Fall 2014 Project in Computer Science

Constant Approximation Algorithm for Node-Weighted Prize-Collecting Steiner Forest on Planar Graphs

Final report (7/01/2015)

Author: Mateusz Lewandowski *Supervisor: Carsten Moldenhauer*

1 Abstract

The 4-approximation Primal-Dual Algorithm for Node-Weighted Prize-Collecting Steiner Forest on Planar Graphs

In this project we give a primal-dual approximation algorithm for node-weighted prize-collecting Steiner forest problem (NWPCSF) and prove that it achieves 4 approximation factor guarantee on planar graphs. We simply use algorithm of Demaine et al. for node-weighted steiner forest on planar graphs [2] and (as they suggest in their *Theorem 4*) use a technique of Hajiaghayi and Jain [3] to generalize it to the prize-collecting version. We show that this gives us 4 approximation by carrying improved analysis given by Moldenhauer [4] of Demaine's algorithm over the prize-collecting setting.

2 Introduction

Consider a graph $G = (V, E)$ with a cost function on nodes $w : V \to Q_+$, a set of pairs of vertices (demands) $D = (s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ and a non-negative penalty function $\pi : D \to Q_+$.
The node-weighted prize-collecting Steiner forest problem is to find a set of vertices $F \subseteq V$ which minimizes the sum of costs of vertices in F plus penalties of pairs of vertices which are not connected in a subgraph of G induced by F.

Note that we can give an equivalent definiton of demands and penalties by specifying a penalties for each unordered pair of vertices. Simply set penalties for pairs of vertices which are not in D to 0. From now on we will use values π_{ij} to denote penalties. Let also $\Gamma(S)$ denote a set of vertices in $V - S$ incident to vertices from $S \subset V$ and let $S \odot (i, j)$ means that $|(i, j) \cap S| = 1$ (we will say that S separates vertices i and j)

Using this notation, we can state our problem with a following integer programming formulation

$$\begin{aligned}
&\min \sum_{v \in V} w_v x_v + \sum_{(i,j) \in V \times V} \pi_{ij} z_{ij} && (IP)\\
&s.t.\\
&\quad \sum_{v \in \Gamma(S)} x_v + z_{i,j} \geq 1 && \forall S \subset V, \quad (i,j) \in V \times V, \quad S \odot (i,j)\\
&\quad x_v \in \{0,1\} && \forall v \in V\\
&\quad z_{i,j} \in \{0,1\} && \forall (i,j) \in S \times S
\end{aligned}$$

Setting $x_v = 1$ corresponds to "buying" a vertex v (including v into solution F) and setting $z_{i,j} = 1$ corresponds to paying a penalty instead of connecting vertices i and j.

The dual of the linear relaxation of this program is:

$$
\begin{aligned}
&\max \sum_{S\subset V, S\odot(i,j)} y_{S_{ij}} && (DLP1)\\
&s.t.\\
&\quad \sum_{S:v\in\Gamma(S), S\odot(i,j)} y_{S_{ij}} \le w_v && \forall v \in V\\
&\quad \sum_{S:S\odot(i,j)} y_{S_{ij}} \le \pi_{i,j} && \forall (i,j) \in V\times V\\
&\quad y_{S_{ij}} \le 0 && \forall S\subset V, S\odot(i,j)
\end{aligned}
$$

The problem with this dual program is that it has different variables for each pair of vertices. Hence in a moat growing approach we don't know how to split the increase of the dual variable y_S corresponding to a moat between variables $y_{S_{ij}}$. Splitting them in some specified way won't give us a constant factor approximation. This is why the algorithm for node-weighted prize-collecting Steiner forest in general graphs by Bateni et al. [1] doesn't work better on planar graphs. (see Appendix B)
Fortunately Hajiaghayi and Jain in [3] gave a general approach of handling this issue of different variables induced by prize-collecting setting. Following their argumentation using their *Lemma 2.1*, Farkas lemma and their *Corollary 2.1* our dual becomes:

