Combinatorial Approximations for Cluster Deletion: Simpler, Faster, and Better

Vicente Balmaseda [1] **Ying Xu** [2] **Yixin Cao** [2] **Nate Veldt** [1]

Abstract

Cluster deletion is an NP-hard graph clustering objective with applications in computational biology and social network analysis, where the goal is to delete a minimum number of edges to partition a graph into cliques. We first provide a tighter analysis of two previous approximation algorithms, improving their approximation guarantees from 4 to 3. Moreover, we show that both algorithms can be derandomized in a surprisingly simple way, by greedily taking a vertex of maximum degree in an auxiliary graph and forming a cluster around it. One of these algorithms relies on solving a linear program. Our final contribution is to design a new and purely combinatorial approach for doing so that is far more scalable in theory and practice.

1. Introduction

Graph clustering is a fundamental task in graph mining where the goal is to partition nodes of a graph into disjoint clusters that have dense internal connections but are only sparsely connected to the rest of the graph. This has a wide variety of applications which include detecting communities in social networks (Fortunato, 2010), identifying related genes in biological networks based on gene expression profiles (Ben-Dor et al., 1999), and finding groups of pixels in an image that belong to the same object (Shi & Malik, 2000). An idealized notion of a cluster in a graph is a set of nodes that is completely connected internally (i.e., a clique) while being completely disconnected from the rest of the graph. *Cluster graph modification problems* (Shamir et al., 2004) are a class of graph clustering objectives that seek to edit the edges in a graph as little as possible in order to achieve this idealized structure. One widely studied problem is correlation clustering (Bansal et al., 2004), which can be cast as adding or deleting a minimum number of edges to convert a graph into a disjoint union of cliques. This problem is also known as *cluster editing*. Designing approximation algorithms for different variants of correlation clustering has a long history, and has also seen extensive interest in the past few years in the machine learning community (Jafarov et al., 2020; 2021; Bun et al., 2021; Cohen-Addad et al., 2021; Veldt, 2022; Stein et al., 2023; Davies et al., 2023; Assadi et al., 2023).

This paper focuses on a variant of correlation clustering called CLUSTER DELETION, which seeks a minimum number of edges to *delete* so that the graph becomes a disjoint set of cliques. CLUSTER DELETION was first motivated by applications in clustering gene networks (Ben-Dor et al., 1999) and arises as an interesting special case of other more general frameworks for clustering (Charikar et al., 2005; Puleo & Milenkovic, 2015; Veldt et al., 2018). The problem is NP-hard, but has been studied extensively from the perspective of parameterized algorithms (Gramm et al., 2005; Damaschke, 2009; Gao et al., 2013; Böcker & Damaschke, 2011; Bathie et al., 2022) and approximation algorithms (Charikar et al., 2005; Dessmark et al., 2007; Puleo & Milenkovic, 2015; Veldt et al., 2018; Veldt, 2022). We provide several improved theoretical results and practical implementations for combinatorial algorithms for this task.

Previous work. The first approximation algorithm for CLUSTER DELETION was based on rounding a linear programming (LP) relaxation and came with a factor 4-approximation guarantee (Charikar et al., 2005). Other approximation algorithms based on the same canonical LP were subsequently developed (van Zuylen & Williamson, 2009; Puleo & Milenkovic, 2015), culminating in the current-best approximation factor of 2 (Veldt et al., 2018). One limitation of all of these algorithms is that the underlying LP relaxation has $O(n^3)$ constraints for a graph with n nodes, and is prohibitively expensive to solve in practice on large instances. Recently, Veldt (2022) provided faster approximation algorithms by rounding different and less expensive lower bounds for CLUSTER DELETION. The first was a 4-approximation algorithm based on rounding an LP relaxation for a related problem called Strong Triadic Cluster (STC) labeling (Sintos & Tsaparas, 2014). The STC LP relaxation has fewer constraints than the canonical CLUSTER DELETION LP relaxation, but still provides

[1]Department of Computer Science and Engineering, Texas A&M University, College Station, Texas, USA [2]Department of Computing, Hong Kong Polytechnic University, Hong Kong, China. Correspondence to: Vicente Balmaseda <vibalcam@tamu.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

a lower bound on CLUSTER DELETION. Veldt also developed the first *combinatorial* approximation algorithm, called *MatchFlipPivot*, which applies a fast algorithm for STC labeling and then rounds the resulting edge labels into a 4-approximate CLUSTER DELETION solution. In numerical experiments, solving and rounding the STC LP relaxation using black-box LP software was shown to be roughly twice as fast as solving and rounding the canonical LP, while *MatchFlipPivot* was shown to be orders of magnitude faster.

Motivating questions. While these recent results lead to more practical algorithms, there is still a gap between theory and practice for CLUSTER DELETION algorithms, and several open questions remain. Although the STC-based algorithms are faster and more practical, their 4-approximation guarantee is still noticeably worse than the 2-approximation based on the canonical LP relaxation. In practice, the STC-based algorithms tend to produce solutions that are much better than just a 4-approximation (Veldt, 2022). A natural direction is to try to improve approximation factors and bridge the gap between theoretical and practical performance of STC-based methods.

Another direction is to address the performance gap between *MatchFlipPivot* and the STC LP rounding algorithm. *MatchFlipPivot* is far faster in practice while satisfying the same worst-case approximation guarantee. At the same time, the STC LP relaxation is guaranteed to return a tighter lower bound for CLUSTER DELETION, and was shown to produce higher quality results in practice. Furthermore, solving the STC LP relaxation was observed to often return the optimal solution for the canonical LP relaxation in practice. In these cases, the LP rounding technique is guaranteed to return a 2-approximate solution. These observations motivate the study of better approximation guarantees and faster techniques for solving the STC LP relaxation.

Finally, existing implementations of the STC-based algorithms are randomized, and their approximation guarantees hold only in expectation. In theory these algorithms can be made deterministic by leveraging existing derandomization techniques (van Zuylen & Williamson, 2009). However, the deterministic versions are more complicated and slower, and as such have not been implemented in practice.

Our contributions. We significantly bridge the theory-practice gap by presenting algorithms that are *simpler*, *faster*, and have *better* approximation guarantees.

- We provide a simplified presentation and a tight analysis of the *MatchFlipPivot* algorithm, proving an improved 3-approximation guarantee for the method and providing instances on which the ratio is asymptotically 3.
- We show a similar tighter analysis for an STC LP rounding algorithm, improving its approximation guarantee to 3.
- We improve the runtime of *MatchFlipPivot* by designing a faster algorithm for a key step: computing a maximal edge-disjoint set of open wedges in a graph.
- We prove that the STC LP relaxation can be reduced to a minimum s-t cut problem, leading to a faster, purely combinatorial version of our LP-based algorithm.
- We prove a simpler and faster new approach for *deterministically* rounding a CLUSTER DELETION lower bound into an approximate solution.

To put the last contribution into context, we note that previous approximations for CLUSTER DELETION rely on (1) computing a lower bound on a graph G, (2) rounding the lower bound into a new graph $\hat{G}$, and (3) forming clusters by *pivoting* in $\hat{G}$ (repeatedly select a node and cluster it with its neighbors). We prove that selecting pivot nodes based simply on degrees in $\hat{G}$ provides the same approximation guarantee as other (more complicated and computationally expensive) deterministic pivoting strategies.

We accompany our theoretical results with practical implementations and numerical experiments[1]. They include the first implemented deterministic algorithms for CLUSTER DELETION, which in practice produce solutions that are typically much less than 3 times the optimal solution. We also implement our combinatorial algorithm for solving the STC LP relaxation and demonstrate in practice that it is significantly faster than using black-box LP software and scales to instances that are orders of magnitude larger.

2. Preliminaries and Related Work

Let $G = (V, E)$ be an unweighted undirected graph with $n = |V|$ and $m = |E|$. We use the $\tilde{O}(\cdot)$ notation to suppress logarithmic factors in runtimes, e.g., $O(\log n) = \tilde{O}(1)$. The problems we consider rely on the concept of open wedges. An *open wedge centered at* k is a node triplet (i, j, k) such that $(i, k) \in E$, $(j, k) \in E$ and $(i, j) \notin E$. The third node indicates the center of the wedge. The order of the first two nodes in an open wedge is irrelevant, hence $(i, j, k) \equiv (j, i, k)$. Let $\mathcal{W}(G)$ be the set of open wedges in G, and $\mathcal{W}_k(G) \subseteq \mathcal{W}(G)$ be the set of open wedges centered at k. When G is clear from context we simply write $\mathcal{W}$ and $\mathcal{W}_k$.

2.1. Cluster Deletion

Given graph G, CLUSTER DELETION seeks a set of edges $E_D \subseteq E$ that minimizes $|E_D|$ such that $G' = (V, E - E_D)$ is a disjoint set of cliques. This is equivalent to forming clusters in a way that minimizes the number of edges between clusters (known as "mistakes") while ensuring all

[1] https://github.com/vibalcam/combinatorial-cluster-deletion

clusters are cliques. This can be formulated as a binary linear program (BLP) as follows:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} x_{ij} \\ \text{s.t.} \quad & x_{ik} + x_{jk} \geq x_{ij} && \forall i,j,k \\ & x_{ij} = 1 && \text{if } (i,j) \notin E \\ & x_{ij} \in \{0,1\} && \forall (i,j) \in E. \end{aligned} \tag{1}$$

This BLP has one variable for each pair of nodes, and $x_{ij} = 0$ if and only if nodes i and j are in the same cluster. The canonical LP relaxation for CLUSTER DELETION is obtained by replacing $x_{ij} \in \{0,1\}$ with nonnegativity constraints $x_{ij} \geq 0$. Charikar et al. (2005) presented a 4-approximation based on this LP relaxation. The results of van Zuylen & Williamson (2009) for constrained variants of correlation clustering imply a 3-approximation algorithm for CLUSTER DELETION by rounding the same LP. The current best approximation factor for CLUSTER DELETION, also obtained by rounding this LP, is 2 (Veldt et al., 2018).

2.2. Strong Triadic Closure Labeling

CLUSTER DELETION has a well-documented connection to another NP-hard graph optimization problem (Sintos & Tsaparas, 2014). The latter problem is derived from the Strong Triadic Closure (STC) principle from social network analysis (Granovetter, 1973; Easley & Kleinberg, 2010), which states that if two individuals both have a strong connection to a mutual friend, they are likely to share at least a weak connection with each other.

Following this principle, we can label the edges in G as either weak or strong such that the STC principle is satisfied, i.e., each open wedge has at least one weak edge. This is called an *STC labeling*, and is encoded by a set of weak edges $E_W \subseteq E$. The *minimum weakness strong triadic closure* (MINSTC) problem (Sintos & Tsaparas, 2014) is then the problem of finding a strong triadic closure labeling of G that minimizes the number of weak edges. Formally this is cast as the following BLP:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} x_{ij} \\ \text{s.t.} \quad & x_{ik} + x_{jk} \geq 1 && \forall (i,j,k) \in \mathcal{W} \\ & x_{ij} \in \{0,1\} && \forall (i,j) \in E. \end{aligned} \tag{2}$$

The variable x_{ij} is equal to 1 if and only if edge (i,j) is a weak edge. The constraints in this BLP are in fact a subset of the constraints in the CLUSTER DELETION BLP in (1). This implies that every feasible solution E_D for CLUSTER DELETION defines a valid STC labeling $E_W = E_D$, and hence MINSTC lower bounds CLUSTER DELETION. However, deleting edges in an arbitrary STC labeling E_W does not necessarily produce a disjoint union of cliques. The relationship between MINSTC and CLUSTER DELETION has been noted in several different contexts (Konstantinidis et al., 2018; Veldt, 2022; Bengali & Veldt, 2023), and there are known graphs where their optimal solutions differ by up to a factor of $8/7$ (Grüttemeier & Komusiewicz, 2020).

