A better approximation algorithm for the budget prize collecting tree problem

Asaf Levin[1]

Faculty of Industrial Engineering and Management

Technion

November 6, 2003

Abstract

Given an undirected graph $G = (V, E)$, an edge cost $c(e) \geq 0$ for each edge $e \in E$, a vertex prize $p(v) \geq 0$ for each vertex $v \in V$, and an edge budget B. The BUDGET PRIZE COLLECTING TREE PROBLEM is to find a subtree $T' = (V', E')$ that maximizes $\sum_{v \in V'} p(v)$, subject to $\sum_{e \in E'} c(e) \leq B$. We present a $(4 + \epsilon)$-approximation algorithm.

Keywords: Approximation algorithms.

1 Introduction

Suppose that we are given an undirected graph $G = (V, E)$, a non-negative edge cost $c(e)$ for each edge $e \in E$, and a non-negative vertex prize $p(v)$ for each vertex $v \in V$. Motivated by applications to local access network design, Johnson, Minkoff and Phillips [6] considered the following problems:

- The QUOTA PRIZE COLLECTING TREE PROBLEM (QPCT): given a prize quota $Q > 0$, find a subtree $T' = (V', E')$ that minimizes $\sum_{e \in E'} c(e)$ subject to $\sum_{v \in V'} p(v) \geq Q$.
- The BUDGET PRIZE COLLECTING TREE PROBLEM (BPCT): given a budget $B \geq 0$, find a subtree $T' = (V', E')$ that maximizes $\sum_{v \in V'} p(v)$, subject to $\sum_{e \in E'} c(e) \leq B$.

The k-MST problem is the special case of the QPCT where $p(v) = 1 \; \forall v \in V$ and $Q = k$. Arora and Karakostas [1] provided a $(2 + \epsilon)$-approximation algorithm for the k-MST, improving the 3-approximation algorithm of Garg [2]. As observed in [6], any α-approximation algorithm for the k-MST problem yields a pseudo-polynomial time α-approximation algorithm for the corresponding QPCT. Moreover, using properties of the Goemans-Williamson primal-dual algorithm [3] applied during Garg's k-MST approximation algorithm, the pseudo-polynomial time algorithm can be

[1]Corresponding author: Asaf Levin, Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel. E-mail: levinas@tx.technion.ac.il

transformed into a polynomial time algorithm. This fact was proved in [6] for Garg's algorithm, however it holds also for the algorithm of Arora and Karakostas. To conclude, the best available approximation ratio for the QPCT is $(2+\epsilon)$.

With respect to the BPCT, Johnson, Minkoff and Phillips [6] provided a $(5+\epsilon)$-approximation algorithm. They also noted that in order to get a better bound for the BPCT, they need an algorithm for the QPCT with a performance guarantee of 2 or less, in which case the performance guarantee for BPCT would drop to $3+\epsilon$.

In this paper we show a $(4+\epsilon)$-approximation algorithm for the BPCT.

Denote by OPT, the maximum value of BPCT. For a tree $T=(V_T,E_T)$, a cost function $c: E \to I\!R^+$ and a prize function $p: V \to I\!R^+$, we denote $c(T)=\sum_{e\in E_T} c(e)$ and $p(T)=\sum_{v\in V_T} p(v)$.

2 A $(4+\epsilon)$-approximation algorithm for BPCT

Algorithm BPCT (see Figure 1) uses a geometric search among values $(1+\epsilon)^k \max_{v\in V} p(v)$, $k=1,2,\ldots$ to find an approximate value for OPT. More precisely, we try to find a value $\hat{Q}$ such that $\frac{1}{1+\epsilon}OPT \le \hat{Q} \le OPT$. Since $\max_{v\in V} p(v) \le OPT \le \sum_{v\in V} p(v) \le n \max_{v\in V} p(v)$, the number of iterations during the binary search is $O(\log_{1+\epsilon} n)$. In each iteration of the binary search, we approximate the QPCT problem with quota $\hat{Q}$. In order to approximate QPCT, we use a $2\frac{1}{4}$-approximation algorithm. We note that we can use any approximation algorithm for QPCT with a performance guarantee of less than $2\frac{1}{2}$. There exists such an algorithm using the scheme of [1] and the equivalence between the QPCT and the k-MST proved by [6]. If $\hat{Q} \le OPT$, then the cost of the resulting tree $T^{\hat{Q}}$ is at most $2\frac{1}{4}B$ (because we use a $2\frac{1}{4}$-approximation algorithm). Then, we decompose $T^{\hat{Q}}$ to at most four subtrees that cover the vertex set of $T^{\hat{Q}}$ (this is done using Algorithm *general_tree_cover* that is presented in Section 3), each of them costs at most B. We pick the candidate $T_{\hat{Q}}$ as the tree that has a maximum prize among them. Among all the values of $\hat{Q}$ that returned a feasible solution, we pick the best tree.

