On the Complexity of Finding an Unknown Cut Via Vertex Queries

Peyman Afshani, Ehsan Chiniforooshan, Reza Dorrigiv, Arash Farzan, Mehdi Mirzazadeh, Narges Simjour, and Hamid Zarrabi-Zadeh

School of Computer Science, University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada

Email: {pafshani,echinifo,rdorrigiv,afarzan, mmirzaza,nsimjour,hzarrabi}@cs.uwaterloo.ca

Abstract. We investigate the problem of finding an unknown cut through querying vertices of a graph G. Our complexity measure is the number of submitted queries. To avoid some worst cases, we make a few assumptions which allow us to obtain an algorithm with the worst case query complexity of $O(k) + 2k \log \frac{n}{k}$ in which k is the number of vertices adjacent to cut-edges. We also provide a matching lowerbound and then prove if G is a tree our algorithm can asymptotically achieve the information theoretic lowerbound on the query complexity. Finally, we show it is possible to remove our extra assumptions but achieve an approximate solution.

1 Introduction

Consider a graph G together with a partition of its set of vertices, $V(G)$, into two sets, A and B. Here, we study the problem of finding the sets A and B by only asking queries about the vertices of G. In other words, the algorithm has only access to the graph G and an oracle which given a vertex v will tell the algorithm whether $v \in A$ or $v \in B$. Although we study this problem from a theoretical point of view, we can establish connections to the existing concepts and problems studied in machine learning.

In the standard learning problems, the learner is given a collection of labeled data items, which is called the *training data*. The learner is required to find a "hypothesis", using the training data and thus predict the labels of all (or most of) the data items, even those not seen by the learner algorithm. In this context, labeling the data points is considered to be an expensive operation. Thus, reducing the size of the training data is one of the important objectives. Semi-supervised learning attempts to accomplish this by using additional information about the whole data set. Recently, new models for semi-supervised learning have emerged which use spectral or graph techniques. We can name the work of Blum et al. [1], in which they built a graph and proposed several strategies to weigh the edges of the graph and proved finding a minimum cut corresponds to several of the previously employed learning algorithms based on Random Markov Fields. This is supported by the fact that it is common to restrict the set of possible labels to $\{+,-\}$ [7] which implies a cut in graph G. We need to mention that spectral clustering techniques (for instance [9, 8]) which are closely related to cuts have also been used in context of learning (for instance see [3, 5]). We refer the reader to a line of papers in this area [1, 10, 4, 2].

Other concepts related to the problem studied here are *Query learning* and *active learning*. Basically, under these assumptions the learner algorithm is allowed to interactively ask for the label of any data item. Thus, we can claim the problem studied in this paper has strong connection to the existing topics in machine learning; in short, our problem can be described as actively learning an unknown cut in a graph G.

In the next section, we define the problem precisely and obtain a simple lower bound on the number of queries. Then, in Section 3 we develop an algorithm that can solve a stronger version of our problem. We prove that the number of queries needed by our algorithm matches the lower bound. We discuss the problem for trees as a special family of graphs in Section 4. Finally, we relax the balancedness assumption and develop an ϵ-approximation algorithm instead of an exact algorithm in Section 5.

2 Preliminaries

Given a graph G, here we choose to represent the cut using a *labeling* $l : V(G) \rightarrow \{+,-\}$ which is the assignment of $+$ or $-$ to the vertices of G. Our goal is to design an algorithm which through querying labels of vertices can detect *all* the cut-edges. Clearly, the challenge is to minimize the number of queries or otherwise n queries can trivially solve the problem. Thus, we measure the complexity of the problem by the number of submitted queries and the parameters involved are the number of vertices, n, number of edges in the cut, k, and number of vertices adjacent to cut-edges, k'.

