Breaking 3-Factor Approximation for Correlation Clustering in Polylogarithmic Rounds*

Nairen Cao† Shang-En Huang† Hsin-Hao Su†

Abstract

In this paper, we study parallel algorithms for the correlation clustering problem, where every pair of two different entities is labeled with similar or dissimilar. The goal is to partition the entities into clusters to minimize the number of disagreements with the labels. Currently, all efficient parallel algorithms have an approximation ratio of at least 3. In comparison with the $1.994+\epsilon$ ratio achieved by polynomial-time sequential algorithms [25], a significant gap exists.

We propose the first poly-logarithmic round parallel algorithm that achieves a better approximation ratio than 3. Specifically, our algorithm computes a $(2.4+\epsilon)$-approximate solution and uses $\tilde{O}(m^{1.5})$ work. Additionally, it can be translated into a $\tilde{O}(m^{1.5})$-time sequential algorithm and a poly-logarithmic rounds sublinear-memory MPC algorithm with $\tilde{O}(m^{1.5})$ total memory.

Our approach is inspired by Awerbuch, Khandekar, and Rao's [6] length-constrained multi-commodity flow algorithm, where we develop an efficient parallel algorithm to solve a truncated correlation clustering linear program of Charikar, Guruswami, and Wirth [16]. Then we show the solution of the truncated linear program can be rounded with a factor of at most 2.4 loss by using the framework of [17]. Such a rounding framework can then be implemented using parallel pivot-based approaches (e.g. [10, 29]).

1 Introduction

We study parallel algorithms for the correlation clustering problem introduced by Bansal, Blum, and Chawla [7], where the goal is to group similar entities and keep different entities apart. In the problem, we are given a complete graph $G=(V,E^+\cup E^-)$ where E^+ are the edges labeled with $+$ (similar) and E^- are the ones labeled with $-$ (different). Given a clustering of the vertices, we say an edge uv *disagrees* with the clustering if either (1) uv is labeled with $-$ and u,v are in the same cluster, or (2) uv is labeled with $+$ and u,v are in the different clusters. The objective is to find a clustering that minimizes the number of edges that disagree.

In contrast to most other clustering methods, correlation clustering does not require the user to specify the number of clusters as the input. Due to its simplicity, this clustering method has various applications in spam detection, gene clustering, chat disentanglement, and co-reference resolution [18, 11, 28, 4, 27].

In the sequential setting, several algorithms have been developed with increasingly better approximation ratios [7, 16, 3, 17, 25]. In particular, [3] introduced the classic Pivot algorithm, which achieves an approximation factor of 3. They also showed how to use it to round a linear programming (LP) solution with a 2.5-approximation ratio. Later, [17] improved the approximation factor to 2.06. Recently, [25] designed a rounding algorithm based on the Sherali-Adams relaxation which achieves an approximation factor of $1.994+\epsilon$. Very recently, the bound has been improved to 1.73 by using a pre-clustering technique [24]. The correlation clustering problem is also known to be APX-hard [16].

The exponential growth of data and advances in parallel architectures have motivated a long line of study on the parallel algorithms for the correlation clustering problem [10, 18, 29, 13, 23, 2, 5, 14, 8]. Algorithms of particular interest are those with small rounds, say, poly-logarithmic rounds. We summarize the results that fit into this category in Table 1 along their round complexities and their targeted models. Noticeably,

*This work is supported by NSF CCF-2008422.

†Boston College

there has been a successful line of work based on parallelization of the PIVOT algorithm of [3], to which they culminated in constant rounds MPC algorithms [8, 14].

Reference	Approx. Ratio	Rounds	Models	Method
[5]	≈ 100000	1	♣	sparse-dense decomposition
[23]	701	$O(1)$	♣	
[2]	3	$O(\log\log n)$	♣	PIVOT-based
[13]	3	$O(\log\Delta \cdot \log\log n)$	♦	
[29]	3	$O(\log n)$	♠	
[10]	3	$O(\log^2 n)$	♠	
[18]	$3+\epsilon$	$O((\log n \cdot \log\Delta)/\epsilon)$	♦	
[14]	$3+\epsilon$	$O(1)$	♣♦	
[8]	$3+\epsilon$	$O(1/\epsilon)$	♣♦	
This paper.	$2.4+\epsilon$	$\text{poly}(1/\epsilon, \log n)$	♦♠	LP + PIVOT-based

Table 1: Low-round algorithms for correlation clustering. The running times are stated with respect to the model of focus in each work, shown in the "Models" column. They may be implemented in other models that are not listed, potentially with additional factors on the running times. ♣ denotes the semi-streaming model. ♦ denotes the MPC model. ♠ denotes the PRAM model.

However, these algorithms hit a barrier at the approximation factor of 3. A natural question is whether parallel algorithms of poly-logarithmic rounds for achieving an approximation factor better than 3 exist. Indeed, [8] also mentioned in the conclusion and open problem section,

> "It would be extremely interesting to study whether a low-round algorithm exists for solving the natural correlation clustering LP." – as it would lead to algorithms with better than 3 approximation ratios.

In this paper, we give the first poly-logarithmic round parallel algorithms that surpass the 3-factor approximation. In previous literature, it is typical to use $m = |E^+|$ to denote the number of positive edges and to obtain bounds in terms of m, as it has been pointed out by [18] that it is common to have a much smaller number of positive edges than negative edges in practical applications.

THEOREM 1.1. *There exists a $\tilde{O}(\epsilon^{-4})$-depth parallel algorithm that achieves a $(2.4+\epsilon)$-approximation for the correlation clustering problem using a total work of $\tilde{O}(\epsilon^{-7}m^{1.5})$.* [1]

Moreover, our parallel algorithms can also be simulated in the sublinear-space MPC model with a total space of $\tilde{O}(\epsilon^{-3}m^{1.5})$. We summarize this result in Corollary 1.1 and prove it in Appendix A.

