Unbalanced Graph Cuts

Ara Hayrapetyan[1] ⋆, David Kempe[2] ⋆⋆, Martin Pál[3] ⋆⋆⋆, and Zoya Svitkina[1] †

[1] Dept. of Computer Science, Cornell University. {ara,zoya}@cs.cornell.edu
[2] Dept. of Computer Science, University of Southern California. dkempe@usc.edu
[3] DIMACS Center, Rutgers University. mpal@dimacs.rutgers.edu

Abstract. We introduce the Minimum size bounded capacity cut (MinSBCC) problem, in which we are given a graph with an identified source, and seek to find a cut minimizing the number of nodes on the source side, subject to the constraint that its capacity not exceed a prescribed bound B. Besides being of interest in the study of graph cuts, this problem arises in many practical settings, such as in epidemiology, disaster control, military containment, as well as finding dense subgraphs and communities in graphs.

In general, the MinSBCC problem is NP-complete. We present an efficient $(\frac{1}{\lambda}, \frac{1}{1-\lambda})$-bicriteria approximation algorithm for any $0 < \lambda < 1$; that is, the algorithm finds a cut of capacity at most $\frac{1}{\lambda}B$, leaving at most $\frac{1}{1-\lambda}$ times more vertices on the source side than the optimal solution with capacity B. In fact, the algorithm's solution *either* violates the budget constraint, *or* exceeds the optimal number of source-side nodes, but not both. For graphs of bounded treewidth, we show that the problem with unit weight nodes can be solved optimally in polynomial time, and when the nodes have weights, approximated arbitrarily well by a PTAS.

1 Introduction

Graph cuts are among the most well-studied objects in theoretical computer science. In the most pristine form of the problem, two given vertices s and t have to be separated by the removal of an edge set of minimum total capacity. By a fundamental result of Ford and Fulkerson [1], such an edge set can be found in polynomial time. Since then, many problems have been shown to reduce to graph cut problems, sometimes quite surprisingly (see, e.g., [2]). One way to view the Min-Cut Problem is to think of "protecting" the sink node t from the presumably harmful node s by way of removing edges: the capacity of the cut then corresponds to the cost of edge removals. This interpretation in turn suggests a very natural variant of the graph cut problem: given a node s and a bound B on the total edge removal cost, try to "protect" as many nodes from s as possible, while cutting at most a total edge capacity of B. In other words,

⋆ Supported in part by NSF grant CCR-0311333.
⋆⋆ Work done while supported by an NSF graduate fellowship.
⋆⋆⋆ Supported by NSF grant EIA 02-05116, and ONR grant N00014-98-1-0589.
† Supported in part by NSF grant CCR-032553.

find an s-t cut of capacity at most B, minimizing the size of the s-side of the cut. This is the MINIMUM SIZE BOUNDED CAPACITY CUT (MINSBCC) problem studied in this paper.

Naturally, the MINSBCC problem has direct applications in the areas of disaster, military, or crime containment. In all of these cases, a limited amount of resources can be used to monitor or block the edges by which the disaster could spread, or people could escape. At the same time, the area to which the disaster is confined should be as small as possible. For instance, in the firefighter's problem [3], a fixed small number of firefighters must confine a fire to within a small area, trying to minimize the value of the property and lives inside.

Perhaps even more importantly, the MINSBCC problem arises naturally in the control of epidemic outbreaks. While traditional models of epidemics [4] have ignored the network structure in order to model epidemic diseases via differential equations, recent work by Eubank et al. [5, 6], using highly realistic large-scale simulations, has shown that the graph structure of the social contacts has a significant impact on the spread of the epidemic, and crucially, on the type of actions that most effectively contain the epidemic. If we assume that patient 0, the first infected member of the network, is known, then the problem of choosing which individuals to vaccinate in order to confine the epidemic to a small set of people is exactly the node cut version of the MINSBCC problem.

Besides the obvious connections to the containment of damage or epidemics, the MINSBCC problem can also be used for finding small dense subgraphs and communities in graphs. Discovering communities in graphs has received much attention recently, in the context of analyzing social networks and the World Wide Web [7, 8]. It involves examining the link structure of the underlying graph so as to extract a small set of nodes sharing a common property. This property is usually expressed by high internal connectivity, sometimes in combination with small expansion. We show how to reduce the community finding problem to MINSBCC.