Approximations based on vertex cover. Solving MINSTC over a graph $G = (V, E)$ is equivalent to finding a minimum vertex cover in the Gallai graph of G, obtained by associating each edge $(i,j) \in E$ with a vertex v_{ij} and introducing an edge (v_{ik}, v_{jk}) in the Gallai graph if (i,j,k) defines an open wedge in G (Le, 1996). Every algorithm for vertex cover instantly implies an algorithm for MINSTC with the same approximation factor. One simple 2-approximation for MINSTC is to find a maximal edge-disjoint set of open wedges in G, then label an edge $(i,j) \in E$ as weak if it is in one of the open wedges in this set. This is equivalent to applying a standard maximal matching 2-approximation for vertex cover in the Gallai graph. Another simple 2-approximation is to solve the LP relaxation of the BLP in (2) and label $(i,j) \in E$ as weak if $x_{ij} \geq 1/2$, analogous to a standard LP rounding algorithm for vertex cover. Nemhauser & Trotter (1975) showed that the LP relaxation for vertex cover is half integral, meaning that every basic feasible solution has LP variables satisfying $x_{ij} \in \{0, 1/2, 1\}$. This property therefore also holds for the STC LP relaxation, obtained by replacing binary constraints in (2) with nonnegativity constraints $x_{ij} \geq 0$.

2.3. STC + Pivot Framework

The Pivot algorithm repeatedly selects an unclustered node (the *pivot*) in a graph and then clusters it with all of its unclustered neighbors. This was first designed as a way to approximate correlation clustering. When pivots are chosen uniformly at random and the procedure is applied directly to a graph G, this is a randomized 3-approximation algorithm for correlation clustering (Ailon et al., 2008). Many algorithms for different variants of correlation clustering and CLUSTER DELETION use Pivot as one step in a broader algorithmic pipeline (van Zuylen & Williamson, 2009; Chawla et al., 2015; Jafarov et al., 2020; Veldt, 2022). Choosing random pivots leads to approximation guarantees that hold only in expectation, but van Zuylen & Williamson (2009) also showed techniques for carefully selecting pivot nodes in order to obtain deterministic approximation guarantees for different problem variants.

Veldt (2022) recently provided a general framework for approximating CLUSTER DELETION by combining STC labelings with pivoting procedures. The framework first (1) obtains an approximately optimal STC labeling E_W for a graph $G = (V, E)$, and then (2) runs Pivot on graph $\hat{G} = (V, E - E_W)$ to form clusters. If pivoting on a node k places two other nodes i and j inside a cluster, then both (i,k) and (j,k) are strong edges, which guarantees $(i,j) \in E$. This

Algorithm 1 Pivot($\hat{G} = (V, \hat{E})$)

$V' \leftarrow V; E' \leftarrow \hat{E}; \mathcal{C} \leftarrow \emptyset$
while V' not empty **do**
  Select pivot $k \in V'$
  $C_k = k \cup \{i \in V' : (i,k) \in E'\}$ `// cluster`
  $\mathcal{C} = \mathcal{C} \cup \{C_k\}$
  $V' \leftarrow V' - C_k$ `// update graph`
  $E' \leftarrow \hat{E} \cap (V' \times V')$
end while
Return clustering $\mathcal{C}$

leads to a useful observation.

Observation 2.1. *If $E_W \subseteq E$ is an STC labeling for $G = (V, E)$, running Pivot on $\hat{G} = (V, E - E_W)$ with any pivot selection strategy produces clusters that are cliques in G.*

Veldt (2022) used this framework to design two 4-approximation algorithms for CLUSTER DELETION: one based on rounding the STC LP relaxation, and a faster purely combinatorial algorithm called *MatchFlipPivot* based on finding a maximal edge-disjoint set of open wedges. The approximation guarantees hold in expectation when pivot nodes are chosen uniformly at random. The derandomized pivoting techniques of van Zuylen & Williamson (2009) can be used to obtain deterministic approximation guarantees, though this is more involved conceptually and far slower computationally.

3. Improved Approximation Analysis

We prove tighter approximations and new deterministic rounding schemes for combining STC labelings with Pivot.

3.1. Pivoting Lemma

Algorithm 1 shows the generic Pivot algorithm applied to a graph $\hat{G}$. The resulting clusters are typically not cliques in $\hat{G}$, but we will combine these strategies with STC labeling techniques and Observation 2.1 in order to design CLUSTER DELETION approximation algorithms. Consider what happens if we have an induced subgraph $G' = (V', E')$ of $\hat{G}$ at some intermediate step of the Pivot algorithm and we pivot on a node $k \in V'$ to form a new cluster $C_k \subseteq V'$. Let $deg_k(G') = |C_k| - 1$ be the degree of node k in G' (the number of neighbors of k in V'), and define two sets of node pairs:

$$B_k(G') = \{(i,j) \in E' : (i,k) \in E', (j,k) \notin E'\},$$
$$N_k(G') = \{(i,j) \notin E' : (i,k) \in E', (j,k) \in E'\}.$$

The set $B_k(G')$ represents edges on the *boundary* of cluster C_k and $N_k(G')$ is the set of *non-edges* inside the cluster. We define three strategies for selecting pivots.

- **Pivot Strategy 1.** Select a pivot k with the maximum degree in G'.
- **Pivot Strategy 2.** Select a pivot k that minimizes $|B_k(G')|/|N_k(G')|$.
- **Pivot Strategy 3.** Select a pivot k uniformly at random.

Lemma 3.1. *Let $\mathcal{B}$ be the set of edges between clusters and $\mathcal{N}$ be the set of non-edges inside clusters that result from running Algorithm 1. If Pivot Strategy 1 or 2 is used, then $|\mathcal{B}| \leq 2|\mathcal{N}|$. If Pivot Strategy 3 is used, this holds in expectation: $\mathbb{E}[|\mathcal{B}|] = 2\mathbb{E}[|\mathcal{N}|]$.*

Proof. Consider the graph $G' = (V', E')$ at a fixed intermediate step of the algorithm. For an arbitrary node $v \in V'$ we write $N_v = N_v(G')$, $B_v = B_v(G')$ and $deg_v = deg_v(G')$

Strategy 1 analysis. Assume that node k is chosen as the pivot when applying Pivot Strategy 1. For an arbitrary node $u \in C_k \setminus \{k\}$, let b_u be the number of edges in G' that are incident to u but not contained in C_k, and let n_u be the number of non-edges involving u that are in C_k. Note that

$$b_u + (|C_k| - 1) - n_u = deg_u \leq deg_k = |C_k| - 1,$$

which implies that $b_u \leq n_u$. Each non-edge in C_k involves two nodes from $C_k \setminus \{k\}$, so

$$|B_k| = \sum_{u \in C_k \setminus \{k\}} b_u \leq \sum_{u \in C_k \setminus \{k\}} n_u = 2|N_k|.$$

Thus, the number of new boundary edges in each iteration is bounded by twice the number of non-edges in the new cluster. Summing across all iterations gives $|\mathcal{B}| \leq 2|\mathcal{N}|$.

Strategy 2 and 3 analysis. Let $\mathcal{W}'$ be the set of open wedges in G'. The following four statements are equivalent: (1) (i,j,k) is an open wedge; (2) $(i,k) \in B_j$, (3) $(j,k) \in B_i$, and (4) $(i,j) \in N_k$. Thus, $\sum_{k \in V'} |N_k| = |\mathcal{W}'|$ and $\sum_{k \in V'} |B_k| = 2|\mathcal{W}'|$. In other words,

$$\sum_{k \in V'} (|B_k| - 2|N_k|) = \sum_{k \in V'} |B_k| - 2 \sum_{k \in V'} |N_k| = 0. \quad (3)$$

Therefore, there is at least one node satisfying $|B_k| - 2|N_k| \leq 0$, and applying Pivot Strategy 2 guarantees that $|B_k| \leq 2|N_k|$, so summing across iterations again gives $|\mathcal{B}| \leq 2|\mathcal{N}|$. Regarding Pivot Strategy 3, Eq. (3) implies that for a uniform random pivot, $\mathbb{E}_{k \in V'}[|B_k| - 2|N_k|] = 0$. Thus, at every iteration, the expected number of new boundary edges is twice the expected number of non-edges inside the cluster. By linearity of expectation, $\mathbb{E}[|\mathcal{B}|] = 2\mathbb{E}[|\mathcal{N}|]$. □

3.2. Rounding a Disjoint Open Wedge Set

One way to approximate MINSTC over a graph $G = (V, E)$ is a straightforward adaptation of the matching-based approximation algorithm for vertex cover. We find a maximal

Algorithm 2 MATCHFLIPPIVOT($G = (V, E)$)

1: $W \leftarrow$ maximal edge-disjoint set of open wedges in G.
2: $E_W \leftarrow$ edges contained in some open wedge of W.
3: Form $\hat{G} = (V, E - E_W)$
4: Run Pivot($\hat{G}$) `// for some pivot strategy`

edge-disjoint set of open wedges $W \subseteq \mathcal{W}$, and then for each $(i, j, k) \in W$, place edges (i, k) and (j, k) into the weak edge set E_W. Note that $|W|$ is a lower bound for MINSTC (and also CLUSTER DELETION) since each open wedge in W must contain at least one weak edge (or in the case of CLUSTER DELETION, one deleted edge) and no two wedges in W share an edge. The edge set E_W is therefore a 2-approximation for MINSTC since $|E_W| = 2|W|$. The randomized *MatchFlipPivot* (MFP) algorithm of Veldt (2022) runs Pivot on $\hat{G} = (V, E - E_W)$ with uniform random pivot nodes. The algorithm was shown to have an expected approximation ratio of 4, and can be derandomized using the techniques of van Zuylen & Williamson (2009) as a black box. We note here that this corresponds to running Algorithm 2 using Pivot Strategy 2.

Our next result improves on this prior work by providing a tighter analysis of MFP to show an improved approximation guarantee of 3. Furthermore, we prove that our simple new degree-based pivoting strategy also provides a deterministic 3-approximation. This is significant given that the bottleneck of the previous deterministic MFP algorithm was computing and updating N_k and B_k values.

