ "matched".
For each child $v$ of $u$ do
    Call DFS-MATCH($v$).
```

Figure 1: A DFS-like procedure for computing a dense matching M on tree T

```
1. Set $G_0 \leftarrow G, T \leftarrow$ any MST in $G$, and $T_0 \leftarrow T$. /* initialization */
2. For $i = 0$ to $k - 1$ do    /* $M_i$'s and $S_i$'s are constructed in this phase */
    2-1. Let $S_i$ be the set of $s$-nodes in $G_i$.    /* set $y^i_D = \Delta_i, \forall D \in \mathcal{D}(S_i)$ */
    2-2. Compute a dense matching $M_i$ on $T_i$.    /* set $y^i_e = \Delta_i, \forall e \in M_i$ */
    2-3. Let $T_{i+1}(G_{i+1})$ be the tree (graph) obtained by contracting all the edges of cost $w_i$ within
         $T_i$.
3. For each leaf edge $e$ of $T$ do
    3-1. Set $\bar{l}_e = l_e - \sum_{i:e \in M_i} \Delta_i$.    /* $= l_e - \sum_{e \in M_i} y^i_e$ */
4. While there exists an edge $f$ between two leaves of $T$, $u$ and $v$, with $\min\{\bar{l}_{e(u)}, \bar{l}_{e(v)}\} > 0$ do
    4-1. Set $y_f = \min\{\bar{l}_{e(u)}, \bar{l}_{e(v)}\}$.
    4-2. Subtract $y_f$ from each of $\bar{l}_{e(u)}$ and $\bar{l}_{e(v)}$.
5. Let $T_{\rm tc} \leftarrow$ ($T$ with any of its leaf edges $e$ removed if $\bar{l}_e > 0$), and output $T_{\rm tc}$.
```

Figure 2: Algorithm TC for computing a tree cover T in G

a sequence of trees, $T_1, \dots, T_{k-1}$, for each $0 \leq i \leq k-2$, by contracting all the edges of cost w_i on T_i (in Step 2). At the same time a set $S_i \subseteq V_i$ of s-nodes and a dense matching $M_i \subseteq T_i$ are constructed for each $0 \leq i \leq k-1$. Call an edge of a tree T *leaf edge* if it is incident to a leaf u of T, and denote it $e(u)$ (or call an arc of $\vec{T}$ *leaf arc*, and denote it $\vec{e}(u)$). In Step 3 the "residual" cost $\bar{l}_e$ on each leaf edge e of T is set to initial cost l_e less $\sum_{i:e \in M_i} \Delta_i$. Using these residual costs, duals on those edges connecting leaves of T are maximally increased in Step 4; for any f between leaves u and v, y_f is set to a maximal value such that y_f does not exceed either of $\bar{l}_{e(u)}$ and $\bar{l}_{e(v)}$, y_f is next subtracted from each of $\bar{l}_{e(u)}$ and $\bar{l}_{e(v)}$, and repeated by going to any other edge connecting leaves of T, until no longer possible to raise duals on such edges. So after this step, no residual cost remains positive on at least one of leaf edges $e(u)$ and $e(v)$ for any pair of leaves u and v of T connected by an edge. The algorithm outputs an r-tree cover T_{tc} by trimming any leaf edge of T with a positive residual cost still remaining on it.

It is rather easy to see that T_{tc} thus computed is indeed an r-tree cover since 1) the internal structure of T (i.e., the subtree of T obtained by removing all the leaves from T) is completely maintained in T_{tc}, and 2) for any edge connecting two leaves of T, at least one of them is kept in T_{tc} as well.

Now the whole algorithm is to pick any edge $e = \{u, v\}$ in given G, compute both of u- and v-tree covers by calling TC twice, and choose the lighter of them as a tree cover for G.

3 Dual Solution and its Feasibility

In this section we show how a dual feasible solution y is computed ***implicitly*** within the algorithm, along with an r-tree cover T_{tc}, and that it is feasible to (D). Let us begin with an easy but very basic observation:

Lemma 1. *For any $e \in E$, $e \notin E_i$ if $l_e < w_i$.*

Proof. For the sake of contradiction, suppose there exists $e \in E_i$ with $l_e < w_i$. Since T_i is the tree resulting from contracting all the edges of cost $< w_i$ within T, e cannot occur within T_i.

