Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp -

Pruning 2-Connected Graphs
(Extended Abstract)

Chandra Chekuri*, **Nitish Korula**†
Dept. of Computer Science
University of Illinois
Urbana, IL 61801
{chekuri,nkorula2}@cs.uiuc.edu

ABSTRACT. Given an edge-weighted undirected graph G with a specified set of terminals, let the *density* of any subgraph be the ratio of its weight/cost to the number of terminals it contains. If G is 2-connected, does it contain smaller 2-connected subgraphs of density comparable to that of G? We answer this question in the affirmative by giving an algorithm to *prune* G and find such subgraphs of any desired size, at the cost of only a logarithmic increase in density (plus a small additive factor). We apply the pruning techniques to give algorithms for two NP-Hard problems on finding large 2-vertex-connected subgraphs of low cost; no previous approximation algorithm was known for either problem. In the k-2VC problem, we are given an undirected graph G with edge costs and an integer k; the goal is to find a minimum-cost 2-vertex-connected subgraph of G containing at least k vertices. In the Budget-2VC problem, we are given the graph G with edge costs, and a budget B; the goal is to find a 2-vertex-connected subgraph H of G with total edge cost at most B that maximizes the number of vertices in H. We describe an $O(\log n \log k)$ approximation for the k-2VC problem, and a bicriteria approximation for the Budget-2VC problem that gives an $O(\frac{1}{\epsilon}\log^2 n)$ approximation, while violating the budget by a factor of at most $3+\epsilon$.

1 Introduction

Connectivity and network design problems play an important role in combinatorial optimization and algorithms both for their theoretical appeal and their usefulness in real-world applications. Many of these problems, such as the well-known minimum cost Steiner tree problem, are NP-hard, and there has been a large and rich literature on approximation algorithms. A number of elegant and powerful techniques and results have been developed over the years. In particular, the primal-dual method [1, 17] and iterated rounding [19] have led to some remarkable results. Occasionally, interesting and useful variants of classical problems are introduced, sometimes motivated by their natural appeal and sometimes motivated by practical applications. One such problem is the k-MST problem introduced by Ravi *et al.* [24]: Given an edge-weighted graph G and an integer k, the goal is to find a minimum-cost subgraph of G that contains at least k vertices. It is not hard to see that the k-MST problem is at least as hard as the Steiner tree problem; moreover an α-approximation for the k-MST problem implies an α-approximation for the Steiner tree problem. The k-MST problem has attracted considerable attention in the approximation algorithms literature and its study has led to several new algorithmic ideas and applications [3, 15, 14, 7, 5]. Closely related to the k-MST problem is the budgeted or Max-Prize Tree problem [21, 5]; here we

* Partially supported by NSF grants CCF 0728782 and CNS-0721899, and a US-Israeli BSF grant 2002276.
†Partially supported by NSF grant CCF 0728782.

are given G and a budget B, and the goal is to find a subgraph H of G of total cost no more than B, that maximizes the number of vertices (or terminals) in H. Interestingly, it is only recently that the rooted version of the Max-Prize Tree problem was shown to have an $O(1)$-approximation [5], although an $O(1)$-approximation was known for the k-MST problem much earlier [6].

Recently, Lau *et al.* [22] considered the natural generalization of k-MST to higher connectivity. In particular they defined the (k, λ)-subgraph problem to be the following: Find a minimum-cost subgraph of the given graph G that contains at least k vertices and is λ-edge connected. We use the notation k-λEC to refer to this problem. In [22, 23] a poly-logarithmic approximation was derived for the k-2EC problem. In this paper, we consider the vertex-connectivity generalizations of the k-MST and Budgeted Tree problems. We define the k-λVC problem as follows: Given an integer k and a graph G with edge costs, find the minimum-cost λ-vertex-connected subgraph of G that contains at least k vertices. Similarly, in the Budget-λVC problem, given a budget B and a graph G with edge costs, the goal is to find a λ-vertex-connected subgraph of G of cost at most B, that maximizes the number of vertices it contains. In particular we focus on $k = 2$ and develop approximation algorithms for both the k-2VC and Budget-2VC problems. We note that the k-λEC problem reduces to the k-λVC problem in an approximation preserving fashion, though the opposite reduction is not known. The k-λEC and k-λVC problems are NP-hard and also APX-hard for any $k \geq 1$. Moreover, Lau *et al.* [22] give evidence that, for large λ, the k-λEC problem is likely to be harder to approximate by relating it to the approximability of the dense k-subgraph problem [12].

