LIPIcs Leibniz International Proceedings in Informatics

Bounded Size Graph Clustering with Applications to Stream Processing

Rohit Khandekar[1], Kirsten Hildrum[1], Sujay Parekh[1], Deepak Rajan[1], Jay Sethuraman[2], and Joel Wolf[1]

[1]IBM T.J.Watson Research Center
{rohitk,hildrum,sujay,drajan,jlwolf}@us.ibm.com

[2]Columbia University
js1353@columbia.edu

ABSTRACT. We introduce a graph clustering problem motivated by a stream processing application. Input to our problem is an undirected graph with vertex and edge weights. A cluster is a subset of the vertices. The *size* of a cluster is defined as the total vertex weight in the subset plus the total edge weight at the boundary of the cluster. The bounded size graph clustering problem (BSGC) is to partition the vertices into clusters of size at most a given budget and minimize the total edge-weight across the clusters. In the *multiway cut* version of the problem, we are also given a subset of vertices called *terminals*. No cluster is allowed to contain more than one terminal. Our problem differs from most of the previously studied clustering problems in that the number of clusters is not specified. We first show that the feasibility version of the multiway cut BSGC problem, i.e., determining if there exists a clustering with bounded-size clusters satisfying the multiway cut constraint, can be solved in polynomial time. Our algorithm is based on the min-cut subroutine and an uncrossing argument. This result is in contrast with the NP-hardness of the min-max multiway cut problem, considered by Svitkina and Tardos (2004), in which the number of clusters must equal the number of terminals. Our results for the feasibility version also generalize to any symmetric submodular function. We next show that the optimization version of BSGC is NP-hard by showing an approximation-preserving reduction from the $\frac{1}{3}$-balanced cut problem. Our main result is an $O(\log^2 n)$-approximation to the optimization version of the multiway cut BSGC problem violating the budget by an $O(\log n)$ factor, where n denotes the number of vertices. Our algorithm is based on a set-cover-like greedy approach which iteratively computes bounded-size clusters to maximize the number of new vertices covered.

1 Introduction

Graph partitioning and clustering are fundamental optimization problems with applications to a variety of areas like VLSI design, divide-and-conquer algorithms, computer vision, data analysis, discovering communities in social networks, and learning. In this paper we introduce a graph clustering problem motivated by *System S*, a stream computing system [1] being developed at IBM research. Consider a system that takes, as input, a high-throughput data stream such as live option trading or stock feeds in financial services, physical link statistics in networking and telecommunications, sensor readings in environmental monitoring and emergency response, or live experimental data in scientific applications. This system is required to generate responses derived from on-line processing of the data

Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 275–286
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2325

in real-time. An application in this system can be modeled as a graph in which the vertices represent domain-specific *operators* consuming and producing streams of data, and the edges represent the *streams* themselves.

It is very convenient to compose these applications out of type-generic, built-in stream processing operators [7]. But these operators are usually small: they typically spend more effort sending and receiving data than they do in actual processing. Taken individually, such operators can therefore become performance bottlenecks in the system. The good news is that it is actually possible to "fuse" operators at compile time. If two adjacent operators are fused, the downstream operator is invoked by means of a function call from the upstream operator. As a result, there is effectively no cost to sending data between fused pairs of adjacent operators.

Thus, to efficiently deploy such a CPU-intensive application in a distributed computing environment, one has to decide how to partition the operators into clusters, for example one for each computing host. The total CPU requirement for a cluster of operators comes from two sources. The first is the computational needs of the operators in the cluster. This can be modeled by associating a non-negative weight with each operator u. The total computational needs of a cluster is then the sum of its operators weights. The second is the communication overhead, incurred at the *boundary* of the cluster, for receiving and sending streams to operators outside the cluster. This can be modeled by associating a non-negative weight with each edge $e = (u, v)$. The total communication needs of a cluster is then the cut-weight with respect to these edge-weights. As noted, an edge contained within a cluster is converted into a function-call, incurring negligible overhead and as such not accounted for in the computational needs of the cluster. The consideration of the CPU requirement imposes a natural constraint on the clustering: the total CPU requirement of each cluster must be at most the capacity of a host.

We frequently encounter additional resource constraints that cannot be captured as CPU requirements. For example, some operators make extensive use of specific hardware (such as a network card). Clustering two such operators together would cause poor performance. To handle such situations, we include in the problem a set of terminals T and insist that each cluster contain at most one terminal from T.