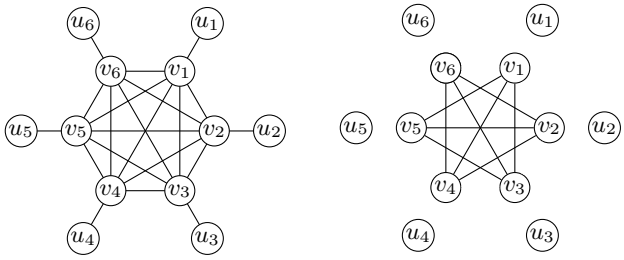
Theorem 3.2. *When using Pivot Strategy 1 or 2 on $\hat{G}$, Algorithm 2 is a deterministic 3-approximation for* CLUSTER DELETION. *When selecting pivots uniformly at random, it is a randomized 3-approximation algorithm.*

Proof. Let m_W denote the number of weak edges between clusters, and m_S the number of other edges between clusters. The three nodes in any open wedge $W \in \mathcal{W}$ must be separated into at least two clusters. Thus, at least one of the two edges of every wedge in W is between clusters. This means that $m_W \geq |E_W|/2$. As in Lemma 3.1, we let $\mathcal{B}$ be the set of edges in $\hat{G}$ between clusters and $\mathcal{N}$ be the set of non-edges in $\hat{G}$ inside clusters. Note then that $m_S = |\mathcal{B}|$ and $|\mathcal{N}| = |E_W| - m_W$ because non-edges of $\hat{G}$ inside clusters are all weak edges. By Lemma 3.1, using Pivoting Strategy 1 or 2 on $\hat{G}$ guarantees that

$$m_S = |\mathcal{B}| \leq 2|\mathcal{N}| = 2(|E_W| - m_W).$$

The total number of edges between clusters is thus:

$$\begin{aligned} m_W + m_S &\leq m_W + 2(|E_W| - m_W) \\ &= 2|E_W| - m_W \\ &\leq \frac{3}{2}|E_W| = 3|W| \leq 3\text{OPT}_{CD}. \end{aligned}$$



(a) G (b) $\hat{G}$

Figure 1: The example for Theorem 3.3.

If we select pivot nodes uniformly at random, m_W and m_S become random variables, but coupling Lemma 3.1 with the fact that $m_W \geq |E_W|/2$ for every choice of pivot nodes provides the same guarantee in expectation. □

The following theorem proves that independent of the pivot strategy used, Algorithm 2 cannot have a ratio better than 3.

Theorem 3.3. *The asymptotic ratio of Algorithm 2 is at least three.*

Proof. For any even integer $n \geq 8$, we can construct a graph of n vertices $\{v_1, v_2, \ldots, v_{n/2}, u_1, u_2, \ldots, u_{n/2}\}$. The vertex set $\{v_1, v_2, \ldots, v_{n/2}\}$ is a clique, and for each $i = 1, \ldots, n/2$, the vertex u_i is adjacent to only v_i. See Figure 1 for the example when $n = 12$. The only optimal solution is

$$E_D = \{(v_1, u_1), (v_2, u_2), \ldots, (v_{n/2}, u_{n/2})\}$$

and it has cost $n/2$. On the other hand, it is easy to see that

$$W = \{(v_1, u_2, v_2), (v_2, u_3, v_3), \ldots, (v_{n/2}, u_1, v_1)\}$$

is a maximal edge-disjoint set of open wedges in G. The set E_W is accordingly $\{(u_1, v_1), (u_2, v_2), \ldots, (u_{n/2}, v_{n/2}), (v_1, v_2), (v_2, v_3), \ldots, (v_{n/2}, v_1)\}$. In $\hat{G} = (V, E - E_W)$, for all $i = 1, \ldots, n/2$, the vertex u_i is isolated, and the vertex v_i has $n/2 - 3$ neighbors. When applying Algorithm 2, whatever the pivot strategy is, the first pivot in $\{v_1, v_2, \ldots, v_{n/2}\}$ decides the solution. The solution has cost

$$n/2 + 2(n/2 - 2) = 3n/2 - 4.$$

Thus, the ratio is asymptotically three. □

3.3. Rounding the STC LP Relaxation

Recall that the LP relaxation of the BLP in (2) provides a natural lower bound for both MINSTC and CLUSTER DELETION. Since every basic feasible solution of (2) is half integral, we can obtain a set of variables $\{x_{ij}\}$ in polynomial time with $x_{ij} \in \{0, 1/2, 1\}$ for every $(i, j) \in E$, that

Algorithm 3 STC-LP-ROUND($G = (V, E)$)

1: $\{x_{ij}\}_{ij \in E} \leftarrow$ (half-integral) solution to STC LP
2: Set $E_W \leftarrow \{(i,j) \in E \colon x_{ij} \in \{1/2, 1\}\}$
3: Form $\hat{G} = (V, E - E_W)$
4: Run Pivot($\hat{G}$) `// for some pivot strategy`

minimizes (2). Given this solution, define $E_1 = \{(i,j) \in E \colon x_{ij} = 1\}$ and $E_h = \{(i,j) \in E \colon x_{ij} = 1/2\}$, and note that $E_W = E_1 \cup E_h$ defines an STC-labeling that is a 2-approximation for MINSTC. We also refer to edges in E_h as "half-edges." Define $E_S = \{(i,j) \in E \colon x_{ij} = 0\} = E - E_W$ to be strong edges. Veldt (2022) showed that with randomized pivot nodes, Algorithm 3 has an expected approximation ratio 4, and can be derandomized using the techniques of van Zuylen & Williamson (2009).[2] Mirroring Theorem 3.2, we provide an improved approximation analysis and show that our (simpler and faster) degree-based pivoting also gives a deterministic 3-approximation.

Theorem 3.4. *When using Pivot Strategy 1 or 2 on $\hat{G}$, Algorithm 3 is a deterministic 3-approximation for* CLUSTER DELETION. *When selecting pivots uniformly at random, it is a randomized 3-approximation algorithm.*

Proof. We begin by proving that for every choice of pivot nodes in $\hat{G}$, at most half of the edges in E_h will end up inside the clusters formed by Algorithm 3. Consider an arbitrary choice of pivots. Let $\mathcal{B}_h$ be the set of half-edges between clusters, and $\mathcal{N}_h$ the set of half-edges inside clusters. We claim that the following set of variables is still feasible for the STC LP:

$$\hat{x}_{ij} = \begin{cases} x_{ij} & \text{if } x_{ij} \in \{0,1\}, \\ 1 & \text{if } x_{ij} \in \mathcal{B}_h, \\ 0 & \text{if } x_{ij} \in \mathcal{N}_h. \end{cases}$$

Consider an arbitrary open wedge $(i,j,k) \in \mathcal{W}$. Assume without loss of generality that $x_{ik} \geq x_{jk}$. If $x_{ik} = 1$, then $\hat{x}_{ik} = 1$ and $\hat{x}_{ik} + \hat{x}_{jk} \geq 1$. Otherwise, $x_{ik} = x_{jk} = 1/2$. The three nodes in W must be separated into at least two clusters. Thus, at least one of x_{ik} and x_{jk} is in $\mathcal{B}_h$, and hence $\hat{x}_{ik} + \hat{x}_{jk} \geq 1$. From the optimality of $\{x_{ij}\}$ and the feasibility of $\{\hat{x}_{ij}\}$ it follows that

$$\sum_{(i,j) \in E} x_{ij} \leq \sum_{(i,j) \in E} \hat{x}_{ij} = \sum_{(i,j) \in E} x_{ij} + \frac{|\mathcal{B}_h|}{2} - \frac{|\mathcal{N}_h|}{2}.$$

Thus, $|\mathcal{B}_h| \geq |\mathcal{N}_h|$ and $|\mathcal{N}_h| \leq (|\mathcal{B}_h| + |\mathcal{N}_h|)/2 = |E_h|/2$.

[2]The deterministic STC-LP algorithm of Veldt (2022) incorporates LP values $\{x_{ij}\}$ more directly in choosing pivot nodes and is different from running Algorithm 3 with Pivot Strategy 2. The latter provides a simplified and unified approach for rounding both types of CLUSTER DELETION lower bounds we consider.

Let m_1 and m_S be the numbers of edges between clusters that are from the sets E_1 and E_S, respectively. Note that $|\mathcal{B}| = m_S$ and $|\mathcal{N}| = |E_1| - m_1 + |\mathcal{N}_h|$. Using Pivot Strategy 1 or 2, Lemma 3.1 implies that $m_S \leq 2(|E_1| - m_1 + |\mathcal{N}_h|)$. We can then bound the total number of edges between clusters in G:

$$\begin{aligned} m_1 + |\mathcal{B}_h| + m_S &\leq m_1 + |\mathcal{B}_h| + 2(|E_1| - m_1 + |\mathcal{N}_h|) \\ &= 2|E_1| - m_1 + |\mathcal{B}_h| + 2|\mathcal{N}_h| \\ &\leq 2|E_1| + \frac{3}{2}|E_h| \\ &= \sum_{(i,j) \in E_1} 2x_{ij} + \sum_{(i,j) \in E_h} 3x_{ij} \\ &\leq 3 \sum_{(i,j) \in E} x_{ij} \\ &\leq 3\text{OPT}_{CD}. \end{aligned}$$

Lemma 3.1 can similarly be used to show the same result in expectation for random pivot nodes. □

4. Faster Algorithms for Lower Bounds

In addition to our improved approximation analysis, we provide faster algorithms for computing a maximal edge-disjoint sets of open wedges and for solving the STC LP.

4.1. Maximal Edge-Disjoint Open Wedge Set

A simple existing approach for finding a maximal edge-disjoint open wedge set is to iterate through each node $k \in V$, and then iterate through pairs $\{i,j\}$ of neighbors of k. If (i,j,k) is an open wedge and edge-disjoint from previously explored open wedges, we can add it to a growing set W of open wedges. We maintain a list E_W of edges that come from wedges in W. This can be implemented in $O(\sum_{k \in V} d_k^2)$-time, which is always larger than $|\mathcal{W}|$ and can be as large as $O(nm)$. While this is already fast in practice, we can further improve the theoretical runtime.

Lemma 4.1. *A maximal edge-disjoint set of open wedges can be found in $O(m^{1.5})$ time and $O(m)$ space.*

The appendix provides pseudocode and a full analysis for an algorithm satisfying these bounds. Similar to the previous approach, our procedure starts by iterating through nodes $k \in V$ and then iterates through pairs of neighbors of k. The key observation is that as soon as we encounter an open wedge (i,j,k) and add its two edges (i,k) and (j,k) to E_W, we can effectively "delete" these edges and avoid exploring triplets involving them in future iterations. Since $|W|$ is always at most $m/2$, the total amount of work for adding edges to E_W and then deleting them is bounded by $O(m)$. Now, when visiting two neighbors $\{i,j\}$ of k

we may find that $\{i, j, k\}$ is a triangle rather than an open wedge. We therefore have nothing to add to W and no edges to delete. However, the amount of work that goes into finding these unwanted triangles is bounded in terms of the number of triangles in the graph, which is known to be at most $O(m^{1.5})$. The result is inspired by the recent work of Cao et al. (2024), who provided the same type of bound for finding a maximal set of disjoint *open triangles* in a complete signed graph. See the appendix for further details on the similarities and differences between these problems and approaches.