Problems such as k-MST, Budget-2VC, k-2VC are partly motivated by applications in network design and related areas where one may want to build low-cost networks including (or servicing) many clients, but there are constraints such as a budget on the network cost, or a minimum quota on the number of clients. Algorithms for these problems also find other uses. For instance, a basic problem in vehicle routing applications is the s-t Orienteering problem in which one seeks an s-t path that maximizes the number of vertices in it subject to a budget B on its length. Approximation algorithms for this problem [5, 4, 11] have been derived through approximation algorithms for the k-MST and the related k-stroll problems; in the latter, the goal is to find a minimum-cost path containing k vertices.

How do we solve these problems? The k-MST problem required several algorithmic innovations which eventally led to the current best approximation ratio of 2 [14]. The main technical tool which underlies $O(1)$ approximations for the k-MST problem [6, 15, 14] is a special property that holds for a LP relaxation of the prize-collecting Steiner tree problem [17] which is a Lagrangian relaxation of the Steiner tree problem. Unfortunately, one cannot use these ideas (at least directly) for more general problems such as k-2VC (or the k-Steiner forest problem [18]) since the LP relaxation for the prize-collecting variant is not known to satisfy the above mentioned property. We therefore rely on alternative techniques that take a more basic approach.

Our algorithms for k-2VC and Budget-2VC use the same high-level idea, relying on the notion of *density*: the density of a subgraph is the ratio of its cost to the number of vertices it contains. The algorithms greedily combine subgraphs of low density until the union of these subgraphs has the desired number of vertices or has cost equal to the budget. They

fail only if we find a subgraph H of good density, but that is far too large. One needs, then, a way to *prune* H to find a smaller subgraph of comparable density. Our main structural result for pruning 2-connected graphs is the following:

THEOREM 1. *Let G be a 2-connected edge-weighted graph with density ρ, and a designated vertex $r \in V(G)$ such that every vertex of G has 2 vertex-disjoint paths to r of total weight/cost at most L. There is a polynomial-time algorithm that, given any integer $k \leq |V(G)|$, finds a 2-connected k-vertex subgraph H of G containing r, of total cost at most $O(\log k)\rho k + 2L$.*

Intuitively, the algorithm of Theorem 1 allows us to find a subgraph of *any* desired size, at the cost of only a logarithmic increase in density. Further, it allows us to require any vertex r to be in the subgraph, and also applies if we are given a *terminal* set S, and the output subgraph must contain k terminals. (In this case, the density of a subgraph is the ratio of its cost to the number of terminals it contains.) In addition, it applies if the terminals/vertices have arbitrary weights, and the density of a subgraph is the ratio of its cost to the sum of the weights of its terminals. All our algorithms apply to these weighted instances, but for simplicity of exposition, we discuss the more restricted unweighted versions throughout. We observe that pruning a tree (a 1-connected graph) is easy and one loses only a constant factor in the density; the theorem above allows one to prune 2-connected graphs. A technical ingredient that we develop is the following theorem: we believe that Theorems 1 and 2 are interesting in their own right and will find other applications besides algorithms for k-2VC and Budget-2VC.

THEOREM 2. *Let G be a 2-vertex-connected graph with edge costs and let $S \subseteq V$ be a set of terminals. Then, there is a simple cycle C containing at least 2 terminals (a non-trivial cycle) such that the density of C is at most the density of G. Moreover, such a cycle can be found in polynomial time.*

Using the above theorem and an LP approach we obtain the following.

COROLLARY 3. *Given a graph $G(V, E)$ with edge costs and ℓ terminals $S \subseteq V$, there is an $O(\log \ell)$ approximation for the problem of finding a minimum-density non-trivial cycle.*

Note that Theorem 2 and Corollary 3 are of interest because we seek a cycle with at least *two* terminals. A minimum-density cycle containing only one terminal can be found by using the well-known min-mean cycle algorithm in directed graphs [2]. We remark, however, that although we suspect that the problem of finding a minimum-density non-trivial cycle is NP-hard, we currently do not have a proof. Theorem 2 shows that the problem is equivalent to the dens-2VC problem, defined in the next section.

Armed with these useful structural results, we give approximation algorithms for both the k-2VC and Budget-2VC problems. Our results in fact hold for the more general versions of these problems where the input also specifies a subset $S \subseteq V$ of *terminals* and the goal is to find subgraphs with the desired number of terminals, or to maximize the number of terminals.‡

‡For k-2EC and k-λEC , the problem with specified terminal set S can be reduced to the problem where every vertex in V is a terminal. Such a reduction does not seem possible for the k-2VC and k-λVC , so we work directly with the terminal version.

