Pruning 2-Connected Graphs*

Chandra Chekuri† Nitish Korula‡

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 (see [20, 28]). In particular, the primal-dual method [1, 18] and iterated rounding [21] 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.* [26]: 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, 16, 15, 7, 5]. Closely related to the k-MST problem is the budgeted or Max-Prize Tree problem [23, 5]; here we 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

*To appear in the 28th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2008.

†Dept. of Computer Science, University of Illinois, Urbana, IL 61801. Partially supported by NSF grants CCF 07-28782 and CNS-0721899, and a US-Israeli BSF grant 2002276. chekuri@cs.uiuc.edu

‡Dept. of Computer Science, University of Illinois, Urbana, IL 61801. Partially supported by NSF grant CCF 07-28782. nkorula2@uiuc.edu

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.* [24] 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 [24, 25] 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.* [24] 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 [13].

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 [15]. The main technical tool which underlies $O(1)$ approximations for k-MST problem [6, 16, 12, 15] is a special property that holds for a LP relaxation of the prize-collecting Steiner tree problem [18] 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 [19]) 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 and rely 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 1.1. *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 1.1 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 1.1 and 1.2 are interesting in their own right and will find other applications besides algorithms for k-2VC and Budget-2VC.

Theorem 1.2. *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 1.3. *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 1.2 and Corollary 1.3 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 1.2 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.[1]

Theorem 1.4. *There is an $O(\log \ell \cdot \log k)$ approximation for the k-2VC problem, where ℓ is the number of terminals.*

Corollary 1.5. *There is an $O(\log \ell \cdot \log k)$ approximation for the k-2EC problem, where ℓ is the number of terminals.*

Theorem 1.6. *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.*[2]

As mentioned before, the k-2EC problem was introduced by Lau *et al.* and an $O(\log^3 k)$ approximation was claimed for this problem in [24]. However, the algorithm and proof in [24] are incorrect. More recently, and in independent work from ours, the authors obtained a different algorithm for k-2EC that yields an $O(\log n \log k)$ approximation [25]; however, their algorithm does not generalize to k-2VC . We give a more detailed comparison of the differences between their approach and ours in the next subsection.

1.1 Overview of Technical Ideas

For this section, 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 relatively straightforward to reduce k-2VC

[1]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.

[2]For the *rooted* version of Budget-2VC (see Section 2), we obtain a subgraph of weight $(2 + \epsilon)B$ with this number of terminals.

to its rooted version (see section 2 for details). We draw inspiration from algorithmic ideas that led to polylogarithmic approximations for the k-MST problem. As described above, our approach focuses on the idea of low-density subgraphs.

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 $\{0, 1, 2\}$ [14]. The bucketing and scaling trick has seen several uses in the past and has recently been highlighted in several applications [9, 10, 8].

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. We remark that this yields a rather straightforward $O(\log n \log k)$ approximation for k-MST and could have been discovered much before a more clever analysis given in [3].

Our main technical contribution is Theorem 1.1, 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 1.2. 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.* [24] 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 [24] on the k-2EC problem. The combinatorial algorithm in [24] is based on finding a low-density cycle or a related structure called a bi-cycle. The algorithm in [24] 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 1.2 and Corollary 1.3. In independent work, Lau *et al.* [25] 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 [27]. 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. The following lemma is proved in Section 2.1. It relies on a 2-approximation, via a natural LP, for the min-cost 2-connectivity problem due to Fleischer, Jain and Williamson [14], and some standard techniques.

Lemma 2.1. *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' ← 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) ≤ k'):
         G' ← G' ∪ H,   k' ← k' − k(H).
         Mark all terminals in H as non-terminals.
   Else:
         Prune H to obtain H' that contains k' terminals.
         G' = G' ∪ H',   k' ← 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 2.2. *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 $O(\log \ell \cdot \log k)$OPT.*

Therefore, it remains only to bound the cost of the graph H' added in the pruning step, and Theorem 1.1, proved in Section 4, is precisely what is needed. We can now prove our main result for the k-2VC problem, Theorem 1.4.

Proof of Theorem 1.4: Let OPT be the cost of an optimal solution to the (rooted) k-2VC problem. By Lemma 2.2, the total cost of the augmentation steps of our greedy algorithm is $O(\log \ell \cdot \log k)$OPT. To bound the cost of the pruning step, let k' be the number of additional terminals that must be covered just prior to this step. The algorithm for the dens-2VC problem returns a graph H with $k(H) > k'$ terminals, and density at most $O(\log \ell)\frac{\text{OPT}}{k'}$. As a result of our pre-processing step, every vertex has 2 vertex-disjoint paths to r of total cost at most OPT. Now, we use Theorem 1.1 to prune H and find a graph H' with k' terminals and cost at most $O(\log k) density(H) k' + 2\text{OPT} \leq O(\log \ell \cdot \log k)\text{OPT} + 2\text{OPT}$. Therefore, the total cost of our solution is $O(\log \ell \cdot \log k)$OPT. □

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 2.3. *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 1.6, giving a bicriteria approximation for Budget-2VC, we only have to bound the cost of the pruning step.

Proof of Theorem 1.6: 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 1.1, 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 2.1, that there is an $O(\log \ell)$-approximation for the dens-2VC problem, and the crucial Theorem 1.1, bounding the cost of the pruning step. We prove the former in Section 2.1 immediately below. Before the latter is proved in Section 4, we develop some tools in Section 3; chief among these tools is Theorem 1.2.