A high-throughput application, if not carefully deployed, may overload the network capacity. Therefore, a natural goal when computing a bounded-size clustering is to minimize the total edge-weights across the clusters.

Formal problem definition. With the above motivation, we introduce the `Bounded-Size Graph Clustering` (`BSGC`) problem, defined formally as follows. Consider an undirected graph $G = (V, E)$ on n vertices with vertex-weights $w_v \in \mathbb{Q}_+$ and edge-weights $w_e \in \mathbb{Q}_+$. Here $\mathbb{Q}_+$ denotes the set of non-negative rational numbers. For a subset $S \subset V$, let $\delta(S)$ denote the set of edges with exactly one end-point in S, $w(S) = \sum_{v \in S} w_v$, $w(\delta(S)) = \sum_{e \in \delta(S)} w_e$, and $\texttt{size}(S) = w(S) + w(\delta(S))$. We are also given a set of *terminals* $T \subseteq V$ and a *budget* $B \in \mathbb{Q}_+$. The `BSGC` problem is to find a partitioning of the vertex set into clusters $S_1, \ldots, S_k$ such that

- the size of each cluster is bounded: $\texttt{size}(S_i) \leq B$ for all i;
- each cluster contains *at most* one terminal, i.e., $|S_i \cap T| \leq 1$ for all i; and

- the total edge-weight across the clusters, $\frac{1}{2}\sum_{i=1}^{k} w(\delta(S_i))$, is minimized.

We call a clustering that satisfies the first two properties *feasible*. From the second condition, it is clear that the number of clusters k is at least the number of terminals $|T|$. However, k is *not* given as input, and it may be *larger* than the number of terminals.

Our results. Our main results are summarized below:

1. First, in Section 2, we consider the *feasibility* version of BSGC, i.e., to compute a feasible clustering without considering the total cut-weight. We show that we can compute a feasible clustering, if it exists, in polynomial time. Our algorithm uses a minimum cut subroutine and an uncrossing argument to make the clusters disjoint. This result generalizes to any symmetric submodular function [6]. See Section 2.1 for more details.
2. In Section 3.1 we show that the BSGC problem is NP-hard by an approximation preserving reduction from the $\frac{1}{3}$-balanced cut problem. Recall that the best-known approximation for the $\frac{1}{3}$-balanced cut problem that does not violate* the balance constraint is $O(\log n)$ [10].
3. Finally, in Section 3.2, we present a pseudo-approximation for the *optimization* version of the problem. More precisely, we present a deterministic polynomial-time algorithm that computes a clustering $\{S_1, \ldots, S_k\}$ such that $|S_i \cap T| \leq 1$ and $w(\delta(S_i)) \leq O(\log n) \cdot (B - w(S_i))$ for all i, and the total cut-weight is $O(\log^2 n)$ times the optimum cut-weight. Note that the above condition implies that $(c \log n) \cdot w(S_i) + w(\delta(S_i)) \leq (c \log n) \cdot B$ for some absolute constant $c > 0$, i.e., the budget is violated by an $O(\log n)$ factor.

Related work. A problem that is closely related to the feasibility version of BSGC was studied by Svitkina and Tardos [11]. In that problem, called *min-max multiway cut*, an edge-weighted undirected graph with terminals $T \subseteq V$ is given. The goal is to partition vertices into $|T|$ clusters such that each cluster contains *exactly* one terminal and the maximum cut value of a cluster is minimized. A crucial difference is that BSGC does not require the number of clusters to be exactly $|T|$. Svitkina and Tardos show that the min-max multiway cut problem is NP-hard and present an $O(\log^2 n)$-approximation† for it. They do so using, as a subroutine, the maximum-size bounded capacity cut problem (MaxSBCC), defined as follows: Given an undirected graph $G = (V, E)$ with vertex and edge weights, two vertices $s, t \in V$, and a budget $B > 0$, find an s-t cut $(S, V \setminus S)$ such that $w(\delta(S)) \leq B$ and $w(S)$ is maximized. Svitkina and Tardos iteratively solve MaxSBCC with varying vertex weights and combine those cuts to compute their final clustering.

Several cut problems with budget constraints were also considered by Engelberg et al. [5]. In particular, they consider budgeted multiway cut problems in which there is a budget on the total cut-value and the objective is either to maximize the number of terminal-pairs separated, to maximize the number of terminals that are completely separated from

*We can obtain an $O(\sqrt{\log n})$ approximation if we violate the budget by a constant factor [2].