Notice that if graph G is a k-regular graph and all the vertices except one vertex v are labeled $+$, then it is easy to see that the algorithm must perform n queries in the worst case to find the single vertex v. This (the unbalanced cut having undesirable properties) is a common phenomenon which also appears in the spectral and clustering techniques. For instance, the definitions of normalized cut and ratio cut both factor in a form of balancedness condition. Here, we require a different form of balancedness: suppose we remove all the cut-edges in the graph. We call a labeling of a graph α-*balanced*, if each connected component in the new graph has at least αn vertices. Now we formally define our first problem:

Definition 1 (Problem A). *Suppose a graph G with an unknown α-balanced labeling of cut-size k is given. Use the structure of G together with the value of α to find this labeling using a small number of queries.*

Now, we show how to reduce the above problem to a seemingly easier problem by a probabilistic reduction. For a cut $\mathcal{C}$, define $G \setminus \mathcal{C}$ as the graph constructed from G by deleting all cut-edges of $\mathcal{C}$. Given a graph G, a *hint-set* is a set S of vertices of G such that S has at least one vertex from each connected component of $G \setminus \mathcal{C}$.

Definition 2 (Problem B). *Given an input graph G and a hint-set for an unknown labeling of G, find the labeling using a small number of queries.*

To show that Problem A can be reduced to Problem B, we select $\frac{c \log n}{\alpha}$ vertices uniformly at random and query the labels of the selected vertices. Since a connected component C contains at least αn vertices, the probability that a randomly selected vertex is not in C is at most $1 - \alpha$. Hence, the probability that all selected vertices fall outside C is at most

$$(1-\alpha)^{\frac{c \log n}{\alpha}} \leq ((1-\alpha)^{\frac{1}{\alpha}})^{c \log n} \leq (\frac{1}{n})^{c}$$

The number of connected components is α^{-1} thus, with high probability we have obtained a hint-set and reduced the Problem A to Problem B.

In addition to this probabilistic reduction, a non-probabilistic one, though with an exponential running time, can also be proposed for the situation in which an upper bound k on the number of cut-edges of the labeling is given. As follows from a theorem proved by Kleinberg [6], for any graph G, parameter α, and integer $k < \frac{\alpha^2}{20}|V(G)|$, any subset of size $O(\frac{1}{\alpha} \log \frac{2}{\alpha})$, with probability at least $\frac{1}{2}$, is a hint-set for all α-balanced cuts of G with at most k cut-edges. So, we may examine each subset of vertices of G against every α-balanced cut of G with at most k cut-edges to find a subset that is a hint-set for all possible input labellings; then we will have an equivalent instance of problem B.

2.1 The Lower Bound

The balancedness condition prevents the problem from having a huge and trivial lowerbound. The idea behind our lowerbound is to construct a large number of balanced cuts on a fixed hint-set S.

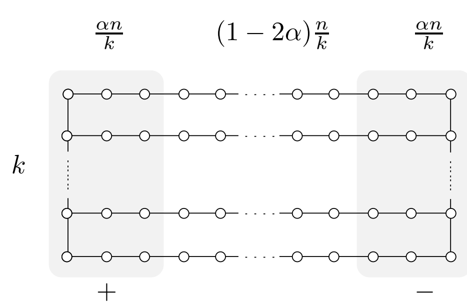

Fig. 1. Construction of the lowerbound.

Lemma 1. *Treating α as a constant, any algorithm that solves Problem A or B needs $k \log \frac{n}{k} - O(k)$ queries in the worst case, where n is the size of the graph and k is the cut-size of the labeling.*

Proof. Let G consist of k paths, each of length n/k, connected to each other through their endpoints as depicted in Figure 1. Suppose that the vertices in the first (respectively, the last) α portion of each path are labeled with $+$ (respectively, with $-$). The cut we are looking for is formed by k edges from the middle $(1 - 2\alpha)$ portion of the paths, one from each path. There are $\left((1-2\alpha)(\frac{n}{k})\right)^k$ choices for selecting these edges. Thus, any algorithm for finding the cut in this graph needs $\log((1-2\alpha)\frac{n}{k})^k = k \log \frac{n}{k} - O(k)$ queries in the worst case. □

3 An Optimal Algorithm

First we examine a very special case in which the graph G is a path and the cut is a single edge. This special case will come handy in the solution for the general case.

3.1 Algorithm for Paths with Cut-size One

In this case the cut essentially is just one edge and all vertices to one side of it are all labeled $+$ and the vertices on the other side are all labeled $-$. The solution is direct and is similar to the binary search algorithm. We start from both ends and query the labels of the endpoints. These will have opposite labels. Then we query the label of the midpoint and depending on the answer our search will be confided to one side of the path. We continue this binary search which eventually will find the cut using $O(\log n)$ queries.