4.2. Combinatorial Solver for the STC LP

Although the STC LP rounding algorithm produces the same 3-approximation guarantee as MFP, the LP relaxation produces a tighter lower bound for CLUSTER DELETION that can be used to obtain better a posteriori approximation guarantees (approximate CLUSTER DELETION solution $|E_D|$ divided by the lower bound) in practice. However, previous implementations rely on simply applying black-box LP solvers, which become a bottleneck both in terms of runtime and memory requirements (Veldt, 2022). In this section we present a faster and purely combinatorial approach for solving the LP by reducing it to a minimum s-t cut problem. This can be accomplished by first proving that the half-integral STC LP can be cast as a so-called monotone IP2 problem—an integer program with two variables per constraint with opposite signed coefficients. This can in turn be cast as a maximum closure problem (Picard, 1976) and then reduced to a maximum s-t flow problem following the approach of Hochbaum (2021). The reduction we present in this section merges several of these steps in order to provide a simplified and more direct reduction for the STC LP.

Converting to BLP. The STC LP relaxation is obtained by replacing constraint $x_{ij} \in \{0, 1\}$ in (2) with $x_{ij} \geq 0$. Since every basic feasible solution of the LP is half-integral, we can equivalently optimize over variables $x_{ij} \in \{0, 1/2, 1\}$. We will show that this is equivalent to the following BLP:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} \frac{1}{2} y_{ij} + \frac{1}{2}(1 - z_{ij}) \\ \text{s.t.} \quad & \left.\begin{aligned} z_{ik} &\leq y_{jk} \\ z_{jk} &\leq y_{ik} \end{aligned}\right\} \quad \forall (i, j, k) \in \mathcal{W}_k \\ & z_{ij}, y_{ij} \in \{0, 1\} \quad \forall (i, j) \in E. \end{aligned} \tag{4}$$

We prove the following result.

Lemma 4.2. *If $\{y_{ij}, z_{ij}\}_{ij\in E}$ is feasible for (4), then*

$$x_{ij} = \frac{1}{2}(y_{ij} - z_{ij} + 1) \quad \forall (i, j) \in E \tag{5}$$

defines a feasible half-integral solution for the STC LP with the same objective value. Conversely, if $\{x_{ij}\}_{ij\in E}$ is a feasible half-integral solution to the STC LP, the variables

$$(y_{ij}, z_{ij}) = \begin{cases} (0, 1) & \text{if } x_{ij} = 0 \\ (1, 1) & \text{if } x_{ij} = 1/2 \\ (1, 0) & \text{if } x_{ij} = 1 \end{cases} \quad \forall (i, j) \in E \tag{6}$$

are feasible for (4) and have the same objective value.

Proof. First, suppose that $\{y_{ij}, z_{ij}\}_{ij\in E}$ is feasible for (4). Then for an open wedge (i, j, k) we have

$$\begin{aligned} x_{ik} + x_{jk} &= \frac{1}{2}(y_{ik} - z_{ik} + 1) + \frac{1}{2}(y_{jk} - z_{jk} + 1) \\ &= \frac{1}{2}(y_{jk} - z_{ik}) + \frac{1}{2}(y_{ik} - z_{jk}) + 1 \\ &\geq 1. \end{aligned}$$

Thus, $\{x_{ij}\}_{ij\in E}$ is feasible for the STC LP. It is obviously half-integral and the objective value is the same.

Now suppose that $\{x_{ij}\}_{ij\in E}$ is a feasible half-integral solution to the STC LP. Consider an open wedge (i, j, k). Assume without loss of generality that $x_{ik} \geq x_{jk}$.

- If $x_{ik} = 1/2$, then $x_{jk} = 1/2$. By Eq. (6), we have $y_{ik} = z_{jk} = 1$ and $y_{jk} = z_{ik} = 1$.
- If $x_{ik} = 1$, then by Eq. (6), we have $y_{ik} = 1 \geq z_{jk}$ and $y_{jk} \geq z_{ik} = 0$.

Thus, $\{y_{ij}, z_{ij}\}_{ij\in E}$, which is integral, is feasible for (4). Since $\frac{1}{2}(y_{ij} + 1 - z_{ij}) = x_{ij}$ for all the possible values of x_{ij}, the objective value is the same. □

We can therefore turn our attention to solving (4), and use Eq. (5) to convert back to a solution to the STC LP.

Casting as a minimum s-t cut problem. The BLP in (4) is equivalent to a minimum s-t cut problem on a graph G_{st}. To construct the graph, we introduce a source node s and a sink node t. Then for each $(i, j) \in E$, we introduce a node Y_{ij} and a node Z_{ij}, then construct two directed edges (s, Z_{ij}) and (Y_{ij}, t) each with weight $1/2$. Finally, for every $(i, j, k) \in \mathcal{W}_k$, we introduce two infinite-weight directed edges: (Z_{ik}, Y_{jk}) and (Z_{jk}, Y_{ik}).

Every s-t cut in G_{st} corresponds to a feasible solution to (4), where the weight of cut edges in G_{st} equals the resulting objective score for (4). In more detail, nodes Y_{ij} and Z_{ij} correspond to binary variables y_{ij} and z_{ij} in (4). Setting a binary variable to 1 corresponds to placing its node on the s-side of the cut. For example, setting $y_{ij} = 1$ means placing Y_{ij} on the s-side, which cuts the edge (Y_{ij}, t). This contributes a $1/2$ to the cut penalty of G_{st}, just as setting $y_{ij} = 1$ contributes $1/2$ to the objective of (4). A similar penalty arises from setting $z_{ij} = 0$, which is equivalent to

placing Z_{ij} on the t-side of an s-t cut in G_{st}. The infinite weight edges in G_{st} encode the constraints of (4). The edge (Z_{ik}, Y_{jk}) has infinite weight to ensure that if Z_{ik} is on the s-side of the cut, then Y_{jk} is as well, just as $z_{ik} = 1$ forces $y_{jk} = 1$, required by the constraint $z_{ik} \leq y_{jk}$.

To understand how this relates to MINSTC, note that an edge (s, Z_{ij}) encourages z_{ij} to be 1 and edge (Y_{ij}, t) encourages y_{ij} to be zero. If both these preferences are satisfied, then $x_{ij} = 0$ meaning that x_{ij} is a strong edge. If neither preference is satisfied, then $x_{ij} = 1$ and the edge is weak, whereas satisfying one preference but not the other leads to $x_{ij} = 1/2$. The reason these preferences typically cannot all be satisfied is because an edge (Z_{jk}, Y_{ik}) indicates that if $z_{jk} = 1$, this forces $y_{ik} = 1$.

4.3. Runtime and Space Analysis

We briefly summarize several improvements in runtime and space requirements that are obtained using our new techniques. More details for proving these bounds are included in the appendix. Lemma 4.1 improves the runtime for computing the MFP lower bound to $O(m^{1.5})$. This is always at least as fast as the previous $O(mn)$ runtime and is strictly faster for sparse graphs. The previous deterministic pivoting scheme (Pivot Strategy 2) has space and runtime requirements that are $\Omega(m + |\mathcal{W}|)$, which can be $\Omega(n^3)$ in the worst case, whereas degree-based pivoting can be implemented in $O(m)$ time and space. The most expensive step of Algorithm 3 is solving the STC LP. Hence, the runtime for our combinatorial STC LP solver is also the asymptotic runtime for Algorithm 3. Using our reduction to minimum s-t cut, we can can get a randomized solver that runs in $(m + |\mathcal{W}|)^{1+o(1)}$ time by applying recent nearly linear time algorithms for maximum s-t flows (Chen et al., 2022). Using the algorithm of Goldberg & Rao (1998) we can get a deterministic algorithm with runtime $\tilde{O}(\min\{(m+|\mathcal{W}|)^{1.5}, (m+|\mathcal{W}|)\cdot m^{2/3}\})$. For comparison, even the best recent theoretical solvers for general LPs would lead to runtimes that are $\Omega((m + |\mathcal{W}|)^2)$ (van den Brand, 2020; Jiang et al., 2021; Cohen et al., 2021).

5. Experimental Results

Veldt (2022) previously showed experimental results for the *randomized* variants of Algorithms 2 and 3 on a large collection of real-world graphs. For these experiments, the STC LP was solved using Gurobi optimization software. For our work we implement both deterministic schemes (Pivot Strategies 1 and 2) and compare them against each other and the randomized variant. We also show that our combinatorial approach for solving the STC LP is much faster and scales to much larger graphs than using a black-box LP solver. Our algorithms are implemented in Julia. Our experiments were run on a laptop with 16 GB of RAM.

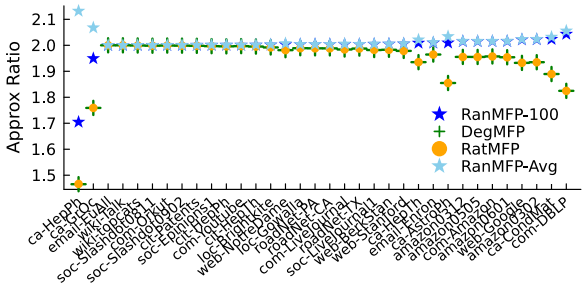


Figure 2: Approximation ratios ($|E_D|/|W|$) for MFP.

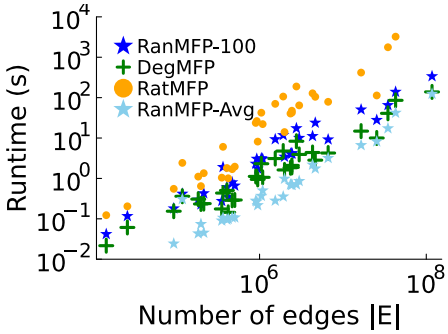


Figure 3: Runtimes of the MFP algorithms using different pivoting strategies. Each point represents one graph.

For the most direct comparison with previous work (Veldt, 2022), we consider the same collection of large graphs from the SNAP Repository (Leskovec & Krevl, 2014), the largest of which (soc-Livejournal1) has 4.2 million nodes and 4.7 billion edges. We also run experiments on one even larger graph (com-Orkut) with 117.2 billion edges. The appendix includes additional details about datasets, implementations, and experimental results.

Comparing pivot strategies. Our degree-based pivoting strategy is fast and practical. In addition to enjoying a deterministic approximation guarantee, it is comparable in speed to choosing random pivot nodes, while achieving much better approximations. Figure 2 shows approximation ratios and Figure 3 shows runtimes achieved by MFP with degree-based pivots (DegMFP), pivots that minimize the ratio $|B_k|/|N_k|$ (RatMFP), and two different approaches to using random pivots. RanMFP-100 runs the random pivot strategy 100 times and takes the best solution found. This is a natural strategy to use since running randomized pivot once is very fast. RanMFP-Avg represents the average performance of the algorithm over these 100 trials.

DegMFP is almost identical to RatMFP in terms of approximation ratio, but is faster by an order of magnitude or more. DegMFP finds better solutions that RanMFP-100 and RanMFP-Avg, and is faster than RanMFP-100. Choosing random pivots once is faster (see runtimes for RanMFP-

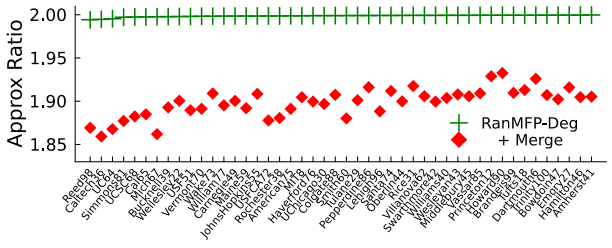


Figure 4: Improved approximation ratios when incorporating a cluster merging step after DegMFP.