†In fact, they present an $O(\log^3 n)$-approximation using a subroutine for finding minimum cuts with the specified number of vertices. Using an improved $O(\log n)$-approximation for the subroutine [10], their algorithm can be shown to yield an $O(\log^2 n)$-approximation.

other terminals, or simply to maximize the number of connected components. They use Räcke's tree decomposition [10] and Gomory-Hu trees [8] to design their approximation algorithms.

Most graph clustering problems in the literature specify the number of clusters required as part of the input. One important exception is *correlation clustering*, introduced by Bansal et al. [3]. In this problem, the edges of an undirected graph are labelled with either "$+$" or "$-$". Given a clustering of the vertices, let the number of "agreements" be the number of $+$ edges inside the clusters plus the number of $-$ edges across the clusters. Similarly the number of "disagreements" is the total number of edges minus the number of agreements. Bansal et al. and several subsequent papers design approximation algorithms for maximizing the number of agreements or minimizing the number of disagreements.

Finally, Khandekar et al. [9] consider a variant of our graph clustering problem (with significantly more elaborate practical constraints). Their (partially heuristic) solution is, in fact, implemented as a key component in *System S* [1].

Our techniques. In contrast to min-max multiway cut, relaxing the constraint on the number of clusters makes it possible to find a polynomial-time algorithm for the feasibility version of BSGC. For each vertex v, our algorithm computes a cluster, of size at most B, containing v and at most one terminal. To this end, we first augment the graph by adding a new vertex with edges to all the old vertices and "translate" the vertex weights into weights on the new edges. Later we show that the problem of computing a desired cluster can be reduced to minimum cut computations in the augmented graph. The clusters thus computed may not be disjoint, however. The algorithm then systematically *uncrosses* the clusters to make them disjoint, using an argument similar to [11], while satisfying the budget and the multiway cut constraints. This result applies more generally: if the size of a cluster S is defined as $\sum_{v \in S} w_v + f(S)$, where f is a symmetric submodular set function, we can determine in polynomial time if there exists a clustering such that each cluster contains at most one terminal and has size at most B.

The optimization version of BSGC is different from the traditional graph partitioning into clusters of bounded-size, because the size of a cluster includes its cut-cost. Therefore the hierarchical partitioning approach – iteratively splitting clusters into two until the size constraints are satisfied – does not work for the BSGC problem. For example, after splitting the given graph into two, there may not exist a feasible clustering respecting this split, even if the original graph has a feasible clustering.

Our approach for the optimization problem resembles that of Svitkina and Tardos [11]. We think of our problem as an instance of the set-cover problem where the sets are the subsets $S \subseteq V$ such that $|S \cap T| \leq 1$ and $\texttt{size}(S) \leq B$. Let the cost of such a set be $w(\delta(S))$. The problem is then to find a minimum-cost collection of sets that covers all the vertices. Now in order to use the greedy algorithm, we need the following *oracle*: given a subset of vertices not yet covered, find a set S that minimizes the ratio of $w(\delta(S))$ and the number of vertices in S that are not yet covered. Unfortunately the oracle itself is NP-hard. We then use a *hierarchical tree-decomposition* of graphs by Räcke [10] to get a $O(\log n)$-approximation

to the oracle. More precisely, we find $S \subseteq V$ such that

$$(c \log n) \cdot w(S) + w(\delta(S)) \leq (c \log n) \cdot B$$

for an absolute constant $c > 0$ that also minimizes the desired ratio to within an $O(\log n)$ factor. This, combined with a standard set-cover analysis, yields our final result. Once again we use an uncrossing argument to make the clusters disjoint.

2 The feasibility version

Since the definition of the size of a cluster involves both the vertex and edge weights, it is not clear a-priori if the feasibility version, i.e., to determine if there *exists* a feasible clustering, is tractable. For example, the clustering obtained by putting each vertex into a separate cluster may not be feasible. Assuming that the problem is feasible, we now present a polynomial-time algorithm for finding a clustering $\{S_1, \ldots, S_k\}$ such that $\texttt{size}(S_i) \leq B$ and $|S_i \cap T| \leq 1$ for all i.

Our idea is to construct a new graph in which the vertex weights are converted into edge weights on artificial edges. This means our algorithm can work just with cuts. We construct this graph $G' = (V', E')$ as follows. (See Figure 1.) Let $V' = V \cup \{s\}$ for a new vertex s and $E' = E \cup \{(s, v) \mid v \in V\}$. Each edge $e \in E' \cup E$ inherits its weight w_e, and $e = (s, v) \in E'$ gets a weight of $w_e = w_v$ for all $v \in V$. Note that for a cluster $S \subseteq V$, we have that $\texttt{size}(S)$ equals the capacity of the cut $(S, V' \setminus S)$ in G'.

