A New 3/2-Approximation Algorithm for the b-Edge Cover Problem

Arif Khan* Alex Pothen†

Abstract

We describe a 3/2-approximation algorithm, LSE, for computing a b-Edge Cover of minimum weight in a graph with weights on the edges. The b-Edge Cover problem is a generalization of the better-known *Edge Cover* problem in graphs, where the objective is to choose a subset C of edges in the graph such that at least a specified number $b(v)$ of edges in C are incident on each vertex v. In the weighted b-Edge Cover problem, we minimize the sum of the weights of the edges in C. We prove that the LSE algorithm computes the same b-Edge Cover as the one obtained by the Greedy algorithm for the problem. However, the Greedy algorithm requires edges to be sorted by their *effective weights*, and these weights need to be updated after each iteration. These requirements make the Greedy algorithm sequential and impractical for massive graphs. The LSE algorithm avoids the sorting step, and is amenable for parallelization. We implement the algorithm on a serial machine and compare its performance against a collection of approximation algorithms for the b-Edge Cover problem. Our results show that the LSE algorithm is $3\times$ to $5\times$ faster than the Greedy algorithm on a serial processor. The approximate edge covers obtained by the LSE algorithm have weights greater by at most 17% of the optimal weight for problems where we could compute the latter. We also investigate the relationship between the b-Edge Cover and the b-Matching problems, show that the latter has a faster implementation since edge weights are static in this algorithm, and obtain a heuristic solution for the former from the latter.

1 Introduction

We describe a new 3/2-approximation algorithm, LSE, for computing a b-Edge Cover of minimum weight in a graph, implement it on a serial machine and compare its performance against a collection of approximation algorithms for this problem. The b-Edge Cover problem is a generalization of the *Edge Cover* problem in graphs, where the objective is to choose a subset C of edges in the graph such that *at least* a specified number $b(v)$ of edges in C are incident on each vertex v, and subject to this restriction we minimize the sum of the weights of the edges in C. The LSE algorithm iteratively adds a set of locally sub-dominant edges to the current edge cover, where an edge (u, v) is locally sub-dominant if it has the minimum weight among all edges incident on its endpoints u and v.

The *Edge Cover* and b-Edge Cover problems are special cases of the *Set Cover* and *Set Multi-cover* problems, respectively. In the Set Multi-cover problem, we are given a universe N of n elements, and a family $\mathcal{F}$ of m sets, where each set in $\mathcal{F}$ is a subset of N. Associated with each set $F \in \mathcal{F}$ is a non-negative cost c_F, and the objective is to find a minimum cost sub-family $\mathcal{F}' \subseteq \mathcal{F}$ such that each element $i \in N$ is covered $b(i)$ times. In the case of the edge cover, each element corresponds to a vertex, each set corresponds to an edge with exactly two elements (vertices), and the cost of a set corresponds to the weight on an edge. Although both *Set Cover* and *Set Multi-cover* problems are NP-complete, the corresponding *Edge Cover* and b-Edge Cover problems can be solved in polynomial time. One context for application of the b-Edge Cover problem is in communication or distribution problems where reliability is important, i.e., each communication node has to be *"covered"* several times to increase reliability in the event of a communication link failing.

Our interest in the b-Edge Cover problem was stimulated by its application to the adaptive anonymity problem [2]. Here we are given a binary matrix consisting of values whose rows correspond to individuals and columns to features. The problem is to publish the data matrix while respecting the privacy preference of every individual. Each individual v wishes to be confused with at least $b(v) - 1$ others in the published data. Choromanski, Jebara and Tang [2] approximately solved this problem by means of a variational optimization algorithm, in each iteration of which they compute a b-Matching using an exact algorithm for the matching. In joint (unpublished) work with these authors, we have shown that the adaptive anonymity problem can be solved approximately using a b-Edge Cover formu-

*Computer Science, Purdue University, West Lafayette IN 47907 USA, (khan58@purdue.edu)

†Computer Science, Purdue University, West Lafayette, IN 47907 USA, (apothen@purdue.edu)

lation, with an approximation ratio of $(3\beta)/2$, where β is the maximum value of $b(v)$ over all vertices. An approximate b-MATCHING algorithm can also be employed here to obtain a heuristic solution that is nonetheless close to the approximate solution, and that in practice can be computed faster than with the b-EDGE COVER approach. Both approximation algorithms are one to three orders of magnitude faster than one that uses exact b-MATCHING; approximate b-MATCHING can also reduce the memory requirements of the adaptive anonmity problem from quadratic to linear in the number of individuals, thus enabling the solution of problems with a million individuals and hundreds of features.

There have been a few recent studies on minimum cardinality *Set Multi-cover*, a special case where each set has unit cost. Optimal algorithms for this problem requiring exponential time were proposed in [8, 15]. The fastest optimal algorithm that is based on a dynamic programming model was proposed by Hua et al. [9]. Let $H_n = 1 + 1/2 + \ldots + 1/n$ denote the n-th harmonic number. The greedy algorithm which iteratively adds the largest number of uncovered elements to a current solution was shown to be H_n-approximate by Johnson [10] and Lovasz [13]. Let a denote the maximum number of elements in a set. Berman et al. [1] described a randomized $(1 + \ln a)$-approximation algorithm for this problem when $b(i) = 1$, and considered an application of minimum cardinality *Set Multi-cover* in systems biology for reverse-engineering protein and gene networks. They also showed that as $b(i)$ increases, the algorithm asymptotically becomes a 2-approximation algorithm. Chvatal [3] extended the results of Johnson [10] and Lovasz [13] to the minimum cost *Set Cover* problem. A useful summary of other approximation algorithms for the minimum cost *Set Cover* problem is discussed by Rajagopalan and Vazirani [18].