COROLLARY 1.1. *There exists a $\tilde{O}(\epsilon^{-4})$-round sublinear memory MPC algorithm that computes a $(2.4+\epsilon)$-approximate solution for the correlation clustering problem using a total memory of $\tilde{O}(\epsilon^{-3}m^{1.5})$.*

In the sequential setting, all previous algorithms with approximation factors better than 3 would require solving the standard linear program by [3], which would take at least $\Omega(n^5)$ time by the fastest solver today. As sequential algorithms can be directly obtained by simulating parallel algorithms, we also obtain a sequential algorithm whose running time equals to the work of our parallel algorithm:

COROLLARY 1.2. *There exists a $\tilde{O}(\epsilon^{-7}m^{1.5})$ time sequential algorithm that computes a $(2.4+\epsilon)$-approximate solution for the correlation clustering problem.*

[1] The *work* of a parallel algorithm is the total number of primitive operations, and its *span* or *depth* is the length of the longest chain of sequential dependencies or, equivalently, the limit of parallel time as processors approach infinity. The bounds are stated for the PRAM CRCW model, although other PRAM variants induce at most a logarithmic factor.

Interestingly, the work bottleneck in Theorem 1.1 comes from the problem of finding a maximal set of edge-disjoint open triangles, where an open triangle is a triangle with 2 positive edges and 1 negative edge. We show that any combinatorial algorithm with better work than $O(m^{1.5-\epsilon})$ for any constant $\epsilon > 0$ for the problem would refute a conjecture on the Boolean matrix multiplication problem [1, 43].

Further Related Works on Correlation Clustering. There have also been studies of the correlation clustering problem on non-complete graphs where there may be missing labels among pairs of vertices. In such a setting, [26] gave an $O(\log n)$-approximation sequential algorithm for the problem. For the agreement maximization version of the problem where the goal is to maximize the number of edges in agreement with the partition, [41] and [16] gave a 0.7666-approximation algorithm and a 0.7664-approximation algorithm for the problem respectively. The correlation clustering problem has also been studied in online settings [36, 33, 22], and settings with differential privacy guarantees [12, 21, 35] and local guarantees [15, 39, 32, 31].

1.1 Technical Challenges

As mentioned in [8], the bottleneck for achieving efficient parallel algorithms with an approximation factor better than 3 is in solving the standard correlation clustering LP (P1). The existing sequential 2.06-approximation algorithm by [17] and sequential 2.5-approximation algorithm by [3] both start with a fractional solution to (P1). Note that the 2.06 approximation is nearly optimal as the integrality to the LP is 2. [8] pointed out that once a fractional solution of the linear program is obtained, one can create an instance in the spirit of the 2.06-approximate algorithm of [17] such that running the Pivot-style parallel algorithms on the instance yields the same approximation factor.

Standard Correlation Clustering LP

$$\text{(P1)} \qquad \begin{aligned} \text{minimize} \quad & \sum_{(u,v)\in E^+} x_{uv} + \sum_{(u,v)\in E^-} (1 - x_{uv}) \\ \text{subject to} \quad & x_{uw} + x_{wv} \geq x_{uv} && \forall u, v, w \in V \\ & x_{uu} = 0 && \forall u \in V \\ & x_{uv} \in [0, 1] && \forall (u, v) \in E^+ \cup E^- \end{aligned}$$

The most efficient solver by using the interior point method today [37, 34] would require at least $\Omega(\sqrt{N})$ iterations, where N is the number of variables, which in our case is $\Theta(n^2)$. This is even not counting the fact each iteration involves complicated volume computation or linear system solving. In the sequential setting, the best known algorithm for solving a general linear program runs in $\tilde{O}(N^\omega + N^{2.5-\alpha/2} + N^{13/6})$ time by [19], where $\omega \sim 2.37$ and $\alpha \sim 0.31$ are the current best-known exponent and dual exponent of matrix multiplication.

The Cut-Flow View. As solving the standard linear program of correlation clustering using the current tools requires at least polynomial iterations, we take a detour from the problem. The correlation clustering problem is known to have a strong connection with the multicut problem [26]. Charikar, Guruswami, and Wirth [16] gave the following alternative linear program formulation which captures such a connection.

(Primal)			(Dual)		
min	$\sum_{(u,v)\in E^+\cup E^-} z_{uv}$		max	$\sum_{P\in\mathcal{P}} y_P$	
s.t.	$z_{uv} + \sum_{e\in P} z_e \geq 1$	$\forall uv \in E^- \ \forall P \in \mathcal{P}_{uv}$	s.t.	$\sum_{P\ni e} y_P \leq 1$	$\forall e \in E^+$
				$\sum_{P\in\mathcal{P}_{uv}} y_P \leq 1$	$\forall uv \in E^-$
	$z_{uv} \geq 0$	$\forall uv \in E^+ \cup E^-$		$y_P \geq 0$	$\forall P \in \mathcal{P}$

where $\mathcal{P}_{uv}$ be the collection of paths in G^+ that connects u and v and $\mathcal{P} = \cup_{uv\in E^-}\mathcal{P}_{uv}$.

[16] showed that an optimal solution to (Primal) is an optimal solution to the standard linear program (P1). In this cut-flow view, the dual linear program (Dual) can be seen as a variant of the multi-commodity flow problem on E^+ where each negative edge $(u, v) \in E^-$ is a source-sink pair. The objective becomes routing as much flow as possible between the source-sink pairs under the constraints that (1) each edge has

capacity 1, and (2) at most 1 unit flow can be routed between each source-sink pair. Although the multi-commodity flow problem has been studied extensively, unfortunately, there are no known parallel algorithms that can solve it in poly-logarithmic iterations.

1.2 Our Approach We truncate the Cut-Flow View linear program, by only keeping the path constraints for bounded-hop paths. The following is the truncated primal program.

The L-hop Cut-Flow View. Fix a positive integer L. For any negative edge $(u,v) \in E^-$, we denote $\mathcal{P}_{uv}^{(L)}$ to be the paths in $\mathcal{P}_{uv}$ with at most L edges (hops). Similarly we define $\mathcal{P}_L = \cup_{uv \in E^-} \mathcal{P}_{uv}^{(L)}$.