In a problem instance *without* terminals, we note that Gomory-Hu trees [8] allow us to determine feasibility. Consider the Gomory-Hu tree $\mathcal{T}$ of G'. In a feasible instance, the minimum cut in G' between s and any other vertex u is at most B, and the edges in $\mathcal{T}$ that are incident to s have weight at most B each. The removal these edges from T gives a partitioning of vertices into clusters, say $S_1, \ldots, S_k$. It is easy to see that this is a feasible clustering for our problem. If there are edges in $\mathcal{T}$ that are incident to s and have weight greater than B, the problem instance is not feasible.

To approach the problem *with* terminals, we start by stating a useful lemma that will simplify the presentation of our algorithm. The following lemma states that it is enough to compute possibly *overlapping* clusters that satisfy the given constraints. The basic technique used in this lemma is systematic *uncrossing*.

LEMMA 1. *Given clusters $\{S_1, \ldots, S_k\}$ such that $\cup_i S_i = V$ and $|S_i \cap T| \leq 1$ for all i, we can compute in polynomial time clusters $\{U_1, \ldots, U_k\}$ such that $\cup_i U_i = V$, $|U_i \cap T| \leq 1$, $w(U_i) \leq w(S_i)$, $w(\delta(U_i)) \leq w(\delta(S_i))$ for all i, and moreover $U_i \cap U_j = \emptyset$ for $i \neq j$*

PROOF. For two disjoint subsets $A, B \subset V$, let $w(A, B) = \{w_e \mid e = (u, v), u \in A, v \in B\}$ be the total edge-weight between A and B. We define an uncrossing operation for two intersecting sets A and B as follows. If $w(A \cap B, A \setminus B) < w(A \cap B, B \setminus A)$, we let $A' \leftarrow A \setminus B$ and $B' \leftarrow B$, else we let $A' \leftarrow A$ and $B' \leftarrow B \setminus A$. Note that we have: $w(A') \leq w(A), w(B') \leq w(B), w(\delta(A')) \leq w(\delta(A)), w(\delta(B')) \leq w(\delta(B))$, and $A' \cap B' = \emptyset$.

We apply the above uncrossing operation to $S_1, \ldots, S_k$ systematically, obtaining $U_1, \ldots, U_k$ as follows. We first let $U_1 = S_1$ and make it disjoint from $S_2, \ldots, S_k$ in that order. Then we let $U_2 = S_2$ and repeat. In the end, we have sets U_i with the desired properties. ∎

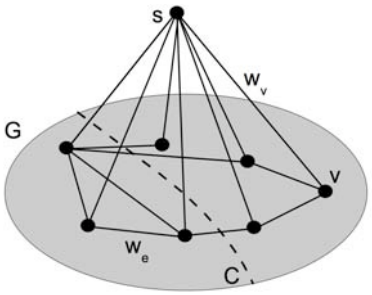

Figure 1: Construction of graph G'. The capacity of the cut $(C, (V \setminus C) \cup \{s\})$ in G' is $\texttt{size}(C)$.

LEMMA 2. *For any $v \in V$, in polynomial-time, we can find a cluster $S_v \subset V$ such that $v \in S_v$, $|S_v \cap T| \leq 1$, and $\texttt{size}(S_v) \leq B$.*

PROOF. Since BSGC is feasible, for any $v \in V$, the cluster S_v^* in a feasible clustering satisfies the above conditions.

If v is a terminal, we find a minimum cut in G' that separates v from $(T \setminus \{v\}) \cup \{s\}$ by doing a single min-cut computation.‡ Let S_v denote the vertices on the v-side of this cut. From the minimality of the cut, we have $\texttt{size}(S_v) \leq \texttt{size}(S_v^*) \leq B$.

If v is not a terminal, we try all possible values of $S_v^* \cap T$. It can either be empty or a singleton set containing a terminal. If $S_v^* \cap T = \emptyset$, we can find a minimum cut in G' that separates v from $T \cup \{s\}$. On the other hand, if $S_v^* \cap T = \{t\}$, we can find a minimum cut in G' that separates $\{v, t\}$ from $(T \setminus \{t\}) \cup \{s\}$. In either case, we can find S_v satisfying the desired properties. ∎

We can now find a feasible clustering in polynomial-time as follows.