2.1 An $O(\log \ell)$-approximation for the dens-2VC problem

Recall that the dens-2VC problem was defined as follows: Given a graph $G(V, E)$ with edge-costs, a set $T \subseteq V$ of terminals, and a root $r \in V(G)$, find a subgraph H of minimum density, in which every terminal of H is 2-connected to r. (Here, the density of H is defined as the cost of H divided by the number of terminals it contains, not including r.) We describe an algorithm for dens-2VC that gives an $O(\log \ell)$-approximation, and sketch its proof. We use an LP based approach and a bucketing and scaling trick (see [8, 9, 10] for applications of this idea), and a constant-factor bound on the integrality gap of an LP for SNDP with vertex-connectivity requirements in $\{0, 1, 2\}$ [14].

We define **LP-dens** as the following LP relaxation of dens-2VC . For each terminal t, the variable y_t indicates whether or not v is chosen in the solution. (By normalizing $\sum_t y_t$ to 1, and minimizing the sum of edge costs, we minimize the density.) $\mathcal{C}_t$ is the set of all simple cycles containing t and the root r; for any $C \in \mathcal{C}_t$, f_C indicates how much 'flow' is sent from v to r through C. (Note that a pair of vertex-disjoint paths is a cycle; the flow along a cycle is 1 if we can 2-connect t to r using the edges of the cycle.) The variable x_e indicates whether the edge e is used by the solution.

$$\min \sum_{e \in E} c(e) x_e$$

$$\sum_{t \in T} y_t = 1$$

$$\sum_{C \in \mathcal{C}_t} f_C \geq y_t \qquad (\forall t \in T)$$

$$\sum_{C \in \mathcal{C}_t | e \in C} f_C \leq x_e \qquad (\forall t \in T, e \in E)$$

$$x_e, f_c, y_t \geq 0$$

It is not hard to see that an optimal solution to **LP-dens** has cost at most the density of an optimal solution to dens-2VC . We now show how to obtain an integral solution of density at most $O(\log \ell)\mathrm{OPT}_{LP}$, where OPT_{LP} is the cost of an optimal solution to **LP-dens** . The linear program **LP-dens** has an exponential number of variables but a polynomial number of non-trivial constraints; it can, however, be solved in polynomial time. Fix an optimal solution to **LP-dens** of cost OPT_{LP}, and for each $0 \leq i < 2 \log \ell$ (for ease of notation, assume $\log \ell$ is an integer), let Y_i be the set of terminals t such that $2^{-(i+1)} < y_t \leq 2^{-i}$. Since $\sum_{t \in T} y_t = 1$, there is some index i such that $\sum_{t \in Y_i} y_t \geq \frac{1}{2 \log \ell}$. Since every terminal $t \in Y_i$ has $y_t \leq 2^{-i}$, the number of terminals in Y_i is at least $\frac{2^{i-1}}{\log \ell}$. We claim that there is a subgraph H of G with cost at most $O(2^{i+2}\mathrm{OPT}_{LP})$, in which every terminal of Y_i is 2-connected to the root. If this is true, the density of H is at most $O(\log \ell \cdot \mathrm{OPT}_{LP})$, and hence we have an $O(\log \ell)$-approximation for the dens-2VC problem.

To prove our claim about the cost of the subgraph H in which every terminal of Y_i is 2-connected to r, consider scaling up the given optimum solution of **LP-dens** by a factor of 2^{i+1}. For each terminal $t \in Y_i$, the flow from t to r in this scaled solution[3] is at least 1, and the cost of the scaled solution is $2^{i+1}\mathrm{OPT}_{LP}$.

In [14], the authors describe a linear program LP_1 to find a minimum-cost subgraph in which a given set of terminals is 2-connected to the root, and show that this linear program has an integrality gap of 2. The variables x_e in the 'scaled solution' to **LP-dens** correspond to a feasible solution of LP_1 with Y_i as the set of terminals; the integrality gap of 2 implies that there is a subgraph H in which every terminal of Y_i is 2-connected to the root, with cost at most $2^{i+2}\mathrm{OPT}_{LP}$.

Therefore, the algorithm for dens-2VC is:

1. Find an optimal fractional solution to **LP-dens** .
2. Find a set of terminals Y_i such that $\sum_{t \in Y_i} y_t \geq \frac{1}{2 \log \ell}$.
3. Find a min-cost subgraph H in which every terminal in Y_i is 2-connected to r using the algorithm of [14]. H has density at most $O(\log \ell)$ times the optimal solution to dens-2VC .

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

[3] This is an abuse of the term 'solution', since after scaling, $\sum_{t \in T} y_t = 2^{i+1}$

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.

First, we prove Theorem 1.2, that a 2-connected graph with edge costs and terminal weights contains a simple non-trivial cycle, with density no more than the average density of the graph. We give two algorithms to find such a cycle; the first, described in Section 3.1, is simpler, but the running time is not polynomial. A more technical proof that leads to a strongly polynomial-time algorithm is described in Section 3.2; we recommend this proof be skipped on a first reading.