$$
\begin{aligned}
&\max \sum_{S\subset V} y_S && (DLP2)\\
&s.t.\\
&\quad \sum_{S:v\in\Gamma(S)} y_S \le w_v && \forall v\in V\\
&\quad \sum_{S\in\mathbb{S}} y_S \le \sum_{(i,j)\in V\times V, \mathbb{S}\odot(i,j)} \pi_{i,j} && \forall \mathbb{S}\in 2^{2^V}\\
&\quad y_S \le 0s && \forall S\subset V
\end{aligned}
$$

where $\mathbb{S}\odot(i,j)$ denotes that there exists $S\in\mathbb{S}$ such that $S\odot(i,j)$ (we say that family $\mathbb{S}$ separates vertices i and j if and only if there exists at least one set S belonging to this family $\mathbb{S}$ which separates vertices i and j).

Fact 1 *Linear programs DLP1 and DLP2 are equivalent*

Proof Follows exactly the same as in [3] ■

Note that $\mathbb{S}$ is a family of subsets of vertices and our dual has double exponential number of constraints. But we have now only one dual variable for each set. Intuitively this double exponential number of constraints implicitly ensures that for given variables y_S there exist feasible variables $y_{S_{ij}}$ of former dual which sum to y_S.

We can define a function $f : 2^{2^V} \to \mathbb{R}_+$ which for every family $\mathbb{S}$ define $f(\mathbb{S})$ to be the right-hand side of corresponding constraint, i.e.:

$$f(\mathbb{S}) = \sum_{(i,j)\in V\times V, \mathbb{S}\odot(i,j)} \pi_{i,j}$$

In their paper, Hajiaghayi and Jain shows that f is submodular which allow them to prove a following fact.

Fact 2 *Suppose y is a feasbile solution to dual DLP2. Suppose the constraints corresponding to families $\mathbb{S}_1$ and $\mathbb{S}_2$ are tight. Then a constraint corresponding to the family $\mathbb{S}_1 \cup \mathbb{S}_2$ is also tight.*

Proof See *Corollary 2.2* in [3] ■

3 Algorithm

Let us call each vertex v_i which belongs to some demand $(i, j) \in D$ (respectively $\pi_{ij} > 0$ for at least one j) a sink. Now without loss of generality we can assume that each sink v_i belongs to only one demand ($\pi_{ij} > 0$ exactly for one j) and weight w_i of each sink equal to 0. To see that, construct a new graph where for each vertex v_i, v_j of each demand $(i, j) \in D$ we have additional two vertices v_i^{ij} and v_j^{ij} connected by a single edge to original verticex (v_i and v_j correspondingly). The penalties are now only between vertices v_i^{ij} and v_j^{ij}. Costs of the new vertices is now 0 while cost of original vertices v_i, v_j remains the same.
It is easy to that every solution on the new graph can be used to construct a solution of the same cost for the original graph and vice-versa.

Now we are ready to give a primal-dual algorithm for the NWPCSF problem in planar graphs. It's a natural generalization of an algorithm by Demaine et al. given in [2] for handling prize-collecting setting.
The algorithm starts with initial solution F in which there are all vertices of cost 0 (hence all sinks). In each iteration algorithm maintains moats which are connected components of graph G induced by vertices of a current solution F. Some of them are active and unactive. Demands can be marked (meaning that we decide to pay a penalty for them) or unmarked. At the beginning all demands are unmarked. Once demand is marked, it stays marked forever. A moat (denoted by corresponding set $S \subset V$) is active in current iteration if and only if there is at least one unmarked demand (i, j) such that $S \odot (i, j)$. Now in each iteration we simultaneously grow each active moat until one of the two events occur:

- a constraint corresponding to a vertex v goes tight.
- a constraint corresponding to a family $\mathbb{S}$ goes tight.