So, $e \in E_i - T_i$. Since we always shrink edges of spanning T, every T_i is a spanning tree in G_i. If e not in T_i is lighter than any edge of T_i, a spanning tree T'_e strictly lighter than T_i would arise in G_i, by adding e to T_i and removing some edge from T_i, say e' (recall that e cannot be a self-loop). But then, $(T - \{e\}) \cup \{e'\}$, which is strictly lighter than T, would be a spanning tree in G, and this contradicts the fact that T is an MST in G. □

During the first phase (i.e., within the for-loop of Step 2) of algorithm TC, a dense matching M_i and a set S_i of s-nodes are computed for each i. Recall that $D(u) \subseteq V$ denotes the set of those vertices merged into an s-node u by edge contraction, and let $\mathcal{D}_i = \{D(u) \mid u \in S_i\}$. A dual solution y is set up by letting each of y_e ($e \in M_i$) and y_D ($D \in \mathcal{D}_i$) be given a fixed nonnegative value uniformly for each i as follows; y will then be determined by the component-wise accumulation of them:

1. For each i
$$y_e^i = \begin{cases} \Delta_i & \text{if } e \in M_i \\ 0 & \text{otherwise} \end{cases}$$
and $y_e = \sum_i y_e^i = \sum_{i:e \in M_i} \Delta_i$ for each edge $e \in T$;
2. For each i
$$y_D^i = \begin{cases} \Delta_i & \text{if } D \in \mathcal{D}_i \\ 0 & \text{otherwise} \end{cases}$$
and $y_D = \sum_i y_D^i = \sum_{D \in \mathcal{D}(S_i)} \Delta_i$ for any dependent set $D \subseteq V - \{r\}$.

(Note: Quite possibly, an edge $e \in M_i$ could happen to be identical to $D(u) \in \mathcal{D}_{i'}$ for some s-node $u \in S_{i'}$ if $i \neq i'$. If so, y_e and $y_{D(u)}$ actually correspond to the same component of y. For a clearer presentation, however, they will be distinguished from each other in the sequel.)

Any edge in any M_i is certainly an edge of an MST T, and any D in any $\mathcal{D}(S_i)$ is the vertex set of some subtree of T as all the vertices in D are merged into an s-node by contracting edges of T. We will also need to use an r-edge cut of form $\delta^-(D)$ such that D is *not* a part of T, and its value y_D is to be explicitly determined by the algorithm:

3. y_e = as assigned at step 4 in algorithm TC, for any e joining leaves of T.

From the way the algorithm assigns values to these y_e's (at step 4) and that any leaf edge f is removed at step 5 if $\bar{l}_f > 0$, it is clear that, for any leaf edge $f = e(u)$, $l_f = \sum_{f \in M_i} \Delta_i + \sum(y_e : e \in E$ connects u with another leaf of $T)$ if f remains in T_{tc}.

By setting all the other dual variables to zero, the r-edge cut packing is completed, and this is the dual solution $y \in \mathbb{R}^{\mathcal{D}}$ which in what follows will be paired with the integral primal solution $\vec{T}_{\mathrm{tc}} \subseteq \vec{E}$, the directed counterpart of the r-tree cover computed.

Lemma 2. *For all $\vec{e} = (u,v) \in \vec{E}$, $Y_{\vec{e}} = \sum_{D\in\mathcal{D}:\vec{e}\in\delta^-(D)} y_D \le l_e$.*

Proof. Let T be an MST used in algorithm TC.

Case v is an internal node of T. Notice first that the edges $\in M_i$ and the vertex sets $\in \mathcal{D}_i = \{D(u) \mid u \in S_i\}$ are mutually vertex disjoint in G for each i since so are the edges $\in M_i$ and the nodes $\in S_i$ in G_i. Therefore, at most one among the edges $\in M_i$ and the vertex sets $\in \mathcal{D}_i$ contains v in it, and hence, the contribution to $Y_{\vec{e}}$ from y^i_f's ($f \in M_i$) and y^i_D's ($D \in \mathcal{D}_i$) together is at most Δ_i, for each $0 \le i \le k-1$. All the other dependent sets with positive duals are such edges that are incident to leaves only, and they do not show up within $\sum_{D\in\mathcal{D}:\vec{e}\in\delta^-(D)} y_D$.

By Lemma 1, if $w_j \le l_e < w_{j+1}$, e does not appear in G_i for $i = j+1, \dots, k-1$. Therefore,

$$\begin{aligned}
Y_{\vec{e}} &\le \sum_{v\in e\in M_i} \Delta_i + \sum_{v\in D\in\mathcal{D}} \Delta_i \\
&= \sum_{i=0}^{j} [(\Delta_i : v \in e \in M_i) + (\Delta_i : v \in D \in \mathcal{D})] \\
&\le \Delta_0 + \Delta_1 + \cdots + \Delta_j \\
&= w_0 + (w_1 - w_0) + \cdots + (w_j - w_{j-1}) = w_j.