THEOREM 4. *There is an $O(\log \ell \cdot \log k)$ approximation for the k-2VC problem, where ℓ is the number of terminals.*

COROLLARY 5. *There is an $O(\log \ell \cdot \log k)$ approximation for the k-2EC problem, where ℓ is the number of terminals.*

THEOREM 6. *There is a polynomial time bicriteria approximation algorithm for Budget-2VC that, for any $0 < \epsilon \leq 1$, outputs a subgraph of edge-weight $(3+\epsilon)B$ containing $\Omega(\epsilon \cdot \mathrm{OPT}/(\log n \log \mathrm{OPT}))$ terminals, where OPT is the number of terminals in an optimum solution of cost B. For the rooted version, the subgraph has weight at most $(2+\epsilon)B$.*

Most of the proofs are omitted from this version due to space limitations. The reader can find a full version on the websites of the authors.

1.1 Overview of Technical Ideas

We focus on the rooted version of k-2VC : the goal is to find a min-cost subgraph that 2-connects at least k terminals to a specified root vertex r. It is easy to reduce k-2VC to its rooted version. We draw inspiration from algorithmic ideas that led to poly-logarithmic approximations for the k-MST problem.

For a subgraph H that contains r, let $k(H)$ be the number of terminals that are 2-connected to r in H. Then the *density* of H is simply the ratio of the cost of H to $k(H)$. The dens-2VC problem is to find a 2-connected subgraph of minimum density. An $O(\log \ell)$ approximation for the dens-2VC problem (where ℓ is the number of terminals) can be derived in a some what standard way by using a bucketing and scaling trick on a linear programming relaxation for the problem. We exploit the known bound of 2 on the integrality gap of a natural LP for the SNDP problem with vertex connectivity requirements in $\{0,1,2\}$ [13].

Our algorithm for k-2VC uses a greedy approach at the high level. We start with an empty subgraph G' and use the approximation algorithm for dens-2VC in an iterative fashion to greedily add terminals to G' until at least $k' \geq k$ terminals are in G'. This approach would yield an $O(\log \ell \log k)$ approximation if $k' = O(k)$. However, the last iteration of the dens-2VC algorithm may add many more terminals than desired with the result that $k' \gg k$. In this case we cannot bound the cost of the solution obtained by the algorithm. To overcome this problem, one can try to *prune* the subgraph H added in the last iteration to only have the desired number of terminals. For the k-MST problem, H is a tree and pruning is quite easy.

Our main technical contribution is Theorem 1, to give a pruning step for the k-2VC problem. To accomplish this, we use two algorithmic ideas. The first is encapsulated in the cycle finding algorithm of Theorem 2. Second, we use this cycle finding algorithm to repeatedly merge subgraphs until we get the desired number of terminals in one subgraph; this latter step requires care. The cycle merging scheme is inspired by a similar approach from the work of Lau *et al.* [22] on the k-2EC problem and in our previous work [11] on the directed orienteering problem. These ideas yield an $O(\log \ell \cdot \log^2 k)$ approximation. We give a modified cycle-merging algorithm with a more sophisticated and non-trivial analysis to obtain an improved $O(\log \ell \cdot \log k)$ approximation.

Some remarks are in order to compare our work to that of [22] on the k-2EC problem. The combinatorial algorithm in [22] is based on finding a low-density cycle or a related structure called a bi-cycle. The algorithm in [22] to find such a structure is incorrect. Further, the cycles are contracted along the way which limits the approach to the k-2EC problem (contracting a cycle in 2-node-connected graph may make the resulting graph not 2-node-connected). In our algorithm we do not contract cycles and instead introduce dummy terminals with weights to capture the number of terminals in an already formed component. This requires us to now address the minimum-density non-trivial simple cycle problem which we do via Theorem 2 and Corollary 3. In independent work, Lau *et al.* [23] obtain a new and correct $O(\log n \log k)$-approximation for k-2EC . They also follow the same approach that we do in using the LP for finding dense subgraphs followed by the pruning step. However, in the pruning step they use a very different approach; they use the sophisticated idea of nowhere-zero 6-flows [25]. Although the use of this idea is elegant, the approach works only for the k-2EC problem, while our approach is less complex and leads to an algorithm for the more general k-2VC problem.

2 The Algorithms for the k-2VC and Budget-2VC Problems

We work with graphs in which some vertices are designated as *terminals*. Henceforth, we use 2-connected graph to mean a 2-vertex-connected graph. Recall that the goal of the k-2VC problem is to find a minimum-cost 2-connected subgraph on at least k terminals. In the rooted k-2VC problem, we wish to find a min-cost subgraph on at least k terminals in which every terminal is 2-connected to the specified root r. The (unrooted) k-2VC problem can be reduced to the rooted version by *guessing* 2 vertices u, v that are in an optimal solution, creating a new root vertex r, and connecting it with 0-cost edges to u and v. It is not hard to show that any solution to the rooted problem in the modified graph can be converted to a solution to the unrooted problem by adding 2 minimum-cost vertex-disjoint paths between u and v. (Since u and v are in the optimal solution, the cost of these added paths cannot be more than OPT.) Similarly, one can reduce Budget-2VC to its rooted version. However, note that adding a min-cost set of paths between the guessed vertices u and v might require us to pay an additional amount of B, so to obtain a solution for the unrooted problem of cost $(3+\epsilon)B$, we must find a solution for the rooted instance of cost $(2+\epsilon)B$.