An integer linear programming formulation for the minimum cost *Set Multi-cover* was given by Hall and Hochbaum [6]. These authors later proposed an a-approximation algorithm for this problem [7]. Dobson [5] extended Chvatal's results and proposed an H_a-approximation algorithm. The authors in [18] showed a parallel randomized algorithm in RNC^4 for this problem which has an approximation guarantee of $32H_n$ with high probability. Heuristic algorithms based on local search have been described by Pessoa et al. [17] and Wang et al [19].

In this paper, we adapt the approximation algorithms for the minimum cost *Set Multi-cover* proposed in [7] and [5] to solve the b-EDGE COVER problem. We also propose a new 3/2-approximation algorithm, the LSE algorithm that iteratively adds locally subdominant edges to the cover. We discuss several algorithmic issues in efficiently implementing these algorithms on a serial machine, and compare the results of these algorithms.

The rest of this paper is organized as follows. We discuss serial approximate algorithms for b-EDGE COVER and the algorithmic insights they are based on in Section 2. Next we discuss our proposed approximation algorithm LSE in Section 3, and the 3/2-approximation guarantee of LSE is proved in Section 4. Our experiments and results are described in Section 5. We provide a summary of our results and conclude in Section 6.

2 b-EDGE COVER, b-MATCHING, and Approximation Algorithms

2.1 Background In the b-EDGE COVER problem we assume that the value of $b(v)$ satisfies $0 \leq b(v) \leq \delta(v)$ for every vertex v, where $\delta(v)$ denotes the degree of a vertex. Then every b-EDGE COVER problem has a feasible solution, namely the graph G itself.

The b-EDGE COVER problem is also related to the b-MATCHING problem, where we choose a set of edges M such that *at most* $b(v)$ edges of M are incident on each vertex v. Here one considers a maximum weight b-MATCHING problem, where subject to the constraints on M, we seek to maximize the sum of the weights of the matched edges. A b-MATCHING is perfect if there are exactly $b(v)$ edges incident on every vertex v. Every b-MATCHING problem has a feasible solution, namely the empty matching, but a perfect b-MATCHING might not exist in a given graph. One easy way to show this is to choose $b(v)$ values so that they sum to an odd value.

Let $B \equiv \sum_{v \in V} b(v)$, and let $\beta \equiv \max_{v \in V} b(v)$. The b-MATCHING problem can be solved in polynomial time, although the $O(mnB)$ time complexity of exact algorithms make them impractical on graphs with billions of edges. A number of half-approximation algorithms have been designed for this problem, and a discussion of several algorithms including the currently fastest algorithm, the b-SUITOR algorithm, is described in [12].

A b-MATCHING can be used to compute a b-EDGE COVER as follows. For each vertex v, denote $b'(v) = \delta(v) - b(v)$. If there is a perfect b'-MATCHING in G, then let $M^{b'}_{opt}$ denote the maximum weight of such a matching. We can obtain a minimum weight b-EDGE COVER in G by taking all the unmatched edges in a maximum weight perfect b'-MATCHING. Let C^b_{opt} denote the minimum weight of a b-EDGE COVER in G, and $w(E)$ denote the sum of the weights of all the edges in G; then we have

$$C^b_{opt} = w(E) - M^{b'}_{opt}.$$

Thus if we have an optimal perfect matching

for a b'-MATCHING problem, then the unmatched edges of that graph constitute an optimal solution for b-EDGE COVER. If we relax one or both of the optimality or perfect matching restriction then the b-EDGE COVER solution will not be optimal but will still be a b-EDGE COVER. We use this observation to implement a heuristic algorithm for b-EDGE COVER using an approximation algorithm for b-MATCHING later in this paper.

There are a few advantages in using b-MATCHING to solve b-EDGE COVER problems. First, approximation algorithms for b-MATCHING do not need to compute and update effective weights in the course of the algorithm, unlike the b-EDGE COVER algorithms we consider here. Hence these algorithms can be implemented faster than b-EDGE COVER algorithms. They are also more amenable to parallelization on both multithreaded shared memory machines and distributed memory machines with large numbers of processors. Second, the approximation algorithms for b-MATCHING usually find a solution that is close (96%-99%) to the weight of the optimal matching, and they compute a b-EDGE COVER that is within a few percent of the optimal as well. We use b-SUITOR, a new $1/2-$ approximation algorithm for b-MATCHING, in order to compute a heuristic solution for b-EDGE COVER.

Together with other colleagues, we have proposed the b-SUITOR algorithm in earlier work [12]. The algorithm is based on a proposal extend-accept-annul scheme where a vertex extends proposals to its neighbors in decreasing order of their edge weights. Vertices may propose in any order. Each vertex v can receive at most $b(v)$ proposals, and it keeps track of the weight of its $b(v)$-th (lowest) proposal thus far. A vertex v extends a proposal to a neighbor w only if the weight of the edge (v, w) is greater than the $b(w)$-th lowest proposal that w currently holds. A vertex v may annul a proposal received by its neighbor w, if the weight of the edge (v, w) is greater than the lowest proposal that w currently has, say from a neighbor x of w. In this case, x would need to extend a proposal to its next eligible neighbor. When two vertices propose to each other, then they are matched.

The b-SUITOR algorithm is currently the fastest algorithm in serial, mutli-threaded, and distributed memory settings. This algorithm is about 300 times faster than an exact b-MATCHING algorithm based on belief propagation on a set of test problems [12]. On the multithreaded shared memory Intel Xeon with 16 threads, b-SUITOR is 14× faster than the Locally Dominant edge algorithm. It scales upto 240 threads of an Intel Xeon Phi as well. The b-SUITOR algorithm is also suitable for distributed memory settings because vertices are free to extend proposals in any order, thus increasing the concurrency in the algorithm. We have shown that the distributed b-SUITOR algorithm scales up to $16K$ processors of an extreme-scale computer such as Cori at NERSC, where it can solve both synthetic and real world problems with billions of edges under 4 seconds [11].