(PRIMAL$^{(L)}$)			(DUAL$^{(L)}$)		
min	$\sum_{(u,v)\in E^+\cup E^-} z_{uv}$		max	$\sum_{P\in\mathcal{P}_L} y_P$	
s.t.	$z_{uv} + \sum_{e\in P_{uv}} z_e \geq 1$	$\forall uv \in E^- \ \forall P \in \mathcal{P}_{uv}^{(L)}$	s.t.	$\sum_{P\ni e} y_P \leq 1$	$\forall e \in E^+$
				$\sum_{P\in\mathcal{P}_{uv}^{(L)}} y_P \leq 1$	$\forall uv \in E^-$
	$z_{uv} \geq 0$	$\forall uv \in E^+ \cup E^-$		$y_P \geq 0$	$\forall P \in \mathcal{P}_L$

The corresponding dual program is also a variant of the multi-commodity flow problem where we can only route flow along paths of at most L-hops between the source sink pairs. The hop-constrained multi-commodity flow problem has been studied by Awerbuch, Khandekar, and Rao [6], where they developed an algorithm that runs in $\tilde{O}(L)$ iterations (note that the standard multiplicative weight update method will take too much time because the width can be very large). Although our dual program has an extra constraint, their result suggests the possibility of having a poly-logarithmic round parallel algorithm, for L up to poly$(\log n)$. However, even if it works, such an approach still faces the following challenges:

1. First, it is unclear how far the optimal solution of PRIMAL$^{(L)}$ is from that of PRIMAL. Moreover, we will need to develop a procedure to convert a solution of PRIMAL$^{(L)}$ to, say some approximation solution of PRIMAL.
2. Second, all the existing rounding algorithms are based on the standard correlation clustering LP. Even if we can convert an optimal solution of PRIMAL$^{(L)}$ to an approximate solution of PRIMAL, it is unclear if it will satisfy the constraints of the standard correlation clustering LP. Note that [16] only showed an optimal solution will satisfy the constraints, but an approximate solution will not necessarily do so.
3. Third, as mentioned previously, we hope to obtain an upper bound on the total work in terms of $m = |E^+|$, as it will result in a faster sequential algorithm on graphs with sparse positive edges. The challenge is two-fold. First, in the fractional primal solution, there can be much more than m negative edges with non-zero values. This means the input to the rounding algorithm could have $\Omega(n^2)$ edges in the worst case. Second, in the algorithm of [6], each iteration requires computing an approximate blocking flow for every source-sink pair. Potentially, there can be $\Omega(n^2)$ such source-sink pairs. For each source-sink pair, the number of paths of length at most L can be as large as $\Omega(n^{L-1})$ between each pair. If we assign a processor to each of the paths directly, the work would be at least $\Omega(n^{L+1})$.

Surprisingly, we found that it is possible to overcome the first two challenges altogether by directly rounding a solution of PRIMAL$^{(L)}$ for $L = 2$ to an integral correlation clustering solution. Our new rounding procedure obtains an approximation factor of 2.4.

Two-Hop Primal ($L = 2$).

(PRIMAL$^{(2)}$)

$$\begin{aligned} \text{minimize} \quad & \sum_{(u,v)\in E^+\cup E^-} z_{uv} \\ \text{subject to} \quad & z_{uv} + z_{uw} + z_{wv} \geq 1 \qquad \forall (u,v) \in E^- \text{ and } (u,w),(w,v) \in E^+ \\ & z_{uv} \geq 0 \qquad \forall (u,v) \in E^+ \cup E^- \end{aligned}$$

Two-Hop Dual. Let $\mathcal{P}_2$ be the collection of length-two paths (u, w, v) such that $(u, w), (w, v) \in E^+$ and $(u, v) \in E^-$.

$$
(\text{DUAL}^{(2)}) \qquad
\begin{aligned}
\text{maximize} \quad & \sum_{(u,w,v)\in\mathcal{P}_2} y_{(u,w,v)} \\
\text{subject to} \quad & \sum_{w:(u,w,v)\in\mathcal{P}_2} y_{(u,w,v)} \le 1 && \forall (u,v) \in E^- \\
& \sum_{v':(u,w,v')\in\mathcal{P}_2} y_{(u,w,v')} + \sum_{u':(u',w,u)\in\mathcal{P}_2} y_{(u',w,u)} \le 1 && \forall (u,w) \in E^+ \\
& y_{(u,w,v)} \ge 0 && \forall (u,w,v) \in \mathcal{P}_2
\end{aligned}
$$

LEMMA 1.1. *There exists a* poly$(\log n)$*-time parallel algorithm that converts a fractional solution of* PRIMAL$^{(2)}$ *to a clustering where the number of disagreements is at most 2.4 times the object value of the fractional solution in expectation. The total work of the algorithm is quasi-linear in the number of non-zero terms of the fractional solution.*

COROLLARY 1.3. *Let* $OPT(\cdot)$ *denote the optimal value of a linear program. Let* PRIMAL$_I$ *be the integral version of* PRIMAL. *We have*

$$OPT(\text{PRIMAL}) \le OPT(\text{PRIMAL}_I) \le 2.4 \cdot OPT(\text{PRIMAL}^{(2)})$$

An *open triangle* is (x, y, z), where $(x, y), (y, z) \in E^+$ and $(x, z) \in E^-$. A *closed triangle* is (x, y, z), where $(x, y), (y, z), (x, z) \in E^+$. We note that $\mathcal{P}_2$ can also be seen as a collection of open triangles. The above dual program can be viewed as an open triangle packing problem, where the goal is to pack as many open triangles as possible fractionally, subject to the condition that for each $e \in E^+ \cup E^-$, the sum of the values over all the triangles containing e is at most 1.