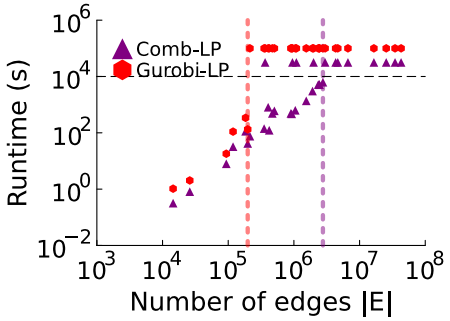


Figure 5: Runtimes of two different solvers for the STC LP. Each point represents a graph. Points above the black dashed line indicate graphs for which the given STC LP solver did not find a solution. The two vertical dashed lines indicate the size of the largest graph (in terms of edges) for which each method was able to successfully solve the LP.

Avg), but especially for larger graphs DegMFP has comparable runtimes. A benefit of DegMFP is that by choosing high-degree nodes, it terminates in fewer pivot steps.

Most approximation ratios achieved by MFP are very close to 2. This can be explained by noting that the method labels a large percentage of edges as weak—between 63.4% and 99.7% for graphs in Figure 2. As a result, MFP deletes nearly all edges for some graphs, which is roughly a 2-approximation since $|E| \approx 2|W|$. It is especially interesting to observe the behavior of different pivoting strategies when fewer edges are labeled weak and approximations factors deviate more from 2 (left- and right-most graphs in Figure 2). In these cases, RanMFP tends to have approximations that are worse than 2, while the deterministic schemes perform the best in these cases and detect more meaningful clusters. This highlights the utility of having a very fast deterministic rounding scheme for CLUSTER DELETION algorithms.

Cluster merging heuristic. Figure 4 shows results for DegMFP on Facebook100 graphs (Traud et al., 2012) with up to 351k edges. For these graphs, finding an edge-disjoint open wedge set labels between 99.6% and 99.95% of edges as weak. MFP essentially achieves a 2-approximation by deleting nearly all edges. We also implement a heuristic for checking when clusters output by MFP can be merged into larger cliques (see appendix for details). Our current implementation is a proof of concept and not optimized for runtime. Nevertheless, this leads to noticeably better approximation ratios for these graphs, as well as for other graphs where a smaller percentage of edges are labeled weak (see appendix). Developing more scalable techniques for improving MFP is a promising direction for future research.

STC LP solvers. Our combinatorial solver for the STC LP enables us to run Algorithm 3 more quickly and on a much larger scale than was previously possible. Figure 5 shows runtimes for solving this LP using our combinatorial min-cut approach (Comb-LP) versus using general-purpose Gurobi optimization software (Gurobi-LP). The resulting objective score is the same for both since they are both finding an optimal solution for the LP. The main bottleneck of both algorithms is memory. For smaller graphs where both algorithms terminate without memory overflow, our combinatorial approach is roughly twice as fast. Overall, Gurobi-LP can only solve the STC LP on 6 of 34 graphs, while Comb-LP can solve it for 21 of 34. The largest graph for which Gurobi finds a solution has 34,546 nodes and 420,877 edges, while our combinatorial approach was able to find solutions for many of the larger graphs, the largest of which has 1,971,281 nodes and 2,766,607 edges. Thus, in addition to being faster, this approach allows us to tackle problems that are an order of magnitude larger.

6. Conclusion

We have developed two combinatorial approximation algorithms for the CLUSTER DELETION problem, with a common degree-based pivoting strategy for derandomization. We managed to show that both algorithms have an approximation guarantee of 3. While the analysis for MatchFlipPivot is tight, it is not for the other. One open question is whether the ratio can be improved by a tighter analysis. Another major open problem is obtaining more efficient implementations of our algorithms. For example, can we exploit the special structure of the minimum s-t cut problem to avoid calling an off-the-shelf algorithm? Another interesting direction for future work is to see whether we can adapt these ideas to obtain faster approximation algorithms for CLUSTER EDITING.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Ailon, N., Charikar, M., and Newman, A. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM*, 55(5):23, 2008. doi: 10.1145/1411509.1411513.

Assadi, S., Shah, V., and Wang, C. Streaming algorithms and lower bounds for estimating correlation clustering cost. In *Advances in Neural Information Processing Systems*, NeurIPS '23, 2023.

Bansal, N., Blum, A., and Chawla, S. Correlation clustering. *Machine Learning*, 56:89–113, 2004. doi: 10.1023/B:MACH.0000033116.57574.95.

Bathie, G., Bousquet, N., Cao, Y., Ke, Y., and Pierron, T. (Sub)linear kernels for edge modification problems toward structured graph classes. *Algorithmica*, 84:3338–3364, 2022. doi: 10.1007/s00453-022-00969-1.

Ben-Dor, A., Shamir, R., and Yakhini, Z. Clustering gene expression patterns. *Journal of computational biology*, 6 (3-4):281–297, 1999. doi: 10.1089/106652799318274.

Bengali, V. and Veldt, N. Faster approximation algorithms for parameterized graph clustering and edge labeling. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, pp. 78–87, New York, NY, USA, 2023. ACM. doi: 10.1145/3583780.3614878.

Böcker, S. and Damaschke, P. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011. doi: 10.1016/j.ipl.2011.05.003.

Bun, M., Elias, M., and Kulkarni, J. Differentially private correlation clustering. In *International Conference on Machine Learning*, ICML '21, pp. 1136–1146, 2021.

Cao, N., Huang, S.-E., and Su, H.-H. Breaking 3-factor approximation for correlation clustering in polylogarithmic rounds. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 4124–4154. SIAM, 2024. doi: 10.1137/1.9781611977912.143.

Charikar, M., Guruswami, V., and Wirth, A. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360 – 383, 2005. doi: 10.1016/j.jcss.2004.10.012. Learning Theory 2003.

Chawla, S., Makarychev, K., Schramm, T., and Yaroslavtsev, G. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pp. 219–228. ACM, 2015. doi: 10.1145/2746539.2746604.

Chen, L., Kyng, R., Liu, Y. P., Peng, R., Gutenberg, M. P., and Sachdeva, S. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 612–623. IEEE, 2022. doi: 10.1109/FOCS54457.2022.00064.

Cohen, M. B., Lee, Y. T., and Song, Z. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021. doi: 10.1145/3424305.

Cohen-Addad, V., Lattanzi, S., Mitrović, S., Norouzi-Fard, A., Parotsidis, N., and Tarnawski, J. Correlation clustering in constant many parallel rounds. In *International Conference on Machine Learning*, pp. 2069–2078. PMLR, 2021.

Damaschke, P. Bounded-degree techniques accelerate some parameterized graph algorithms. In *Parameterized and Exact Computation: 4th International Workshop*, IWPEC '09, pp. 98–109. Springer, 2009. doi: 10.1007/978-3-642-11269-0_8.

Davies, S., Moseley, B., and Newman, H. Fast combinatorial algorithms for min max correlation clustering. In *International Conference on Machine Learning*, ICML '23, 2023.

Davis, T. A. and Hu, Y. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), dec 2011. ISSN 0098-3500. doi: 10.1145/2049662.2049663.

Dessmark, A., Jansson, J., Lingas, A., Lundell, E.-M., and Person, M. On the approximability of maximum and minimum edge clique partition problems. *International Journal of Foundations of Computer Science*, 18(02):217–226, 2007.

Easley, D. and Kleinberg, J. *Networks, crowds, and markets*. Cambridge university press Cambridge, 2010. ISBN 9781139490306.

Fortunato, S. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

Gao, Y., Hare, D. R., and Nastos, J. The cluster deletion problem for cographs. *Discrete Mathematics*, 313(23):2763–2771, 2013. doi: 10.1016/j.disc.2013.08.017.

Goldberg, A. V. and Rao, S. Beyond the flow decomposition barrier. *Journal of the ACM (JACM)*, 45(5):783–797, 1998. doi: 10.1145/290179.290181.

Gramm, J., Guo, J., Hüffner, F., and Niedermeier, R. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, Jul 2005. doi: 10.1007/s00224-004-1178-y.

Granovetter, M. S. The strength of weak ties. *American journal of sociology*, 78(6):1360–1380, 1973. doi: 10.1086/225469.

Grüttemeier, N. and Komusiewicz, C. On the relation of strong triadic closure and cluster deletion. *Algorithmica*, 82(4):853–880, 2020. doi: 10.1007/s00453-019-00617-1.

Hochbaum, D. S. Applications and efficient algorithms for integer programming problems on monotone constraints. *Networks*, 77(1):21–49, January 2021. doi: 10.1002/net.21983.

Jafarov, J., Kalhan, S., Makarychev, K., and Makarychev, Y. Correlation clustering with asymmetric classification errors. In *International Conference on Machine Learning*, ICML '20, pp. 4641–4650, 2020.

Jafarov, J., Kalhan, S., Makarychev, K., and Makarychev, Y. Local correlation clustering with asymmetric classification errors. In *International Conference on Machine Learning*, ICML '21, pp. 4677–4686, 2021.

Jiang, S., Song, Z., Weinstein, O., and Zhang, H. A faster algorithm for solving general LPs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pp. 823–832, 2021. ISBN 9781450380539. doi: 10.1145/3406325.3451058.

Konstantinidis, A. L., Nikolopoulos, S. D., and Papadopoulos, C. Strong triadic closure in cographs and graphs of low maximum degree. *Theoretical Computer Science*, 740:76–84, 2018. doi: 10.1016/j.tcs.2018.05.012.

Le, V. B. Gallai graphs and anti-gallai graphs. *Discrete Mathematics*, 159(1-3):179–189, 1996. doi: 10.1016/0012-365X(95)00109-A.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

Nemhauser, G. L. and Trotter, L. E. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975. doi: 10.1007/BF01580444.

Picard, J.-C. Maximal Closure of a Graph and Applications to Combinatorial Problems. *Management Science*, 22 (11):1268–1272, 1976. doi: 10.1287/mnsc.22.11.1268.

Puleo, G. and Milenkovic, O. Correlation clustering with constrained cluster sizes and extended weights bounds. *SIAM Journal on Optimization*, 25(3):1857–1872, 2015. doi: 10.1137/140994198.

Shamir, R., Sharan, R., and Tsur, D. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2): 173–182, 2004. doi: 10.1016/j.dam.2004.01.007.

Shi, J. and Malik, J. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000. doi: 10.1109/34.868688.

Sintos, S. and Tsaparas, P. Using strong triadic closure to characterize ties in social networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '14, pp. 1466–1475, 2014. doi: 10.1145/2623330.2623664.

Stein, D., Di Gregorio, S., and Andres, B. Partial optimality in cubic correlation clustering. In *International Conference on Machine Learning*, ICML '23, 2023.

Traud, A. L., Mucha, P. J., and Porter, M. A. Social structure of Facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012.

van den Brand, J. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 259–278. SIAM, 2020.

van den Brand, J., Lee, Y. T., Liu, Y. P., Saranurak, T., Sidford, A., Song, Z., and Wang, D. Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pp. 859–869, 2021. doi: 10.1145/3406325.3451108.

van Zuylen, A. and Williamson, D. P. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research*, 34(3): 594–620, 2009. doi: 10.1287/moor.1090.0385.