3.1 An Algorithm to Find Cycles of Average Density

To find a non-trivial cycle of density at most that of the 2-connected input graph G, we will start with an arbitrary non-trivial cycle, and successively find cycles of better density until we obtain a cycle with density at most *density*(G). The following lemma shows that if a cycle C has an ear with density less than *density*(C), we can use this ear to find a cycle of lower density.

Lemma 3.1. *Let C be a non-trivial cycle, and H an ear incident to C at u and v, such that* $\frac{cost(H)}{weight(H-\{u,v\})} <$ density(C)*. Let S_1 and S_2 be the two internally disjoint paths between u and v in C. Then $H \cup S_1$ and $H \cup S_2$ are both simple cycles and one of these is non-trivial and has density less than* density(C)*.*

Proof. C has at least 2 terminals, so it has finite density; H must then have at least 1 terminal. Let c_1, c_2 and c_H be, respectively, the sum of the costs of the edges in S_1, S_2 and H, and let w_1, w_2 and w_H be the sum of the weights of the terminals in S_1, S_2 and $H - \{u, v\}$.

Assume w.l.o.g. that S_1 has density at most that of S_2. (That is, $c_1/w_1 \leq c_2/w_2$.)[4] S_1 must contain at least one terminal, and so $H \cup S_1$ is a simple non-trivial cycle. The statement *density*$(H \cup S_1) <$ *density*(C) is equivalent to $(c_H + c_1)(w_1 + w_2) < (c_1 + c_2)(w_H + w_1)$.

$$\begin{aligned}
(c_H + c_1)(w_1 + w_2) &= c_1w_1 + c_1w_2 + c_H(w_1 + w_2) \\
&\leq c_1w_1 + c_2w_1 + c_H(w_1 + w_2) && (\textit{density}(S_1) \leq \textit{density}(S_2)) \\
&< c_1w_1 + c_2w_1 + (c_1 + c_2)w_H && (c_H/w_H < \textit{density}(C)) \\
&= (c_1 + c_2)(w_H + w_1)
\end{aligned}$$

Therefore, $H \cup S_1$ is a simple cycle containing at least 2 terminals of density less than *density*(C). □

Lemma 3.2. *Given a cycle C in a 2-connected graph G, let G' be the graph formed from G by contracting C to a single vertex v. If H is a connected component of $G' - v$, $H \cup \{v\}$ is 2-connected in G'.*

Proof. Let H be an arbitrary connected component of $G' - v$, and let $H' = H \cup \{v\}$. To prove that H' is 2-connected, we first observe that v is 2-connected to any vertex $x \in H$. (Any set that separates x from v in H' separates x from the cycle C in G.)

It now follows that for all vertices $x, y \in V(H)$, x and y are 2-connected in H'. Suppose deleting some vertex u separates x from y. The vertex u cannot be v, since H is a connected component of $G' - v$. But if $u \neq v$, v and x are in the same component of $H' - u$, since v is 2-connected to x in H'. Similarly, v and y are in the same component of $H' - u$, and so deleting u does not separate x from y. □

[4]It is possible that one of S_1 and S_2 has cost 0 and weight 0. In this case, let S_1 be the component with non-zero weight.

We now show that given any 2-connected graph G, we can find a non-trivial cycle of density no more than that of G.

Theorem 3.3. *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*.** 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 *density*$(C) >$ *density*(G), we give an algorithm that finds a new non-trivial cycle C' such that *density*$(C') <$ *density*(C). Repeating this process, we obtain cycles of successively better densities until eventually finding a non-trivial cycle X of density at most *density*(G).

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' (from Lemma 3.2, 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, *density*$(H) <$ *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 *density*$(C') \leq$ *density*$(H) <$ *density*(C). If C' exists in the given graph G, it has the desired properties, and 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, so we can apply Lemma 3.1 to obtain a non-trivial cycle in G that has density less than *density*(C).

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 $cost(H)$, and concatenating these 2 paths corresponds to an ear of C in G. The density of this ear is less than *density*(C); again, we use Lemma 3.1 to obtain a cycle in G with the desired properties. □

We remark again that the algorithm of Theorem 3.3 does not lead to a polynomial-time algorithm, even if all edge costs and terminal weights are polynomially bounded. In Section 3.2, we describe a strongly polynomial-time algorithm that, given a graph G, finds a non-trivial cycle of density at most that of G. 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 3.3 to prove the following theorem:

Theorem 3.4. *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.*

***Proof*.** Assume we have a $\gamma(\ell)$-approximation for the dens-2VC problem; we use it to find a low-density non-trivial cycle. Solve the dens-2VC problem on the given graph; since the optimal cycle is a 2-connected graph, our solution H to the dens-2VC problem has density at most $\gamma(\ell)$ times the density of this cycle. Find a non-trivial cycle in H of density at most that of H; it has density at most $\gamma(\ell)$ times that of an optimal non-trivial cycle.

Note that any instance of the (unrooted) dens-2VC problem has an optimal solution that is a non-trivial cycle. (Consider any optimal solution H of density ρ; by Theorem 1.2, H contains a non-trivial cycle of density at most ρ. This cycle is a valid solution to the dens-2VC problem.) Therefore, a $\beta(\ell)$-approximation for the min-density non-trivial cycle problem gives a $\beta(\ell)$-approximation for the dens-2VC problem. □

Theorem 3.4 and Lemma 2.1 imply an $O(\log \ell)$-approximation for the minimum-density non-trivial cycle problem; this proves Corollary 1.3.