In a first case we simply add v to our solution F and continue to a next iteration (observe that some new moats in next iteration can become inactive - this is when we connect some demand).
In a second case, we mark each demand (i, j) such that $\mathbb{S} \odot (i, j)$. Hence in next iteration all moats which separates these demands (i, j) will be inactive, so we won't violate any constraint during growth process.

We repeat this until all moats become inactive.

After that we have an additional prune phase in which we process all vertices of F in reverse order we added them and remove a vertex v from F if after removing it from F, all unmarked demands are still connected in graph induced by F. We output this pruned set of vertices as F' which is our final solution.

```
input  : A graph $G = (V, E)$ with non-negative weights $w_i$ on the nodes and non-negative
         penalties $\pi_{ij}$ between each pair of vertices such that if $\pi_{ij} > 0$ then $w_i = 0$ and $w_j = 0$
output: A set of vertices $F'$ representing a forest and a set of pairs $Q'$ representing not connected
         demands
1  begin
2      $F \leftarrow \{v_i \in V : w_i = 0\}$;
3      $Q \leftarrow \emptyset$ // set all demands unmarked
4      $y_S \leftarrow 0$ // implicitly
5      $Active\_Moats \leftarrow \left\{ S \subset V : S \in SCC\,(G[F]) \wedge \underset{(i,j)\in V\times V-Q}{\exists} \pi_{ij} > 0 \wedge S \odot (i,j) \right\}$;
       // identify active moats as components of graph $G$ induced by vertices $F$ for
          which there is at least one unmarked demand $(i, j)$ which is separated by
          corresponding set
6      while $Active\_Moats$ ¡¿ $\emptyset$ do
7          find minimum $\epsilon_1$ s.t if we increase $y_S$ for each $S \in Active\_Moats$ by $\epsilon_1$ we get a new tight
           vertex $v$;
8          find minimum $\epsilon_2$ s.t if we increase $y_S$ for each $S \in Active\_Moats$ by $\epsilon_2$ we get a new tight
           family $\mathbb{S}$;
9          $\epsilon \leftarrow min(\epsilon_1, \epsilon_2)$;
10         $y_S \leftarrow y_S + \epsilon$ for all $S \in Active\_Moats$;
11         if $\epsilon = \epsilon_1$ then
12             $F \leftarrow F \cup \{v\}$;
13         else
14             $Q \leftarrow Q \cup \{(i,j) \in V \times V : \mathbb{S} \odot (i,j)\}$
15         end
16         $Active\_Moats \leftarrow \left\{ S \subset V : S \in SCC\,(G[F]) \wedge \underset{(i,j)\in V\times V-Q}{\exists} \pi_{ij} > 0 \wedge S \odot (i,j) \right\}$;
17     end
       // prune phase
18     Derive $F'$ from $F$ by removing vertices in reverse of the order in which they were added so that
       every unmarked demand is connected in $F'$.
19     Let $Q'$ be all demands not connected via $F'$
20 end
```

Algorithm 1: Primal-Dual Algorithm for NWPCSF in planar graphs

Obtaining ϵ_1 and a tight vertex in line 7 is easy.
On the other hand obtaining ϵ_2 in line 8 and a tight family $\mathbb{S}$ seems to be much harder, since the number of corresponding constraint is double exponential. Fortunately Hajiaghayi and Jain in section 4 of [3] gave a polynomial time algorithm for computing ϵ_2 and corresponding tight family $\mathbb{S}$.
Since algorithm terminates after at most $2|V| - 1$ iterations (in each iteration number of active moats or number of connected components decreases), the running time of this algorithm is polynomial.