Note that k-2VC and Budget-2VC are equivalent from the viewpoint of exact optimization, but this is not true from an approximation perspective. Still, we solve them both via the dens-2VC problem, where the goal is to find a subgraph H of minimum density in which all terminals of H are 2-connected to the root. We use the following lemma, which relies on a 2-approximation, via a natural LP for the min-cost 2-connectivity problem, due to Fleischer, Jain and Williamson [13], and some standard techniques.

LEMMA 7. *There is an $O(\log \ell)$-approximation algorithm for the dens-2VC problem, where ℓ is the number of terminals in the given instance.*

We first describe our algorithm for the k-2VC problem. Let OPT be the cost of an optimal solution to the k-2VC instance. We assume knowledge of OPT; this can be dispensed with using standard methods. We pre-process the graph by deleting any terminal that does

not have 2 vertex-disjoint paths to the root r of total cost at most OPT. The high-level description of the algorithm for the rooted k-2VC problem is given below.

$k' \leftarrow k$, G' is the empty graph.
While ($k' > 0$):
  Use the approximation algorithm for dens-2VC to find a subgraph H in G.
  If ($k(H) \leq k'$):
    $G' \leftarrow G' \cup H$, $k' \leftarrow k' - k(H)$.
    Mark all terminals in H as non-terminals.
  Else:
    Prune H to obtain H' that contains k' terminals.
    $G' = G' \cup H'$, $k' \leftarrow 0$.
Output G'.

At the beginning of any iteration of the while loop, the graph contains a solution to the dens-2VC problem of density at most $\frac{\text{OPT}}{k'}$. Therefore, the graph H returned always has density at most $O(\log \ell)\frac{\text{OPT}}{k'}$. If $k(H) \leq k'$, we add H to G' and decrement k'; we refer to this as the *augmentation* step. Otherwise, we have a graph H of good density, but with too many terminals. In this case, we prune H to find a graph with the required number of terminals; this is the *pruning step*. A simple set-cover type argument shows the following lemma:

LEMMA 8. *If, at every augmentation step, we add a graph of density at most $O(\log \ell)\frac{\text{OPT}}{k'}$ (where k' is the number of additional terminals that must be selected), the total cost of all the augmentation steps is at most $O(\log \ell \cdot \log k)\text{OPT}$.*

Therefore, it remains only to bound the cost of the graph H' added in the pruning step, and Theorem 1, proved in Section 4, is precisely what is needed. Our main result for the k-2VC problem, Theorem 4, follows easily from Lemma 8 and Theorem 1.

We now describe the similar algorithm for the Budget-2VC problem. Given budget B, preprocess the graph as before by deleting vertices that do not have 2 vertex-disjoint paths to r of total cost at most B. Let OPT denote the number of vertices in the optimal solution, and $k = \text{OPT}/c \log \ell \log \text{OPT}$, for some constant $c = O(1/\epsilon)$. We run the same greedy algorithm, using the $O(\log \ell)$-approximation for the dens-2VC problem. Note that at each stage, the graph contains a solution to dens-2VC of density at most $B/(\text{OPT} - k) < 2B/\text{OPT}$. Therefore, we have the following lemma:

LEMMA 9. *If, at every augmentation step of the algorithm for Budget-2VC, we add a graph of density at most $O(\log \ell)(2B/\text{OPT})$, the total cost of all augmentation steps is at most $O(B/\log \text{OPT}) \leq \epsilon B$.*

Again, to prove Theorem 6, giving a bicriteria approximation for Budget-2VC, we only have to bound the cost of the pruning step.

PROOF OF THEOREM 6. From the previous lemma, the total cost of the augmentation steps is at most ϵB. The graph H returned by the dens-2VC algorithm has density at most $O(\log \ell \cdot B/\text{OPT})$, and $k(H) > k'$ terminals. Now, from Theorem 1, we can prune H to find a graph H' containing k' terminals and cost at most $O(\log k' \log \ell \cdot B/\text{OPT}) \cdot k' + 2B$. As $k' \leq k = \text{OPT}/(c \log \ell \log \text{OPT})$, a suitable choice of c ensures that the total cost of the pruning step is at most $\epsilon B + 2B$.

It remains only to prove Lemma 7, that there is an $O(\log \ell)$-approximation for the dens-2VC problem, and the crucial Theorem 1, bounding the cost of the pruning step. We omit the proof of Lemma 7 from this extended abstract. Before the latter is proved in Section 4, we develop some tools in Section 3; chief among these tools is Theorem 2.