Our Results. Formally, we define the MINSBCC problem as follows. Given an (undirected or directed) graph $G = (V, E)$ with edge capacities c_e, source and sink nodes s and t, as well as a total capacity bound (also called the *budget*) B, we wish to find an s-t cut $(S, \overline{S}), s \in S$ of capacity no more than B, which leaves as few nodes on the source side as possible. We will also consider a generalization in which the nodes are assigned weights w_v, and the objective is to minimize the total node weight $\sum_{v \in S} w_v$, subject to the budget constraint. [1]

We show in Sections 2 and 4.2 that MINSBCC is NP-hard on general graphs with uniform node weights, and on trees with non-uniform node weights. We therefore develop two $(\frac{1}{\lambda}, \frac{1}{1-\lambda})$-bicriteria approximation algorithms for MINSBCC, where $0 < \lambda < 1$. These algorithms, in polynomial time, find a cut $(S, \overline{S})$ of capacity at most $\frac{1}{\lambda}B$, such that the size of S is at most $\frac{1}{1-\lambda}$ times that of S^*, where $(S^*, \overline{S^*})$ is the optimal cut of capacity at most B. The first algorithm

[1] Some of our motivating examples and applications do not specify a sink; this can be resolved trivially by adding an isolated sink to the graph.

obtains this guarantee by a simple rounding of a linear programming relaxation of MinSBCC. The second one bypasses solving the linear program by running a single parametric maximum flow computation and is thus very efficient [9]. It also has a better guarantee: it outputs either a $(\frac{1}{\lambda}, 1)$-approximation or a $(1, \frac{1}{1-\lambda})$-approximation, thus violating at most one of the constraints by the corresponding factor. The analysis of this algorithm is based on the same linear programming formulation of MinSBCC and its Lagrangian relaxation.

We then investigate the MinSBCC problem for graphs of bounded treewidth in Section 3. We give a polynomial-time algorithm based on dynamic programming to optimally solve MinSBCC for graphs of bounded treewidth with unit node weights. We then extend the algorithm to a PTAS for general node weights.

Section 4 discusses the reductions from node cut and dense subgraph problems to MinSBCC, thus showing that our algorithms can be used to address the problems described above. Finally, we conclude with directions for future work in Section 5.

Related Work. Minimum cuts have a long history of study and form part of the bread-and-butter of everyday work in algorithms [10]. While minimum cuts can be computed in polynomial time, additional constraints on the size of the cut or on the relationship between its capacity and size (such as its density) usually make the problem NP-hard.

Much recent attention has been given to the computation of *sparse cuts*, partly due to their application in divide-and-conquer algorithms [11]. The seminal work of Leighton and Rao [12] gave the first $O(\log n)$ approximation algorithm for sparsest and balanced cut problems using region growing techniques. This work was later extended by Garg, Vazirani, and Yannakakis [13]. In a recent breakthrough result, the approximation factor for these problems was improved to $O(\sqrt{\log n})$ by Arora, Rao, and Vazirani [14].

A problem similar to MinSBCC was studied in [15, 16]: given a number k, find an s-t cut $(S, \overline{S})$ with $|S| = k$ of minimum capacity. They obtain an $O(\log^2 n)$ approximation algorithm in the general case [15], and improved approximation guarantees when k is small [16].

MinSBCC has a natural maximization version MaxSBCC, where the goal is to maximize the size of the s-side of the cut instead of minimizing it, while still obeying the capacity constraint. This problem was recently introduced by Svitkina and Tardos [17]. Obviously, an exact solution to MaxSBCC will yield an exact solution to MinSBCC. However, the approximation guarantees may differ significantly. Based on the work of Feige and Krauthgamer [15], Svitkina and Tardos give an $(O(\log^2 n), 1)$-bicriteria approximation which is used as a black box to obtain an approximation algorithm for the min-max multiway cut problem, in which one seeks a multicut minimizing the number of edges leaving any one component. The techniques in [17] readily extend to an $(O(\log^2 n), 1)$-bicriteria approximation for the MinSBCC problem.

2 Bicriteria Approximation Algorithms

We first establish the NP-completeness of MinSBCC.

Proposition 1. *The* MinSBCC *problem with arbitrary edge capacities and node weights is NP-complete even when restricted to trees.*

Proof. We give a reduction from Knapsack. Let the Knapsack instance consist of items $1, \ldots, n$ with sizes $s_1, \ldots, s_n$ and values $a_1, \ldots, a_n$, and let the total Knapsack size be B. We create a source s, a sink t, and a node v_i for each item i. The node weight of v_i is a_i, and it is connected to the source by an edge of capacity s_i. The sink t has weight 0, and is connected to v_1 by an edge of capacity 0. The budget for the MinSBCC problem is B.

The capacity of any s-t cut is exactly the total size of the items on the t-side, and minimizing the total node weight on the s-side is equivalent to maximizing the total value of items corresponding to nodes on the t-side. ■

Now, we turn our attention to our main approximation results, which are the two $(\frac{1}{\lambda}, \frac{1}{1-\lambda})$-bicriteria approximation algorithms for MinSBCC on general graphs. For the remainder of the section, we will use $\delta(S)$ to denote the capacity of the cut $(S, \overline{S})$ in G. We use S^* to denote the minimum-size set of nodes such that $\delta(S^*) \leq B$, i.e. $(S^*, V \setminus S^*)$ is the optimum cut of capacity at most B.