2.2 Approximation Algorithms In this Section, we describe two approximation algorithms for b-EDGE COVER: GREEDY and DELTA. Both of these are special cases of the algorithms for set multi-cover proposed in [5] and [7] respectively. We discuss algorithmic issues in implementing these, and motivate the need to develop a new approximation algorithm which solves these issues.

2.2.1 GREEDY algorithm: The GREEDY algorithm is a 3/2-approximation algorithm for b-EDGE COVER. We will need the concept of the *effective weight,* $w'(u, v)$ of an edge (u, v), which is the weight of an edge divided by the number of its endpoints v whose $b(v)$ values are not satisfied by the edges in the current b-EDGE COVER (this value is either zero or one or two). We will say that such vertices are *uncovered*. We also need the concept of a neighboring edge of an edge (u, v), an edge that shares one of its endpoints, u or v. Hence the effective weight of an edge that is not in the current edge cover is its weight divided by the number of uncovered endpoints of the edge. Initially the edge cover is empty, and the effective weight of each edge is half its weight, but as the algorithm adds an edge to the cover, the effective weight of its neighboring edges could double, and finally when both end points are covered become infinite.

Algorithm 1 Algorithm GREEDY($G(V, E, w), b$)

```
1:  EC = ∅;
2:  Compute the effective weights, w' of E;
3:  Create a min priority heap, Q using w' of E;
4:  while Graph Q ≠ ∅ AND constraints are not satisfied do
5:      e(u, v) = Q.pop();
6:      EC = EC ∪ e(u, v);
7:      for x ∈ {u, v} do
8:          if b(x) > 0 then
9:              b(x) = b(x) − 1
10:             if b(x) == 0 then
11:                 Update effective weights of
12:                 e(x, y) ∈ E ∧ y ≠ v;
13:     if any change in effective weights then
14:         Q.heapify(); ▷ Update Q with new weights
15: return b-EDGE COVER EC
```

The GREEDY Algorithm 1, considers edges to add to the b-EDGE COVER in increasing order of effective weights. It uses a minimum priority queue which enables it to find an edge with globally lowest effective weight. It adds an edge (u, v) of lowest effective weight to the cover, and decrements the values $b(u)$ and $b(v)$ by one. If either value $b(u)$ (or $b(v)$) becomes zero, then the effective weights of all other edges incident on that endpoint u (v) are updated. If there is a change in any existing effective weight then the algorithm rebuilds the queue to bring it to heap order. When there are no more edges to process or all the $b(v)$ constraints are satisfied then the algorithm terminates. The time complexity of the algorithm is $O(\beta m \log m)$.

THEOREM 2.1. *The* GREEDY *algorithm computes a* 3/2*-approximation to the minimum weight* b-EDGE COVER *problem.*

Proof. The *Set Multi-cover* problem defined in Section 1 reduces to the b-EDGE COVER problem on a simple graph when each set has exactly two elements. [5] (Theorem 3.1) proves that the GREEDY algorithm has the approximation ratio of $H(a)$ for the *Set Multi-cover* problem, where a is the maximum number of elements in any set. In the context of b-EDGE COVER, $a = 2$, and thus Algorithm 1 is an $H(2) = 1 + 1/2 = 3/2$-approximation algorithm. ∎

The GREEDY algorithm does not have much concurrency since it must process the edges in increasing order of effective weights, and these weights need to be updated during the algorithm. Maintaining the edges in increasing order of effective weights is also expensive.

2.2.2 DELTA **algorithm:** We describe the DELTA Algorithm 2, a Δ-approximation algorithm for the b-EDGE COVER, where Δ is the maximum degree in the graph. At each step, the algorithm arbitrarily chooses an uncovered vertex v, and adds the lightest weight edge $e(v, u)$ incident on it to the solution. The algorithm then updates $b(v)$ and $b(u)$, and subtracts the edge weight $w(v, u)$ from all other edges incident on v. It terminates when all $b(v)$ constraints are satisfied. The algorithm has time complexity of $O(\beta m)$. From the discussion, it should be clear that this algorithm is highly sensitive to the order in which vertices are processed. This is specifically problematic in parallel computing: in the shared memory context the scheduling of threads is determined by the underlying operating system and is non-deterministic. From one parallel run to another, the order in which threads are scheduled will also determine the order which vertices are processed. Another issue with the DELTA algorithm is that its approximation guarantee is high relative to the GREEDY algorithm, since the maximum degree of a graph can be large.

Algorithm 2 Algorithm DELTA$(G(V, E, w), b)$

```
1:  EC = ∅;
2:  while constraints are not satisfied do
3:      arbitrarily choose a vertex v;
4:      if b(v) > 0 then
5:          find e(v,u) ∉ EC, the lightest weight edge,
6:            w(v,u) incident on v;
7:          EC = EC ∪ e(v,u)
8:          b(v) = b(v) − 1;
9:          if b(u) > 0 then
10:             b(u) = b(u) − 1;
11:         for each edge (v,x) ∧ x ≠ u do
12:             w(v,x) = max(0, w(v,x) − w(v,u));
13: return b-EDGE COVER EC
```

3 A new 3/2−Approximate Algorithm: LSE

In this Section, we describe the LSE algorithm that iteratively computes a set of locally sub-dominating edges to add to the edge cover. Recall that an edge (u, v) is a *locally sub-dominating* edge if its effective weight is minimum relative to the effective weights of all neighboring edges. The effective weight of an edge is calculated as in the GREEDY algorithm. Ties are broken by say, choosing an edge with a lower numbered endpoint. With this tie-breaking scheme, locally sub-dominant edges in each iteration are uniquely defined, and are independent of each other, i.e., they do not share an endpoint.