Notice this binary search approach can still be used to find a cut-edge provided we start from two vertices with opposite labels.

3.2 Algorithm for Balanced Cuts

In this section, we develop an algorithm which matches the lowerbound proved in Section 2.1. We also focus on Problem B and assume a hint set $S = \{v_1, \ldots, v_c\}$ is given. First, the algorithm uses c queries to find out labels of vertices in S. It then computes a sequence $G_0 = G, G_1, \ldots, G_k = G \setminus \mathcal{C}$ of subgraphs of G where, for $1 \leq i \leq k$, $G_i = G_{i-1} - e_i$ for a cut-edge e_i of $\mathcal{C}$. To find a cut-edge e_i in G_{i-1}, it selects vertices u and v of S that are in the same connected component of G_i but have different labels. Then a "binary search" on a path between u and v in G_i is used to find a cut-edge. If computed naively, this path can have $\Omega(n)$ nodes, causing the algorithm to perform too many queries ($\Omega(\log n)$ in the worst case) at each step. Thus this naive approach will only result in the bound $O(k \log n)$. Although this bound of $O(k \log n)$ seems efficient, it still does not match the lowerbound in the previous section. To obtain a matching upperbound, we need one last ingredient: "domination sets" of G_i.

Definition 3. *In a connected graph G, a set of vertices R is an* r-domination set*, if the distance of any vertex in G to the set R is at most r. We use $f(r)$ to denote the size of an r-domination set with the least number of vertices among all r-domination sets of G.*

The next two lemma shows how domination sets are used to find a path of length at most $2r + 1$ connecting two vertices with different labels in G_{i-1}.

Lemma 2. *Given a graph G and an r-domination set R of G, for every two vertices u and v of the same connected component of G, one can construct a walk W from u to v such that every $2r + 1$ consecutive vertices in W contain at least one vertex from R.*

Proof. Let $P = (u = p_1, p_2, \ldots, p_l = v)$ be an arbitrary path between u and v in G. Since R is an r-domination, for each vertex $p_i \in P$ there is a vertex $r_i \in R$ such that there is a path P_i of length at most r connecting p_i to r_i. We define P'_i to be the reverse of P_i, for all $1 \leq i \leq l$, and W to be the walk $(P_1, P'_1, P_2, P'_2, \ldots, P_l, P'_l)$. Intuitively, the walk W starts walking along the vertices in P and at each vertex p_i, it first goes to r_i and then returns to p_i. So, each segment of size $2r + 1$ of W contains at least one vertex from R. □

Lemma 3. *Suppose l is a labeling of a graph G, R is an r-domination set in G, and u_1 and u_2 are vertices with $l(u_1) \neq l(u_2)$. There are vertices v_1 and v_2 of distance at most $2r + 1$ in $R \cup \{u_1, u_2\}$ such that $l(v_1) \neq l(v_2)$.*

Proof. Consider the walk W form u_1 to u_2 described in Lemma 2 and let $r_1, r_2, \ldots, r_p$ be the vertices of R appearing in W in order from u_1 to u_2. Also, define $r_0 = u_1$ and $r_{p+1} = u_2$. By Lemma 2, there are at most $2r$ vertices in W between r_i and r_{i+1}, exclusive, for $0 \leq i \leq p$. Since $l(r_0) \neq l(r_{p+1})$, there is an $0 \leq i \leq p$ such that $l(r_i) \neq l(r_{i+1})$. The correctness of the lemma follows by setting $v_1 = r_i$ and $v_2 = r_{i+1}$. □

Our algorithm will use the above two lemmas to iteratively find and extract the cut edges. Since at each step of the algorithm we will remove a cut edge, we must be able to update the r-domination set under edge deletions. Furtunately, this can be done trivially as the next lemma shows. We omit the prove since the proof is mostly intuitive.

Lemma 4. *If R is an r-domination set in G and $uv \in E$, then $R' = R \cup \{u, v\}$ is an r-domination set in $G' = G - uv$.*

The following property of r-domination sets allow the algorithm to asymptotically achieve the query complexity of the existing lower bound.