The analysis of both of our algorithms is based on the following Linear Programming (LP) relaxation of the natural integer program for MinSBCC. We use a variable x_v for every vertex $v \in V$ to denote which side of the cut it is on, and a variable y_e for every edge e to denote whether or not the edge is cut.

$$\begin{array}{lll} \text{Minimize} & \sum_{v \in V} x_v & \\ \text{subject to} & x_s = 1 & \\ & x_t = 0 & \\ & y_e \geq x_u - x_v & \text{for all } e = (u,v) \in E \\ & \sum_{e \in E} y_e \cdot c_e \leq B & \\ & x_v, y_e \geq 0 & \end{array} \tag{1}$$

2.1 Randomized Rounding-Based Algorithm

Our first algorithm is based on a randomized rounding of the solution to (1).

Algorithm 1 Randomized LP-rounding algorithm with parameter λ

1: Let (x^*, y^*) be the optimal solution to LP (1).
2: Choose $\ell \in [1-\lambda, 1]$ uniformly at random.
3: Let $S = \{v \mid x^*_v \geq \ell\}$, and output S.

Theorem 1. *The Randomized Rounding algorithm (Algorithm 1) outputs a set S of size at most $\frac{1}{1-\lambda}$ times the LP objective value. The expected capacity of the cut $(S, \overline{S})$ is at most $\frac{1}{\lambda}B$.*

Proof. To prove the first statement of the theorem, observe that for each $v \in S$, $x_v^* \geq \ell \geq 1 - \lambda$. Therefore $\sum_{v \in V} x_v^* \geq \sum_{v \in S} x_v^* \geq (1-\lambda)|S|$.

For the second statement, observe that ℓ is selected uniformly at random from an interval of size λ. Furthermore, an edge $e = (u, v)$ will be cut only if ℓ lies between x_u^* and x_v^*. The probability of this happening is thus at most $\frac{|x_u^* - x_v^*|}{\lambda} \leq \frac{y_e^*}{\lambda}$. Summing over all edges yields that the expected total capacity of the cut is at most $\sum_e \frac{c_e y_e^*}{\lambda} \leq \frac{1}{\lambda}B$. ■

This algorithm can easily be derandomized: instead of choosing ℓ randomly, we can try all possibilities and output the smallest cut we encounter. Since there are at most $|V|$ different values of x_v^*, we need to try at most $|V|$ cuts. The algorithm can also be extended to instances with non-negative node weights w_v: simply add the weights in the objective function — the analysis stays the same.

2.2 A Parametric Flow-Based Algorithm

Next, we show how to avoid solving the LP, and instead compute the cuts directly via a parametric max-flow computation. The somewhat more involved analysis will also show that in fact, at most one of the two criteria is approximated, while the other is not violated.

Algorithm Description: The algorithm searches for candidate solutions among the parametrized minimum cuts in the graph G^α, which is obtained from G by adding an edge of capacity α from every vertex v to the sink t (introducing parallel edges if necessary). Here, α is a parameter ranging over non-negative values. Observe that the capacity of a cut $(S, \overline{S})$ in the graph G^α is $\alpha|S| + \delta(S)$, so the minimum s-t cut in G^α minimizes $\alpha|S| + \delta(S)$.

Initially, as $\alpha = 0$, the min-cut of G^α is the min-cut of G. As α increases, the source side of the min-cut of G^α will contain fewer and fewer nodes, until eventually it contains the single node $\{s\}$. All these cuts for the different values of α can be found efficiently using a single run of the push relabel algorithm. Moreover, the source sides of these cuts form a nested family $S_0 \supset S_1 \supset ... \supset S_k$ of sets [9]. (S_0 is the minimum s-t cut in the original graph, and $S_k = \{s\}$) . Our solution will be one of these cuts S_j.

We first observe that $\delta(S_i) < \delta(S_j)$ if $i < j$; for if it were not, then S_j would be a superior cut to S_i for all values of α. If $\delta(S_k) \leq B$, then, of course, $\{s\}$ is the optimal solution. On the other hand, if $\delta(S_0) > B$, then no solution exists. In all other cases, choose i such that $\delta(S_i) \leq B \leq \delta(S_{i+1})$. If $\delta(S_{i+1}) \leq \frac{1}{\lambda}B$, then output S_{i+1}; otherwise, output S_i.

Theorem 2. *The above algorithm produces either (1) a cut S^- such that $\delta(S^-) \leq B$ and $|S^-| \leq \frac{1}{1-\lambda}|S^*|$, or (2) a cut S^+ such that $\delta(S^+) \leq \frac{1}{\lambda}B$ and $|S^+| \leq |S^*|$.*

Proof. For the index i chosen by the algorithm, we let $S^- = S_i$ and $S^+ = S_{i+1}$. Hence, $\delta(S^-) \leq B \leq \delta(S^+)$.