Algorithm LSE iteratively finds a set of locally sub-dominant edges, adds them to the edge cover, and updates effective weights and $b(.)$ values. It is described in Algorithm 3.

At each iteration, we calculate the set of locally sub-dominant edges S as follows. Each vertex u sets a pointer to the edge of least effective weight incident on it. If the end points of an edge point each other, then the edge is locally sub-dominating. We pick each such edge, add it to the cover, remove it from further consideration, and finally decrement the $b(.)$ values of the end points.

When the graph becomes empty, we break the loop and algorithm terminates with a b-EDGE COVER, EC. The time complexity of the algorithm is $O(\beta m)$.

4 Proofs

In this section, we prove that the GREEDY and the LSE algorithms compute the same b-EDGE COVER, provided ties in weights are broken consistently.

Algorithm 3 Algorithm LSE($G(V, E, w), b$)

```
EC = ∅
Compute the effective weights of E
while Graph G is not exhausted AND constraints are not satisfied do
    Compute locally sub-dominating edges S of G
    for each e(u, v) ∈ S do
        EC = EC ∪ e(u, v)
        E = E \ e(u, v)
        for x ∈ {u, v} do
            if b(x) > 0 then
                b(x) = b(x) − 1
                if b(x) == 0 then
                    Update effective weights of e(x, y) ∈ E
return b-Edge Cover EC
```

Lemma 4.1. *The effective weight of any edge will only increase during the execution of the* Greedy *or* LSE *algorithm.*

Proof. The effective weight of an edge is its weight divided by the number of its uncovered end points. As the algorithm progresses, the number of uncovered vertices can only decrease, and hence the effective weight can only increase. ∎

Lemma 4.2. *Locally sub-dominating edges in an iteration of the* LSE *algorithm are independent.*

Proof. By definition, when we use the tie-breaking scheme on effective weights, each vertex can be incident on at most one locally sub-dominant edge. Therefore within an iteration, locally sub-dominating edges are independent. ∎

Lemma 4.3. *If an edge becomes locally sub-dominating at some point in the* LSE *algorithm, then it remains so until the algorithm adds it to the b-*Edge Cover.

Proof. This Lemma is a direct consequence of Lemmas 4.1 and 4.2. We define the edge neighborhood of an edge $(u, v) \in E$ as $N(u, v) = \{(u, w) \in E \wedge (w \neq v)\} \cup \{(z, v) \in E \wedge (z \neq u)\}$. Consider two arbitrary locally sub-dominating edges (u, v) and (x, y) that belong to the same iteration of the LSE algorithm. Without loss of generality, assume that the edge (u, v) is added to the edge cover first. When (u, v) is added to the edge cover, the effective weights of the edges in $N(u, v)$ increase by Lemma 4.1. But the edge (x, y) is independent of (u, v) by Lemma 4.2, and hence it does not belong to the neighborhood set $N(u, v)$. Thus its effective weight remains unchanged, and it continues to be a locally sub-dominating edge. ∎

Lemma 4.3 is an important property not only for the proof of the approximation guarantee but also in the context of concurrent computations. The Lemma shows that the order in which the algorithm adds the locally sub-dominating edges into the cover does not matter. Hence we can choose all these edges in parallel without changing the computed b-Edge Cover.

Lemma 4.4. *When the* Greedy *algorithm adds an edge to the b-*Edge Cover*, it is a locally sub-dominating edge in the current graph.*

Proof. The Greedy algorithm chooses the next edge which is currently an edge with the least effective weight among all uncovered edges. Denote this edge by (u, v). Then any other edge that belongs to the neighborhood $N(u, v)$ (hence of the form (w, u) or (v, z)), has either a higher effective weight than (u, v), or the same effective weight but has a higher numbered endpoint than the minimum value of $\{u, v\}$. Thus (u, v) is a locally sub-dominant edge in the current graph. ∎

The Greedy and LSE algorithms do not necessarily choose edges to include in the b-Edge Cover in the same order; since the effective weights of neighboring edges could change after an edge is added to the cover, at some point during their execution, an (uncovered) edge could have two different weights in these algorithms. We now prove a result that holds despite these differences.

Lemma 4.5. *For any vertex v, the* Greedy *and* LSE *algorithms choose the same set of edges incident on v to cover v; furthermore, the edges are chosen in the same order by both algorithms.*

Proof. Let the degree of the vertex v be $\delta(v) \equiv \delta$, and let the edges incident on v be numbered from 1 to δ. The algorithms choose at least $b(v) \leq \delta$ edges to add to the b-Edge Cover. Let the Greedy algorithm choose edges denoted $g_1, g_2, \ldots, g_b$ in that order; and let the LSE algorithm choose edges denoted $l_1, l_2, \ldots, l_{b'}$ in that order. Note that $b \geq b(v)$ and $b' \geq b(v)$. The Lemma claims that $b = b'$, and $g_j = l_j$, for $j = 1, 2, \ldots, b$.

We prove this result by contradiction. If the two sets are not identical, then choose the smallest index j such that $g_j \neq l_j$.