We note that the Two-Hop Dual LP has been seen in the analysis of the 3-approximation PIVOT algorithm of [3], where they refer it as *bad triangle packing*. Moreover, recently, Veldt [42], also considered the Two-Hop Primal LP for correlation clustering. He developed a 4-approximation rounding algorithm for correlation clustering from the LP. Moreover, he noted that such a linear program can be solved faster *empirically* by LP solvers. Although we use the same LP, the paths of how we arrive at such an LP are different, yet coincidental. Veldt [42] drew and strengthened an interesting connection between correlation clustering and *strong triadic closure labeling* [40, 38], while we derive such a linear program from the lens of efficient optimization algorithms. We also emphasize that the LP is different than the standard LP (P1) studied by [3] as they consider *all* (u, w, v) tuples in the constraint instead of only the ones in $\mathcal{P}_2$.

A 2.4-Approximation Rounding Algorithm. Ailon, Charikar, and Newman [3] first proposed a 2.5-approximation algorithm by rounding the linear programming solution $\{x_e\}_{e\in E^+\cup E^-}$ of (P1). They interpret x_e as the probability that the edge e should be cut, with higher values of x_e indicating a lower probability that the endpoints e should be included in the same cluster. Their algorithm is based on their PIVOT algorithm and works as follows: Choose a random vertex w as the pivot. Put every other node v into the cluster of w with probability $1 - x_{wv}$. Remove the cluster and then repeat on the remaining graph.

Later, Chawla et al. [17] improved this method to a 2.06 approximation ratio by making the crucial observation that clustering a node u with probability $1 - x_{uw}$ may not lead to the best approximation ratio. Instead, they cluster the node with probability $1 - f(x_{uw})$, for some carefully chosen function f.

The key in upper bounding the approximation ratio the PIVOT-based rounding algorithm, as developed in [3, 17] and elaborated in [25], is in bounding the following ratio:

$$\rho(u, v, w) = \frac{cost(u, v \mid w) + cost(u, w \mid v) + cost(w, v \mid u)}{lp(u, v \mid w) + lp(u, w \mid v) + lp(w, v \mid u)}$$

for every triangle (u, v, w). The term $cost(u, v \mid w)$ denotes the probability that the disagreement of (u, v) occurs when w is chosen as the pivot, whereas $lp(u, v \mid w)$ denotes the probability that the edge uv is

removed from the graph when w is chosen as the pivot, multiplied by the contribution of the edge uv in the LP. Intuitively, the numerator can be understood as the actual cost caused by the algorithm, while as the denominator can be understood as the cost in the LP that we are charging to. Their analysis shows that $\max_{(u,v,w)} \rho(u,v,w)$ is an upper bound of the approximation ratio of the rounding algorithm.

[17] noted that $\rho(u,v,w)$ is a multivariate polynomial over x_{uv}, x_{uw}, x_{vw} and conducted a case-by-case analysis of $\rho(\cdot)$ over four types of triangles based on the sign of their edges: $(-,-,-), (+,-,-), (+,+,-), (+,+,+)$. Their carefully designed rounding function, $f(\cdot)$, led to an upper bound of $\rho(u,v,w)$ by 2.06 for every type of triangle. The constraints of (P1), namely the triangle inequality $x_{uw} + x_{vw} \geq x_{uv}$ over all the triples (u,w,v), play a critical role in their analysis.

One of our main technical contributions is that we show that we do not need triangle inequality for all types of triangles to obtain a good approximate ratio. Specifically, we show that for all triangles (u,w,v) such that $uw, vw \in E^+$ and $uv \in E^-$ (the $(+,+,-)$ triangle), if we have $x_{uw} + x_{vw} \geq x_{uv}$, then it is possible to obtain an upper bound of 2.4 on $\rho(u,v,w)$ for *all* types of triangles, by using a different rounding function f. In addition, we show that 2.4 is the best ratio one can obtain when under such a framework.

Focusing solely on $(+,+,-)$ triangles greatly simplifies the process of solving linear programs. Instead of solving (P1), the solution to our $\textsc{Primal}^{(2)}$ provides us with the triangle inequality for $(+,+,-)$ triangles. This can be done by setting $x_e = z_e$ for all positive edges and $x_e = 1 - z_e$ for all negative edges. Thus, a feasible solution from $\textsc{Primal}^{(2)}$ can be converted into an assignment that satisfies the triangle inequality for $(+,+,-)$ triangles.

Solving $\textsc{Primal}^{(2)}$. Awerbuch, Khandekar, and Rao [6] proposed a distributed *approximate steepest descendant framework* that gives approximate solutions to multi-commodity flow problems efficiently. In their framework, there is a *convex* length function with respect to the congestion $cong_e$ of each edge (say $(m^{1/\epsilon})^{cong_e}$), and the objective is to minimize the sum Φ of all edge lengths while maximizing the total flow $\sum y_P$. Let k be the number of commodities. In each step, for each commodity, the algorithm chooses a set of *approximately shortest* source-sink paths and runs a *blocking flow* through these paths, where the capacities of blocking flow are set to be tiny, roughly ϵ/k. By the pigeonhole principle, all approximately shortest source-sink paths will be eliminated after a certain number of steps. It turns out that, by choosing only the shortest paths in each step, one can bound the growth rate of Φ. Furthermore, these edge lengths, divided by the shortest paths' length, can be used to define a feasible primal solution. This establishes a $(1 + O(\epsilon))$ factor difference between primal and dual objective values, which certifies a desired $(1 + O(\epsilon))$ approximation ratio.

Inspired by the steepest descendant framework, we extend the length function to not only the positive edges (the edges presented in the input) but also the negative edges. Moreover, each source-sink pair corresponds to a negative edge $uv \in E^-$ where there exist two-hop paths in E^+ connecting u and v, forming $(+,+,-)$ triangles. Sending a flow from u to v is then equivalent to adding a circulation to a $(+,+,-)$ triangle involving uv. This makes the entire framework applicable to solve $\textsc{Primal}^{(2)}$ and $\textsc{Dual}^{(2)}$.