4 Analysis

Theorem 3 *The algorithm described in section 3 outputs a set of vertices F' and a set of demands Q' which are not connected via F' such that*

$$\sum_{v\in F'} w_v + \sum_{(i,j)\in Q'} \pi_{ij} \leq 4 \sum_{S\subset V} y_S \leq 4OPT$$

In order to prove Theorem 3 it is enough to prove following two lemmas:

Lemma 4

$$\sum_{(i,j)\in Q'} \pi_{ij} \le \sum_{S\subset V} y_S$$

Proof First observe that $Q' \subset Q$ where Q are marked pairs. Now consider families $\mathbb{S}_1, \ldots, \mathbb{S}_f$ which went tight during running of the algorithm. Observe that each marked pair was separated by some $\mathbb{S}_i$.
Hence family $\mathbb{S}_{all} = \bigcup_{j=1}^{f} \mathbb{S}_j$ separates each marked pair. From fact 2 union of tight families is tight. Putting it all together gives:

$$\sum_{(i,j)\in Q'} \pi_{ij} \le \sum_{(i,j)\in Q} \pi_{ij} \le \sum_{\mathbb{S}_{all}\odot(i,j)} \pi_{ij} = \sum_{S\in\mathbb{S}} y_S \le \sum_{S\subset V} y_S$$

■

Lemma 5

$$\sum_{v\in F'} w_v \le 3 \sum_{S\subset V} y_S$$

To prove Lemma 5 we will use auxiliary lemma. But let's first introduce one definition.
For a set of nodes F and a set of unmarked demands $R = D - Q$ define a minimal feasible augmentation F_{aug} of F with respect to R to be a set of vertices F_{aug} containing F such that every pair of vertices from R is connected in subgraph of G induced by F_{aug} and such that removal of any $v \in F_{aug} - F$ from F_{aug} disconnects some pair from R.

Lemma 6 *Let G be planar, R be a set of unmarked demands after running the above algorithm, F^j be a set of bought vertices before running iteration j and F_{aug} be a minimal feasible augmentation of F^j with respect to R. Let also A^j be a set of active moats before running iteration j. Then*

$$\sum_{S\in A^j} |F_{aug} \cap \Gamma(S)| \le 3|A^j|$$

Before we prove Lemma 6 we will show how it helps us in proving Lemma 5.
Proof [Proof of Lemma 5]
Since we add a vertex v to F only if it is tight, and after then we don't modify variables corresponding to sets adjacent to v, we have following equality

$$\begin{aligned}\sum_{v\in F'} w_v &= \sum_{v\in F'} \sum_{S: v\in\Gamma(S)} y_S \\ &= \sum_{S\subset V} |F' \cap \Gamma(S)| y_S\end{aligned}$$

(in last step we changed the order of summation).

Now let F' be an output of the algorithm, $R = D - Q$ be a set of all unmarked demands, F^j be a set of bought vertices before running iteration j, A^j be a set of active moats before running iteration j and ϵ_j be the increase of the dual variables in iteration j. Then for each $S \subset V$ we have $y_S = \sum_{j:S\in A^j} \epsilon_j$ hence following holds:

$$\sum_{S\subset V} y_S = \sum_j |A^j| \epsilon_j$$

and

$$\begin{aligned}\sum_{v\in F'} w_v &= \sum_{S\subset V} |F'\cap\Gamma(S)| y_S \\ &= \sum_{S\subset V} |F'\cap\Gamma(S)| \sum_{j:S\in A^j} \epsilon_j \\ &= \sum_j \left(\sum_{S\in A^j} |F'\cap\Gamma(S)|\right)\epsilon_j\end{aligned}$$

Observe now, that $F_{aug} = F^j \cup F'$ is a minimal feasible augmentation of F^j with respect to R. Obviously, every demand from R is connected in F_{aug}. Try now picking any vertex $v \in F' - F^j$. Removing v from our F_{aug} will make some pair from R unconnected because otherwise v would be deleted in pruning phase. Hence we can use Lemma 6:

$$\sum_{S\in A^j} |(F^j\cup F')\cap\Gamma(S)| \le 3|A^j|$$

Since $|F'\cap\Gamma(S)| \le |(F^j\cup F')\cap\Gamma(S)|$ we have

$$\begin{aligned}\sum_{v\in F'} w_v &= \sum_j \left(\sum_{S\in A^j} |F'\cap\Gamma(S)|\right)\epsilon_j \\ &= \sum_j 3|A^j|\epsilon_j \\ &= 3\sum_{S\subset V} y_S\end{aligned}$$