First, observe that $|S^+| \leq |S^*|$, as otherwise, the parametric cut procedure would have returned S^* instead of S^+. If S^+ also satisfies $\delta(S^+) \leq \frac{1}{\lambda}B$, then we are done, so we focus on the case that $\delta(S^+) > \frac{1}{\lambda}B$. In that case, we will prove that $|S^-| \leq \frac{1}{1-\lambda}|S^*|$.

Because S^+ and S^- are neighbors in our sequence of parametric cuts, there is a value of α, call it α^*, for which both are minimum cuts of G^{α^*}. Applying the Lagrangian Relaxation technique, we remove the constraint $\sum_e y_e c_e \leq B$ from LP (1) and put it into the objective function using the constant α^*.

$$\begin{array}{lll} \text{Minimize} & \alpha^* \cdot \sum_{v \in V} x_v + \sum_{e \in E} y_e \cdot c_e & \\ \text{subject to} & x_s \quad = 1 & \\ & x_t \quad = 0 & \\ & y_e \quad \geq x_u - x_v & \text{for all } e = (u,v) \in E \\ & x_v, y_e \geq 0 & \end{array} \tag{2}$$

Lemma 1. *LP (2) has an integer optimal solution.*

Proof. Recall that in G^{α^*} we added edges of capacity α^* from every node to the sink. Extend any solution of LP (2) to these edges by setting $y_e = x_v - x_t = x_v$ for the newly added edge e connecting v to t. We claim that after this extension, the objective function of LP (2) is equivalent to $\sum_{e \in G^{\alpha^*}} y_e c'_e$, where c'_e is the edge capacity in the graph G^{α^*}. Indeed, this claim follows from observing that the first part of the objective of LP (2) is identical to the contribution that the newly added edges of G^{α^*} are making towards $\sum_{e \in G^{\alpha^*}} y_e c'_e$.

Consider some fractional optimal solution $(\hat{x}, \hat{y})$ to LP (2) that has objective function value $L^* = \sum_{e \in G^{\alpha^*}} \hat{y}_e c'_e$. As this is an optimal solution, we can assume without loss of generality that $y_e = \max(0, x_u - x_v)$ for all edges $e = (u,v)$. So if we define $w_x = \sum_{u,v: x_u \geq x \geq x_v} c'_{uv}$, then the value of the objective function is $L^* = \int_0^1 w_x \mathrm{d}x$.

Also, for any $x \in (0,1)$, we can obtain an integral solution to LP (2) whose objective function value is w_x by rounding $\hat{x}_v$ to 0 if it is no more than x, and to 1 otherwise (and setting $y_{uv} = \max(0, x_u - x_v)$). Since this process yields feasible solutions, we know that $w_x \geq L^*$ for all x. On the other hand, L^* is a weighted average (integral) of w_x's, and hence in fact $w_x = L^*$ for all x, and any of the rounded solutions is an integral optimal solution to LP (2). ■

Notice that feasible integral solutions to LP (2) correspond to s-t cuts in G^{α^*}. Therefore, by Lemma 1, the optimal solutions to LP (2) are the minimum s-t cuts in G^{α^*}. In particular, S^+ and S^- are two such cuts. From S^+ and S^-, we naturally obtain solutions to LP (2), by setting $x^+_v = 1$ for $v \in S^+$ and $x^+_v = 0$ otherwise, with $y^+_e = 1$ if e is cut by $(S^+, \overline{S^+})$, and 0 otherwise (similarly for S^-). By definition of α^*, both (x^+, y^+) and (x^-, y^-) are then optimal solutions to LP (2). Thus, their linear combination $(x^*, y^*) = \ell \cdot (x^+, y^+) + (1-\ell) \cdot (x^-, y^-)$ is also an optimal feasible solution. Choose ℓ such that

$$\ell \cdot \sum_{e\in E} y_e^+ c_e + (1-\ell) \cdot \sum_{e\in E} y_e^- c_e = B. \tag{3}$$

Such an ℓ exists because our choice of S^- and S^+ ensured that $\delta(S^-) \leq B \leq \delta(S^+)$. For this choice of ℓ, the fractional solution (x^*, y^*), in addition to being optimal for the Lagrangian relaxation, also satisfies the constraint $\sum_e y_e^* c_e \leq B$ of LP (1) with equality, implying that it is optimal for LP (1) as well. Crudely bounding the second term in Equation (3) by 0, we obtain that $\delta(S^+) = \sum_{e\in E} y_e^+ c_e \leq \frac{B}{\ell}$.