3 Finding Low-density Non-trivial Cycles

A cycle $C \subseteq G$ is *non-trivial* if it contains at least 2 terminals. We define the min-density non-trivial cycle problem: Given a graph $G(V, E)$, with $S \subseteq V$ marked as terminals, edge costs and terminal weights, find a minimum-density cycle that contains at least 2 terminals. Note that if we remove the requirement that the cycle be non-trivial (that is, it contains at least 2 terminals), the problem reduces to the min-mean cycle problem in directed graphs, and can be solved exactly in polynomial time (see [2]). Algorithms for the min-density non-trivial cycle problem are a useful tool for solving the k-2VC and k-2EC problems. In this section, we give an $O(\log \ell)$-approximation algorithm for the minimum-density non-trivial cycle problem.

THEOREM 10. *Let G be a 2-connected graph with at least 2 terminals. G contains a simple non-trivial cycle X such that* $\text{density}(X) \leq \text{density}(G)$.

PROOF SKETCH. Let C be an arbitrary non-trivial simple cycle; such a cycle always exists since G is 2-connected and has at least 2 terminals. If $\mathit{density}(C) > \mathit{density}(G)$, we give an algorithm that finds a new non-trivial cycle C' such that $\mathit{density}(C') < \mathit{density}(C)$. Repeating this process gives us the desired cycle. Let G' be the graph formed by contracting the given cycle C to a single vertex v. In G', v is not a terminal, and so has weight 0. Consider the 2-connected components of G' (each such component is formed by adding v to a connected component of $G' - v$), and pick the one of minimum density. If H is this component, $\mathit{density}(H) < \mathit{density}(G)$ by an averaging argument.

H contains at least 1 terminal. If it contains 2 or more terminals, recursively find a non-trivial cycle C' in H such that $\mathit{density}(C') \leq \mathit{density}(H) < \mathit{density}(C)$. If C' exists in the given graph G, we are done. Otherwise, C' contains v, and the edges of C' form an ear of C in the original graph G. The density of this ear is less than the density of C, and we can find a non-trivial cycle in the union of C and the ear of density at most that of G.

Finally, if H has exactly 1 terminal u, find any 2 vertex-disjoint paths using edges of H from u to distinct vertices in the cycle C. (Since G is 2-connected, there always exist such paths.) The cost of these paths is at most $\mathit{cost}(H)$, and concatenating these 2 paths corresponds to an ear of C in G. The density of this ear is less than $\mathit{density}(C)$; again, the union of the ear and C has a desired non-trivial cycle. ∎

We remark that the algorithm of Theorem 10 does not lead to a polynomial-time algorithm, even if all edge costs and terminal weights are polynomially bounded. We give a strongly polynomial time algorithm to find such a cycle in the full version of this paper. Note that neither of these algorithms may directly give a good approximation to the min-density non-trivial cycle problem, because the optimal non-trivial cycle may have density much less than that of G. However, we can use Theorem 10 to prove the following theorem:

THEOREM 11. *There is an α-approximation to the (unrooted) dens-2VC problem if and only if there is an α-approximation to the problem of finding a minimum-density non-trivial cycle.*

Theorem 11 and Lemma 7 imply an $O(\log \ell)$-approximation for the minimum-density non-trivial cycle problem; this proves Corollary 3.

4 Pruning 2-connected Graphs of Good Density

In this section, we prove Theorem 1. We are given a graph G and $S \subseteq V$, a set of at least k terminals. Further, every terminal in G has 2 vertex-disjoint paths to the root r of total cost at most L. Let ℓ be the number of terminals in G, and $cost(G)$ its total cost; $\rho = \frac{cost(G)}{\ell}$ is the density of G. We describe an algorithm that finds a subgraph H of G that contains at least k terminals, each of which is 2-connected to the root, and of total edge cost $O(\log k)\rho k + 2L$.

We can assume $\ell > (8 \log k) \cdot k$, or the trivial solution of taking the entire graph G suffices. The main phase of our algorithm proceeds by maintaining a set of 2-connected subgraphs that we call *clusters*, and repeatedly finding low-density cycles that merge clusters of similar weight to form larger clusters. (The weight of a cluster X, denoted by w_X, is (roughly) the number of terminals it contains.) Clusters are grouped into *tiers* by weight; tier i contains clusters with weight at least 2^i and less than 2^{i+1}. Initially, each terminal is a separate cluster in tier 0. We say a cluster is *large* if it has weight at least k, and *small* otherwise. The algorithm stops when most terminals are in large clusters.