However, the third challenge still remains, even if we restrict the number of hops to $L = 2$. This is mainly because there can be $k = \Omega(n^2)$ source-sink pairs, one per negative edge. In the algorithm of [6], they compute an approximate shortest blocking flow for every source-sink pair. This can cause the algorithm to send flow through $\Omega(n^2)$ edges in a single iteration. Additionally, computing the blocking flow for $k = \Omega(n^2)$ may require a significant amount of work. If one would like to efficiently utilize Lemma 1.1, which requires quasi-linear work in the number of non-zero terms of the fractional solution, we have to first ensure that our fractional solution has a $\tilde{O}(m)$-sized support.

To resolve this, our idea is to mix up all the commodities. That is, we let each commodity block other commodities, instead of computing the blocking flow per commodity. In the view of triangles, this corresponds to computing a maximal set of edge-disjoint open triangles. There can be at most $O(m)$ edge-disjoint open triangles in such a set because each open triangle consumes two positive edges. At the same time, instead of sending ϵ/k flow for each commodity, we can send up to $\epsilon/\log m$ flow for each commodity in the maximal set. This, combined with the fact each triangle contains at least two positive edges and one negative edge, gives an upper bound of $\tilde{O}(m)$ on the number of non-zero duals. However, these do not

necessarily translate directly to bounds on the number of non-zero or relatively small primal variables. We devise an additional simple post-processing step to construct a primal solution where all but $\tilde{O}(m)$ primal variables are small enough to be truncated.

LEMMA 1.2. *Given the set E^+ of m positive edges and a parameter $\epsilon > 0$, there exists a parallel algorithm that computes an $(1+\epsilon)$-approximate solution to PRIMAL$^{(2)}$ using $\tilde{O}(\epsilon^{-7}m^{1.5})$ work and $\tilde{O}(\epsilon^{-4})$ span. In addition, the support size of the solution is at most $\tilde{O}(\epsilon^{-2}m)$.*

The last mile to our $\tilde{O}(m^{1.5})$-work poly$(\log n)$-depth parallel algorithm is the problem of computing a maximal set of edge-disjoint open triangles. Although, it can be shown that the number of closed triangles is upper bounded by $\tilde{O}(m^{1.5})$, it is not necessarily the case for open triangles. The number of open triangles can be as large as $\Omega(n^2)$ even when there are $O(n)$ edges (e.g. a star of positive edges). Inspired by the triangle enumeration algorithms, we observe that it is possible to compute a maximal set of edge-disjoint open triangles without checking all open triangles. To this end, we obtain a parallel algorithm that gradually searches for more edge-disjoint open triangles in rounds. In each round, the algorithm explores a collection of open triangles $\mathcal{C}$ and runs a maximal independent set (MIS) algorithm in the conflict graph built on $\mathcal{C}$. As long as these open triangles in $\mathcal{C}$ are edge-disjoint to the current found set S of edge-disjoint open triangles, the open triangles selected in the MIS can be added to S and all open triangles in S are still edge-disjoint. By carefully controlling the exploration rate, we obtain the parallel algorithm with desired work and depth, summarized below in Lemma 1.3 and proved in Section 5.

LEMMA 1.3. *Let $G = (V, E^+ \cup E^-_{>1})$, and $l : \binom{V}{2} \to [1, \infty)$ be a length function with $l(u,v) = 1$ for $uv \in \binom{V}{2} \setminus (E^+ \cup E^{-1}_{>1})$, and $L > 0$ be a length limit. Let $m_f = |E^+ \cup E^-_{>1}|$. Then, there exists a parallel algorithm such that, in $O(m_f^{1.5} \log^3 n)$ work and $O(\log^3 n)$ span, the algorithm returns a maximal edge-disjoint set S of open triangles with length less than L.*

Conditional Lower Bound for Maximal Edge-Disjoint Open Triangles. Unfortunately, it seems that $\tilde{O}(m^{1.5})$ is the best total work one can hope for, if we want a combinatorial algorithm that computes a maximal set of edge-disjoint open triangles. In terms of the number of total input edges m, the problem of finding a maximal set of edge-disjoint open triangles could be as hard as searching for just one (regular, non-open) triangle. The latter problem (triangle detection) has a conditional lower bound based on the combinatorial Boolean matrix multiplication (BMM) problem [43]. In Section 5.2 we give a randomized reduction from the triangle detection problem to our maximal open triangle problem. Such a reduction also implies that *any* algorithms using $O(m^{1.185-\delta})$ work for any constant $\delta > 0$ for our problem would lead to an improvement over the best known algorithm for the triangle detection problem [9].

1.3 Open Problems We give the first poly-logarithmic depth parallel algorithm that achieves an approximation ratio better than 3. We hope that our work can shed some light on the search of low-round algorithms with an approximation ratio of less than 3 in other models that exploit parallelism, such as the streaming model, the MPC model (with nearly-linear total memory), and the CONGEST model. The main bottleneck in adapting our algorithm to those models is in finding a maximal set of edge-disjoint open triangles. It is interesting to investigate whether there are ways to sparsify the process in these models.

We have shown that the optimal solution of PRIMAL$^{(2)}$ is at most 2.4 times that of PRIMAL. It is interesting to see if there are tighter relations between PRIMAL$^{(L)}$ and PRIMAL for $L > 2$. Note that [16] showed the optimal values to both linear programs are the same when $L = n - 1$. Perhaps one may obtain a trade-off between L and the quality of the solution. If this is the case, we may be able to obtain a result that exhibits a trade-off between the running time/work of the algorithms and the quality of the solutions.

Finally, we have shown a conditional lower bound on the problem of finding a maximal set of edge-disjoint open triangles. It is interesting to directly investigate the fine-grained complexity of the c-approximate correlation clustering for $c < 3$. We already know that the 3-approximation algorithm of [3] can be implemented in $\tilde{O}(m)$ time in the sequential setting, and $\tilde{O}(m)$ total work in the parallel setting [29]. Now

the question is whether it is possible to obtain a nearly-linear time algorithm for c-approximate algorithm for $c < 3$ or it can be shown to be (conditionally)-hard.