We say that a graph $G(V, E)$ is minimally 2-connected on its terminals if for every edge $e \in E$, some pair of terminals is not 2-connected in the graph $G - e$. Section 3.2 shows that in any graph which is

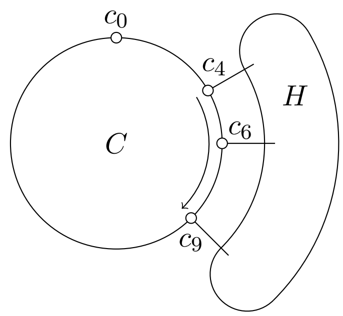

Figure 1: H is an earring of G, with clasps c_4, c_6, c_9; c_4 is its first clasp, and c_9 its last clasp. The arrow indicates the arc of H.

minimally 2-connected on its terminals, every cycle is non-trivial. Therefore, the problem of finding a minimum-density non-trivial cycle in such graphs is just that of finding a minimum-density cycle, which can be solved exactly in polynomial time. However, as we explain at the end of the section, this does not directly lead to an efficient algorithm for arbitrary graphs.

3.2 A Strongly Polynomial-time Algorithm to Find Cycles of Average Density

In this section, we describe a strongly polynomial-time algorithm which, given a 2-connected graph $G(V, E)$ with edge costs and terminal weights, finds a non-trivial cycle of density at most that of G.

We begin with several definitions: Let C be a cycle in a graph G, and G' be the graph formed by deleting C from G. Let $H_1, H_2, \ldots H_m$ be the connected components of G'; we refer to these as *earrings* of C.[5] For each H_i, let the vertices of C incident to it be called its *clasps*. From the definition of an earring, for any pair of clasps of H_i, there is a path between them whose internal vertices are all in H_i.

We say that a vertex of C is an *anchor* if it is the clasp of some earring. (An anchor may be a clasp of multiple earrings.) A *segment* S of C is a path contained in C, such that the endpoints of S are both anchors, and no internal vertex of S is an anchor. (Note that the endpoints of S might be clasps of the same earring, or of distinct earrings.) It is easy to see that the segments partition the edge set of C. By deleting a segment, we refer to deleting its edges and internal vertices. Observe that if S is deleted from G, the only vertices of $G - S$ that lose an edge are the endpoints of S. A segment is *safe* if the graph $G - S$ is 2-connected.

Arbitrarily pick a vertex o of C as the *origin*, and consecutively number the vertices of C clockwise around the cycle as $o = c_0, c_1, c_2, \ldots, c_r = o$. The first clasp of an earring H is its lowest numbered clasp, and the last clasp is its highest numbered clasp. (If the origin is a clasp of H, it is considered the first clasp, not the last.) The *arc* of an earring is the subgraph of C found by traversing clockwise from its first clasp c_p to its last clasp c_q; the length of this arc is $q - p$. (That is, the length of an arc is the number of edges it contains.) Note that if an arc contains the origin, it must be the first vertex of the arc. Figure 1 illustrates several of these definitions.

Theorem 3.5. *Let H be an earring of minimum arc length. Every segment contained in the arc of H is safe.*

Proof. Let $\mathcal{H}$ be the set of earrings with arc identical to that of H. Since they have the same arc, we refer to this as the arc of $\mathcal{H}$, or the *critical arc*. Let the first clasp of every earring in $\mathcal{H}$ be c_a, and the last clasp of

[5]If H_i were simply a path, it would be an ear of C, but H_i may be more complex.