■

The last thing to have a complete proof of Theorem 3 is to prove Lemma 6 which we can do by following Moldenhauer in [4]. We note that the only difference is that the white vertices in his proof correspond now to all active moats in considered iteration. The key property that removing a black vertex splits the graph into multiple components is still satisfied, because of fact that removing any vertex $v \in F_{aug} - F^j$ from F_{aug} will make some pair from R unconnected. Note also that for every unmarked pair R there must be two corresponding active sets in A^j (otherwise the demand would be marked). Hence removing v from F_{aug} will disconnect two previously connected white vertices.

5 Open problems

- It is possible to obtain ≈ 2.93-approximation for the same problem using threshold rounding. However, it requires solving LP. Providing better algorithm than our 4-approximation which doesn't solve LP is an interesting task.
- We weren't able to provide an example proving that our analysis is tight. Simple adaptation of an example given in [3] to node-weighted version shows only that the maximum ratio between the cost of the solution and the sum of dual variables is at least 3. The same gap can be shown by simply taking tight example from [4]. We weren't able to combine these two examples to prove that ratio 4 is met.

References

[1] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Vahid Liaghat, *Improved Approximation Algorithms for (Budgeted) Node-weighted Steiner Problems, ICALP 2013*

[2] Erik D. Demaine, MohammadTaghi Hajiaghayi, Philip N. Klein, *Node-Weighted Steiner Tree and Group Steiner Tree in Planar Graphs, ICALP 2009*

[3] MohammadTaghi Hajiaghayi, Kamal Jain, *The Prize-Collecting Generalized Steiner Tree Problem Via A New Approach Of Primal-Dual Schema, SODA 2006*

[4] Carsten Moldenhauer, *Primal-dual Approximation Algorithms for Node-Weighted Steiner Forest on Planar Graphs, ICALP 2011*

Appendices

A Implementation of an algorithm

We implemented our algorithm using C++ (see *4approx.cpp*). In following, we give some important details.

Input

The first line of the input contains three space-separated integers n, m and k - the number of vertices in graph, the number of edges in graph and the number of demands. Each of the next n lines contains one float number. The i^th line contains the weight of the vertex $i-1$ (vertices are numbered from 0 to $n-1$). After that there are m lines - each containing two integers $0 \leq a, b < n$ - a pair of vertices connected by an undirected edge. At the end there are k lines - each containing two integers and one float number a, b, c - the value c of penalty for not connecting demand (a, b).

Output

The algorithm outputs a list of nodes which are bought, the value of the sum of dual variables, the cost of bought nodes, the total penalty for not connected demands and the ratio between the cost of the solution and sum of dual variables. Above analysis guarantee that this ratio is at most 4.

Complexity

The algorithm works in iterations. There are at most $O(n)$ iterations. Cost of the iteration is dominated by finding minimum epsilon for which some family of sets becomes tight (second inequalites in 1). To find this epsilon we perform at most $O(k)$ max-flow computations on graph of $O(n+k)$ vertices and $O(n \cdot k)$ edges. Assuming that $M(a, b)$ is a complexity of max-flow computation with a vertices and b edges, we have total complexity $O(n \cdot k \cdot M(n+k, n \cdot k))$ which equals $\approx O(n^3k^2)$ using Dinic's algorithm.

Test cases

We tested our implementation against different test cases (see files with *.in* extension).

- *0.in, 1.in, 2.in* are some simple but interesting instances
- *k8.in, k20.in, k50.in* are NWSF instances of pockets tending to reach 18/7 approximation ratio (see $fig.8$ in [4]). See *18_7_family_generator.cpp* for generator of these instances.
- *pr8.in, pr20.in, pr50.in* are the same instances but with some random penalties. They were generated using *18_7_pcst_generator.cpp*
- *hiv-1.in, hiv-2.in* - instances for NWSF from a Computational Biology application, contributed by Sharon Hffner and Falk Hffner (took from 11th DIMACS Implementation Challenge: Steiner Tree Problems); in addition some random penalties were assigned. Not attached because of large size of files. Instead we attach *stptransformator.cpp* which transforms *.stp* instances into our format.