First consider the case that $1 \leq j \leq \min\{b, b'\}$. Delete the $j - 1$ edges from the two lists chosen earlier to cover v, since these are identical by choice of the index j. Since the Greedy algorithm chooses the edge g_j, it is now the edge of least effective weight incident on v, and it is also a locally subdominant edge incident

on v by Lemma 4.4. Now since the LSE algorithm picks the edge $l_j \neq g_j$ to add to the b-Edge Cover l_j is also a locally subdominant edge incident on vertex j. But since the edges are ordered by weight and ties are broken by the indices of their endpoints, the locally subdominant edge incident on v at this point in each algorithm must be unique. Hence we have $g_j = l_j$ contrary to assumption.

Now we consider the case that $j > \min\{b, b'\}$. Note that the minimum of the two values could be greater than $b(v)$. In this case either the Greedy or LSE algorithm adds an edge to the b-Edge Cover, while the other algorithm does not. Let the other end point of the edge incident on v added to the b-Edge Cover be x. Now this edge is added to the cover because x is unsaturated although v already has at least $b(v)$ edges incident on it in the cover. Since both the Greedy and the LSE algorithms add edges to the cover in the same order for the unsaturated vertex x (by the previous paragraph since v was an arbitrary vertex), if one of these algorithms includes the edge in the cover, the other also includes it in the cover. ∎

Theorem 4.1. *Both the* Greedy *and* LSE *algorithms choose exactly the same set of edges provided that both algorithms break ties consistently.*

Proof. We prove this result by induction on the number of iterations in the Greedy algorithm. We claim that every edge (u, v) chosen by the Greedy algorithm will also be chosen by the LSE algorithm at some iteration of the latter algorithm.

Initially both the Greedy and the LSE algorithms begin with the empty edge cover, and the claim is true. Hence by the inductive hypothesis assume that claim is true at the end of the k-th iteration of the Greedy algorithm, and consider its $(k+1)$-st iteration.

By Lemma 4.4, when an edge (u, v) is chosen by the Greedy algorithm to belong to the b-Edge Cover, it is a locally sub-dominating edge in the current graph. Let $N(u, v)$ denote the neighboring edges in the current graph relative to the Greedy algorithm. Only edges in the set $N(u, v)$ can satisfy the currently unmet $b(u)$ or $b(v)$ requirements of the b-Edge Cover. At this stage in the Greedy algorithm, fewer than $b(u)$ or $b(v)$ (or both) edges belong to the partial edge cover computed by it, for otherwise its effective weight would be infinite (both endpoints covered), and the algorithm would not choose this edge. When the LSE algorithm considers the edge (u, v) to add to the edge cover at the iteration when it becomes locally subdominant, by Lemma 4.5 edges incident on u (and v) are added to the b-Edge Cover in the same order in the Greedy and LSE algorithms. Hence at least one of the $b(u)$ or $b(v)$ requirements have not been met in the LSE algorithm also, and this edge will be added to the b-Edge Cover. ∎

Theorem 4.2. *Algorithm* LSE *is a 3/2-approximation algorithm.*

Proof. This follows immediately from the previous Theorem since the Greedy algorithm satisfies this approximation ratio. ∎

5 Experiments & Results

5.1 Experimental Setup: For the experiments, we used an Intel Xeon E5-2660 processor based system (part of the Purdue University Community Cluster), called *Snyder*[1]. The machine consists of two processors, each with ten cores running at 2.6 GHz (20 cores in total) with 25 MB unified L3 cache and 256 GB of memory. The operating system is Red Hat Enterprise Linux 6. All the code was developed using C++ and compiled using the Intel C++ Composer XE 2013 compiler (version: 1.1.163) using the -O3 flag.

Our testbed consists of both real-world and synthetic graphs. Synthetic datasets were generated using the Graph500 RMAT data generator [14]. We generate three different synthetic datasets varying the RMAT parameters (similar to previous work [12]). These are (i) *rmat_b* with parameter set (0.55, 0.15, 0.15, 0.15), (ii) *rmat_g* with parameter set (0.45, 0.15, 0.15, 0.25), and (iii) *rmat_er* with parameter set (0.25, 0.25, 0.25, 0.25). These graphs have varying degree distributions representing different application areas. We also consider a random geometric graph (*geo_14*) [16] that has recently attracted attention in the study of neural networks, astrophysics, etc.

Additionally we consider eight real-world datasets taken from the University of Florida Matrix collection [4] covering mutually exclusive application areas such as medical science, structural engineering, and sensor data.

Table 1 shows the sizes of our testbed. Our testbed is divided into two subsets: the first nine problems with tens of millions of edges or greater, and the last three with low edge counts. The larger problems are the five largest symmetric problems from Florida Matrix collection. The other three out of these eight real world problems have fewer than a million edges, and we use these to compare the weight of the edge covers computed by the approximation algorithms relative to the exact edge cover. The reason for using smaller datasets for comparing edge cover quality is that we do not have a fast implementation for exact weighted b-Edge Cover.

[1] https://www.rcac.purdue.edu/compute/snyder/

Problem	Vertices	Edges	Avg. Deg	Max. Deg
Fault_639	638,802	13,987,881	11	317
mouse_gene	45,101	14,461,095	160	8,031
bone010	986,703	35,339,811	18	80
dielFil.V3real	1,102,824	44,101,598	20	269
kron.logn21	2,097,152	91,040,932	22	213,904
geo_14	16,384	54,689,168	3274	10,365
rmat_b	1,048,576	123,599,502	58	63,605
rmat_g	1,048,576	133,056,675	64	7,998
rmat_er	1,048,576	134,201,240	64	337
astro-ph	16,706	121,251	8	360
Reuters911	13,332	148,038	12	2265
cond-mat-2005	40,421	175,693	5	278

Table 1: The structural properties of our testbed for b-Edge Cover.

The exact b-Edge Cover algorithm that we compare LSE to is based on integer linear programming and is programmed in Matlab. This implementation can run only on the smaller datasets due to memory limitations, and hence we selected the largest symmetric problems that this implementation could solve.