Lemma 5. *In any n-vertex connected graph, $f(r) \leq 2n/r$.*

Algorithm FindCutEdges(graph G, hint-set S, integers r and κ)

1. Find an r-domination set R for G using the greedy approach
2. Query labels of vertices in S and R
3. Set $C = \emptyset$
4. **while** *there are* u_1, u_2 *in* S *in the same component of* G *with* $l(u_1) \neq l(u_2)$ **do**
5. Find a path P between two vertices v_1 and v_2 in $R \cup \{u_1, u_2\}$ with length at most $2r + 1$ such that $l(v_1) \neq l(v_2)$
6. Use binary search to find an edge $e = wx$ of P such that $l(w) \neq l(x)$
7. Set $C = \mathcal{C} \cup e$, $G = G - e$, and $R = R \cup \{w, x\}$
8. **if** $|C| > \kappa$ **then** Fail;
9. Return C

Fig. 2. Algorithm for finding the cut-edges of an unknown labeling of a graph.

Proof. This upper bound can be achieved by a naive greedy algorithm. Start with one vertex as the initial domination set R and progressively add vertices to R in steps. In each step, find a vertex with distance more than r to R and add it to R (if there is no such vertex, we are finished). We claim $|R| \leq 2n/r$.

We define the *d-neighborhood* of a vertex u, denoted by $N_u(d)$, as the set of all vertices within distance d of u. If v and w are in the greedily-selected r-domination set R, then $N_v(r/2)$ and $N_w(r/2)$ have an empty intersection. Hence, there are $|R|$ $\frac{r}{2}$-neighborhoods formed around vertices of R that are pairwise disjoint. Since, there are at least $r/2$ vertices (including v) in $N_v(r/2)$, for any vertex v, $r|R|/2$ is a lower bound on the number of vertices n. This immediately implies that $|R| \leq 2n/r$. $\square$

The greedy algorithm above seems naive as it can be far from achieving the optimal value of $f(r)$. However we prove, in the next lemma, that the size of its output can be upper bounded by the function f in some way.

Lemma 6. *Suppose, on an input graph G, the greedy algorithm, as described in Lemma 5, comes up with an r-domination set of size $Greedy(r)$. Then,*

$$f(r) \leq Greedy(r) \leq f(r/2).$$

Proof. It is obvious that $f(r) \leq Greedy(r)$, as $f(r)$ is the minimum size of a domination set and $Greedy(r)$ is the size one such set. To prove $Greedy(r) \leq f(r/2)$, it is sufficient to consider the $\frac{r}{2}$-neighborhoods formed around the selected vertices in Lemma 5. For a set to be an eligible $\frac{r}{2}$-domination set, it has to include at least one vertex from each of these $\frac{r}{2}$-neighborhoods; otherwise the center would be at a distance more than $r/2$ from all the vertices in the domination set. Since these neighborhoods are pairwise disjoint (see the proof of Lemma 5), the size of any eligible $\frac{r}{2}$-domination set must be at least the number of $\frac{r}{2}$-neighborhoods which is exactly $Greedy(r)$. Hence, $f(r/2) \geq Greedy(r)$. $\square$

Consider the algorithm shown in Figure 2. It accepts additional parameters r and κ where κ is enforced to be an upper-bound on the cut-size. The algorithm starts by constructing an r-domination set R of G. Then, it performs $|R| + |S|$ queries to find out the labels of the vertices in R and in S. The algorithm runs in at most κ steps and in the i-th step it constructs the graph G_i. We use G_i, R_i, and C_i, respectively, to denote values of variables G, R, and C at the beginning of the i-th iteration of the while loop ($i \geq 0$).

At the beginning of the iteration i, we have a partially computed cut C_i, a graph $G_i = G \setminus C_i$, and an r-domination set R_i for G_i. Also, the algorithm knows the labels of vertices in S and R_i. Next, it uses Lemma 3 to find a path P of length at most $2r+1$ between two vertices with different labels. Hence, the algorithm uses at most $\lceil \log{(2r+1)} \rceil$ queries to find a new cut-edge e in G_i. The edge e is removed from the graph and the r-domination set is updated based on Lemma 4. If $\kappa \geq k$, removing all edges of $\mathcal{C}$ one by one in this way, the algorithm finds the solution using $|S| + \mathit{Greedy}(r) + \kappa \log r$ queries. The algorithm fails, when $\kappa < k$, before submitting more than $|S| + \mathit{Greedy}(r) + \kappa \log r$ queries.