Performing in practice

Running tests showed that our algorithm solves in seconds instances with 2000 nodes. We observed also very small ratios between sums of dual variables and costs of the solutions. These facts leads us to conclusion that our algorithm could be used in practice (for example as a good initial solution in some heuristics).

B A planar counterexample for algorithm by Bateni

At the beginning of the project we tried to obtain our goal by analysing the existing $O(\log k)$-approximation algorithm for NWPCSF by Bateni [1]. We hoped it could work better on planar graphs. In the following we show that it isn't the case by providing a family of planar instances for which the algorithm gives $\Omega(\log k)$-approximation.
Brief description of $O(\log k)$-approximation algorithm for NWPCSF by Bateni et al.

The algorithm iteratively builds solution X and works with dual programs of a following form:

$$\begin{array}{lll} \text{Maximize} & \sum_{S\in\mathcal{S}} y(S) & \\ \text{subject to} & \sum_{S\in\overline{\mathcal{S}(c,\mathcal{L})}:v\in\delta(S)} y(S) \le c(v) & \forall v\in V \\ & \sum_{S\prime:core(S)\subseteq S\prime\subseteq S} y(S\prime) \le \pi_{\mathcal{L}}(S) & \forall S\in\overline{\mathcal{S}(c,\mathcal{L})} \\ & y(s)\ge 0 & \end{array}$$

where a cost function c, a set of active demands $\mathcal{L}$, collections of sets separating one core from the other cores $\overline{\mathcal{S}(c,\mathcal{L})}$ change in each iteration.

It starts with identifying so called "cores" which are simply connected components of graph induced by vertices with cost 0.
Then it "grows" each core (by simultaneously increasing corresponding dual variable y_S). It grows only cores (disks) having at least one so far not connected terminal. If during this process some constraint for vertex v gets tight (but left hand side of this constraint must be contributed by only one disk) then v is added to this disk and growth proceed.
At some point one of the event will happen (stopping growth procedure):

- Constraint for a vertex v gets tight (contributed by ≥ 2 disks).
- Constraint for a disk S gets tight.

In the first case, we (buy) add to solution X shortest paths from each terminal inside disks contributing to the constraint. We update cost function to be 0 for each bought node. Then we deactivate every demand which gets connected and start over.