5.2 Results: We analyze the serial performance of three algorithms: Greedy, LSE and Delta. We have considered two sets of $b(v)$ values for the experiments. First we fix $b(v) = \min\{b, \delta(v)\}$, where $b \in \{1, 3, 5, 10, 15\}$ for all the vertices. The reason for using constant values of $b(v)$ is to study how the algorithms perform as the value of $b(v)$ is varied. The second set of $b(v)$ values is motivated by the privacy application in [2]. Here we randomly assign values of $b(v)$ for each vertex v, choosing $b(v)$ uniformly between 1 and $\delta(v)^{1/2}$. We generate three sets of $b(v)$ values for each problem, and select the set which leads to the median weight for its b-Edge Cover. We use these randomized $b(v)$ values for all the experiments. The average randomized $b(v)$ values for all the problems are between 3 and 9, except for the relatively dense (geo_14) problem, where the average value of $b(v)$ is 40. In the experiments, we explicitly list the b values unless we use randomized values for them.

5.2.1 Cover Weight: We have compared the weight of the solution for all three algorithms. Both the Greedy and the LSE algorithms find the identical edge cover irrespective of vertex ordering, and their weight is not affected by the order in which the vertices are processed. However, the Delta algorithm is sensitive both in terms of runtimes and solution weight to the order in which the vertices are processed.

In Figure 1, we compare the total edge weight computed by LSE (as well as Greedy) and Delta with respect to the edge weight of an optimal b-Edge Cover for different choices of $b(v)$ for the smaller three problems. It is to be noted that LSE is a $3/2-$approximation algorithm, i.e., the weight of the edge cover should be less than 150% of the optimal algorithm. However,

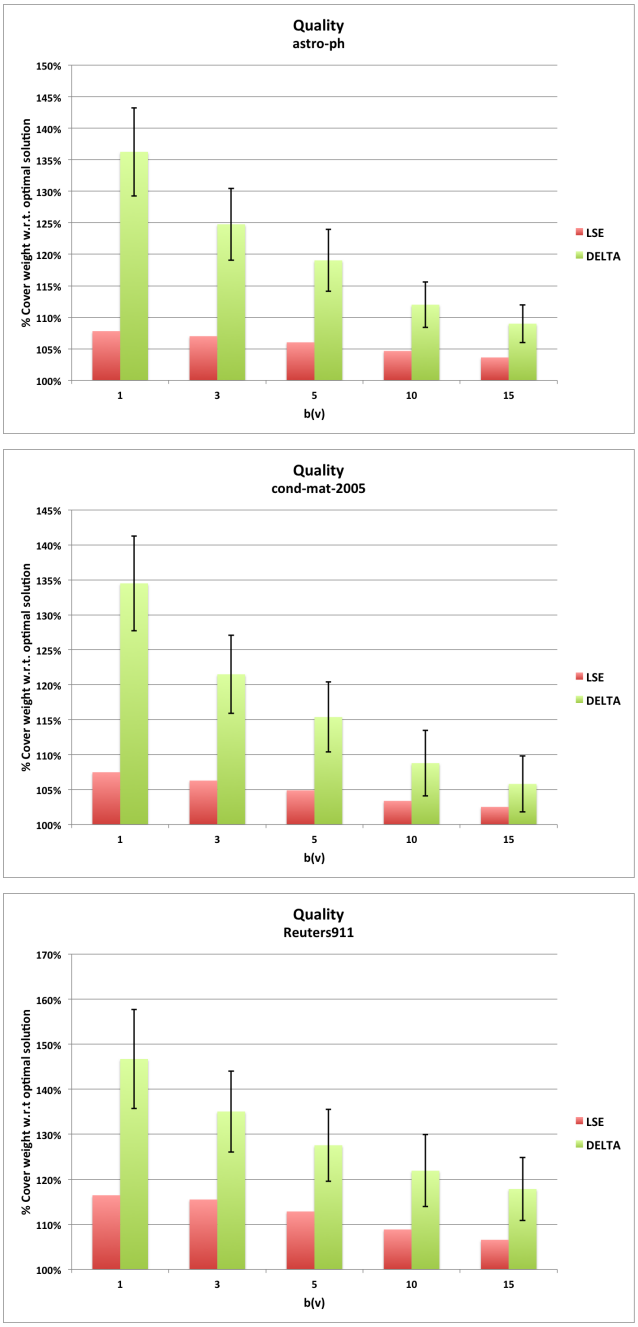

Figure 1: Solution quality w.r.t. optimal weight of LSE with varying values of $b(v)$.

DELTA is a $\Delta-$approximation algorithm, and for the three problems *astro-ph*, *reuters911* and *cond-mat-2005* the maximum degrees are 360, 2265 and 278, respectively. We see that both algorithms find solutions much better than their approximation guarantees. In particular, LSE gives less than 117% of the optimal weight when $b(v) = 1$ for the three smaller problems where we could run the exact algorithm, and it gives better solution than the DELTA algorithm. The figure also shows the variations (as big as 11%) in the quality of solutions for different vertex ordering for the DELTA algorithm. However, the gap between the algorithms decreases with increasing $b(v)$ values. The variability in the solution for the DELTA algorithm also decreases with the increase of $b(v)$. This is an important insight for the b-EDGE COVER problem overall relative to the *Edge Cover* problem. Given a fixed b value, the expected number of edges in the cover is at least $(n \times b)/2$, meaning that the number of edges in the cover is larger with higher values of $b(v)$. As the solution set gets larger, it is less likely that an approximate b-EDGE COVER algorithm will miss out on a good edge. Therefore, the difference in solution quality will tend to be smaller among the b-EDGE COVER algorithms than for *Edge Cover* algorithms; this difference should also decrease with increasing $b(v)$ values.