2 Preliminaries

Let $m = |E^+|$ be the number of positive edges in the unweighted and undirected input graph $G^+ = (V, E^+)$. We will use the triple (u, w, v) to denote the triangle with edges (u, v), (v, w) and (u, w). We note that the same triangle and the same undirected edge can be identified in different ways. In particular, (u, w, v) and (v, w, u) refer to the same triangle, and (u, v), (v, u), uv, and vu refer to the same edge. Given an edge e and a triangle (u, w, v), we say that e is *on the triangle*, denoted as $e \in (u, w, v)$, if e is one of the edges of (u, v), (v, w) and (u, w).

Sometimes we denote the same triangle with an *ordered* 3-edge triple (uw, wv, vu) when we are mapping certain attributes to the edges. Let $s_{uw}, s_{wv}, s_{vu} \in \{+, -\}$, a (s_{uw}, s_{wv}, s_{vu}) triangle is where $uw \in E^{s_{uw}}$, $wv \in E^{s_{wv}}$, and $vu \in E^{s_{vu}}$. For example, a $(+, +, -)$ triangle is a triangle (uw, wv, vu) such that $uw, wv \in E^+$ and $vu \in E^-$. Such a triangle is called an *open triangle*. Although $\mathcal{P}_2$ was defined be the collection of length-two paths (u, w, v) such that $(u, w), (w, v) \in E^+$ and $(u, v) \in E^-$, we can also view it as a collection of open triangles. A $(+, +, +)$ triangle is called a *closed triangle*.

Assumptions. We assume without loss of generality that the graph $G^+ = (V, E^+)$ is connected. Otherwise, we can process each connected component induced by positive edges separately. Also, we assume that G^+ is not a complete graph, so the optimal objective value is at least 1. Otherwise, we may just output the entire graph as a cluster.

3 An $(1+\epsilon)$-Approximation Algorithm for Primal$^{(2)}$

In this section, we propose Algorithm 1, an algorithm that computes a $(1+\epsilon)$-approximate solution $\{z_{uv}\}$ for Primal$^{(2)}$. Our algorithm is inspired by the distributed steepest descent framework [6] for the *most beneficial flow* (MBF), and an earlier sequential multicommodity flow algorithm by Garg and Könemann [30]. In our case, as mentioned in the introduction, the algorithm focuses on triples in $\mathcal{P}_2$ and sends flows along the *most beneficial triangle*.

The algorithm runs in *iterations*. Intuitively, in each iteration t, the algorithm seeks a set S of the approximately shortest length triangles from $\mathcal{P}_2$. Then, the algorithm pushes some tiny flow along each triangle, which by correspondence (see Invariant 1 below) increases the length of each edge with a multiplicative factor of $\exp(\epsilon) \approx 1+\epsilon$. The iterations end once the total length of each edge exceeds a certain value, and the algorithm is then able to produce the $(1 + O(\epsilon))$-approximate solutions to both Primal$^{(2)}$ and Dual$^{(2)}$.

Explicitly Maintained Variables and Invariants. Our algorithm mainly operates on Dual$^{(2)}$, that is, the algorithm explicitly stores all non-zero y values for triangles in $\mathcal{P}_2$. For the ease of analysis, we will use $y^{(t)}_{(u,w,v)}$ to denote the dual variables at the beginning of the iteration t. If we treat each $y^{(t)}_{(u,w,v)}$ as a circulation on its own commodity, then it makes sense to define *congestion* of an edge e, to be the sum of all flow values passing through that edge. Specifically, for each edge $e \in E^+ \cup E^-$ we define $cong^{(t)}_e = \sum_{(u,w,v)\in\mathcal{P}_2, e\in(u,w,v)} y^{(t)}_{(u,w,v)}$. The algorithm also explicitly maintains the *length* of an edge e, which is defined by $l^{(t)}_e = (m^{1/\epsilon})^{cong^{(t)}_e}$. This leads to a definition for the length of a triangle (u, w, v) in t-th iteration to be $l^{(t)}_{(u,w,v)} := l^{(t)}_{wu} + l^{(t)}_{wv} + l^{(t)}_{uv}$. Moreover, the algorithm maintains a variable $\alpha^{(t)}$ for lower bounding the shortest open triangle. Throughout the execution, the algorithm maintains the following invariant between the dual variables, length variables, and the shortest open triangle estimate:

Invariant 1. *At the beginning of any iteration t,*

1. *For all $(u, w, v) \in \mathcal{P}_2$, $l^{(t)}_{(u,w,v)} \geq \alpha^{(t)}$,*
2. *For any edge $e \in E^+ \cup E^-$, $l^{(t)}_e = \left(m^{1/\epsilon}\right)^{cong^{(t)}_e}$.*

Initialization. Initially, the algorithm sets the dual variable $y^{(0)}_{(u,w,v)} \leftarrow 0$ for each triangle $(u,w,v) \in \mathcal{P}_2$. Each edge has an edge length $l^{(0)}_e := (m^{1/\epsilon})^{cong^{(0)}_e}$ which has an initial value 1. Since $\alpha^{(0)}$ is a lower bound for the shortest open triangle, we can safely set $\alpha^{(0)} \leftarrow 3$ initially.

Termination Condition. Since the congestion of each edge is non-decreasing, the length of each edge is also non-decreasing. The algorithm terminates when the total length of all edges becomes too large. Specifically, we define the following potential function

$$\Phi(t) = \sum_{e \in E^+ \cup E^-} (m^{1/\epsilon})^{cong^{(t)}_e}$$

and terminate the algorithm once $\Phi(t)$ surpasses $m^{1/\epsilon}/\exp(\epsilon)$.