Notice that we can compute $\mathit{Greedy}(r)$ for different values of r in advance and choose the value which minimizes the query complexity. Then, the number of queries will be at most $|S| + O(\kappa) + \min_r\{\mathit{Greedy}(r) + \kappa \log r\}$ which is at most

$$|S| + O(\kappa) + \min_r\{f(r/2) + \kappa \log r\} = |S| + O(\kappa) + \min_r\{f(r) + \kappa \log r\}$$

according to Lemma 6.

Let U be the set of endpoints of all cut-edges. We can further reduce the number of queries by noticing that every edge between a vertex u with positive label and a vertex v with negative label must be a cut-edge. Once we remove all such trivial cut-edges, the next step of the algorithm will find a new vertex $v \in U$. This implies we can bound the number of steps by $k' = |U|$. The final theorem is as follows:

Theorem 1. *For a graph G, a labeling l, and an integer κ, one can use $|S| + O(\kappa) + \min_r\{f(r) + \kappa \log r)\}$ queries to discover if $\kappa < k'$ and to solve the problem B when $\kappa \geq k'$, where S is a hint-set for l.*

According to Lemma 5, $f(2n/\kappa) \leq \kappa$. Thus, if we run the algorithm of Theorem 1 with parameters $\kappa = 1, \kappa = 2, \kappa = 4, \ldots$ and $r = 2n/\kappa$, until κ becomes as large as k', we get the following theorem.

Corollary 1. *For a graph G and a labeling l, the problem B can be solved using $|S| + O(k') + 2k' \log \frac{n}{k'}$ queries, where k' is the number of vertices adjacent to cut-edges and S is a hint-set for l.*

Finally, we must mention that our probabilistic reduction of problem A to problem B used $\frac{c \log n}{\alpha}$ queries which is always asymptotically smaller than the above query complexity and thus we can safely omit this term.

4 The Tightness of the Bounds

The result of the Corollary 1 is tight with respect to parameters n and k'. However, for specific graphs better bounds might be possible. Note that given a graph G and a hit-set S one available lowerbound is the logarithm of the number of cuts for which S is the hint-set. Clearly, this number depends entirely on the structure of graph G. Same can be said about the result of Theorem 1. For instance, if G is a full binary tree then we have $f(r) = \theta(\frac{n}{2^r})$ which means the query complexity of Theorem 1 is in fact $\kappa \log\log \frac{n}{\kappa} + O(\kappa)$ with a matching asymptotic lowerbound for this particular tree. In this section we generalize this observation for all trees. In other words, we prove if G is a tree then the result of Theorem 1 is asymptotically tight by providing a matching lowerbound which entirely depends on the structure of G and thus throughout this section we always assume G is a tree.

We use the output S of the greedy algorithm of Lemma 5 to construct many of labellings for G, all with the same hint-set S. Consider an arbitrary r and the r-domination set S returned by greedy algorithm. This special r-domination set has the property that for every $u, v \in S$, $dist(u, v) \geq r + 1$. We call any r-domination set with this property a *distributed r-domination set*. For a vertex $v \in S$ let N_v be the $\lceil \frac{r}{2} \rceil$-neighborhood of v. If $u \neq v$, then the edge-sets of the graphs induced by N_u and N_v do not intersect, that is, $E(G[N_u]) \cap E(G[N_v]) = \emptyset$. Suppose $v_0, \ldots v_{m-1}$ is an ordering of the vertices of S. We have assumed G is a tree and thus there is a unique path connecting v_i to v_j. Define P_{ij} to be the portion of this path which falls inside N_{v_i}. We have $|P_{ij}| \geq 1 + \lceil r/2 \rceil$, for every $0 \leq i < j \leq m-1$. The fact that $E(G[N_{v_{i_1}}]) \cap E(G[N_{v_{i_2}}]) = \emptyset$ implies that the edge sets of the paths $P_{i_1 j_1}$ and $P_{i_2 j_2}$ do not intersect, for $i_1 \neq i_2$. This results in the following lemma.