5.2.2 Runtimes of the algorithms: Our main contribution in this paper is to show that it is possible to obtain an approximation algorithm as good as GREEDY without sorting the edges. Hence the algorithm is expected to run faster on a serial machine. We provide evidence for our claim in Figure 2, and show that indeed LSE is $2\times$ to $5\times$ faster (with geometric mean $3\times$) than the GREEDY algorithm. The difference is bigger with larger input instances. It is to be noted that LSE has much more concurrency than GREEDY, but a parallel implementation is the scope of future work.

We also show the comparison between LSE and DELTA in Figure 3 with the variations in runtimes for DELTA algorithm for five different vertex orderings. The variability in run times ($\pm$12 sec) for the DELTA algorithm makes it difficult to draw any conclusion. We compute how much LSE is faster with respect to the average runtimes for DELTA and on average LSE is $1.1\times$ faster, i.e., these algorithms are comparable in terms of their serial performance. However, the weights of the edge covers computed by the DELTA algorithm varies with vertex orderings, and this makes it unsuitable for parallel implementation.

5.2.3 Relationship with the b-MATCHING problem: In this Subsection, we discuss how a b-MATCHING

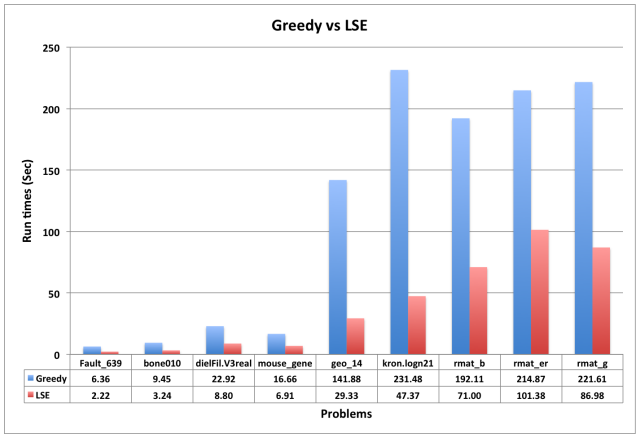

Figure 2: Comparison of serial run times between GREEDY and LSE.

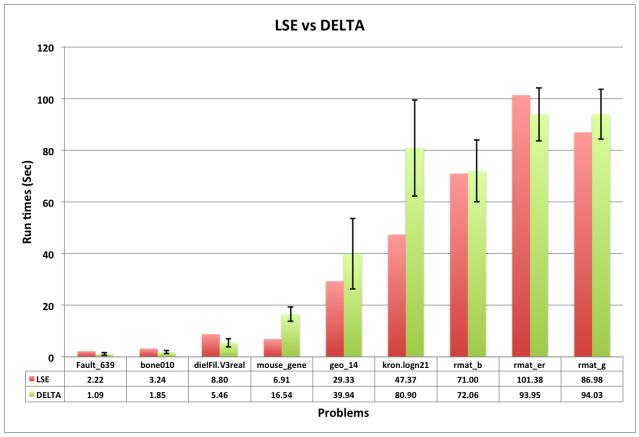

Figure 3: Comparison of serial run times between DELTA and LSE.

could be used to compute a b-EDGE COVER. Currently the fastest $1/2-$approximation algorithm for b-MATCHING is the b-SUITOR algorithm proposed in [12]. Both b-SUITOR and LSE employ similar schemes in iteratively finding a set of edges which have local importance. More precisely, b-SUITOR(LSE) finds locally heaviest (lightest) edges to maximize (minimize) the sum of weights of the solution edges where each vertex v has at most (at least) $b(v)$ edges incident on it. These are different algorithms for solving different problems with different approximation bounds, but procedurally they are similar except that LSE has an additional step that involves effective weight updates, whereas b-SUITOR works with static weights.

Although these are different problems, we compare the serial run times of the fastest variant of b-SUITOR (the Delayed Partial variant) and LSE with the same $b(v)$ values for a set of problems in Figure 4. We have

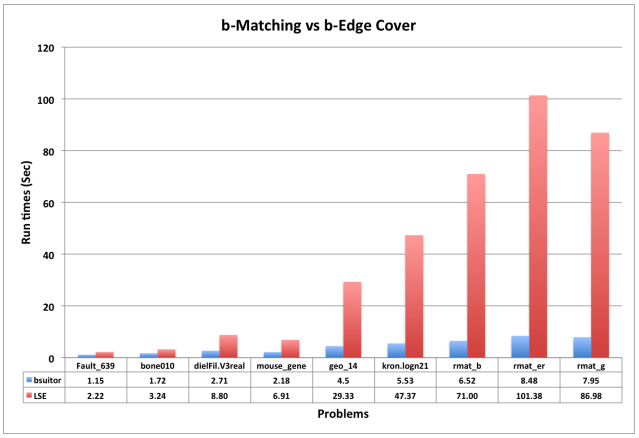

Figure 4: Comparison of serial run times between b-SUITOR and LSE.

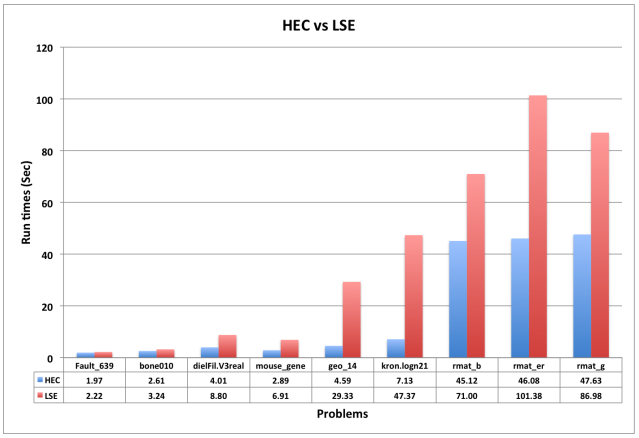

Figure 5: Comparison of serial run times between *HEC* and LSE.

observed that b-SUITOR is about an order of magnitude faster than LSE on larger problems (5× on an average). We claim that the difference is mainly due to the dynamic weight update phase.