As we assumed that $\delta(S^+) > \frac{B}{\lambda}$, we conclude that $\ell < \lambda$. Because (x^*, y^*) is an optimal solution to LP (1), it provides the lower bound $\sum_v x_v^* \leq |S^*|$, and the fact that $x_v^* \geq (1-\ell)x_v^-$ now implies that $|S^-| = \sum_{v\in V} x_v^- \leq \frac{|S^*|}{1-\ell} \leq \frac{1}{1-\lambda} \cdot |S^*|$.

Hence, in this case, S^- meets the capacity constraint, and exceeds the optimal size by at most a factor of $\frac{1}{1-\lambda}$. ■

As in the case of the LP rounding algorithm presented above, some simple cosmetic modifications allow for the inclusion of node weights in addition to edge capacities.

3 Bounded Treewidth

As we saw in Section 2, the MinSBCC problem is NP-complete even on trees when both node weights and edge capacities are considered. However, if all nodes have unit weights, then the problem can be solved in polynomial time for graphs of bounded treewidth, via a dynamic programming algorithm. In order to present the intuition behind our algorithm, we first describe it for trees, and then extend it to graphs of bounded treewidth (see [2] for a review of tree decompositions and graphs of bounded treewidth).

3.1 An Algorithm for Trees

We root the tree at the source node s and direct all edges away from s. When all edges have capacity 1, then clearly, only edges incident with s should be cut. They must include the edge on the unique s-t path, and in addition, the edges to the roots of the largest subtrees. Choosing these B edges gives the smallest possible size for the s-side of the cut.

For the case of general edge capacities, consider the tree T_v rooted at a node v, together with the edge e_v into v. We define the quantity a_v^k to be the smallest total capacity of edges in T_v that must be cut if at most k nodes of T_v are to be included. Notice that $a_v^0 = c_{e_v}$. Also, as the sink must always be excluded, we have $a_t^k = c_{e_t}$ for all k.

For a leaf v, we have $a_v^0 = c_{e_v}$, and $a_v^k = 0$ for $k > 0$. For an internal node v with children $v_1, \ldots, v_d$, we can either cut the edge e_v into v, or otherwise include v and solve the problem recursively for the children of v, hence

$$a_v^k = \min\left(c_{e_v}, \min_{k_1\geq 0,\ldots,k_d\geq 0: \sum k_i = k-1} \sum_i a_{v_i}^{k_i}\right) \qquad \text{for } k > 0, v \neq t.$$

Note that the optimal partition into k_i's can be found in polynomial time by a nested dynamic programming subroutine that uses optimal partitions of each k into $k_1 \dots k_j$ in order to calculate the optimal partition into $k_1 \dots k_{j+1}$.

Once we have computed a_s^k at the source s for all values of k, we simply pick the smallest k^* such that $a_s^{k^*} \leq B$.

3.2 An Algorithm for Graphs with Bounded Treewidth

Recall [2] that a graph $G = (V, E)$ has treewidth θ if there exists a tree T, and subsets $V_w \subseteq V$ of nodes associated with each vertex w of T, satisfying the following properties:

1. Every node $v \in V$ is contained in some subset V_w.
2. For every edge $e = (u, v) \in E$, some set V_w contains both u and v.
3. If $\hat{w}$ lies on the path between w and w' in T, then $V_w \cap V_{w'} \subseteq V_{\hat{w}}$.
4. $|V_w| \leq \theta + 1$ for all vertices w of the tree T.

The pair $(T, \{V_w\})$ is called a *tree decomposition* of G, and the sets V_w will be called *pieces*. It can be shown that for any two neighboring vertices w and w' of the tree T, the deletion of $V_w \cap V_{w'}$ from G disconnects G into two components, just as the deletion of the edge (w, w') would disconnect T into two components.

We assume that we are given a tree decomposition $(T, \{V_w\})$ for G with treewidth θ. [4] We associate with each node $w \in T$ a set $E_w \subseteq E \cap (V_w \times V_w)$ of edges both of whose endpoints lie in V_w, such that each edge appears in exactly one set E_w; if an edge lies entirely in V_w for several nodes w, we assign it arbitrarily to one of them. We will identify some node r of the tree T with $s \in V_r$ as being the *root*, and consider the edges of T as directed away from r.

Let $W \subseteq T$ be the set of nodes in the subtree rooted at some node w, $E_W = \bigcup_{u \in W} E_u$, and $V_W = \bigcup_{u \in W} V_u$. Also, let $U, U' \subseteq V_w$ be arbitrary disjoint sets of nodes. We define $a_w^k(U, U')$ to be the minimum capacity of edges from E_W that must be cut by any set $S \subseteq V_W$ such that $S \supseteq U$, $S \cap U' = \emptyset$, the sink t is not included in S (i.e., $t \notin S$), and $|S \setminus U| \leq k$. But for the extension regarding the sets U and U', this is exactly the same quantity we were considering in the case of trees. Also, notice that the minimum size of any cut of capacity at most B is the smallest k for which $a_r^{k-1}(\{s\}, \emptyset) \leq B$.