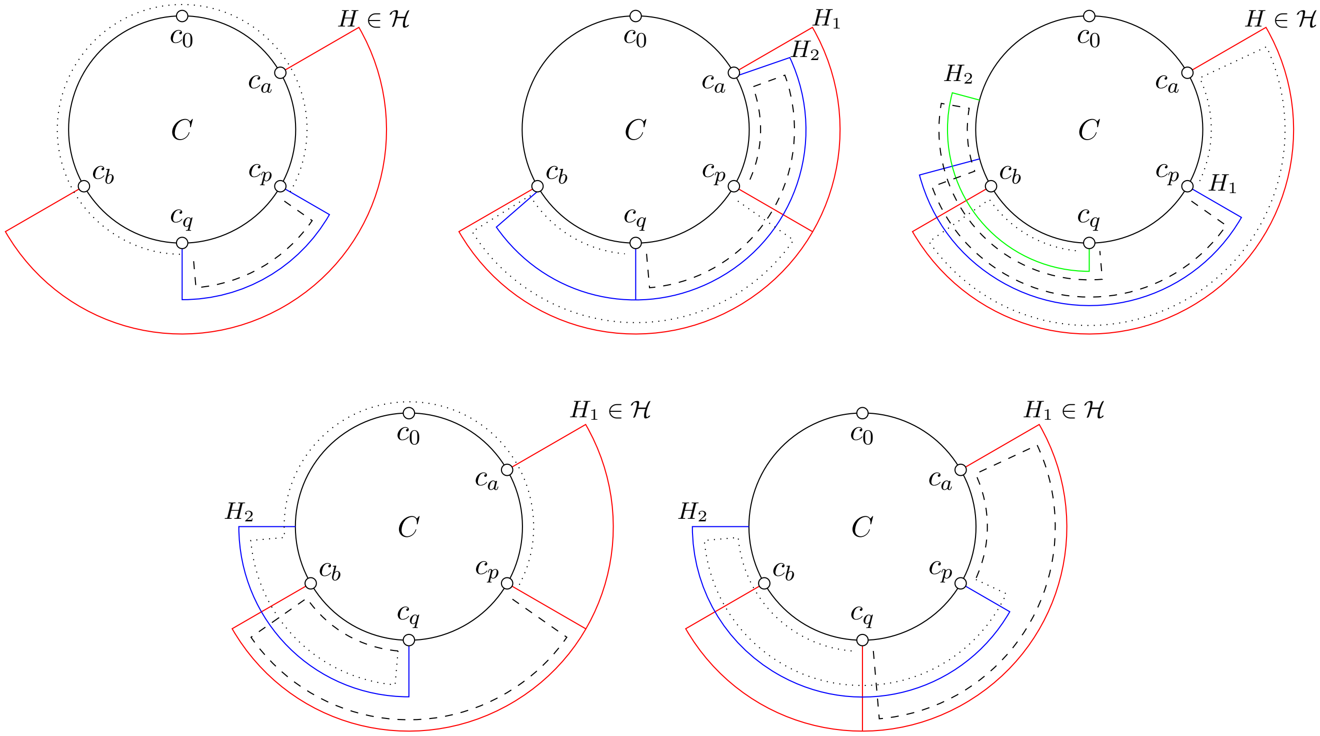

Figure 2: The various cases of Theorem 3.5 are illustrated in the order presented. In each case, one of the 2 vertex-disjoint paths from c_p to c_q is indicated with dashed lines, and the other with dotted lines.

each earring in $\mathcal{H}$ be c_b. Because the earrings in $\mathcal{H}$ have arcs of minimum length, any earring $H' \notin \mathcal{H}$ has a clasp c_x that is not in the critical arc. (That is, $c_x < c_a$ or $c_x > c_b$.)

We must show that every segment contained in the critical arc is safe; recall that a segment S is safe if the graph $G - S$ is 2-connected. Given an arbitrary segment S in the critical arc, let c_p and c_q ($p < q$) be the anchors that are its endpoints. We prove that there are always 2 internally vertex-disjoint paths between c_p and c_q in $G - S$; this suffices to show 2-connectivity.

We consider several cases, depending on the earrings that contain c_p and c_q. Figure 2 illustrates these cases. If c_p and c_q are contained in the same earring H', it is easy to find two vertex-disjoint paths between them in $G - S$. The first path is clockwise from q to p in the cycle C. The second path is entirely contained in the earring H' (an earring is connected in $G - C$, so we can always find such a path.)

Otherwise, c_p and c_q are clasps of distinct earrings. We consider three cases: Both c_p and c_q are clasps of earrings in $\mathcal{H}$, one is (but not both), or neither is.

1. We first consider that both c_p and c_q are clasps of earrings in $\mathcal{H}$. Let c_p be a clasp of H_1, and c_q a clasp of H_2. The first path is from c_q to c_a through H_2, and then clockwise along the critical arc from c_a to c_p. The second path is from c_q to c_b clockwise along the critical path, and then c_b to c_p through H_1. It is easy to see that these paths are internally vertex-disjoint.

2. Now, suppose neither c_p nor c_q is a clasp of an earring in $\mathcal{H}$. Let c_p be a clasp of H_1, and c_q be a clasp of H_2. The first path we find follows the critical arc clockwise from c_q to c_b (the last clasp of the critical arc), from c_b to c_a through $H \in \mathcal{H}$, and again clockwise through the critical arc from c_a to

c_p. Internal vertices of this path are all in H or on the critical arc. Let $c_{p'}$ be a clasp of H_1 not on the critical arc, and $c_{q'}$ be a last clasp of H_2 not on the critical arc. The second path goes from c_p to $c_{p'}$ through H_1, from p' to q' through the cycle C outside the critical arc, and from $c_{q'}$ to c_q through H_2. Internal vertices of this path are in H_1, H_2, or in C, but not part of the critical arc (since each of $c_{p'}$ and $c_{q'}$ are outside the critical arc). Therefore, we have 2 vertex-disjoint paths from c_p to c_q.

3. Finally, we consider the case that exactly one of c_p, c_q is a clasp of an earring in $\mathcal{H}$. Suppose c_p is a clasp of $H_1 \in \mathcal{H}$, and c_q is a clasp of $H_2 \notin \mathcal{H}$; the other case (where $H_1 \notin \mathcal{H}$ and $H_2 \in \mathcal{H}$ is symmetric, and omitted, though figure 2 illustrates the paths.) Let q' be the index of a clasp of H_2 outside the critical arc. The first path is from c_q to c_b through the critical arc, and then from c_b to c_p through H_1. The second path is from c_q to $c_{q'}$ through H_2, and from $c_{q'}$ to c_p clockwise through C. Note that the last part of this path enters the critical arc at c_a, and continues through the arc until c_p. Internal vertices of the first path that are in C are on the critical arc, but have index greater than q. Internal vertices of the second path that belong to C are either not in the critical arc, or have index between c_a and c_p. Therefore, the two paths are internally vertex-disjoint. □

We now describe our algorithm to find a non-trivial cycle of good density, proving Theorem 1.2: *Let G be a 2-connected graph with edge-costs and terminal weights, and at least 2 terminals. There is a polynomial-time algorithm to find a non-trivial cycle X in G such that* $\text{density}(X) \leq \text{density}(G)$.