1. Compute clusters S_v for all v satisfying the conditions in Lemma 2.
2. Systematically uncross clusters S_v to make them disjoint using Lemma 1.

2.1 Generalizations to symmetric submodular functions

A function $f : 2^V \to \Re_+$ is called *submodular* if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ holds for all $A, B \subseteq V$, and it is called *symmetric* if $f(A) = f(V \setminus A)$ holds for all $A \subseteq V$. For an undirected graph $G = (V, E)$ with edges weights w_e, the function $f(S) = w(\delta(S))$ for $S \subseteq V$ is symmetric and submodular. We can extend the results for the feasibility version of the problem to general symmetric submodular functions. The feasibility version of the bounded size clustering problem for symmetric submodular function is defined as follows. Given a symmetric submodular function $f : 2^V \to \Re_+$, a weight function $w : V \to \Re_+$, a set of terminals $T \subseteq V$, and a budget B, find a partitioning of V into clusters such that for each cluster $S \subseteq V$ we have $|S \cap T| \leq 1$ and $\texttt{size}(S) = \sum_{v \in S} w_v + f(S) \leq B$.

We now briefly outline how Lemmas 1 and 2, and hence our algorithm for the feasibility version, can be generalized to symmetric submodular functions. The generalization of the

‡This can be done by shrinking $(T \setminus \{v\}) \cup \{s\}$ into a super-vertex s' (or alternately adding very high weight edges between vertices in $(T \setminus \{v\}) \cup \{s\}$) and finding a min-cut separating v and s'.

proof of Lemma 1 follows from the observation that for any two sets $A, B \subseteq V$, we have $f(A \setminus B) + f(B \setminus A) = f(A \cap \overline{B}) + f(B \cap \overline{A}) = f(A \cap \overline{B}) + f(\overline{B} \cup A) \leq f(A) + f(\overline{B}) = f(A) + f(B)$. Thus either $f(A \setminus B) \leq f(A)$ or $f(B \setminus A) \leq f(B)$ holds.

To generalize the proof of Lemma 2, we introduce a new element s to the ground set V and define a symmetric submodular function $g : V \cup \{s\} \to \Re_+$ as

$$g(A) = \begin{cases} \sum_{v \in A} w_v, & \text{if } s \notin A, \\ \sum_{v \notin A} w_v, & \text{if } s \in A. \end{cases}$$

The function g corresponds to adding edges of weight w_v between s and $v \in V$. We also lift f from V to $V \cup \{s\}$ by defining $f(A) = f(V \cap A)$ for $A \subseteq V \cup \{s\}$. It is easy to see that for any $A \subseteq V$, we have $\texttt{size}(A) = f(A) + g(A)$. Now note that a set separating two subsets $A_1, A_2 \subset V$ of elements that minimizes the symmetric submodular function $f + g$ can be computed by "merging" the elements A_1 (respectively, A_2) into a super-element a_1 (respectively, a_2) and using standard algorithms for symmetric submodular function minimization [6] to separate elements a_1 and a_2. The proof of Lemma 2 thus holds for symmetric submodular functions as well.

3 The optimization version

3.1 NP-hardness

We present an approximation preserving reduction from the $\frac{1}{3}$-balanced cut problem, which is NP-hard, to the BSGC problem with $T = \emptyset$. The $\frac{1}{3}$-balanced cut problem is defined as follows: given undirected graph $G = (V, E)$ on n vertices with vertex weights $w_v \geq 0$ and edge weights $w_e \geq 0$, partition the vertices into two non-empty clusters $S \subset V$ and $V \setminus S$ such that $\min\{w(S), w(V \setminus S)\} \geq \frac{1}{3}w(V)$ and $w(\delta(S))$ is minimized. This problem is NP-hard [4] and the best-known approximation for this problem that does not violate* the balance constraint is $O(\log n)$ [10].

LEMMA 3. *If there is a ρ-approximation for the* BSGC *problem with $T = \emptyset$, there is a ρ-approximation to the $\frac{1}{3}$-balanced cut problem.*

PROOF. Given an instance (G, w) of the $\frac{1}{3}$-balanced cut problem, we create an instance of the BSGC problem as follows. We scale the vertex and edge weights so that $1 = \min_{v \in V} w_v > 2\sum_{e \in E} w_e$ and let $B = \frac{2}{3}w(V) + \frac{1}{2}$ and $T = \emptyset$. We then compute a ρ-approximation for the BSGC problem. We can assume, without loss of generality, that the output consists of *exactly* two clusters, as follows. As long as we have at least three clusters, say S_1, S_2, S_3 with $w(S_1) \leq w(S_2) \leq w(S_3)$, we can merge S_1 and S_2 into a single cluster without violating the budget constraint. This merge does not increase the total edge-weight across the clusters. Since $\min_v w_v = 1$, it is now easy to see that the resulting two clusters, say $\tilde{S}_1$ and $\tilde{S}_2 = V \setminus \tilde{S}_1$, satisfy the balance condition and form a ρ-approximation for the $\frac{1}{3}$-balanced cut problem. ∎