Iterations. Within the t-th iteration of the algorithm, the algorithm attempts to send flows through the most beneficial triangle. Specifically, we identify some triangle in $\mathcal{P}_2$ with $(1+\epsilon)$-approximate shortest distance and increase its dual value (and the corresponding edge lengths). To accelerate the process, instead of sending flow through the most beneficial triangle one by one, the algorithm repeatedly selects a *maximal* edge-disjoint set of triangles $S \subseteq \mathcal{P}_2$ such that $l^{(t)}_{(u,w,v)} < (1+\epsilon)\alpha^{(t)}$ for every triangle $(u,w,v) \in S$, where $l^{(t)}$ is the current length function. Then, the algorithm increases the dual variables $y^{(t)}_{(u,w,v)}$ for all triangles $(u,w,v) \in S$ by a fixed amount $\Delta y^{(t)}_{(u,w,v)} := \epsilon^2/\ln m$. To maintain Invariant 1, the algorithm increases the length of e by an $\exp(\epsilon)$ factor, whenever the congestion is increased in the t-th iteration. If S is empty, then there is no triangle $(u,w,v) \in \mathcal{P}_2$ with $l^{(t)}_{(u,w,v)} < (1+\epsilon)\alpha^{(t)}$, which implies that the shortest triangle now has a length at least $(1+\epsilon)\alpha^{(t)}$. In this case, the algorithm increases $\alpha^{(t)}$ by a $(1+\epsilon)$ factor, i.e., $\alpha^{(t+1)} \leftarrow (1+\epsilon)\alpha^{(t)}$.

Computing the Primal Solution with a Small Support Size. It turns out the length function $l^{(t)}_e$ itself, when divided by $\alpha^{(t)}$ is feasible for PRIMAL$^{(2)}$ (see Lemma 3.1). Let $z^{(t)}_e := l^{(t)}_e/\alpha^{(t)}$. The primal objective then becomes $\sum_e z^{(t)}_e = \Phi(t)/\alpha^{(t)}$. To compute the smallest primal objective value, Algorithm 1 selects an iteration T that minimizes $\sum_{e \in E^+ \cup E^-} z^{(T)}_e$.

To ensure the primal solution has a small support size, note that, although we will be able to bound the number of negative edges with non-zero flows, it does not necessarily translate to an upper bound on the number of negative edges with non-zero primal values, as an edge with zero flow has a non-zero primal value of $1/\alpha^{(T)}$. To overcome this issue, we set the primal values of negative edges to be 0 when there is no flow. However, doing so might violate some constraints. To compensate this, we may re-adjust the primal values of some positive edges by increasing the primal value of *all* positive edges by $1/(2\alpha^{(T)})$. When $\alpha^{(T)}$ is sufficiently large, we can upper bound the total increase of the primal values. To ensure our $\alpha^{(T)}$ is large enough, we show that a high value of $\Phi(T)$ implies a high value of $\alpha^{(T)}$. Then, when taking T to be the iteration with the minimum $\sum_{e \in E^+ \cup E^-} z^{(T)}_e$, we restrict $T \geq T_{\min}$, where $T_{\min}$ is the first iteration the potential Φ grows to be at least m^3/ϵ.

In the following subsections, we will prove that the primal solution $\{z_e\}$ output from Algorithm 1 is both feasible and $(1+O(\epsilon))$-approximate. While we have not yet provided details on how to compute the maximal edge-disjoint eligible open triangles, we will discuss the number of iterations needed at the end of this section. By combining this with a maximal edge-disjoint eligible open triangles algorithm in Section 5, we are able to derive the bounds on the running time.

3.1 Feasibility

We will begin by showing the feasibility of $\{z^{(t)}_{uv} = l^{(t)}_{uv}/\alpha^{(t)}\}$ for any t.

LEMMA 3.1. *For any iteration t, $\{z^{(t)}_{uv} = l^{(t)}_{uv}/\alpha^{(t)}\}$ is feasible for PRIMAL$^{(2)}$.*

Proof. Invariant 1 guarantees that $l^{(t)}_{(u,w,v)} \geq \alpha^{(t)}$ for any $(u,w,v) \in \mathcal{P}_2$, which ensures that all the constraints of PRIMAL$^{(2)}$ are satisfied. ∎

Algorithm 1 A $(1+O(\epsilon))$-approximate algorithm for $\textsc{Primal}^{(2)}$.
Input: A vertex set V, a set of m undirected unweighted edges E^+, and a parameter $\epsilon > 0$.
Output: An $(1+O(\epsilon))$-approximate solution $\{z_{uv}\} \in [0,1]^{\binom{V}{2}}$ to $\textsc{Primal}^{(2)}$
Auxiliary Information: $E^- := \binom{V}{2} \setminus E^+$; $\mathcal{P}_2 :=$ the set of open triangles.