Remark 1 *Using Hassin's geometric-mean binary search method [5], we can find in $O(\log\log_{1+\epsilon} n)$ iterations (instead of $O(\log_{1+\epsilon} n)$ iterations) a value $\hat{Q}$ such that the $2\frac{1}{4}$-approximate solution for $QPCT(G,c,p,\hat{Q})$ costs at most $2\frac{1}{4}B$ whereas the approximate solution for $QPCT(G,c,p,(1+\epsilon)\hat{Q})$ costs more than $2\frac{1}{4}B$.*

BPCT

input

An undirected graph $G = (V, E)$.

A cost $c : E \to I\!R^+$.

A prize $p : V \to I\!R^+$.

A real number $B > 0$.

returns

A tree $T' = (V', E')$.

begin

$\hat{Q} := \max_{v \in V} p(v)$.

$T' = (\emptyset, \emptyset)$.

while $\hat{Q} \leq \sum_{v \in V} p(v)$

$T^{\hat{Q}} :=$ *a* $2\frac{1}{4}$*-approximate solution for* $QPCT(G, c, p, \hat{Q})$.

if $c(T^{\hat{Q}}) \leq 2\frac{1}{4}B$

then

$TS :=$*general_tree_cover*$(T^{\hat{Q}}, c, B)$.

$T_{\hat{Q}} := argmax_{\hat{T} \in TS} p(\hat{T})$.

if $p(T_{\hat{Q}}) \geq p(T')$

then

$T' := T_{\hat{Q}}$.

end if

$\hat{Q} := \hat{Q}(1 + \epsilon)$.

else $\{\hat{Q} > OPT$ *and we stop the loop* $\}$

$\hat{Q} := \sum_{v \in V} p(v) + 1$.

end if

end while

return T'.

end *BPCT*

Figure 1: Algorithm *BPCT*

In order to analyze the performance of the algorithm, it suffices to show that during the iteration in which $\frac{1}{1+\epsilon}OPT \leq \hat{Q} \leq OPT$ Algorithm *BPCT* returns a solution whose prize is at least $\frac{1}{(4+\epsilon)}OPT$.

This last claim holds by the following argument: since $p(T_{\hat{Q}}) \geq \frac{1}{4}p(T^{\hat{Q}})$ and $p(T^{\hat{Q}}) = \hat{Q} \geq \frac{1}{1+\epsilon}OPT$, we get $p(T') \geq p(T_{\hat{Q}}) \geq \frac{1}{4(1+\epsilon)}OPT$. T' is feasible by Lemma 5. We conclude by the following theorem:

Theorem 2 *For every $\epsilon \geq 0$, there is a polynomial time $(4+\epsilon)$-approximation algorithm for the BPCT.*

3 Algorithm general_tree_cover

In this section we first assume that $c(e) \leq \frac{B}{2}$ for each edge $e \in E_T$, and present Algorithm *tree_cover* for this case. Afterwards, we show how the assumption that $c(e) \leq \frac{B}{2}$ for each edge $e \in E_T$ can be removed without any loss in performance.

Algorithm *tree_cover* (see Figure 2) is a weighted variant of the *rooted_tree_cover* procedure of Goldschmidt, Hochbaum, Levin and Olinick [4]. We root T at a vertex *root*, and assume that the vertices of T passed to *tree_cover* have been assigned labels according to a post-order listing. For a vertex i, denote by T_i the subtree of T rooted at i (T changes throughout the algorithm and T_i changes accordingly). For a vertex i, we denote by $\pi(i)$ its father in T, and by $ch(i)$ a list of all the children of i in T. Given a tree T and a subtree T' of T, $T \setminus T'$ is the tree obtained from T after we remove from T the edge set of T' and afterwards we remove any isolated vertex in the resulting tree. Note that in general $T \setminus T'$ need not be a tree, however in Algorithm *tree_cover* the resulting forest is indeed a tree. Given a tree $T' = (V', E')$ and an edge (i, j) where $i \in V'$ and $j \notin V'$, $T' \cup \{(i, j)\}$ is the tree $(V' \cup \{j\}, E' \cup \{(i, j)\})$. Given a pair of subtrees $T' = (V', E')$ and $T'' = (V'', E'')$ of a common tree T such that $V' \cap V'' \neq \emptyset$, $T' \cup T''$ is the subtree of T $(V' \cup V'', E' \cup E'')$. We assume that $c(e) \leq \frac{B}{2}$ for every edge $e \in E_T$.