We now describe the algorithm MERGECLUSTERS (see next page). To simplify notation, let α be the quantity $2\lceil \log k \rceil \rho$. We say that a cluster is *good* if it has density at most α; that is, good cycles have density at most $O(\log k)$ times the density of the input graph.

```
MERGECLUSTERS:
For (each i in {0, 1, ..., (⌈log_2 k⌉ − 1)}) do:
  If (i = 0):
      Every terminal has weight 1
  Else:
      Mark all vertices as non-terminals
      For (each small 2-connected cluster X in tier i) do:
          Add a (dummy) terminal v_X to G of weight w_X
          Add (dummy) edges of cost 0 from v_X to two (arbitrary) distinct vertices of X
  While (G has a non-trivial cycle C of density at most α = 2⌈log k⌉ρ):
      Let X_1, X_2, ... X_q be the small clusters that contain a terminal or an edge of C.
      (Note that the terminals in C belong to a subset of {X_1, ... X_q}.)
      Form a new cluster Y (of a higher tier) by merging the clusters X_1, ... X_q
      w_Y ← Σ_{j=1}^{q} w_{X_j}
      If (i = 0):
          Mark all terminals in Y as non-terminals
      Else:
          Delete all (dummy) terminals in Y and the associated (dummy) edges.
```

We briefly remark on some salient features of this algorithm and our analysis before presenting the details of the proofs.

1. In iteration i, terminals correspond to tier i clusters. Clusters are 2-connected subgraphs of G, and by using cycles to merge clusters, we preserve 2-connectivity as the

clusters become larger.

2. When a cycle C is used to merge clusters, all small clusters that contain an edge of C (regardless of their tier) are merged to form the new cluster. Therefore, at any stage of the algorithm, all currently small clusters are edge-disjoint. Large clusters, on the other hand, are *frozen*; even if they intersect a good cycle C, they are not merged with other clusters on C. Thus, at any time, an edge may be in multiple large clusters and up to one small cluster.
3. In iteration i of MERGECLUSTERS, the density of a cycle C is only determined by its cost and the weight of terminals in C corresponding to tier i clusters. Though small clusters of other (lower or higher) tiers might be merged using C, we do *not* use their weight to pay for the edges of C.
4. The ith iteration terminates when no good cycles can be found using the remaining tier i clusters. At this point, there may be some terminals remaining that correspond to clusters which are not merged to form clusters of higher tiers. However, our choice of α (which defines the density of good cycles) is such that we can bound the number of terminals that are "left behind" in this fashion. Therefore, when the algorithm terminates, most terminals are in large clusters.

We prove that after MERGECLUSTERS terminates most terminals are in large clusters and that each large cluster has good density. The proof proceeds via several properties that we establish formally.

Remarks: Throughout the algorithm, the graph G is always 2-connected. The weight of a cluster is at most the number of terminals it contains.

LEMMA 12. *The clusters formed by* MERGECLUSTERS *are all* 2*-connected.*

LEMMA 13. *The total weight of small clusters in tier i that are not merged to form clusters of higher tiers is at most $\frac{\ell}{2\lceil \log k \rceil}$.*

COROLLARY 14. *When the algorithm* MERGECLUSTERS *terminates, the total weight of large clusters is at least $\ell/2 > (4 \log k) \cdot k$.*

So far, we have shown that most terminals reach large clusters, all of which are 2-connected, but we have not argued about the density of these clusters. The next lemma says that if we can find a large cluster of good density, we can find a solution to the k-2VC problem of good density.

LEMMA 15. *Let Y be a large cluster formed by* MERGECLUSTERS*. If Y has density at most δ, we can find a graph Y' with at least k terminals, each of which is* 2*-connected to r, of total cost at most $2\delta k + 2L$.*

The following lemma allows us to show that every large cluster has density at most $O(\log^2 k)\rho$.

LEMMA 16. *For any cluster Y formed by* MERGECLUSTERS *during iteration i, the total cost of edges in Y is at most $(i+1) \cdot \alpha w_Y$.*

Let Y be an arbitrary large cluster; since we have only $\lceil \log k \rceil$ tiers, the previous lemma implies that the cost of Y is at most $\lceil \log k \rceil \cdot \alpha w_Y = O(\log^2 k)\rho w_Y$. That is, the density of Y

is at most $O(\log^2 k)\rho$, and we can use this fact together with Lemma 15 to find a solution to the rooted k-2VC problem of cost at most $O(\log^2 k)\rho k + 2L$. This completes the 'weaker' analysis, but this does not suffice to prove Theorem 1; to prove the theorem, we would need to use a large cluster Y of density $O(\log k)\rho$, instead of $O(\log^2 k)\rho$.