Proof of Theorem 1.2: Let G be a graph with ℓ terminals and density ρ; we describe a polynomial-time algorithm that either finds a cycle in G of density less than ρ, or a proper subgraph G' of G that contains all ℓ terminals. In the latter case, we can recurse on G' until we eventually find a cycle of density at most ρ.

We first find, in $O(n^3)$ time, a minimum-density cycle C in G. By Theorem 3.3, C has density at most ρ, because the minimum-density *non-trivial* cycle has at most this density. If C contains at least 2 terminals, we are done. Otherwise, C contains exactly one terminal v. Since G contains at least 2 terminals, there must exist at least one earring of C.

Let v be the origin of this cycle C, and H an earring of minimum arc length. By Theorem 3.5, every segment in the arc of H is safe. Let S be such a segment; since v was selected as the origin, v is not an internal vertex of S. As v is the only terminal of C, S contains no terminals, and therefore, the graph $G' = G - S$ is 2-connected, and contains all ℓ terminals of G. □

The proof above also shows that if G is minimally 2-connected on its terminals (that is, G has no 2-connected proper subgraph containing all its terminals), every cycle of G is non-trivial. (If a cycle contains 0 or 1 terminals, it has a safe segment containing no terminals, which can be deleted; this gives a contradiction.) Therefore, given a graph that *is* minimally 2-connected on its terminals, finding a minimum-density non-trivial cycle is equivalent to finding a minimum-density cycle, and so can be solved exactly in polynomial time. This suggests a natural algorithm for the problem: Given a graph that is not minimally 2-connected on its terminals, delete edges and vertices until the graph is minimally 2-connected on the terminals, and then find a minimum-density cycle. As shown above, this gives a cycle of density no more than that of the input graph, but this may not be the minimum-density cycle of the original graph. For instance, there exist instances where the minimum-density cycle uses edges of a safe segment S that might be deleted by this algorithm.

4 Pruning 2-connected Graphs of Good Density

In this section, we prove Theorem 1.1. 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 cycle 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 in G):
      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, the 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.

By bounding the density of large clusters, we can find a solution to the rooted k-2VC problem of bounded density. Because we always use cycles of low density to merge clusters, an analysis similar to that of [24] and [11] shows that every large cluster has density at most $O(\log^2 k)\rho$. We first present this analysis, though it does not suffice to prove Theorem 1.1. A more careful analysis shows that there is at least one large cluster of density at most $O(\log k)\rho$; this allows us to prove the desired theorem.

We now formally prove that MERGECLUSTERS has the desired behavior. First, we present a series of claims which, together, show that when the algorithm terminates, most terminals are in large clusters, and all clusters are 2-connected.

Remark 4.1. *Throughout the algorithm, the graph G is always 2-connected. The weight of a cluster is at most the number of terminals it contains.*

Proof. The only structural changes to G are when new vertices are added as terminals; they are added with edges to two distinct vertices of G. This preserves 2-connectivity, as does deleting these terminals with the associated edges.

To see that the second claim is true, observe that if a terminal contributes weight to a cluster, it is contained in that cluster. A terminal can be in multiple clusters, but it contributes to the weight of exactly one cluster. □

We use the following simple proposition in proofs of 2-connectivity; the proof is straightforward, and hence omitted.

Proposition 4.2. *Let $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$ be 2-connected subgraphs of a graph $G(V, E)$ such that $|V_1 \cap V_2| \geq 2$. Then the graph $H_1 \cup H_2 = (V_1 \cup V_2, E_1 \cup E_2)$ is 2-connected.*

Lemma 4.3. *The clusters formed by* MERGECLUSTERS *are all 2-connected.*

Proof. Let Y be a cluster formed by using a cycle C to merge clusters $X_1, X_2, \ldots X_q$. The edges of the cycle C form a 2-connected subgraph of G, and we assume that each X_j is 2-connected by induction. Further, C contains at least 2 vertices of each X_j: if C contains an edge of X_j, this follows immediately, and if it contains a (dummy) terminal, it must contain at least the two edges of X_j incident to this terminal.[6] Therefore, we can use induction and Proposition 4.2 above: We assume $C \cup \{X_l\}_{l=1}^{j}$ is 2-connected by induction, and C contains 2 vertices of X_{j+1}, so $C \cup \{X_l\}_{l=1}^{j+1}$ is 2-connected.

Note that we have shown $Y = C \cup \{X_j\}_{j=1}^{q}$ is 2-connected, but C (and hence Y) might contain dummy terminals and the corresponding dummy edges. However, each such terminal with the 2 associated edges is a ear of Y; deleting them leaves Y 2-connected. More formally, if u, v are the other endpoints of the edges incident to the dummy terminal in X_j, there are at least 2 disjoint paths remaining between u and v even after deleting the dummy edges, as X_j was 2-connected prior to the introduction of the dummy terminal. □

Lemma 4.4. *The total weight of small clusters in tier i that are not merged to form clusters of higher tiers is at most $\frac{\ell}{2\lceil \log k \rceil}$.*

[6] A cluster X_j may be a singleton vertex (for instance, if we are in tier 0), but such a vertex does not affect 2-connectivity.