3.2 The algorithm

In this section, we show how to find $\{S_1, \ldots, S_k\}$ such that $|S_i \cap T| \leq 1$ and $w(\delta(S_i)) \leq O(\log n) \cdot (B - w(S_i))$ for all i such that the total cut-weight is $O(\log^2 n)$ times the optimum

1. Initialize $U \leftarrow V$ be to the set of not-yet-covered vertices.
2. Initialize the set of clusters $\mathcal{S} \leftarrow \emptyset$.
3. While $U \neq \emptyset$ do:
   (a) Find an approximately valid set $S \subseteq V$ such that
   $$\frac{w(\delta(S))}{|S \cap U|} \leq (c \log n) \cdot \frac{2\text{OPT}}{|U|}.$$
   (b) Add S to $\mathcal{S}$.
   (c) Let $U \leftarrow U \setminus S$.
4. Uncross the clusters in $\mathcal{S}$.

Figure 2: Algorithm for BSGC

cut-weight. We think of BSGC as a set-cover problem. The elements to be covered are the vertices and the sets are "valid" subsets of V.

DEFINITION 4. *A subset $S \subseteq V$ is called* valid *if $|S \cap T| \leq 1$ and $w(S) + w(\delta(S)) \leq B$. A subset $S \subseteq V$ is called* approximately valid *if $|S \cap T| \leq 1$ and*

$$(c \log n) \cdot w(S) + w(\delta(S)) \leq (c \log n) \cdot B$$

holds, where $c > 0$ is an absolute constant, the value of which will be fixed later.

Let the cost of S be $w(\delta(S))$. Clearly the optimum covers all the elements using only valid subsets. Let OPT denote the cost of this optimum set cover. Note that the number of sets is exponential in general. However the greedy set cover algorithm only needs the following oracle: given a subset $U \subseteq V$ of "yet to be covered" vertices, find a valid set S that minimizes the ratio $\frac{w(\delta(S))}{|S \cap U|}$. Unfortunately, even this oracle is NP-hard, and hence we use an approximation for the oracle. Our algorithm, given in Figure 2, picks approximately valid subsets one by one to cover all the vertices. Then, using Lemma 1, it uncrosses the clusters to make them disjoint.

Finding a minimum ratio approximately valid set

LEMMA 5. *Given a non-empty subset $U \subseteq V$, we can find in polynomial time an approximately valid subset $S \subseteq V$ such that*

$$\frac{w(\delta(S))}{|S \cap U|} \leq (c \log n) \cdot \frac{2\text{OPT}}{|U|}.$$

It is easy to see that this lemma combined with the analysis of the greedy set-cover algorithm yields our result.

Proof of Lemma 5. We argue that there exists a valid subset $S^* \subseteq V$ such that $\frac{w(\delta(S^*))}{|S^* \cap U|} \leq \frac{2\text{OPT}}{|U|}$. Consider the optimum clustering $\{S_i^*\}$. Note that

$$\min_i \frac{w(\delta(S_i^*))}{|S_i^* \cap U|} \leq \frac{\sum_i w(\delta(S_i^*))}{\sum_i |S_i^* \cap U|} = \frac{2\text{OPT}}{|U|}.$$

Thus the cluster S_i^* that minimizes the ratio $\frac{w(\delta(S_i^*))}{|S_i^* \cap U|}$ is a candidate set.

We next use the following tree decomposition result of Räcke [10]. Given an edge-weighted undirected graph $G = (V, E)$, a tree decomposition $\mathcal{T}$ is an edge-weighted rooted tree which has a one-to-one correspondence between the vertices V and the leaves of $\mathcal{T}$.

THEOREM 6. *[Räcke [10]] There exists a probability distribution on polynomially many tree decompositions $\mathcal{T}$ such that for all sets $S \subseteq V$ and all $\mathcal{T}$, we have $w(\delta(S)) \leq w_{\mathcal{T}}(\delta(S))$ and*

$$\mathbb{E}_{\mathcal{T}}[w_{\mathcal{T}}(\delta(S))] \leq (c \log n) \cdot w(\delta(S))$$

for an absolute constant $c > 0$. Here $w_{\mathcal{T}}(\delta(S))$ denotes the minimum cut in $\mathcal{T}$ that separates leaves in S from the other leaves. Moreover such a distribution and tree decompositions can be found in polynomial time.