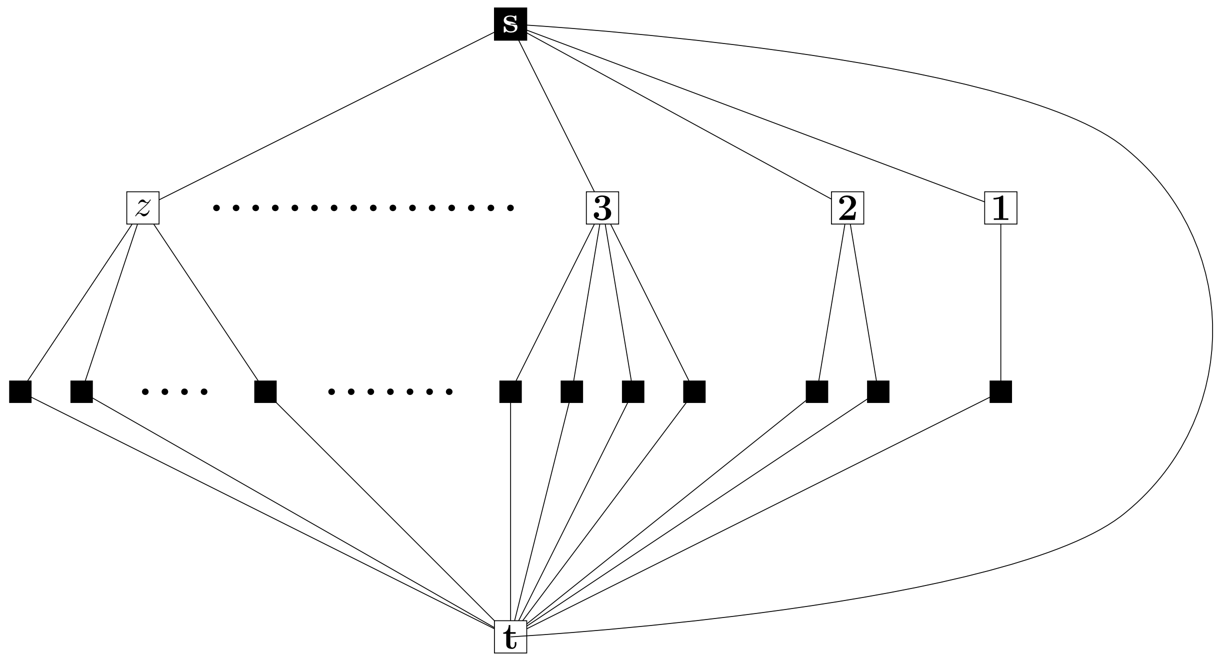
In the second case we deactivate every demand with exactly one endpoint in S, pay a penalty for not connecting them and continue to the next iteration.
The algorithms stops when there is no active demand remaining (at each iteration we deactivate at least one demand).

For the full description see original paper [1]

Example of a bad planar instance

Consider following graph consisting of $2^z + z + 1$ nodes where each white node $i \in \{1, 2 \ldots z\}$ has 2^{i-1} black nodes below.

Penalty assigned to each node is ∞ (meaning that it's large enough to be irrelevant).
Filled nodes denote terminals and there are $2^z - 1$ demands consisting of node s and each other terminal. Such a graph can be think of as a plain Node-Weighted Steiner Tree instance.
All terminals have cost 0 and non-terminals $1, 2, \ldots z$ have cost 1 and non-terminal t has cost 2.



It is clear that optimal solution for this instance has cost 2 (simply buy node t). We will now show that algorithm of Bateni et al. returns a solution with cost z.

Since demands have infinite penalties, in every iteration of the algorithm one constraint for a vertex becomes tight.
In first iteration each terminal is a single core.
Consider how much growth must be assigned to each core to get vertex z tight.
It is easy to see that this quantity is:

$$\epsilon_z = \frac{1}{2^{z-1}+1}$$

while to get vertex t tight we would need to increase dual variables of each core by:

$$\epsilon_t = \frac{2}{2^z}$$

So we have

$$\frac{\epsilon_z}{\epsilon_t} = \frac{1}{2^{z-1}+1}.\frac{2^z}{2} = \frac{2^{z-1}}{2^{z-1}+1} < 1$$

Growth needed to get other vertices tight is bigger than for vertex z.
So in first iteration algorithm buys vertex z.

We will now see that algorithm will buy all vertices $1, 2, \ldots z$.
Let's consider iteration i and assume that the algorithm have already bought vertices $z, z-1, \ldots z-i+2$.
Let's compare growth needed to be assigned to get vertex $z-i+1$ or vertex t tight.

$$\epsilon_{z-i+1} = \frac{1}{2^{z-i}+1}$$

$$\epsilon_{t_i} = \frac{2}{2^{z-i+1}}$$

which gives us:

$$\frac{\epsilon_{z-i+1}}{\epsilon_{t_i}} = \frac{1}{2^{z-i}+1} . \frac{2^{z-i+1}}{2} = \frac{2^{z-i}}{2^{z-i}+1} < 1$$

which by induction proves the claim.

So the ratio between solution returned by algorithm and optimal solution is:

$$\frac{ALG}{OPT} = \frac{z}{2} = \Theta(\log |V|)$$

which shows that above algorithm will not get us a constant approximation guarantee for planar graphs.