Veldt, N. Correlation clustering via strong triadic closure labeling: Fast approximation algorithms and practical lower bounds. In *International Conference on Machine Learning*, pp. 22060–22083. PMLR, 2022.

Veldt, N., Gleich, D. F., and Wirth, A. A correlation clustering framework for community detection. In *Proceedings of the 2018 World Wide Web Conference*, pp. 439–448, 2018. doi: 10.1145/3178876.3186110.

Algorithm 4 Faster maximal edge-disjoint open wedge set

Input. Graph $G = (V, E)$
Output. Maximal edge-disjoint set of open wedges $W \subseteq \mathcal{W}(G)$
$W = \emptyset$ `// edge-disjoint open wedge set`
$E_W = \emptyset$ `//edges within W, will be labeled "weak"`
for $v \in V$ **do**
  $S_v \leftarrow$ array of neighbors u of v such that $(u, v) \notin E_W$
  if $|S_v| < 2$ **then**
    continue `// no open wedges centered at` v
  end if
  $next = [2, 3, \cdots, |S_v|, \text{NIL}]$ `//` $next[t]$ `= neighbor of` v `to visit after` $S_v[t]$
  $i = 1; j = 2; j_{old} = 1$
  $finished =$ FALSE
  while $finished =$ FALSE **do**
    $u = S_v[i]; w = S_v[j]$ `//` (u, w, v) `is the current wedge under consideration`
    if $(u, w) \in E$ **then**
      `// triangle found; do normal increment`
      $[i, j, j_{old}, finished] = \textsc{TriangleIncrement}(i, j, j_{old}, next, finished)$
    else
      `// open wedge found; do special increment`
      $E_W \leftarrow E_W \cup \{(u, v), (w, v)\}$
      $W \leftarrow W \cup \{(u, w, v)\}$
      $[i, j, j_{old}, next, finished] = \textsc{OpenWedgeIncrement}(i, j, j_{old}, next, finished)$
    end if
  end while
end for
Return W

A. Finding a Maximal Open Wedge Set

Algorithm 4 is pseudocode for our $O(m^{1.5})$-time algorithm for finding a maximal edge-disjoint set of open wedges.

Related work. Our algorithm is loosely inspired by the recent work of (Cao et al., 2024), who provided the same type bound for finding a maximal set of open triangles (i.e., $\{+, +, -\}$ triangles) in a complete signed graph. This is equivalent to finding a *pair*-disjoint set of open wedges in a graph $G = (V, E)$, which is more restrictive than finding an *edge*-disjoint set. The setting considered by Cao et al. (2024) is also slightly different in that they associate lengths to different edges and restrict their search to open triangles satisfying certain length restrictions. Nevertheless, two key ideas remain the same: (1) once we add an open wedge to W, we never have to explore triplets of nodes involving its edges ever again, and (2) the amount of time we spend exploring node triplets that turn out to be triangles can be bounded in terms of number of triangles in the graph, which is $O(m^{1.5})$.

Algorithm explanation. For our analysis we fix an ordering of the vertices $V = \{v_1, v_2, \ldots v_n\}$. We also assume that for each node $v_i, i = 1, \ldots, n$, we have an array that stores the neighborhood of v_i in this same order. The basic structure of Algorithm 4 is outlined in the main text: for each node $v \in V$, iterate through pairs of neighbors $\{u, w\}$ of v, checking each time if (u, w, v) is an open wedge that can be added to a growing edge-disjoint open wedge set W. The key idea is that we must find a way to effectively "delete" the edges (u, v) and (w, v) if we add (u, w, v) to W, so that we never waste time visiting another open wedge that involves either of these edges. This will ensure that we can charge all of the work done by the algorithm to an iteration where we add a new open wedge to W (which happens $O(m)$ times), or to an iteration where we "visit" a triangle (which will happen $O(m^{1.5})$ times).

When we first visit a node $v \in V$, we extract an array of neighbors S_v (ordered by node label) with the property that for every node u in S_v, (u, v) is not in E_W (the set of edges from the open wedge set W). This is the first way in which we "delete" edges adjacent to v that we no longer wish to consider. As we iterate through pairs of nodes of v, we may encounter new open wedges, leading to other edges we must "delete" and skip over as we explore pairs of neighbors of v. We keep track of edges to skip over using an array $next$, where $next[t]$ represents the next index in S_v to visit—in other words, the neighbor

Algorithm 5 $[i, j, j_{old}, \textit{finished}] = \text{TRIANGLEINCREMENT}(i, j, j_{old}, \textit{next}, \textit{finished})$

if $\textit{next}[j] \neq$ NIL **then**
  $j_{old} = j$ // record j_{old} such that $\textit{next}[j_{old}] == j$
  $j = \textit{next}[j]$ // increment j
else if $\textit{next}[j] ==$ NIL and $\textit{next}[i] \neq j$ **then**
  $i = \textit{next}[i]$ // increment i
  $j = \textit{next}[i]$ // i comes right after
  $j_{old} = i$
else
  $\textit{finished} =$ TRUE // done exploring neighbors since $j = \textit{next}[i]$ and $\textit{next}[j] ==$ NIL
end if
Return $i, j, j_{old}, \textit{finished}$

Algorithm 6 $[i, j, j_{old}, \textit{next}, \textit{finished}] = \text{OPENWEDGEINCREMENT}(i, j, j_{old}, \textit{next}, \textit{finished})$

$\textit{next}[j_{old}] = \textit{next}[j]$ // skip past (v, v_j) in the future
$i = \textit{next}[i]$ // increment i
if $i ==$ NIL or $\textit{next}[i] ==$ NIL **then**
  $\textit{finished} =$ TRUE
else
  $j = \textit{next}[i]$
  $j_{old} = i$
end if
Return $i, j, j_{old}, \textit{next}, \textit{finished}$

of v that we should visit directly after visiting the neighbor $S_v[t]$. The array is initialized to $\textit{next} = [2, 3, \ldots, |S_v|, \text{NIL}]$ when we begin iterating through neighbors of v, which signifies that we default to visiting node $S_v[i+1]$ after visiting $S_v[i]$. Including NIL at the end of the array tells us when we have reached the end of v's neighbor list.

We maintain two indices i and j that point to distinct neighbors of v: $S_v[i]$ and $S_v[j]$. If $\{S_v[i], S_v[j], v\}$ is a triangle, no edges are deleted and we just update indices i and j with help from $\textit{next}$ to determine which new pair of neighbors of v to explore next. If however this node triplet defines an open wedge, then we delete $(S_v[i], v)$ and $(S_v[j], v)$ by updating $\textit{next}$ so that we skip over $S_v[i]$ and $S_v[j]$ (see Algorithm 6) as we continue to iterate over pairs of neighbors of v. With this careful update, we ensure that every time we consider a new pair of neighbors of v, it defines either a new open wedge to add to W or a triangle.

Runtime bound. The time it takes to initialize S_v and $\textit{next}$ for each $v \in V$ is $O(d_v)$, where d_v is the degree of v. The overall runtime for these steps (and all of the work done in lines 6–12 of Algorithm 6) is therefore $O(m)$. The bottleneck of the algorithm comes from the while loop in lines 13–24. The total number of times we call TRIANGLEINCREMENT (line 17) is $O(m^{1.5})$, since each triangle is visited at most 3 times (once for every node in the triangle) over the course of the algorithm. Each call to TRIANGLEINCREMENT takes $O(1)$ time. We call OPENWEDGEINCREMENT at most $O(m)$ times, which again takes $O(1)$ time each time. Finally, in order to figure out which of these two increment subroutines to call, we must check if there is an edge between two given neighbors of v (line 15). This can be done in $O(1)$ time and $O(n^2)$ space by storing the full adjacency matrix. We improve this to $O(1)$ time and $O(m)$ space by storing all of the edges in a hash table that is queried to check adjacency.

B. Theoretical Runtime Analysis

Here we provide additional details behind the runtime and space analysis in Section 4.3.

MFP algorithms. Appendix A provided a detailed analysis to show that we can find a maximal edge-disjoint set of open wedges in $O(m^{1.5})$ time and $O(m)$ space. This is always at least as fast as the previous runtime of $O(mn)$ and is strictly better for sparse graphs. Computing this lower bound is the most expensive step for the randomized variant of MFP, so our work improves the best runtime for this randomized variant, while maintaining the same $O(m)$ space requirement.

Algorithm 7 Combinatorial solver for STC LP

Input. Graph $G = (V, E)$
Output. Optimal solution to STC LP
Initialize graph G_{st} with two nodes $V_{st} = \{s, t\}$
for $(i, j) \in E$ **do**
  Add nodes Z_{ij}, Y_{ij} to V_{st}
  Add edges (s, Z_{ij}) and (Y_{ij}, t) with weight $1/2$
end for
for $(i, j, k) \in \mathcal{W}_k$ **do**
  Add edges (Z_{ik}, Y_{jk}) and (Z_{jk}, Y_{ik}) with weight ∞
end for
Find min s-t cut set $S \subseteq V_{st}$ of G_{st}
for $(i, j) \in E$ **do**
  Set $y_{ij} = \chi(Y_{ij} \in S)$, $z_{ij} = \chi(Z_{ij} \in S)$
  Set $x_{ij} = \frac{1}{2}(y_{ij} + 1 - z_{ij})$
end for
Return $\{x_{ij}\}_{(i,j)\in E}$

For deterministic MFP, our runtime (and space) improvement is even more dramatic, thanks to our simplified deterministic rounding scheme. The previous deterministic rounding scheme (Pivot Strategy 2) requires identifying all open wedges and storing a map back and forth between open wedges and edges they contain. This means that the space and runtime requirements are both $\Omega(|W| + m)$. In contrast, pivoting based on degrees can be implemented in $O(m)$ time and space. This is done by computing node degrees and then placing nodes in one of n bins based on their degree. We can then iterate through bins, greedily selecting a node with the highest degree and forming a cluster around it. For each node u added to a cluster, we can update the degrees of its neighbors, and update their location in the bins, in $O(d_u)$ time. Overall the runtime is $O(m)$. With this new approach, we now have randomized and deterministic variants of MFP with a runtime of $O(m^{1.5})$ and space constraint of $O(m)$.

STC LP solvers. Algorithm 7 is the pseudocode for the combinatorial procedure for solving the STC LP relaxation given in Section 4.2. When finding a minimum s-t cut of G_{st}, we can double the weight of edges adjacent to s and t and replace infinite weight edges with edges of weight $m + 1$, without changing the optimal solution. This means we are solving an s-t cut problem in a directed graph with $2m + 2$ nodes and and $2m + 2|\mathcal{W}|$ edges, where all edge weights are integers that are polynomially bounded in the input size. Using the nearly linear time maximum s-t flow solver of Chen et al. (2022), the STC LP relaxation can be solved in $(m + |\mathcal{W}|)^{1+o(1)}$-time. We can also obtain a runtime of $\tilde{O}(|\mathcal{W}| + m^{1.5})$ using the results of van den Brand et al. (2021), which is slightly better if $m^{1.5} = O(|\mathcal{W}|)$. We get a deterministic algorithm with a runtime of $\tilde{O}(\min\{(m + |\mathcal{W}|)^{1.5}, (m + |\mathcal{W}|) \cdot m^{2/3}\}$ by using the well-known maximum s-t flow algorithm of Goldberg & Rao (1998). Note that the relative size of $(m + |\mathcal{W}|)^{1/2}$ and $m^{2/3}$ depends on the graph. For example, it is possible for $|\mathcal{W}|$ to be very small (even zero), in which case the former is better. It is also possible to have $|\mathcal{W}| = \Theta(n^3)$ and $m = \Theta(n^2)$, in which case $m^{2/3}$ is smaller.