Our goal is to derive a recurrence relation for $a_w^k(U, U')$. At any stage, we will be taking a minimum over all subsets that meet the constraints imposed by the size k and the sets U, U'. We therefore write $\mathcal{S}_w^k(U, U') = \{S \mid U \subseteq S \subseteq V_w, S \cap U' = \emptyset, t \notin S, |S| \leq k\}$. The size of $\mathcal{S}_w^k(U, U')$ is $O(2^\theta)$. The cost incurred by cutting edges assigned to w is denoted by $\beta_w(S) = \sum_{e \in E_w \cap e(S, V_w \setminus S)} c_e$, and can be computed efficiently.

If w is a leaf node, then we can include up to k additional nodes, so long as the constraints imposed by the sets U and U' are not violated. Hence,

$$a_w^k(U, U') = \min_{S \in \mathcal{S}_w^k(U, U')} \beta_w(S).$$

[4] Although NP-hard in general, a tree decomposition can be found in linear time for graphs of bounded treewidth [18].

For a non-leaf node w, let $w_1, \ldots, w_d$ denote its children. We can include an arbitrary subset of nodes, so long as we add no more than k nodes, and do not violate the constraints imposed by the sets U and U'. The remaining additional nodes can then be divided among the children of w in any way desired. Once we have decided to include (or exclude) a node $v \in V_w$, this decision must be respected by all children, i.e., we obtain different sets as constraints for the children. Notice that any node v contained in the pieces at two descendants of w must also be in the piece at w itself by property 3 of a tree decomposition. Also, by the same property, any node v from V_w that is not in V_{w_i} (for some child w_i of w) will not be in the piece at any descendant of w_i, and hence the information about v being forbidden or forced to be included is irrelevant in the subtree rooted at w_i. We hence have the following recurrence:

$$a_w^k(U,U') = \min_{S \in \mathcal{S}_w^k(U,U')} \quad \min_{(k_i)_i : \sum k_i = k - |S \setminus U|} \left(\beta_w(S) + \sum_i a_{w_i}^{k_i}(S \cap V_{w_i}, (V_w \setminus S) \cap V_{w_i}) \right).$$

As before, for any fixed set S, the minimum over all combinations of k_i values can be found by a nested dynamic program.

By induction over the tree, we can prove that this recursive definition actually coincides with the initial definition of $a_w^k(U,U')$, and hence that the algorithm is correct. The computation of $a_w^k(U,U')$ takes time $O(d \cdot k \cdot 2^\theta) = O(n^2 \cdot 2^\theta)$. For each node, we need to compute $O(n^2 \cdot 4^\theta)$ values, so the total running time is $O(n^4 \cdot 8^\theta)$, and the space requirement is $O(n^2 \cdot 4^\theta)$. To summarize, we have proved the following theorem:

Theorem 3. *For graphs of treewidth bounded by θ, there is an algorithm that finds, in polynomial time $O(8^\theta n^4)$, an optimal* MinSBCC.

3.3 A PTAS for the node-weighted version

When the input graph has bounded treewidth, the above algorithm for solving MinSBCC with unit node weights can be used to obtain a polynomial-time approximation scheme (PTAS) for MinSBCC with arbitrary node weights. We only sketch the algorithm here; the details will be given in the full version.

Suppose we want to achieve a $(1+\epsilon)$-guarantee. We guess the weight W of the source side of the cut in the optimal solution, and first remove all *heavy* nodes, those whose original weight is more than W. We then rescale the remaining node weights w_v to $w'_v := \lceil \frac{w_v n}{\epsilon W} \rceil$. Notice that the largest node weight is now at most $\frac{n}{\epsilon}$. Hence, we can run the above dynamic programming algorithm, in time polynomial in n. It is not hard to see that the total weight of the source-side nodes in the solution obtained is within $(1+\epsilon)$ of the optimum, and the budget constraint is satisfied. Hence, this gives a PTAS for the MinSBCC problem.

4 Applications

4.1 Epidemiology and Node Cuts

Several of the problems given in the introduction are phrased much more naturally in terms of node cuts than edge cuts. For instance, while military or other disaster containment problems can be thought of as controlling escape or spread pathways, i.e., edges, controlling the outbreak of a disease involves the vaccination of individuals, and thus the removal of nodes from the graph. Therefore, it is natural to study the node cut version. Here, each node has a *weight* w_v, the cost of including it on the s-side of the cut, and a *capacity* c_v, the cost of removing (cutting) it from the graph. The goal is to find a set $R \subseteq V$, not containing s, of capacity $c(R)$ not exceeding a budget B, such that after removing R, the connected component S containing s has minimum total weight $w(S)$.