For the purpose of the more careful analysis, implicitly construct a forest $\mathcal{F}$ on the clusters formed by MERGECLUSTERS. Initially, the vertex set of $\mathcal{F}$ is just S, the set of terminals, and $\mathcal{F}$ has no edges. Every time a cluster Y is formed by merging $X_1, X_2, \ldots X_q$, we add a corresponding vertex Y to the forest $\mathcal{F}$, and add edges from Y to each of $X_1, \ldots X_q$; Y is the parent of $X_1, \ldots X_q$. We also associate a cost with each vertex in $\mathcal{F}$; the cost of the vertex Y is the cost of the cycle used to form Y from $X_1, \ldots X_q$. We thus build up trees as the algorithm proceeds; the root of any tree corresponds to a cluster that has not yet become part of a bigger cluster. The leaves of the trees correspond to vertices of G; they all have cost 0. Also, a large cluster Y formed by the algorithm is at the root of its tree; we refer to this tree as T_Y.

For each large cluster Y after MERGECLUSTERS terminates, say that Y is of type i if Y was formed during iteration i of MergeClusters. We now define the *final-stage* clusters of Y: They are the clusters formed during iteration i that became part of Y. (We include Y itself in the list of final-stage clusters; even though Y was formed in iteration i of MERGECLUSTERS, it may contain other final-stage clusters. For instance, during iteration i, we may merge several tier i clusters to form a cluster X of tier $j > i$. Then, if we find a good-density cycle C that contains an edge of X, X will merge with the other clusters of C.) The *penultimate* clusters of Y are those clusters that exist just before the beginning of iteration i and become a part of Y. Equivalently, the penultimate clusters are those formed before iteration i that are the immediate children in T_Y of final-stage clusters. Figure 1 illustrates the definitions of final-stage and penultimate clusters.

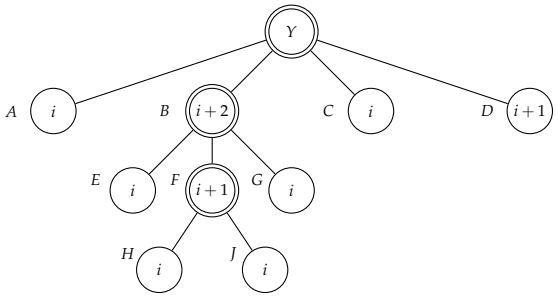

Figure 1: A part of the Tree T_Y corresponding to Y, a large cluster of type i. The number in each vertex indicates the tier of the cluster. Only final-stage and penultimate clusters are shown: final-stage clusters are shown with a double circle; the rest are penultimate.

An edge of a large cluster Y is said to be a *final edge* if it is used in a cycle C that produces a final-stage cluster of Y. All other edges of Y are called *penultimate edges*; note that any penultimate edge is in some penultimate cluster of Y. We define the *final cost* of Y to be the sum of the costs of its final edges, and its *penultimate cost* to be the sum of the costs of its penultimate edges; clearly, the cost of Y is the sum of its final and penultimate costs. We bound the final costs and penultimate costs separately.

Recall that an edge is a final edge of a large cluster Y if it is used by MERGECLUSTERS to

form a cycle C in the final iteration during which Y is formed. The reason we can bound the cost of final edges is that the cost of any such cycle is at most α times the weight of clusters contained in the cycle, and a cluster does not contribute to the weight of more than one cycle in an iteration. (This is also the essence of Lemma 16.) We formalize this intuition below.

LEMMA 17. *The final cost of a large cluster Y is at most αw_Y, where w_Y is the weight of Y.*

LEMMA 18. *If Y_1 and Y_2 are distinct large clusters of the same type, no edge is a penultimate edge of both Y_1 and Y_2.*

THEOREM 19. *After* MERGECLUSTERS *terminates, at least one large cluster has density at most $O(\log k)\rho$.*

PROOF. We define the *penultimate density* of a large cluster to be the ratio of its penultimate cost to its weight. Consider the total penultimate costs of all large clusters: For any i, each edge $e \in E(G)$ can be a penultimate edge of at most 1 large cluster of type i. This implies that each edge can be a penultimate edge of at most $\lceil \log k \rceil$ clusters. Therefore, the sum of penultimate costs of all large clusters is at most $\lceil \log k \rceil cost(G)$. Further, the total weight of all large clusters is at least $\ell/2$. Therefore, the (weighted) average penultimate density of large clusters is at most $2\lceil \log k \rceil \frac{cost(G)}{\ell} = 2\lceil \log k \rceil \rho$, and hence there exists a large cluster Y of penultimate density at most $2\lceil \log k \rceil \rho$. The penultimate cost of Y is, therefore, at most $2\lceil \log k \rceil \rho w_Y$, and from Lemma 17, the final cost of Y is at most αw_Y. Therefore, the density of Y is at most $\alpha + 2\lceil \log k \rceil \rho = O(\log k)\rho$. ∎

Theorem 19 and Lemma 15 together imply that we can find a solution to the rooted k-2VC problem of cost at most $O(\log k)\rho k + 2L$. This completes our proof of Theorem 1.