\end{aligned}$$

Case v is a leaf of T. Let f denote $e(v)$, the leaf edge of T incident to v. Suppose $w_j \le l_e < w_{j+1}$ and $l_f = w_{j'}$. Because T is an MST in G it must be the case that $l_f \le l_e$, and thus, $j' \le j$. Since $l_f = \sum_{i=0}^{j'} \Delta_i$, $l_e \ge w_j = \sum_{i=0}^{j} \Delta_i = l_f + \sum_{i=j'+1}^{j} \Delta_i$.

Since v is a leaf of T, among the duals assigned on the edges of dense matchings, only those placed on f can contribute to $Y_{\vec{e}}$.

Certainly, v does not become an s-node before $f = e(v)$ gets contracted at $i = j'+1$, whereas e disappears at $i = j+1$ and thereafter. Therefore, if $\vec{e} \in \delta^-(D(w))$ for some s-node w, w can occur only in G_i's for $i = j'+1, \dots, j$, and the total contribution of $y_{D(w)}$'s for s-nodes w to $Y_{\vec{e}}$ is at most $\sum_{i=j'+1}^{j} \Delta_i$.

What remains to be accounted for are the duals on those edges joining v with other leaves of T. Whereas actual values placed on those edges are determined within the algorithm, it can be observed, from the way it works, that

$$\sum(y_g : g \in E \text{ joins } v \text{ with another leaf of } T) \le \bar{l}_f$$

where $\bar{l}_f = l_f - \sum(\Delta_i : f \in \text{a dense matching } M_i)$. Therefore,

$$\begin{aligned}
Y_{\vec{e}} &\le \sum(\Delta_i : f \in \text{a dense matching } M_i) + \\
&\quad \sum_{i=j'+1}^{j} \Delta_i + \bar{l}_f \\
&= l_f + \sum_{i=j'+1}^{j} \Delta_i = w_j.
\end{aligned}$$

□

It follows from this lemma that the dual solution $y \in \mathbb{R}^{\mathcal{D}}$ set up as above is feasible to (D).

4 Approximation Ratio

In approximation algorithms based on the primal-dual schema, the approximation ratios are obtained by relating the value of a computed integral (primal) solution with that of a simultaneously computed dual solution, and these values are usually related by means of complementary slackness conditions, in somehow relaxed forms. In case of (P) and (D), these conditions can be stated as follows, where α and β (with each ≥ 1) denote relaxation factors of the respective conditions:

PCSC (Primal Complementary Slackness Conditions): For each $\vec{e} \in \vec{E}$, $x_{\vec{e}} > 0$ implies that $l_e/\alpha \le \sum_{D\in\mathcal{D}:\vec{e}\in\delta^-(D)} y_D \le l_e$.

DCSC (Dual Complementary Slackness Conditions): For each $D \in \mathcal{D}$, $y_D > 0$ implies that $1 \le x(\delta^-(D)) \le \beta$.

It can be shown that, if an algorithm produces x and y satisfying the conditions above, its approximation ratio is at most $\alpha\beta$.

In case of algorithm TC, however, the primal solution $\vec{T}_{tc}$ and the dual solution y are not related in such a simple manner (even in the unweighted case of $l = \vec{1}$), and the way they satisfy PCSC and DCSC can be seen to be as follows:

- for each $\vec{e} \in \vec{E}$ with $x_{\vec{e}} > 0$ (i.e., $\vec{e} \in \vec{T}_{tc}$),
  - PCSC may not hold for any α if $\vec{e}$ is not a leaf arc of $\vec{T}$,
  - PCSC is satisfied at $\alpha = 1$ if $\vec{e}$ is a leaf arc of $\vec{T}$, and
- for each $D \in \mathcal{D}$ with $y_D > 0$,
  - DCSC is satisfied at $\beta = 1$ if D is an edge in a dense matching M_i or $D = D(u)$ for some s-node u (Note: in either case D corresponds to a subtree of T),

- DCSC is satisfied at $\beta = 2$ if D is an edge connecting two leaves of T.

We will use more direct arguments in what follows to show that an r-tree cover $\vec{T}_{tc}$ computed by the algorithm is of cost no more than twice the cost of the dual feasible solution y computed simultaneously.