This problem can be reduced to (node-weighted) MinSBCC in the standard way. First, if the original graph G was undirected, we bidirect each edge. Now, each vertex v is split into two vertices v_{in} and v_{out}; all edges into v now enter v_{in}, while all edges out of v now leave v_{out}. We add a directed edge from v_{in} to v_{out} of capacity c_v. Each originally present edge, i.e., each edge into v_{in} or out of v_{out}, is given infinite capacity. Finally, v_{in} is given node weight 0, and v_{out} is given node weight w_v. Call the resulting graph G'.

Now, it is easy to verify that (1) no edge cut in G' ever cuts any originally present edges, (2) the capacity of an edge cut in G' is equal to the node capacity of a node cut in G, and (3) the total node weight on the s-side of an edge cut in G' is exactly the total node weight in the s component of the corresponding node cut in G. Hence, we have reduced the node cut problem to MinSBCC, and any approximation guarantees obtained for MinSBCC will equally carry over.

4.2 Graph Communities

Identifying "communities" has been an important and much studied problem for social or biological networks, and more recently, the web graph [7, 19]. Different mathematical formalizations for the notion of a community have been proposed, but they usually share the property that a community is a node set with high edge density within the set, and comparatively small expansion.

It is well known [20] that the *densest subgraph*, i.e., the set S maximizing $\frac{c(S)}{|S|} := \frac{c(S,S)}{|S|}$ can be found in polynomial time via a reduction to Min-Cut; Charikar [21] recently showed that a simple greedy algorithm gives a 2-approximation for this problem. On the other hand, if the size of the set S is prescribed to be at most k, then the problem is the well-known *densest k-subgraph problem* [22–24], which is known to be NP-complete (via a simple reduction from Clique), and was recently shown to be hard to approximate to within $(1+\epsilon)$ for some $\epsilon > 0$ [25]. The best known approximation ratio is $O(n^{1/3-\epsilon})$ for some $\epsilon > 0$ [23].

Here, we study the variant where the density of the subgraph is given, and the size needs to be minimized. In addition, we allow nodes to have non-negative weights w_v: in particular, by choosing the node weights equal to the degrees, we obtain a variant closely related to the "community" definition of Flake et al. [7]. We show how to reduce this general size minimization problem to MinSBCC in an approximation-preserving way. In particular, by applying this reduction to the densest k-subgraph problem, we show that MinSBCC is NP-hard even for the case of unit node weights.

Given a graph $G = (V, E)$ with edge capacities c_e, node weights w_v, and a specified node $s \in V$, we consider the problem of finding an *α-community containing* s: that is finding the smallest set S containing s with $\frac{c(S)}{w(S)} \geq \alpha$. (The version where s is not specified can obviously be reduced to this one by trying all nodes s.) We modify G to obtain a graph G' as follows. Add a sink t, connect each vertex v to the source s with an edge of capacity $d(v) := \sum_u c_{(v,u)}$, and to the sink with an edge of capacity $2\alpha w_v$. The capacity for all edges $e \in E$ stays unchanged.

Theorem 4. *A set $S \subseteq V$ with $s \in S$ has $\frac{c(S)}{w(S)} \geq \alpha$ if and only if $(S, \overline{S} \cup \{t\})$ is an s-t cut of capacity at most $2c(V) = 2\sum_{e \in E} c_e$ in G'.*

Notice that this implies that any approximation guarantees on the size of S carry over from the MinSBCC problem to the problem of finding communities. Also notice that by making all node weights and edge capacities 1, and setting $\alpha = \frac{k-1}{2}$, a set S is an α-community of size at most k if and only if S is a k-clique. Hence, the MinSBCC problem is NP-hard even with unit node weights. However, the approximation hardness of Clique does not carry over, as the reduction requires the size k to be known.

Proof. The α-community condition can be rewritten as $c(S) - \alpha w(S) \geq 0$. As

$$2\big(c(S) - \alpha w(S)\big) = 2c(V) - \big(c(S, \overline{S}) + \sum_{v \in \overline{S}} d(v) + 2\alpha w(S)\big),$$

we find that S is an α-community iff $c(S, \overline{S}) + \sum_{v \in \overline{S}} d(v) + 2\alpha w(S) \leq 2c(V)$. The quantity on the left-hand side is exactly the capacity of the cut $(S, \overline{S} \cup \{t\})$, which proves the theorem. ■

5 Conclusion

In this paper, we present a new graph-theoretic problem called the minimum size bounded capacity cut problem, in which we seek to find unbalanced cuts of bounded capacity. Much attention has already been devoted to balanced and sparse cuts [11–14]; we believe that unbalanced cut problems will pose an interesting new direction of research and will enhance our understanding of graph cuts. In addition, as we have shown in this paper, unbalanced cut problems have applications in disaster and epidemics control as well as in computing small dense subgraphs and communities in graphs. Together with the problems discussed in [15–17], the MinSBCC problem should be considered part of a more general framework of finding *unbalanced* cuts in graphs.