Acknowledgments: We thank Mohammad Salavatipour for helpful discussions on k-2EC and related problems. We thank Erin Wolf Chambers for useful suggestions on notation.

References

[1] A. Agrawal, P. N. Klein, and R. Ravi. When trees collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks. *SIAM J. on Computing*, 24(3):440–456, 1995.

[2] R. Ahuja, T. Magnanti, and J. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Upper Saddle River, NJ, 1993

[3] B. Awerbuch, Y. Azar, A. Blum and S. Vempala. New Approximation Guarantees for Minimum Weight k-Trees and Prize-Collecting Salesmen. *SIAM J. on Computing*, 28(1):254–262, 1999. Preliminary version in *Proc. of ACM STOC*, 1995.

[4] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows. *Proc. of ACM STOC*, 166–174, 2004.

[5] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation Algorithms for Orienteering and Discounted-Reward TSP. *SIAM J. on Computing*, 37(2):653–670, 2007.

[6] A. Blum, R. Ravi and S. Vempala. A Constant-factor Approximation Algorithm for the k-MST Problem. *J. of Computer and System Sciences*, 58:101–108, 1999.

[7] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. *Proc. of IEEE FOCS*, 36–45, 2003.

[8] C. Chekuri, G. Even, A. Gupta, and D. Segev. Set Connectivity Problems in Undirected Graphs and the Directed Steiner Network Problem. *Proc. of ACM-SIAM SODA*, 532–541, 2008.

[9] C. Chekuri, M. T. Hajiaghayi, G. Kortsarz, and M. R. Salavatipour. Approximation algorithms for Non-uniform Buy-at-bulk Network Design. *Proc. of IEEE FOCS*, 677–686, 2006.

[10] C. Chekuri, M. T. Hajiaghayi, G. Kortsarz, and M. R. Salavatipour. Approximation Algorithms for Node-weighted Buy-at-bulk Network Design. *Proc. of ACM-SIAM SODA*, 1265–1274, 2007.

[11] C. Chekuri, N. Korula, and M. Pál. Improved Algorithms for Orienteering and Related Problems. *Proc. of ACM-SIAM SODA*, 661–670, 2008.

[12] U. Feige, G. Kortsarz and D. Peleg. The Dense k-Subgraph Problem. *Algorithmica*, 29(3):410–421, 2001. Preliminary version in *Proc. of IEEE FOCS*, 1993.

[13] L. Fleischer, K. Jain, D. P. Williamson. Iterative Rounding 2-approximation Algorithms for Minimum-cost Vertex Connectivity Problems. *J. of Computer and System Sciences*, 72(5):838–867, 2006.

[14] N. Garg. Saving an ϵ: A 2-approximation for the k-MST Problem in Graphs. *Proc. of ACM STOC*, 396–402, 2005.

[15] N. Garg. A 3-approximation for the Minimum Tree Spanning k Vertices. *Proc. of IEEE FOCS*, 302–309, 1996.

[16] M. X. Goemans and D. P. Williamson. A General Approximation Technique for Constrained Forest Problems. *SIAM J. on Computing*, 24(2):296–317, 1995.

[17] M. X. Goemans and D. P. Williamson. The Primal-Dual method for Approximation Algorithms and its Application to Network Design Problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1996.

[18] M. T. Hajiaghayi and K. Jain. The Prize-Collecting Generalized Steiner Tree Problem via a New Approach of Primal-Dual Schema. *Proc of ACM-SIAM SODA*, 631–640, 2006.

[19] K. Jain. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem *Combinatorica*, 21(1):39–60, 2001.

[20] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *J. of Computer and System Sciences*, 9(3):256–278, 1974.

[21] D. S. Johnson, M. Minkoff, and S. Phillips. The Prize Collecting Steiner Tree Problem: Theory and Practice. *Proc. of ACM-SIAM SODA*, 760–769, 2000.

[22] L.C. Lau, J. Naor, M. Salavatipour and M. Singh. Survivable Network Design with Degree or Order Constraints. *Proc. of ACM STOC*, 2007.

[23] L.C. Lau, J. Naor, M. Salavatipour and M. Singh. Survivable Network Design with Degree or Order Constraints. To Appear in *SIAM J. on Computing*.

[24] R. Ravi, R. Sundaram, M. Marathe, D. Rosenkrantz, and S. Ravi. Spanning trees short and small. *SIAM J. Disc. Math.* 9 (2): 178–200, 1996.

[25] P. D. Seymour. Nowhere-zero 6-flows. *J. Comb. Theory B*, 30: 130–135, 1981.