Recall from Section 2 that we can use a perfect b'-MATCHING of maximum weight to compute a b-EDGE COVER of minimum weight, where $b'(v) = \delta(v) - b(v)$. If we relax one or both of the optimality or perfect matching restriction then the b-EDGE COVER will not be one of minimum weight. Nevertheless, we can use this observation to implement a heuristic algorithm for b-EDGE COVER using the b-SUITOR algorithm; we call the resulting algorithm the heuristic algorithm for b-EDGE COVER, *HEC*. We observe that the weight of the b-EDGE COVER thus obtained is within ±1.5% that of LSE. We also compare the serial run times with that of LSE in Figure 5, and b-SUITOR is roughly 2.3× faster than LSE algorithm on the average.

6 Conclusions

We have designed and implemented a new 3/2-approximation algorithm, the LSE algorithm, for computing a minimum weighted b-EDGE COVER in graphs. It computes the same edge cover as the one that would be obtained by the GREEDY algorithm. We show that LSE computes edge covers with weights that are comparable to or better than the DELTA algorithm for the same problem; the latter algorithm has a worse approximation ratio, and is also sensitive to the order in which the vertices are processed. The LSE algorithm does not require edges to be sorted by weight, and has more concurrency relative to the GREEDY algorithm. The solution obtained by LSE algorithm is insensitive to the order in which vertices are processed. The LSE algorithm outperforms the GREEDY algorithm by a factor of three (geometric mean) with respect to runtimes on an Intel Xeon processor.

Acknowledgements

This work was supported by the U.S. National Science Foundation grant CCF-1552323, the Office of Advanced Scientific Computing Research within the Office of Science of the Department of Energy under grant DE-SC0010205, and a gift from the Intel Corporation for a Parallel Computing Center at Purdue.

References

[1] P. BERMAN, B. DASGUPTA, AND E. SONTAG, *Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks*, Discrete Applied Mathematics, 155 (2007), pp. 733–749.

[2] K. M. CHOROMANSKI, T. JEBARA, AND K. TANG, *Adaptive anonymity via b-matching*, in Advances in Neural Information Processing Systems, 2013, pp. 3192–3200.

[3] V. CHVATAL, *A greedy heuristic for the set-covering problem*, Mathematics of Operations Research, 4 (1979), pp. 233–235.

[4] T. DAVIS AND Y. HU, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1:1–1:25.

[5] G. DOBSON, *Worst-case analysis of greedy heuristics for integer programming with nonnegative data*, Mathematics of Operations Research, 7 (1982), pp. 515–531.

[6] N. G. HALL AND D. S. HOCHBAUM, *The multicovering problem: The use of heuristics, cutting planes, and subgradient optimization for a class of integer programs*, College of Administrative Science, Ohio State University, 1985.

[7] N. G. Hall and D. S. Hochbaum, *A fast approximation algorithm for the multicovering problem*, Discrete Applied Mathematics, 15 (1986), pp. 35–40.

[8] Q.-S. Hua, Y. Wang, D. Yu, and F. C. Lau, *Set multi-covering via inclusion–exclusion*, Theoretical Computer Science, 410 (2009), pp. 3882–3892.

[9] Q.-S. Hua, Y. Wang, D. Yu, and F. C. Lau, *Dynamic programming based algorithms for set multicover and multiset multicover problems*, Theoretical Computer Science, 411 (2010), pp. 2467–2474.

[10] D. S. Johnson, *Approximation algorithms for combinatorial problems*, Journal of Computer and System Sciences, 9 (1974), pp. 256–278.

[11] A. Khan, A. Pothen, M. M. Patwary, M. Halappanavar, N. Satish, N. Sundaram, and P. Dubey, *Designing scalable b-Matching algorithms on distributed memory multiprocessors by approximation*, in Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing), 2016.

[12] A. Khan, A. Pothen, M. M. Patwary, N. Satish, N. Sundaram, F. Manne, M. Halappanavar, and P. Dubey, *Efficient approximation algorithms for weighted b-matching*, SIAM Journal on Scientific Computing, (2016). To appear.

[13] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Mathematics, 13 (1975), pp. 383–390.

[14] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, *Introducing the Graph 500*, Cray User's Group, (2010).

[15] J. Nederlof, *Inclusion exclusion for hard problems*, Master's thesis, Utrecht University (August 2008), (2008).

[16] M. Penrose, *Random Geometric Graphs*, vol. 5, Oxford University Press, 2003.

[17] L. S. Pessoa, M. G. Resende, and C. C. Ribeiro, *A hybrid lagrangean heuristic with grasp and path-relinking for set k-covering*, Computers & Operations Research, 40 (2013), pp. 3132–3146.

[18] S. Rajagopalan and V. V. Vazirani, *Primal-dual rnc approximation algorithms for set cover and covering integer programs*, SIAM Journal on Computing, 28 (1998), pp. 525–540.

[19] Y. Wang, M. Yin, D. Ouyang, and L. Zhang, *A novel local search algorithm with configuration checking and scoring mechanism for the set k-covering problem*, International Transactions in Operational Research, (2016), doi:10.1111/itor.12280, `http://dx.doi.org/10.1111/itor.12280`.