There are several recent theoretical algorithms for solving an LP of the form $\min_{\mathbf{Ax}=\mathbf{b};\mathbf{x}\geq 0} \mathbf{c}^T\mathbf{x}$, that run in current matrix multiplication time (van den Brand, 2020; Jiang et al., 2021; Cohen et al., 2021). When written in this format, the STC LP has $(m + |\mathcal{W}|)$ constraints, and applying these solvers yields runtimes that are all at least $\Omega((m + |\mathcal{W}|)^2)$.

C. Additional Implementation and Experimental Details

Implementation details. Section 4.3 and Appendix B provide details for the best runtimes that can be obtained in theory. As is often the case, there is a gap between the best theoretical algorithms and the most practical approaches. As a key example, the current best algorithms for maximum s-t flows do not come with practical implementations. For solving minimum s-t cut problems in practice, we instead apply a fast Julia implementation of the push-relabel algorithm for maximum s-t flows. When computing a maximal edge-disjoint set of open wedges in practice, we apply the simpler approach that iterates through nodes and then pairs of neighbors of nodes to check for open wedges. In our experimental results, computing this lower bound tends to be very fast, and is often even faster than the pivot step.

Table 1: Improved approximation ratios (CLUSTER DELETION solution divided by MFP lower bound) for several small graphs when using the merge heuristic. The % weak column reports the percentage of edges that MFP labels weak before performing the pivot step.

Graph	$\|V\|$	$\|E\|$	% weak	MFP approx	+*merge* approx
Netscience	379	914	68.93	1.72	1.58
Harvard500	500	2043	75.97	1.95	1.79
football	115	613	83.52	1.79	1.47
Erdos991	446	1413	95.4	1.99	1.67
celegansmetabolic	453	2025	95.41	1.99	1.78
polbooks	105	441	95.69	1.98	1.68
email	1133	5451	95.98	1.98	1.80
SmaGri	1024	4916	98.05	2.00	1.85
Roget	994	3640	98.24	2.00	1.69
celegansneural	297	2148	98.88	2.00	1.81
adjnoun	112	425	99.29	2.00	1.76
polblogs	1222	16714	99.75	2.00	1.92

The example in the proof of Theorem 3.3 motivates the following postprocessing step. After a set of clusters is obtained, we check each pair of them. If all the edges between two clusters are present in the input graph, we merge them into a single cluster. If the algorithm above has produced c clusters, this step may take $\Omega(c^2)$ time, which can be prohibitive. Since this preprocessing can be stopped anytime safely, in practice, we can apply it with a fixed time. It can be completely turned off to save time, or carried out exhaustively to achieve the best effect. Our experiments, shown in Table 1 and Figure 4, demonstrate that for most graphs, it significantly improves the outcomes. It is worth noting that this postprocessing is independent on the algorithms used to produce the clusters, and can be used for other algorithms for CLUSTER DELETION.

All of our algorithms are implemented in Julia. Code for our algorithms and experiments can be found at `https://github.com/vibalcam/combinatorial-cluster-deletion`. Given the overlap with the previous work, our new implementations build directly on and improve the open source implementations of Veldt (2022), available at `https://github.com/nveldt/FastCC-via-STC`, released under an MIT License.

Datasets. The graphs in Figure 2 all come from the SNAP network repository (Leskovec & Krevl, 2014) and come from several different types of graph classes. This includes social networks, road networks, citation networks, collaboration networks, and web graphs. All graphs have been standardized to remove weights, directions, and self-loops. The number of nodes and edges for each dataset are shown in Table 2. The graphs in Figure 4 are from the Facebook100 dataset (Traud et al., 2012). We specifically consider 46 smallest graphs in terms of the number of edges. The largest of these is Cal65, which has 11,247 nodes and 351,358 edges. Running MFP is very fast on all of these graphs (always less than 0.02 seconds for DegMFP). We only considered the smallest graphs in the collection since (in its current form) our cluster merging heuristic does not scale as easily (taking over 2 hours for the largest graph). The small graphs considered in Table 1 are all available from the Suitesparse matrix collection (Davis & Hu, 2011). We specifically consider graphs from the Arenas collection (email, celegansmetabolic), Newman collection (Netscience, polblogs, polbooks, football, adjnoun, celegansneural), Pajek collection (Erdos991, Roget, SmaGri), and Mathworks collection (Harvard500).

Additional experimental results. In Table 1 we display improved approximation ratios that can be achieved using our cluster merging heuristic on several small graphs. Although our current implementation of the algorithm is not optimized for runtime, it leads to noticeably better approximation ratios for all graphs where we ran it.

In Table 2 we provide a more detailed look at the performance of our algorithms on the large SNAP graphs. This includes runtimes for computing lower bounds, the value of lower bounds, and the a posteriori approximation ratios (CLUSTER DELETION solution divided by the lower bound computed by each method) achieved using our STC LP rounding method and MFP with three different pivoting strategies. For rounding the STC LP, we use the degree-based pivoting strategy as this is both fast and deterministic.

It is interesting to note that rounding the output to our combinatorial STC LP solver (the Comb-LP column in Table 2) tends to produce worse approximation ratios than MFP because it produces worse CLUSTER DELETION solutions. This may be

because the STC LP rounding technique is somewhat simplistic in that it treats all edges in $E_h = \{(i,j) \in E \colon x_{ij} = 1/2\}$ and $E_1 = \{(i,j) \in E \colon x_{ij} = 1\}$ in the same way (namely, delete these edges and then perform a pivoting step). One open question is whether it is possible to develop an improved rounding scheme that better leverages the difference between edges in E_h and edges in E_1. As a related observation, rounding the STC LP solution returned by Gurobi tends to produce better results, beating MFP in some cases. This is likely due to slight differences in *which* solution each method finds for the STC LP. An interesting direction for future research is to better understand how different optimal solutions to the LP affect downstream CLUSTER DELETION solutions, and then design techniques for quickly finding the most favorable LP solution for CLUSTER DELETION.

It is worth noting that despite the lower approximation ratios produced by Comb-LP, the STC LP lower bound for CLUSTER DELETION is always at least as tight as the lower bound computed by MFP. Furthermore, this lower bound is strictly tighter for all instances observed in practice. Thus, by combining the output of MFP with the tighter STC LP lower bound, we immediately obtain strictly better approximation guarantees than we obtain by running either method by itself.

Table 2: Detailed results ($n = |V|$, and $m = |E|$). Asterisks indicate the method ran out of memory. Dashes indicate Gurobi failed to produce an optimal solution for a reason other than memory. For all graphs, we allotted a maximum runtime of 2 days. The RanMFP column shows results specifically for RanMFP-100.

Graph		**RanMFP**	**DegMFP**	**RatMFP**	**Comb-LP**	**Gur-LP**
AMAZON0302	LB	402,933	402,933	402,933	438,400.0	***
	UB	814,852	779,652	779,523	855,620.0	***
$n = 262{,}111$	Ratio	2.022	1.935	1.935	1.952	***
$m = 899{,}792$	Run LB	0.262	0.196	0.282	488.675	***
	Run round	2.043	0.927	23.413	0.848	***
	Run total	2.305	1.123	23.694	489.523	***
AMAZON0312	LB	1,099,863	1,099,863	1,099,863	1.1616155e6	***
	UB	2,216,290	2,150,602	2,150,294	2.292491e6	***
$n = 400{,}727$	Ratio	2.015	1.955	1.955	1.974	***
$m = 2{,}349{,}869$	Run LB	0.673	0.608	0.685	5,170.367	***
	Run round	3.406	1.231	39.710	0.166	***
	Run total	4.079	1.839	40.395	5,170.533	***
AMAZON0505	LB	1,141,860	1,141,860	1,141,860	1.205665e6	***
	UB	2,300,995	2,232,537	2,232,231	2.378196e6	***
$n = 410{,}236$	Ratio	2.015	1.955	1.955	1.973	***
$m = 2{,}439{,}437$	Run LB	0.660	0.812	0.645	5,204.671	***
	Run round	3.435	1.290	46.796	0.152	***
	Run total	4.095	2.102	47.441	5,204.823	***
AMAZON0601	LB	1,142,262	1,142,262	1,142,262	1.2069215e6	***
	UB	2,302,232	2,232,009	2,231,696	2.37985e6	***
$n = 403{,}394$	Ratio	2.016	1.954	1.954	1.972	***
$m = 2{,}443{,}408$	Run LB	0.669	0.618	0.672	5,102.761	***
	Run round	3.535	1.258	41.885	0.180	***
	Run total	4.204	1.876	42.557	5,102.941	***

Continued on next page

Graph		RanMFP	DegMFP	RatMFP	Comb-LP	Gur-LP
CA-ASTROPH	LB	87,632	87,632	87,632	91,260.0	91,260.0
	UB	176,099	162,552	162,550	175,540.0	175,078.0
$n = 18{,}772$	Ratio	2.01	1.855	1.855	1.924	1.918
$m = 198{,}050$	Run LB	0.073	0.074	0.075	41.889	132.727
	Run round	0.155	0.159	0.570	0.043	0.458
	Run total	0.228	0.233	0.645	41.933	133.185
CA-CONDMAT	LB	39,764	39,764	39,764	42,019.0	42,019.0
	UB	80,464	75,133	75,127	81,989.0	80,860.0
$n = 23{,}133$	Ratio	2.024	1.889	1.889	1.951	1.924
$m = 93{,}439$	Run LB	0.023	0.022	0.021	7.961	18.080
	Run round	0.158	0.132	0.532	0.067	0.161
	Run total	0.181	0.154	0.553	8.028	18.241
CA-GRQC	LB	4,789	4,789	4,789	5,196.0	5,196.0
	UB	9,336	8,424	8,424	9,046.0	8,528.0
$n = 5{,}242$	Ratio	1.949	1.759	1.759	1.741	1.641
$m = 14{,}484$	Run LB	0.005	0.004	0.005	0.318	1.038
	Run round	0.037	0.018	0.119	0.018	0.054
	Run total	0.042	0.022	0.123	0.336	1.092
CA-HEPPH	LB	37,602	37,602	37,602	41,147.5	41,147.5
	UB	64,101	55,102	55,141	61,117.0	60,556.0
$n = 12{,}008$	Ratio	1.705	1.465	1.466	1.485	1.472
$m = 118{,}489$	Run LB	0.307	0.322	0.314	32.220	110.161
	Run round	0.111	0.040	2.110	0.031	0.132
	Run total	0.418	0.362	2.424	32.251	110.293
CA-HEPTH	LB	10,855	10,855	10,855	11,516.5	11,516.5
	UB	21,806	21,006	21,006	22,538.0	21,870.0
$n = 9{,}877$	Ratio	2.009	1.935	1.935	1.957	1.899
$m = 25{,}973$	Run LB	0.005	0.005	0.005	0.815	2.033
	Run round	0.112	0.056	0.199	0.026	0.069
	Run total	0.117	0.061	0.204	0.841	2.101
CIT-HEPPH	LB	208,953	208,953	208,953	210,366.0	–
	UB	418,074	417,075	417,069	420,702.0	–
$n = 34{,}546$	Ratio	2.001	1.996	1.996	2.0	–
$m = 420{,}877$	Run LB	0.097	0.095	0.100	123.203	–
	Run round	0.247	0.049	0.908	0.008	–
	Run total	0.345	0.145	1.008	123.210	–
CIT-HEPTH	LB	174,612	174,612	174,612	175,975.5	–
	UB	349,413	348,385	348,383	351,817.0	–
$n = 27{,}770$	Ratio	2.001	1.995	1.995	1.999	–
$m = 352{,}285$	Run LB	0.089	0.143	0.100	141.213	–
	Run round	0.187	0.033	0.946	0.013	–
	Run total	0.276	0.176	1.046	141.226	–