This paper raises many interesting questions for future research. In particular, the main open question is how well the MinSBCC problem can be approximated in a single-criterion sense. At this time, we are not aware of any non-trivial upper or lower bounds for its approximability. The work of [15, 17] implies a $(\log^2 n, 1)$

approximation — however, it approximates the capacity instead of the size, and thus cannot be used for approximating dense subgraphs or communities.

Obtaining better approximation algorithms will likely require using techniques different from the ones used in this paper. Specifically, the linear program that was used in both of the bicriteria approximation algorithms has a large integrality gap, as evidenced by a graph consisting of an isolated sink t, and a very large dense subgraph connected to the source s with one edge of capacity $1 + \epsilon$. The only s-t cut of capacity at most 1 is then the trivial one separating t from the rest of the graph, but the optimal fractional solution will fractionally cut the edge from s to the large subgraph, and have only an ϵ fraction of the cost of the optimum integral solution.

Further open directions involve more realistic models of the spread of diseases or disasters. The implicit assumption in our node cut approach is that each social contact will always result in an infection. If edges have infection probabilities, for instance based on the frequency or types of interaction, then the model becomes significantly more complex. We leave a more detailed analysis for future work.

Acknowledgments We would like to thank Tanya Berger-Wolf, Venkat Guruswami, Jon Kleinberg, and Éva Tardos for useful discussions.

References

1. Ford, L., Fulkerson, D.: Maximal flow through a network. Can. J. Math **8** (1956) 399–404
2. Kleinberg, J., Tardos, E.: Algorithms Design. Addison-Wesley (2005) to appear.
3. Develin, M., Hartke, S.G.: Fire containment in grids of dimension three and higher (2004) Submitted.
4. Bailey, N.: The Mathematical Theory of Infectious Diseases and its Applications. Hafner Press (1975)
5. Eubank, S., Guclu, H., Kumar, V., Marathe, M., Srinivasan, A., Toroczkai, Z., , Wang, N.: Modelling disease outbreaks in realistic urban social networks. Nature **429** (2004) 180–184
6. Eubank, S., Kumar, V., Marathe, M., Srinivasan, A., , Wang, N.: Structural and algorithmic aspects of massive social networks. In: Proc. 15th ACM Symp. on Discrete Algorithms. (2004) 711–720
7. Flake, G., Lawrence, S., Giles, C.L., Coetzee, F.: Self-organization of the web and identification of communities. IEEE Computer **35** (2002)
8. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Trawling the web for emerging cyber-communities. In: 8th International World Wide Web Conference. (1999)
9. Gallo, G., Grigoriadis, M.D., Tarjan, R.E.: A fast parametric maximum flow algorithm and applications. SIAM J. on Computing **18** (1989) 30–55
10. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice Hall (1993)
11. Shmoys, D.: Cut problems and their application to divide-and-conquer. In Hochbaum, D., ed.: Approximation Algorithms for NP-hard problems. PWD Publishing (1995) 192–235

12. Leighton, F., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. Journal of the ACM **46** (1999)
13. Garg, N., Vazirani, V.V., Yannakakis, M.: Approximate max-flow min-(multi)cut theorems and their applications. SIAM J. on Computing **25** (1996) 235–251
14. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings and graph partitioning. In: Proc. 35th ACM Symp. on Theory of Computing. (2004)
15. Feige, U., Krauthgamer, R.: A polylogarithmic approximation of the minimum bisection. SIAM J. on Computing **31** (2002) 1090–1118
16. Feige, U., Krauthgamer, R., Nissim, K.: On cutting a few vertices from a graph. Discrete Applied Mathematics **127** (2003) 643–649
17. Svitkina, Z., Tardos, E.: Min-max multiway cut. In: APPROX. (2004)
18. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM J. on Computing **25** (1996) 1305–1317
19. Flake, G., Tarjan, R., Tsioutsiouliklis, K.: Graph clustering techniques based on minimum cut trees. Technical Report 2002-06, NEC, Princeton (2002)
20. Lawler, E.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehard and Winston (1976)
21. Charikar, M.: Greedy approximation algorithms for finding dense components in graphs. In: APPROX. (2000)
22. Asahiro, Y., Iwama, K., Tamaki, H., Tokuyama, T.: Greedily finding a dense subgraph. Journal of Algorithms **34** (2000)
23. Feige, U., Kortsarz, G., Peleg, D.: The dense k-subgraph problem. In: Proc. 25th ACM Symp. on Theory of Computing. (1993)
24. Feige, U., Seltser, M.: On the densest k-subgraph problem. Technical report, The Weizmann Institute, Rehovot (1997)
25. Khot, S.: Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique. In: Proc. 45th IEEE Symp. on Foundations of Computer Science. (2004)