Let the constant $c > 0$ be as given in Theorem 6. Our algorithm first computes the tree decompositions given in Theorem 6 and assigns a weight of w_v to each leaf corresponding to vertex v in each of these tree decompositions. From Theorem 6 and an averaging argument, there exists a tree decomposition, say $\mathcal{T}^*$, in this collection such that

$$\frac{w_{\mathcal{T}^*}(\delta(S^*))}{|S^* \cap U|} \leq (c \log n) \cdot \frac{\text{OPT}}{|U|} \quad \text{and} \quad w_{\mathcal{T}^*}(\delta(S^*)) \leq (c \log n) \cdot (B - w(S^*)).$$

Of course, we do not know which of the polynomially many tree decompositions $\mathcal{T}^*$ corresponds to a-priori. Therefore our algorithm tries each of these tree decompositions $\mathcal{T}$ and computes the set S, if it exists, such that

$$|S \cap T| \leq 1 \quad \text{and} \quad w_{\mathcal{T}}(\delta(S)) \leq (c \log n) \cdot (B - w(S)) \tag{1}$$

holds and such that

$$\frac{w_{\mathcal{T}}(\delta(S))}{|S \cap U|} \tag{2}$$

is minimized. Finally, it outputs the set computed in this manner with the minimum ratio (2).

Now fix a tree decomposition $\mathcal{T}$. In order to compute a set S satisfying (1) with the minimum ratio (2), the algorithm runs the following dynamic program. For each value of $k \in \{1, \ldots, |U|\}$ and each possible weight $w \leq B$, it computes S, if it exists, such that $w(S) = w$, $|S \cap T| \leq 1$, $|S \cap U| = k$, and $w_{\mathcal{T}}(\delta(S))$ is minimized. If $w_{\mathcal{T}}(\delta(S)) \leq (c \log n) \cdot (B - w)$ holds, it stores the set S as a candidate set. In the end, it outputs the candidate set with minimum ratio (2).

The dynamic program. To this end, using standard scaling techniques we assume that the vertex and edge weights in $\mathcal{T}$ are polynomially bounded in n. More precisely, we can assume without loss of generality that $w_v \leq B$ for all $v \in V$; otherwise no feasible clustering exists. Next we shrink all the edges $e \in E$ with $w_e > B$, since such edges cannot cross clusters in a feasible clustering. Furthermore, for all v such that $w_v \leq B/n$, we set $w_v = 0$, and for all e such that $w_e \leq B/n^2$, we set $w_e = 0$. In doing so, we can only violate the budget by an extra constant factor. By scaling if necessary, we assume that the vertex and edge weights and the budget B are non-negative integers. We also assume for simplicity that $c \log n$ is an integer.

We can assume, without loss of generality, that $\mathcal{T}$ is a *binary* tree. If some internal node v has $l > 2$ children, we can replace v by a binary tree with l leaves and attach the l children to the l leaves one-to-one. We also give a cost of $1 + (c \log n) \cdot B$ to the edges of this new binary tree. Since $w_{\mathcal{T}}(\delta(S^*)) \leq (c \log n) \cdot B$, no edge of such a high cost will be present in the cut $w_{\mathcal{T}}(\delta(S))$ output by the dynamic program. Thus, computing the desired set S in the original tree is equivalent to computing S in the transformed binary tree.

For a node $v \in \mathcal{T}$, let $\mathcal{T}_v$ denote the subtree hanging from node v (including node v). Now for each node $v \in \mathcal{T}$, our dynamic program builds the following table. For each $I \subset T$ with $|I| \leq 1$, integer weights $w \leq B$ and $w_1, w_2 \leq (c \log n) \cdot B$, and an integer $k \leq |U|$, we store a subset $S[v, I, w, w_1, w_2, k]$ of the leaves in $\mathcal{T}_v$, if it exists, such that

1. $S \cap T = I$,
2. $w(S) = w$,
3. the minimum cut in $\mathcal{T}_v$ separating S from the remaining leaves in $\mathcal{T}_v$ has weight w_1,
4. the minimum cut in $\mathcal{T}_v$ separating S from the remaining leaves in $\mathcal{T}_v$ *as well as* v has weight w_2, and
5. $|S \cap U| = k$.