The first idea is to "decompose" T into the uniformly weighted trees, $T_0, T_1, \ldots, T_{k-1}$, where every edge of T_i is of cost Δ_i, and then to pay for at least *half* the costs of all the internal (i.e., non-leaf) arcs of $\vec{T}_i$ (and possibly more) by the duals associated with the edges in M_i and the nodes in S_i, for each i. The dual value placed on each $e \in M_i$ and each $u \in S_i$ is Δ_i, and so we may use it to pay for half the costs of two arcs in $\vec{T}_i$. Suppose we use y_e^i to pay for the costs of $\vec{e}$ itself and the arc preceding $\vec{e}$ in $\vec{T}_i$, $\Delta_i/2$ to each, for each $e \in M_i$. Likewise, $y_{D(u)}^i$ is used to pay for the costs of the arc of $\vec{T}_i$ entering to an s-node u and its predecessor in $\vec{T}_i$, again $\Delta_i/2$ to each.

Lemma 3. *Every non-leaf arc of $\vec{T}_i$ gets at least half paid by the duals on M_i and S_i.*

Proof. Let $\vec{e} = (u, v)$ be a non-leaf arc of $\vec{T}_i$. If v is an s-node, $\vec{e}$ gets half paid by $y_{D(v)}$. Since v is not a leaf, it has at least one child, and if any of them is an s-node, say w, then, as $\vec{e}$ is a predecessor of (v, w), $\vec{e}$ gets half paid by $y_{D(w)}$. So, assume that neither of v nor any of its children is an s-node. Then, a matching M_i on T_i must match v if it is dense. If so, $e \in M_i$, or otherwise, $\{v, w\} \in M_i$ for some child w of v, and in either case, $\vec{e}$ gets half paid by either e or $\{v, w\}$. □

Let us now say that an arc $\vec{e}$ in $\vec{T}$ is *half paid* if at least half of its cost, $l_e/2$, is paid in total to e by the duals on M_i's and $\mathcal{D}(S_i)$'s.

Lemma 4. *Every non-leaf arc of $\vec{T}$ gets half paid.*

Proof. By Lemma 3, every non-leaf arc of $\vec{T}$ gets paid for at least $\Delta_i/2$ everytime it occurs in $\vec{T}_i$. Suppose $l_e = w_j$. Then, e appears in $G_0, G_1, \ldots,$ up to G_j (but no further). The total amount paid to $\vec{e}$ is hence at least $\Delta_0/2 + \Delta_1/2 + \cdots + \Delta_j/2 = (\Delta_0 + \Delta_1 + \cdots + \Delta_j)/2 = w_j/2 = l_e/2$. □

To account next for the cost of any leaf arc $\vec{e}(u)$ of $\vec{T}$ remaining in $\vec{T}_{tc}$, let us recall that $\bar{l}_e = l_e - \sum_{i:e \in M_i} \Delta_i$ and that $\vec{e}(u)$ remains in $\vec{T}_{tc}$ only if the dual values on the edges joining u with other leaves total to $\bar{l}_e$; that is, $\sum(y_f : f \in E$ joins u with another leaf of $T) = \bar{l}_e$. Thus, if we spend y_f, for any f joining two leaves w and z of T, to pay for the costs of the leaf arcs of $\vec{T}$, $\Delta_i/2$ each to $\vec{e}(w)$ and $\vec{e}(z)$, a leaf arc $\vec{e}$ avoids being got rid of only if it gets half paid. To be precise, one half of $\sum_{i:e \in M_i} \Delta_i$ and one half of $\bar{l}_e$ get paid to $\vec{e}$, totaling to $(\sum_{i:e \in M_i} \Delta_i)/2 + (l_e - \sum_{i:e \in M_i} \Delta_i)/2 = l_e/2$. Therefore,

Lemma 5. *Every leaf edge of $\vec{T}$ remaining in $\vec{T}_{tc}$ also gets half paid.*

It follows immediately from Lemmas 4 and 5 that algorithm TC computes an r-tree cover $\vec{T}_{tc} \subseteq \vec{T}$ and a dual feasible $y \in \mathbb{R}^{\mathcal{D}}$ such that the cost $l(\vec{T}_{tc})$ of $\vec{T}_{tc}$ is no larger than twice the value $\sum_{D \in \mathcal{D}} y_D$ of y. Therefore,

Theorem 6. *The algorithm TC approximates the minimum cost r-tree cover to within a factor of* 2*; consequently, the approximation ratio of the whole algorithm for approximating the minimum cost tree cover is bounded by* 2.