Lemma 7. *If S is a distributed r-domination set of size m in a tree G and $k < m$ be an arbitrary integer then, there are at least $\lceil \frac{r}{2} \rceil^k$ cuts each having k cut-edges such that S is a hint-set for every one of them.*

Proof. Consider the notation above and the following algorithm:

Set $C = \emptyset$
while $|C| < k$ **do**
- Choose the lexicographically smallest pair (i, j), $0 \leq i < j \leq m-1$ such that v_i and v_j are in the same component of $G \setminus C$.
- Nondeterministically select an edge of P_{ij} and add it to C

Since $k \leq m-1$, the selection of the pair (i, j) in each execution of the body of the while loop is feasible. As each path P_{ij} has $\lceil \frac{r}{2} \rceil$ edges and the body of the while loop is executed k times, there are $\lceil \frac{r}{2} \rceil^k$ possibilities, in overall, for nondeterministic choices of the algorithm. Moreover, each time that an edge is deleted from $G \setminus C$ and a connected component of $G \setminus C$ is split into two, each new connected component still includes a vertex of S (either v_i or v_j). Therefore, S is a hint-set for the cut specified by any set C generated by this algorithm.

It remains to show that all the $\lceil \frac{r}{2} \rceil^k$ sets C generated by the algorithm are distinct. Consider two different executions $\mathcal{E}_1$ and $\mathcal{E}_2$ of the algorithm and consider the first point that the algorithm makes different decisions in $\mathcal{E}_1$ and in $\mathcal{E}_2$. Suppose $\mathcal{E}_1$ chooses an edge e_1 of P_{ij} while $\mathcal{E}_2$ selects a different edge e_2 of P_{ij}. Without loss of generality, assume e_1 appears before e_2 in P_{ij}. The edge e_2 is in $\lceil \frac{r}{2} \rceil$-neighborhood of v_i and after adding e_1 to C in $\mathcal{E}_1$, e_2 is not in the same component as v_i anymore; so $\mathcal{E}_1$ will never add e_2 to C. Thus, the value of C in $\mathcal{E}_2$ will be different form the value of C in $\mathcal{E}_1$. So, each execution of the algorithm generates a distinct set of edges. □

Next theorem uses this set of cuts to give a lowerbound.

Theorem 2. *For every tree G with n vertices and integer $k > 0$, there is an integer r such that any algorithm solving problem B must perform $f(r)+k\log r-O(k)$ queries.*

Proof. Due to the term $-O(k)$, we can assume $k \leq \frac{n}{2}$. Define m_t as the maximum size of any distributed t-domination set. The size of the maximum independent set of a tree is at least $\frac{n}{2}$ which implies $m_1 \geq \frac{n}{2}$. As r increases, m_r must decrease, and in particular $m_n = 1$. We know $k \leq \lceil \frac{r}{2} \rceil$ and thus there is an integer $t > 1$ such that $f(t+1) \leq m_{t+1} \leq k < m_t$. According to Lemma 7 any algorithm solving problem B must perform $k \log \frac{t}{2}$ queries and a simple calculation shows that $f(t+1) + k\log(t+1) - O(k) \leq k \log t$ which means $f(r) + k\log r - O(k)$ is a lowerbound for the number of queries for $r = t+1$. □

5 Relaxing the Balancedness Assumption

In this section we show how to remove the balancedness assumption at the expense of obtaining an approximation algorithm. Thus, we present an ε-approximation algorithm that performs $k\ln(3/\varepsilon) + k\log(n/k) + O(k)$ queries on a given graph with n vertices and cut-size k and reports a labeling which with high probability has at most εn vertices mislabeled.

The algorithm is fundamentally same as before. We select a random set of vertices uniformly as the hint-set and perform Algorithm 2 just once for $\kappa = k$ and $r = n/k$. Nevertheless, the probabilistic analysis presented in Section 2 holds no more; the connected components here can be arbitrarily small so there could be components which do not contain any vertex from the hint set.