Proof. Assume this were not true; this means that MERGECLUSTERS could find no more cycles of density at most α using the remaining small tier i clusters. But the total cost of all the edges is at most $cost(G)$, and the sum of terminal weights is at least $\frac{\ell}{2\lceil \log k \rceil}$; this implies that the density of the graph (using the remaining terminals) is at most $2\lceil \log k \rceil \cdot \frac{cost(G)}{\ell} = \alpha$. But by Theorem 3.3, the graph must then contain a good non-trivial cycle, and so the while loop would not have terminated. □

Corollary 4.5. *When the algorithm* MERGECLUSTERS *terminates, the total weight of large clusters is at least $\ell/2 > (4 \log k) \cdot k$.*

Proof. Each terminal not in a large cluster contributes to the weight of a cluster that was not merged with others to form a cluster of a higher tier. The previous lemma shows that the total weight of such clusters in any tier is at most $\frac{\ell}{2\lceil \log k \rceil}$; since there are $\lceil \log k \rceil$ tiers, the total number of terminals not in large clusters is less than $\lceil \log k \rceil \cdot \frac{\ell}{2\lceil \log k \rceil} = \ell/2$. □

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 4.6. *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 $2\delta k + 2L$.*

Proof. Let $X_1, X_2, \ldots X_q$ be the clusters merged to form Y in order around the cycle C that merged them; each X_j was a small cluster, of weight at most k. A simple averaging argument shows that there is a consecutive segment of X_js with total weight between k and $2k$, such that the cost of the edges of C connecting these clusters, together with the costs of the clusters themselves, is at most $2\delta k$. Let X_a be the "first" cluster of this segment, and X_b the "last". Let v and w be arbitrary terminals of X_a and X_b respectively. Connect each of v and w to the root r using 2 vertex-disjoint paths; the cost of this step is at most $2L$. (We assumed that every terminal could be 2-connected to r using disjoint paths of cost at most L.) The graph Y' thus constructed has at least k terminals, and total cost at most $2\delta k + 2L$.

We show that every vertex z of Y' is 2-connected to r; this completes our proof. Let z be an arbitrary vertex of Y'; suppose there is a cut-vertex x which, when deleted, separates z from r. Both v and w are 2-connected to r, and therefore neither is in the same component as z in $Y' - x$. However, we describe 2 vertex-disjoint paths P_v and P_w in Y' from z to v and w respectively; deleting x cannot separate z from both v and w, which gives a contradiction. The paths P_v and P_w are easy to find; let X_j be the cluster containing z. The cycle C contains a path from vertex $z_1 \in X_j$ to $v' \in X_a$, and another (vertex-disjoint) path from $z_2 \in X_j$ to $w' \in X_b$. Concatenating these paths with paths from v' to v in X_a and w' to w in X_b gives us vertex-disjoint paths P_1 from z_1 to v and P_2 from z_2 to w. Since X_j is 2-connected, we can find vertex-disjoint paths from z to z_1 and z_2, which gives us the desired paths P_v and P_w.[7] □

We now present the two analyses of density referred to earlier. The key difference between the weaker and tighter analysis is in the way we bound edge costs. In the former, each large cluster pays for its edges separately, using the fact that all cycles used have density at most $\alpha = O(\log k)\rho$. In the latter, we crucially use the fact that small clusters which share edges are merged. Roughly speaking, because small clusters are edge-disjoint, the average density of small clusters must be comparable to the density of the input graph G.

[7] The vertex z may not be in any cluster X_j. In this case, P_v is formed by using edges of C from z to $v' \in X_a$, and then a path from v' to v; P_w is formed similarly.

Once an edge is in a large cluster, we can no longer use the edge-disjointness argument. We must pay for these edges separately, but we can bound this cost.

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

Lemma 4.7. *For any cluster Y formed by* MERGECLUSTERS *during iteration i, the total cost of edges in Y is at most $(i+1) \cdot \alpha w_Y$.*

Proof. We prove this lemma by induction on the number of vertices in a cluster. Let $\mathcal{S}$ be the set of clusters merged using a cycle C to form Y. Let $\mathcal{S}_1$ be the set of clusters in $\mathcal{S}$ of tier i, and $\mathcal{S}_2$ be $\mathcal{S} - \mathcal{S}_1$. ($\mathcal{S}_2$ contains clusters of tiers less or greater than i that contained an edge of C.)

The cost of edges in Y is at most the sum of: the cost of C, the cost of $\mathcal{S}_1$, and the cost of $\mathcal{S}_2$. Since all clusters in $\mathcal{S}_2$ have been formed during iteration i or earlier, and are smaller than Y, we can use induction to show that the cost of edges in $\mathcal{S}_2$ is at most $(i+1)\alpha \sum_{X \in \mathcal{S}_2} w_X$. All clusters in $\mathcal{S}_1$ are of tier i, and so must have been formed before iteration i (any cluster formed during iteration i is of a strictly greater tier), so we use induction to bound the cost of edges in $\mathcal{S}_1$ by $i\alpha \sum_{X \in \mathcal{S}_1} w_X$.