Continued on next page

Graph		RanMFP	DegMFP	RatMFP	Comb-LP	Gur-LP
CIT-PATENTS	LB	8,141,691	8,141,691	8,141,691	***	***
	UB	16,288,581	16,267,462	16,267,460	***	***
$n = 3{,}774{,}768$	Ratio	2.001	1.998	1.998	***	***
$m = 16{,}518{,}947$	Run LB	6.331	6.125	6.969	***	***
	Run round	44.362	8.704	410.382	***	***
	Run total	50.693	14.830	417.351	***	***
COM-AMAZON	LB	426,524	426,524	426,524	455,045.5	***
	UB	859,596	834,788	834,693	897,651.0	***
$n = 334{,}863$	Ratio	2.015	1.957	1.957	1.973	***
$m = 925{,}872$	Run LB	0.243	0.234	0.282	472.103	***
	Run round	2.812	0.871	26.045	0.213	***
	Run total	3.056	1.104	26.327	472.316	***
COM-DBLP	LB	404,019	404,019	404,019	449,921.5	***
	UB	825,495	737,218	737,172	853,743.0	***
$n = 317{,}080$	Ratio	2.043	1.825	1.825	1.898	***
$m = 1{,}049{,}866$	Run LB	0.344	0.374	0.396	631.427	***
	Run round	2.811	1.937	41.965	0.985	***
	Run total	3.155	2.311	42.362	632.412	***
COM-LIVEJOURNAL	LB	16,772,636	16,772,636	16,772,636	***	***
	UB	33,608,381	33,275,587	33,277,914	***	***
$n = 3{,}997{,}962$	Ratio	2.004	1.984	1.984	***	***
$m = 34{,}681{,}189$	Run LB	17.002	17.034	25.346	***	***
	Run round	48.159	23.847	1,742.355	***	***
	Run total	65.161	40.880	1,767.701	***	***
COM-ORKUT	LB	58,505,482	58,505,482	58,505,482	***	***
	UB	117,026,574	116,953,503	116,953,361	***	***
$n = 3{,}072{,}441$	Ratio	2.0	1.999	1.999	***	***
$m = 117{,}185{,}083$	Run LB	122.594	121.138	135.962	***	***
	Run round	215.619	17.428	17,305.204	***	***
	Run total	338.213	138.566	17,441.166	***	***
COM-YOUTUBE	LB	1,429,785	1,429,785	1,429,785	***	***
	UB	2,860,915	2,855,070	2,855,066	***	***
$n = 1{,}134{,}890$	Ratio	2.001	1.997	1.997	***	***
$m = 2{,}987{,}624$	Run LB	0.775	0.788	1.024	***	***
	Run round	9.276	3.136	63.352	***	***
	Run total	10.051	3.924	64.376	***	***
EMAIL-ENRON	LB	84,385	84,385	84,385	87,861.0	87,861.0
	UB	169,567	165,774	165,765	174,693.0	172,762.0
$n = 36{,}692$	Ratio	2.009	1.964	1.964	1.988	1.966
$m = 183{,}831$	Run LB	0.041	0.042	0.042	113.530	339.700
	Run round	0.257	0.266	1.067	0.069	0.238
	Run total	0.298	0.307	1.109	113.599	339.938

Continued on next page

Graph		RanMFP	DegMFP	RatMFP	Comb-LP	Gur-LP
EMAIL-EUALL	LB	174,651	174,651	174,651	***	***
	UB	349,298	349,296	349,296	***	***
$n = 265{,}214$	Ratio	2.0	2.0	2.0	***	***
$m = 364{,}481$	Run LB	0.088	0.087	0.098	***	***
	Run round	1.822	0.348	5.963	***	***
	Run total	1.911	0.435	6.061	***	***
LOC-BRIGHTKITE	LB	101,924	101,924	101,924	106,429.0	***
	UB	204,011	203,016	203,016	212,545.0	***
$n = 58{,}228$	Ratio	2.002	1.992	1.992	1.997	***
$m = 214{,}078$	Run LB	0.042	0.041	0.043	76.008	***
	Run round	0.387	0.197	1.306	0.018	***
	Run total	0.428	0.238	1.349	76.026	***
LOC-GOWALLA	LB	456,499	456,499	456,499	***	***
	UB	914,484	907,916	907,897	***	***
$n = 196{,}591$	Ratio	2.003	1.989	1.989	***	***
$m = 950{,}327$	Run LB	0.203	0.245	0.213	***	***
	Run round	1.539	0.701	7.949	***	***
	Run total	1.742	0.946	8.162	***	***
ROADNET-CA	LB	1,275,870	1,275,870	1,275,870	1.379125e6	***
	UB	2,556,485	2,536,702	2,536,702	2.745368e6	***
$n = 1{,}971{,}281$	Ratio	2.004	1.988	1.988	1.991	***
$m = 2{,}766{,}607$	Run LB	0.547	0.985	0.406	6,263.413	***
	Run round	16.925	7.356	188.404	1.377	***
	Run total	17.472	8.341	188.810	6,264.791	***
ROADNET-PA	LB	711,585	711,585	711,585	769,226.0	***
	UB	1,425,699	1,414,908	1,414,908	1.531777e6	***
$n = 1{,}090{,}920$	Ratio	2.004	1.988	1.988	1.991	***
$m = 1{,}541{,}898$	Run LB	0.193	0.195	0.189	1,367.728	***
	Run round	9.130	2.911	70.004	0.570	***
	Run total	9.323	3.106	70.193	1,368.297	***
ROADNET-TX	LB	883,250	883,250	883,250	958,145.5	***
	UB	1,769,957	1,755,407	1,755,407	1.9056e6	***
$n = 1{,}393{,}383$	Ratio	2.004	1.987	1.987	1.989	***
$m = 1{,}921{,}660$	Run LB	0.245	0.245	0.239	3,041.782	***
	Run round	11.510	4.519	106.177	0.749	***
	Run total	11.755	4.764	106.416	3,042.530	***
SOC-EPINIONS1	LB	197,337	197,337	197,337	202,521.0	***
	UB	394,809	394,032	394,031	404,825.0	***
$n = 75{,}888$	Ratio	2.001	1.997	1.997	1.999	***
$m = 405{,}740$	Run LB	0.091	0.091	0.103	808.772	***
	Run round	0.516	0.413	1.706	0.035	***
	Run total	0.607	0.504	1.809	808.807	***

Continued on next page

Graph		RanMFP	DegMFP	RatMFP	Comb-LP	Gur-LP
SOC-LIVEJOURNAL1	LB	20,681,921	20,681,921	20,681,921	***	***
	UB	41,452,776	40,988,289	40,999,181	***	***
$n = 4{,}847{,}571$	Ratio	2.004	1.982	1.982	***	***
$m = 42{,}851{,}237$	Run LB	41.378	38.295	21.741	***	***
	Run round	97.697	47.469	3,212.986	***	***
	Run total	139.075	85.764	3,234.728	***	***
SOC-SLASHDOT0811	LB	229,500	229,500	229,500	234,429.5	***
	UB	459,048	458,653	458,653	468,752.0	***
$n = 77{,}360$	Ratio	2.0	1.998	1.998	2.0	***
$m = 469{,}180$	Run LB	0.101	0.101	0.104	488.743	***
	Run round	0.698	0.186	1.594	0.024	***
	Run total	0.800	0.287	1.697	488.767	***
SOC-SLASHDOT0902	LB	247,059	247,059	247,059	251,944.0	***
	UB	494,192	493,777	493,778	503,815.0	***
$n = 82{,}168$	Ratio	2.0	1.999	1.999	2.0	***
$m = 504{,}230$	Run LB	0.103	0.103	0.112	604.756	***
	Run round	0.560	0.190	1.844	0.030	***
	Run total	0.663	0.293	1.956	604.786	***
WEB-BERKSTAN	LB	3,101,047	3,101,047	3,101,047	***	***
	UB	6,217,043	6,148,769	6,148,776	***	***
$n = 685{,}230$	Ratio	2.005	1.983	1.983	***	***
$m = 6{,}649{,}470$	Run LB	3.144	3.179	2.982	***	***
	Run round	6.065	1.077	75.884	***	***
	Run total	9.209	4.256	78.866	***	***
WEB-GOOGLE	LB	1,889,936	1,889,936	1,889,936	***	***
	UB	3,820,726	3,652,072	3,652,510	***	***
$n = 916{,}428$	Ratio	2.022	1.932	1.933	***	***
$m = 4{,}322{,}051$	Run LB	2.109	2.534	2.151	***	***
	Run round	9.049	1.851	125.882	***	***
	Run total	11.158	4.385	128.034	***	***
WEB-NOTREDAME	LB	414,253	414,253	414,253	***	***
	UB	829,768	820,767	820,771	***	***
$n = 325{,}729$	Ratio	2.003	1.981	1.981	***	***
$m = 1{,}090{,}108$	Run LB	0.479	0.472	0.642	***	***
	Run round	2.574	0.593	21.478	***	***
	Run total	3.053	1.064	22.120	***	***
WEB-STANFORD	LB	948,859	948,859	948,859	***	***
	UB	1,903,048	1,877,270	1,877,228	***	***
$n = 281{,}903$	Ratio	2.006	1.978	1.978	***	***
$m = 1{,}992{,}636$	Run LB	0.961	1.035	0.911	***	***
	Run round	2.476	0.587	13.140	***	***
	Run total	3.437	1.621	14.052	***	***

Continued on next page

Graph		RanMFP	DegMFP	RatMFP	Comb-LP	Gur-LP
WIKI-TALK	LB	2,313,080	2,313,080	2,313,080	***	***
	UB	4,626,181	4,626,014	4,626,014	***	***
$n = 2{,}394{,}385$	Ratio	2.0	2.0	2.0	***	***
$m = 4{,}659{,}565$	Run LB	1.546	1.600	1.557	***	***
	Run round	22.358	1.233	103.372	***	***
	Run total	23.904	2.834	104.929	***	***
WIKI-TOPCATS	LB	12,689,197	12,689,197	12,689,197	***	***
	UB	25,380,869	25,369,215	25,369,203	***	***
$n = 1{,}791{,}489$	Ratio	2.0	1.999	1.999	***	***
$m = 25{,}444{,}207$	Run LB	8.091	8.290	8.534	***	***
	Run round	19.867	1.737	104.694	***	***
	Run total	27.958	10.026	113.228	***	***