We call the components that have a vertex representative in the sample as the *represented* components and the rest as the *unrepresented*. Firstly, we claim that if we select $(k+1)\ln(\varepsilon/3)$ sample vertices, with high probability, the number of vertices in an unrepresented component is at most εn. The argument is probabilistic; we compute the expected number of vertices in unrepresented components. We denote by $n_1, \ldots, n_t$ the sizes of connected components $C_1, \ldots, C_t$

in the initial graph respectively. Using basic probability arguments, the probability that the component C_i is unrepresented is at most $\left(1-\frac{n_i}{n}\right)^{(k+1)\ln(\varepsilon/3)}$. Therefore, the expected total number of vertices in unrepresented components is

$$\mathbf{E}=\sum_{i=1}^{t} n_i\left(1-\frac{n_i}{n}\right)^{(k+1)\ln(\varepsilon/3)} \leq n\sum_{i=1}^{t}\frac{n_i}{n}e^{-\frac{n_i}{n}(k+1)\ln(\varepsilon/3)}.$$

As the function xe^{-cx} is convex for any fixed c, the maximum happens when $\frac{n_i}{n}$ are equal and thus $\mathbf{E}\leq n\left(t\frac{1}{t}e^{-\frac{1}{t}(k+1)\ln(\varepsilon/3)}\right) \leq n\varepsilon/3$. The last inequality is due to the fact that k cannot be less than $t-1$. By using Markov inequality, one can assert that the probability that there are more than εn vertices in unreported components is less than $1/3$.

Secondly, we observe that by a run of Algorithm 2, labels of vertices of represented components are reported correctly. This fact follows from the correctness of the algorithm, and the observation that it never reports an edge as a cut-edge if it is not indeed a cut-edge. The algorithm might miss some cut-edges in contrast to the previous setting, as some components are unrepresented, and so the labeling of such components can be reported arbitrarily.

Combining the two latter facts, we conclude that the total number of vertices whose labels are misreported is less than εn for an arbitrary small $\varepsilon < 1$.

6 Conclusion

In this paper we studied the query learning problem, while we assumed both labeled and unlabeled data were available to us and we had a similarity graph constructed on data items. The problem was discussed in the following two settings: We studied the case where we are given a hint set of vertices in which there exists at least one vertex from each connected component of the same label. We gave the lower bound $k\log n/k - O(k)$ on the number of queries required for this case. We provided an algorithm that finds the optimal hypothesis using at most $O(k\log n/k)$ queries, which matched our lower bound for general graphs. Hence, our proposed algorithm is optimal in terms of the number of queries.

In the second setting, we showed that if the labeling is α-balanced, i.e. each connected component of the vertices with the same label has at least α fraction of all vertices, we could find a hint set with high probability using $\frac{c\log n}{\alpha}$ random queries. Note that our previous lower bound works for the α-balanced graphs, too. This gives an evidence of the optimality of our algorithm for general graphs in the latter setting.

Although our algorithm for the first setting was proved to be optimal for general graphs, we may have different lower bounds for special families of graphs. In Section 4, we investigated the problem for trees and proved for tree our algorithm is asymptotically optimal. One natural question is whether it is possible to extend this result to other classes of graphs or not.

Finally, we considered the problem for non-balance cuts. We developed an ε-approximation algorithm for this case. However, we assumed that the number of cut-edges is given to the algorithm. Thus, the following problem remained open:

Open Problem: *Suppose that G is a graph, l is a labeling of G with k cut-edges, and $0 < \varepsilon \leq 1$ is a real number. Does there exist an ε-approximation algorithm that runs in polynomial time and submits at most $O(\mathrm{poly}(1/\varepsilon) k \log n/k)$ queries without knowing k in advance?*

References

1. A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 19–26. Morgan Kaufmann Publishers Inc., 2001.
2. A. Blum, J. Lafferty, M. R. Rwebangira, and R. Reddy. Semi-supervised learning using randomized mincuts. In *Proceedings of the twenty-first international conference on Machine learning*, page 13. ACM Press, 2004.
3. T. Joachims. Transductive learning via spectral graph partitioning. In *Twentieth International Conference on Machine Learning*, 2003.
4. T. Joachims. Transductive learning via spectral graph partitioning. In *Proceedings of the International Conference on Machine Learning*, pages 290–297, 2003.
5. S. Kamvar, D. Klein, and C. Manning. Spectral learning. In *International Joint Conference On Artificial Intelligence*, 2003.
6. J. Kleinberg. Detecting a network failure. In *Proceedings of the Forty-First Annual Symposium on Foundations of Computer Science*, page 231. IEEE Computer Society, 2000.
7. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
8. A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, 2001.
9. Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.
10. X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 912–919, 2003.