We use Algorithm *tree_cover* with a tree T such that $c(T) \leq 2\frac{1}{4}B$. We note that Algorithm *tree_cover* finds a set of trees that cover E_T, however we need only a cover of V_T. We generalize Lemma 3.1 of [4] to our weighted case.

Lemma 3 *The collection SOL returned by Algorithm tree_cover satisfies the following properties:*

1. *Each tree in SOL costs at most B.*

tree_cover
input
A tree $T = (V_T, E_T)$.
A real number $B > 0$.
A cost $c : E_T \rightarrow I\!R^+$, such that $c(e) \leq \frac{B}{2}\ \forall e \in E_T$.
returns
A set of trees, each of weight at most B, covering E_T.
begin
if $c(T) \leq B$
then return $\{T\}$.
end if
$SOL := \emptyset$.
{Phase 1}
$i := 1$.
while $i \leq n$
if $c(T_i) < \frac{B}{2}$ and $c(T_i) + c_{i,\pi(i)} \geq \frac{B}{2}$
then
$T' := T_i \cup \{(i, \pi(i))\}$.
$SOL := SOL \cup \{T'\}$.
$T := T \setminus T'$.
elseif $\frac{B}{2} \leq c(T_i) \leq B$
$SOL := SOL \cup \{T_i\}$.
$T := T \setminus T_i$.
elseif $c(T_i) > B$
$p := \max\{l \in ch(i) | \sum_{\substack{j \in ch(i) \\ j \leq l}} (c(T_j) + c_{i,j}) \leq B\}$.
$T' := \bigcup_{\substack{j \in ch(i) \\ j \leq p}} (T_j \cup \{(i,j)\})$.
$SOL := SOL \cup \{T'\}$.
$T := T \setminus T'$.
$i := i - 1$. {i does not change in the current iteration}
end if
$i := i + 1$.
end while
{Phase 2}
if T is not empty
then
$T' :=$ a tree from SOL such that T, T' have a common vertex.
if $c(T) + c(T') > B$
then
$SOL := SOL \cup \{T\}$.
else
$SOL := (SOL \setminus \{T'\}) \cup \{T \cup T'\}$.
end if
return SOL.
end tree_cover

Figure 2: Algorithm *tree_cover*

2. The average cost of a tree in SOL is at least $\frac{B}{2}$.

Proof: We first argue that Algorithm *tree_cover* maintains the invariant that $c(T_i) + c_{i,\pi(i)} \leq \frac{B}{2}$ after it visits vertex i. To see this note that when the algorithm visits i:

- If $c(T_i) < \frac{B}{2}$, then assume that the claim does not hold. I.e., we assume that $c(T_i)+c_{i,\pi(i)} > \frac{B}{2}$. In this case the algorithm adds $T' := T_i \cup \{(i, \pi(i))\}$ to SOL, and removes T' from T. Then, after the algorithm visits i, $c(T_i) = 0$, and therefore $c_{i,\pi(i)} > \frac{B}{2}$. This contradicts the assumption that $c(e) \leq \frac{B}{2}\ \forall e \in E_T$.

- If $\frac{B}{2} \leq c(T_i) \leq B$, then after it visits i, $c(T_i) = 0$, and the claim holds by the assumption that $c(e) \leq \frac{B}{2}\ \forall e \in E_T$.

- If $c(T_i) > B$, then in the current iteration of Phase 1 we do not increase i. Therefore, in the last iteration with the current value of i, we have $c(T_i) \leq B$, and the claim follows by one of the previous cases.

We assumed that each edge in T costs at most $\frac{B}{2}$. Therefore, by construction, during Phase 1 the algorithm never adds a tree with cost greater than B. During Phase 2, by the algorithm's invariant $c(T) \leq \frac{B}{2}$. Therefore, if the algorithm adds T to SOL then it never adds a tree with cost greater than B. If the algorithm adds $T' \cup T$ instead, then by construction $c(T' \cup T) \leq B$. Therefore, during Phase 2 the algorithm never adds a tree with cost greater than B.