Observe that a cut separating S from the remaining leaves in $\mathcal{T}_v$ may contain node v on either side of the cut. Therefore, $w_1 \leq w_2$. It is easy to see that this table is of polynomial size. The final output of the dynamic program is computed as follows: among all possible sets $S[r, I, w, w_1, w_2, k]$, where r is the root of $\mathcal{T}$, output a set satisfying (1) that minimizes the ratio (2).

We next show how to compute this table in bottom-up fashion in polynomial time. If v is a leaf node, the table has no entries. For internal nodes v that have leaf nodes as its children, it is easy to compute such a table. For all other internal nodes v, let p and q be its children and assume that such tables are already computed for nodes p and q. Let w_{vp} (resp. w_{vq}) denote the weight of edge (v, p) (resp. (v, q)) in $\mathcal{T}$.

Given values of (I, w, w_1, w_2, k), we find disjoint sets

$$S^p = S[p, I^p, w^p, w_1^p, w_2^p, k^p]$$

and

$$S^q = S[q, I^q, w^q, w_1^q, w_2^q, k^q]$$

if they exist, for all possible decompositions $I = I^p \cup I^q$, $w = w^p + w^q$, and $k = k^p + k^q$, and all possible choices of $w_1^p, w_1^q, w_2^p, w_2^q$, such that the following conditions hold:

$$w_1 = \min\{w_1^p + w_1^q + \min\{w_{vp}, w_{vq}\}, w_2^p + w_1^q, w_1^p + w_2^q\}, \tag{3}$$

and

$$w_2 = \min\{w_2^p, w_1^p + w_{vp}\} + \min\{w_2^q, w_1^q + w_{vq}\}. \quad (4)$$

Note that the expression (3) considers all possible ways of realizing a cut with weight w_1 that separates S from the remaining leaves in $\mathcal{T}_v$. In fact, the three terms correspond to whether nodes p and q are on the same side of the cut as sets S^p and S^q, respectively. Similarly, the expression (4) considers all possible ways of realizing a cut with weight w_2 that separates S from the remaining leaves in $\mathcal{T}_v$ as well as v.

If there exist such sets S^p and S^q for any such decomposition, ties broken arbitrarily, we store $S = S^p \cup S^q$ as the entry $S[v, I, w, w_1, w_2, k]$. Otherwise we leave the entry empty. The correctness and the polynomial size of the dynamic program follows easily.

4 Conclusions

A consequence of our work is that the min-max multiway cut problem becomes polynomial-time solvable if there are allowed to be clusters *without* terminals. This raises a question of whether other graph problems become similarly easier if the number of clusters is not specified as part of the input. Our work also introduces many interesting open questions. Since the feasibility version of BSGC is solvable in polynomial time, can one approximate BSGC, say within a poly-logarithmic factor, *without* violating the budget constraint? In stream processing applications, it is often important to find a clustering to minimize the maximum *latency* of a path taken by a data stream, where an edge on a path contributes to the latency only if it goes between two clusters. Studying the approximability of this problem is an important research direction.

Acknowledgments

We would like to thank Nikhil Bansal for useful discussions.

References

[1] System S stream computing system. http://www-01.ibm.com/software/data/infosphere/streams.

[2] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings, Symposium on Theory of Computing (STOC)*, pages 222–231, 2004.

[3] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004.

[4] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Inform. Process. Lett.*, 42:153–159, 1992.

[5] Roee Engelberg, Jochen Könemann, Stefano Leonardi, and Joseph Naor. Cut problems in graphs with a budget constraint. *J. Discrete Algorithms*, 5(2):262–279, 2007.

[6] S. Fujishige. *Submodular Functions and Optimization*. North-Holland, 1991.

[7] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. SPADE: The System S declarative stream processing engine. In *Proceedings, ACM SIGMOD Conference*, 2008.

[8] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *J. Soc. Indust. Appl. Math.*, 9(4):551–570, 1961.

[9] R. Khandekar, K. Hildrum, S. Parekh, D. Rajan, J. Wolf, K.-L. Wu, H. Andrade, and B. Gedik. COLA: Optimizing stream processing applications via graph partitioning. In *Proceedings, Middleware Conference*, 2009.

[10] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings, Symposium on Theory of Computing (STOC)*, pages 255–264, 2008.

[11] Zoya Svitkina and Éva Tardos. Min-max multiway cut. In *Proceedings, Approx-Random*, pages 207–218, 2004.