The integrality gap of (P) is known to be no smaller than 2 [15].

Corollary 7. *The integrality gap of (P) is* 2 *when the graph is in the bidirected form of an undirected graph.*

It is also clear, from the way the dual solution $y \in \mathbb{R}^{\mathcal{D}}$ is determined as above, that y can be ensured to be integral if l is integral:

Corollary 8. *When l is integral and the graph is in the bidirected form of an undirected graph in (D), there exists an integral r-edge cut packing the cost of which is at least* 1/2 *of the cost of an optimal fractional r-edge cut packing. Moreover, such an integral r-edge cut packing is efficiently computable.*

5 Final Remarks

The paper has shown that the minimum cost tree cover can be efficiently approximated to within a factor of 2 of the optimum. As the minimum tree cover problem is as hard to approximate as the minimum vertex cover problem [11], a further improvement on this factor would imply that the minimum vertex cover problem is approximable within a factor better than 2, which has been conjectured by some to be highly unlikely [14].

A natural and equally interesting direction of further research would be in the directed version of the tree cover (DTC) problem; given here is a *directed* graph G, and it is required to compute a directed tree (a branching) T of minimum cost in G such that either head or tail (or both of them) of every arc in G is touched by T. As mentioned in [15], the problem has remained wide open. If G is unweighted, however, it is not hard to find a 2-approximation for it, by extending the approach of the current paper a bit further. Letting V' be the set of vertices reachable from the root vertex r, compute an arborescence T spanning V' entirely (and V' must be a vertex cover for G if it is a feasible instance of DTC). Compute next a dense matching M on T (with no s-nodes), and while there exists an arc connecting two unmatched leaves, add it to M. Finally trim any leaf u from T if u is unmatched *and* there is no arc entering to u from $V - V'$.

Such an approach appears to fall short, however, once arbitrary arc costs are allowed on G. In fact one can come up with an instance in which any feasible solution contained in any spanning arborescence incurs such a cost larger than the optimum by an unbounded factor. Thus, the approximability of minimum cost DTC problem, as well as a related issue of the integrality gap of (P) on arbitrary directed graphs, still remain wide open.

References

[1] E.M. Arkin, M.M. Halldórsson and R. Hassin. Approximating the tree and tour covers of a graph. *Inform. Process. Lett.*, 47:275–282, 1993.

[2] R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *Algorithmica*, 27(2):131–144, 2000.

[3] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.

[4] R. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. *SIAM J. Discrete Math.*, 19(3):762–797, 2005.

[5] F.A. Chudak, M.X. Goemans, D.S. Hochbaum, and D.P. Williamson. A primal-dual interpretation of recent 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Oper. Res. Lett.*, 22:111–118, 1998.

[6] I. Dinur and S. Safra. The importance of being biased. In *Proc. 34th ACM Symp. Theory of Computing*, pages 33–42, 2002.

[7] J. Edmonds. Optimum branchings. *J. Res. Nat. Bur. Standards B*, 71:233–240, 1967.

[8] T. Fujito. On approximability of the independent/connected edge dominating set problems. *Inform. Process. Lett.*, 79(6):261–266, 2001.

[9] T. Fujito and H. Nagamochi. A 2-approximation algorithm for the minimum weight edge dominating set problem. *Discrete Appl. Math.*, 118:199–207, 2002.

[10] D.R. Fulkerson. Packing rooted directed cuts in a weighted directed graph. *Math. Programming*, 6:1–13, 1974.

[11] M.R. Garey and D.S. Johnson. The rectilinear Steiner-tree problem is NP-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977.

[12] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM J. Comput.*, 31(5): 1608-1623, 2002.

[13] G. Karakostas. A better approximation ratio for the vertex cover problem. In *Proc. 32nd ICALP*, pages 1043–1050, 2005.

[14] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2 - \epsilon$. In *Proc. 18th IEEE Conf. Computational Complexity*, pages 379–386, 2003.

[15] J. Könemann, G. Konjevod, O. Parekh and A. Sinha. Improved approximations for tour and tree covers. *Algorithmica*, 38(3): 441–449, 2003.

[16] B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informat.*, 22:115–123, 1985.

[17] O. Parekh. Edge dominating and hypomatchable sets. In *Proc. 13th ACM-SIAM Symp. Discrete Algorithms*, pages 287–291, 2002.

[18] C. Savage. Depth-first search and the vertex cover problem. *Inform. Process. Lett.*, 14(5):233–235, 1982.