Finally, because C was a good-density cycle, and only clusters of tier i contribute to calculating the density of C, the cost of C is at most $\alpha \sum_{X \in \mathcal{S}_1} w_X$. Therefore, the total cost of edges in Y is at most $(i+1)\alpha \sum_{X \in \mathcal{S}} w_X = (i+1)\alpha w_Y$. □

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 4.6 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 1.1; 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, any 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. Such a tree could be formed if, in iteration $i-1$, 4 clusters of this tier merged to form D, a cluster of tier $i+1$. Subsequently, in iteration i, clusters H and J merge to form F. We next find a good cycle containing E and G; F contains an edge of this cycle, so these three clusters are merged to form B. Note that the cost of this cycle is paid for the by the weights of E and G only; F is a tier $i+1$ cluster, and so its weight is not included in the density calculation. Finally, we find a good cycle paid for by A and C; since B and D share edges with this cycle, they all merge to form the large cluster Y.

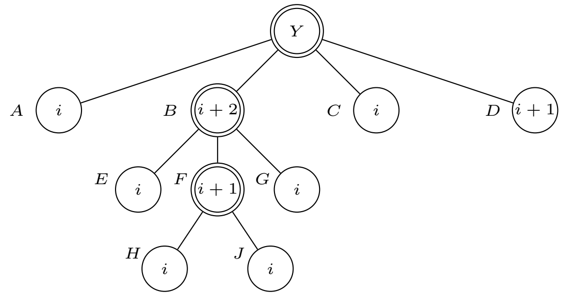

Figure 3: A part of the Tree T_Y corresponding to Y, a large cluster of type i. The number in each vertex indicates the tier of the corresponding cluster. Only final-stage and penultimate clusters are shown: final-stage clusters are indicated with a double circle; all other clusters are penultimate.

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 4.7.) We formalize this intuition in the next lemma.

Lemma 4.8. *The final cost of any large cluster Y is at most αw_Y, where w_Y is the weight of Y.*

Proof. Let Y be an arbitrary large cluster. In the construction of the tree T_Y, we associated with each vertex of T_Y the cost of the cycle used to form the corresponding cluster. To bound the total final cost of Y, we must bound the sum of the costs of vertices of T_Y associated with final-stage clusters. The weight of Y, w_Y is at least the sum of the weights of the penultimate tier i clusters that become a part of Y. Therefore, it suffices to show that the sum of the costs of vertices of T_Y associated with final-stage clusters is at most α times the sum of the weights of Y's penultimate tier i clusters. (Note that a tier i cluster must have been formed prior to iteration i, and hence it cannot itself be a final-stage cluster.)

A cycle was used to construct a final-stage cluster X only if its cost was at most α times the sum of weights of the penultimate tier i clusters that become a part of X. (Larger clusters may become a part of X, but they do not contribute weight to the density calculation.) Therefore, if X is a vertex of T_Y corresponding to a final-stage cluster, the cost of X is at most α times the sum of the weights of its tier i immediate children in T_Y. But T_Y is a tree, and so no vertex corresponding to an penultimate tier i cluster has more than one parent. That is, the weight of a penultimate cluster pays for only one final-stage cluster. Therefore, the sum of the costs of vertices associated with final-stage clusters is at most α times the sum of the weights of Y's penultimate tier i clusters, and so the final cost of Y is at most αw_Y. □

Lemma 4.9. *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.*

Proof. Suppose, by way of contradiction, that some edge e is a penultimate edge of both Y_1 and Y_2, which are large clusters of type i. Let X_1 (respectively X_2) be a penultimate cluster of Y_1 (resp. Y_2) containing e.

As penultimate clusters, both X_1 and X_2 are formed before iteration i. But until iteration i, neither is part of a large cluster, and two small clusters cannot share an edge without being merged. Therefore, X_1 and X_2 must have been merged, so they cannot belong to distinct large clusters, giving the desired contradiction. □

Theorem 4.10. *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 4.8, 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 4.10 and Lemma 4.6 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 1.1.

5 Conclusions

We list the following open problems:

- Can the approximation ratio for the k-2VC problem be improved from the current $O(\log \ell \log k)$ to $O(\log n)$ or better? Removing the dependence on ℓ to obtain even $O(\log^2 k)$ could be interesting. If not, can one improve the approximation ratio for the easier k-2EC problem?

- Can we obtain approximation algorithms for the k-λVC or k-λEC problems for $\lambda > 2$? In general, few results are known for problems where vertex-connectivity is required to be greater than 2, but there has been more progress with higher edge-connectivity requirements.

- Given a 2-connected graph of density ρ with some vertices marked as terminals, we show that it contains a non-trivial cycle with density at most ρ, and give an algorithm to find such a cycle. We have also found an $O(\log \ell)$-approximation for the problem of finding a minimum-density non-trivial cycle. Is there a constant-factor approximation for this problem? Can it be solved *exactly* in polynomial time?

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. Preliminary version in *Proc. of ACM STOC*, 1996.

[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] F. A. Chudak, T. Roughgarden, and D. P. Williamson. Approximate k-MSTs and k-Steiner Trees via the Primal-Dual Method and Lagrangean Relaxation. *Math. Program.* 100(2): 411-421 (2004). Preliminary version in *Proc. of IPCO*, 60–70, 2001.

[13] 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.

[14] 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.

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

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

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

[18] 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.

[19] 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.

[20] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1996.

[21] K. Jain. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem *Combinatorica*, 21(1):39–60, 2001. Preliminary version in *Proc. of IEEE FOCS*, 448–457, 1998.

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

[23] 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.

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

[25] 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*

[26] 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.

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

[28] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.