We now show that during Phase 1 it never adds a tree with cost smaller than $\frac{B}{2}$. Let q be the smallest index such that $q \in ch(i)$ and $q > p$. By definition, $\sum_{\substack{j \in ch(i) \\ j \leq p}} (c(T_j)+c_{i,j})+(c(T_q)+c_{i,q}) > B$. By the algorithm's invariant, $c(T') = \sum_{\substack{j \in ch(i) \\ j \leq p}} (c(T_j) + c_{i,j}) > B - (c(T_q) + c_{i,q}) \geq \frac{B}{2}$. Therefore, any tree that we add to SOL during Phase 1, costs at least $\frac{B}{2}$.

If during Phase 2 we add T to SOL, then the average cost of T and T' is at least $\frac{B}{2}$. This holds because in this case $c(T) + c(T') > B$. If during Phase 2 we replace T' by $T' \cup T$ in SOL, then since T' was added to SOL during Phase 1, we conclude that $c(T' \cup T) \geq c(T') \geq \frac{B}{2}$. Therefore, the average cost of a tree in SOL is at least $\frac{B}{2}$. ∎

Corollary 4 *If we apply Algorithm tree_cover with a tree T such that $c(T) \leq 2\frac{1}{4}B$, then the resulting collection SOL includes at most 4 trees, each of them costs at most B.*

Next we generalize Algorithm *tree_cover* to find a cover of V_T in the case in which there are edges in T that cost more than $\frac{B}{2}$.

Lemma 5 *Consider a tree T such that $c(T) \leq 2\frac{1}{4}B$. We can find in polynomial time a collection $\{T^i = (V_i, E_i)\}$ of at most four trees, such that $c(T^i) \leq B$ $\forall i$, and $V_T = \bigcup_i V_i$ (even if T contains edges that cost more than $\frac{B}{2}$).*

Proof: Let $S = \{e \in E_T | c(e) > \frac{B}{2}\}$.

- If $S = \emptyset$, the claim holds by Corollary 4.
- If $|S| \geq 3$, then the removal of any three edges of S from T forms four connected components, each of them costs at most B.
- If $|S| = 2$, then after we remove S from T we get three connected components. Among these components, there is at most one that costs more than B (but at most $\frac{5}{4}B$). We use Algorithm *tree_cover* to cover this component by a pair of spanning trees. Each of the remaining components can be covered by a single tree. Together we obtain a collection of at most four spanning trees that covers V_T.
- If $|S| = 1$, then we remove S from T, and obtain two connected components. Among these components, there is at most one that costs more than B. The other component can be covered by a single tree. The larger component costs less than $1\frac{3}{4}B$ and therefore, Algorithm *tree_cover* (when applied to this component) returns a collection of at most three trees. This gives at most four trees that cover V_T.

■

Algorithm *general_tree_cover* is the algorithm described in the proof of Lemma 5.

4 Concluding remarks

We conclude this paper with the following remarks:

Remark 6 *Using an algorithm that is similar to Algorithm general_tree_cover, for every $\epsilon > 0$ given a tree $T = (V_T, E_T)$ whose cost is at most $(2 + \epsilon)B$, we can cover V_T by a set of three trees, each of them costs at most $(1\frac{1}{3} + \epsilon)B$.*

Remark 7 *Using the set of trees implied by Remark 6 during Algorithm BPCT instead of TS, we obtain a bicriteria $(3 + \epsilon, 1\frac{1}{3} + \epsilon)$-approximation algorithm for BPCT. I.e., for every instance, the*

algorithm returns a tree whose total cost is at most $(1\frac{1}{3} + \epsilon)B$*, and whose total prize is at least* $\frac{1}{3+\epsilon}OPT$*.*

References

[1] S. Arora and G. Karakostas, "A $2 + \epsilon$ approximation algorithm for the k-MST problem," *Proc. ACM-SIAM Symposium on Discrete Algorithms 2000*, 754-759, 2000.

[2] N. Garg, "A 3-approximation for the minimum tree spanning k vertices," *Proc. 37th Annual IEEE Symposium on Foundation of Computer Science 1996*, 302-309, 1996.

[3] M. X. Goemans and D. P. Williamson, "A general approximation technique for constrained forest problems," *SIAM J. Comput.*, **24**, 296-317, 1995.

[4] O. Goldschmidt, D. S. Hochbaum, A. Levin, E. V. Olinick, "The SONET edge-partition problem," *Networks*, **41**, 13-23, 2003.

[5] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, **17**, 36-42, 1992.

[6] D. S. Johnson, M. Minkoff and S. Phillips, "The prize collecting Steiner tree problem: theory and practice," *Proc. ACM-SIAM Symposium on Discrete Algorithms 2000*, 760-769, 2000.