function ParallelSteepestDescent($G^+ = (V, E^+), \epsilon$)
▷ *Initialization*
  $t \leftarrow 0$ and $\alpha^{(0)} \leftarrow 3$ and $T_{\min} \leftarrow +\infty$
  $l_{uv}^{(0)} \leftarrow 1$ for all $(u,v) \in E^+ \cup E^-$.
  $y_{(u,w,v)}^{(0)} \leftarrow 0$ for all $(u,w,v) \in \mathcal{P}_2$.
▷ *Compute the primal and dual values*
  while $\Phi(t) := \sum_{uv} l_{uv}^{(t)} < m^{1/\epsilon}/\exp(\epsilon)$ **do**
    If $\Phi(t) \geq m^3/\epsilon$, set $T_{\min} \leftarrow \min(T_{\min}, t)$
    Let $\mathcal{P}_2' := \{(u,w,v) \in \mathcal{P}_2 \mid l_{(u,w,v)}^{(t)} < (1+\epsilon)\alpha^{(t)}\}$ be the set of eligible triangles;
    Compute any maximal edge-disjoint set S of $\mathcal{P}_2'$. ▷ See Section 5.
    if $S \neq \varnothing$ **then**
      For all $(u,w,v) \in \mathcal{P}_2$, set $y_{(u,w,v)}^{(t+1)} \leftarrow \begin{cases} y_{(u,w,v)}^{(t)} + \epsilon^2/\ln m & \text{if } (u,w,v) \in S, \\ y_{(u,w,v)}^{(t)} & \text{otherwise.} \end{cases}$
      For all $(u,v) \in E$, set $l_{uv}^{(t+1)} \leftarrow \begin{cases} l_{uv}^{(t)} \cdot \exp(\epsilon) & \text{if } (u,v) \text{ occur in some triangle in } S, \\ l_{uv}^{(t)} & \text{otherwise.} \end{cases}$
      $\alpha^{(t+1)} \leftarrow \alpha^{(t)}$
    else
      $\alpha^{(t+1)} \leftarrow (1+\epsilon)\alpha^{(t)}$ ▷ Update the lower bound estimate of the shortest triangle.
    end if
    $t \leftarrow t+1$
  end while
▷ *Compute the primal solution*
  $T \leftarrow \arg\min_{t \geq T_{\min}} \dfrac{\Phi(t)}{\alpha^{(t)}}$
  For $(u,v) \in E^+ \cup E^-$, set $z_{uv}^{(T)} \leftarrow \dfrac{l_{uv}^{(T)}}{\alpha^{(T)}}$
  For $(u,v) \in E^+$, set $z_{uv} \leftarrow \min\left\{z_{uv}^{(T)} + \dfrac{\epsilon}{m}, 1\right\}$.
  For $(u,v) \in E^-$, set $z_{uv} \leftarrow \begin{cases} \min\left\{z_{uv}^{(T)}, 1\right\} & \text{if } l_{uv}^{(T)} > 1, \\ 0 & \text{otherwise.} \end{cases}$
  return $\{z_{uv}\}$
end function

However, when setting up the final $\{z_e\}$, if $e \in E^-$ and $l_e^{(T)} = 1$, the algorithm sets z_e to be 0 in Line 20, which reduces the value z_e from what it should be by $1/\alpha^{(T)}$. This may lead to a violation of a triangle (u, w, v)'s primal constraint if $(u, v) \in E^-$ and $l_{uv}^{(T)} = 1$. To address this, we increase $z_e^{(T)}$ by ϵ/m for all positive edge $e \in E^+$. We will now show that $\alpha^{(T)} \geq m/(2\epsilon)$, which implies that the reduction to a negative edge's primal variable is at most $2\epsilon/m$. Therefore, it suffices to increase the z_e values for all positive edges by ϵ/m, as a triangle in $\mathcal{P}_2$ contains exactly two positive edges.

LEMMA 3.2. $\alpha^{(T)} \geq m/(2\epsilon)$.

Proof. Since $\Phi(T_{\min}) \geq m^3/\epsilon$, by an averaging argument, there exists an edge $e \in \binom{V}{2}$ such that $l_e^{(T_{\min})} \geq \Phi(T_{\min})/\binom{n}{2} \geq m^3/(\epsilon n(n-1)/2) \geq m/\epsilon$. This implies that $cong_e^{(T_{\min})} > 0$ and at some iteration prior to $T_{\min}$ the algorithm has sent some flow on some triangle (u, w, v) containing e. Moreover, using the fact that in each iteration t the algorithm only selects the triangles with length less than $(1+\epsilon)\alpha^{(t)}$ and that $\alpha^{(t)}$ is non-decreasing, we know that the triangles selected at the iteration $T_{\min} - 1$ must have length at least $m/(\epsilon \cdot \exp(\epsilon))$.

Hence, whenever $\epsilon < 1/8$ we have

$$\alpha^{(T)} \geq \frac{m}{\epsilon \cdot \exp(\epsilon) \cdot (1+\epsilon)} \geq \frac{m}{2\epsilon}.$$

∎

LEMMA 3.3. *Algorithm 1 outputs a feasible solution* $\{z_e\}$ *for* PRIMAL$^{(2)}$.

Proof. Let (u, v, w) be an open triangle with $uv, vw \in E^+$ and $uw \in E^-$. We will show that $z_{uw}+z_{uv}+z_{vw} \geq 1$. First note that if any of $z_{uw}, z_{uv}^{(T)}, z_{vw}^{(T)}$ is greater than 1 then we are done, as at least one of z_{uw}, z_{uv} or z_{vw} will be equal to 1 by Line 19 and 20 of Algorithm 1. Otherwise, by Line 19 and 20 of Algorithm 1, we have $z_{uw} = z_{uw}^{(T)} + \epsilon/m$, $z_{uv} \geq z_{uv}^{(T)} - 1/\alpha^{(T)}$, and $z_{vw} \geq z_{vw}^{(T)} - 1/\alpha^{(T)}$. Therefore,

$$\begin{aligned} z_{uw} + z_{uv} + z_{vw} &\geq \left(z_{uw}^{(T)} + \epsilon/m\right) + \left(z_{uv}^{(T)} - 1/\alpha^{(T)}\right) + \left(z_{vw}^{(T)} - 1/\alpha^{(T)}\right) \\ &\geq z_{uw}^{(T)} + z_{uv}^{(T)} + z_{vw}^{(T)} + \epsilon/m - \epsilon/(2m) - \epsilon/(2m) && \text{(by Lemma 3.2)} \\ &\geq z_{uw}^{(T)} + z_{uv}^{(T)} + z_{vw}^{(T)} \geq 1 && \text{(by Lemma 3.1)} \end{aligned}$$

∎

We now turn our attention to showing that we always maintain a feasible dual $\{y_{(u,w,v)}^{(t)}\}$ throughout the algorithm. First, we show that the potential increases by at most a factor of $\exp(\epsilon)$ in each iteration.

LEMMA 3.4. $\Phi(t) \leq \exp(\epsilon) \cdot \Phi(t-1)$.

Proof. Note that $\Phi(t) = \sum_e l_e^{(t)} \leq \sum_e \exp(\epsilon) \cdot l_e^{(t-1)} \leq \exp(\epsilon) \cdot \Phi(t-1)$. ∎

LEMMA 3.5. *At the beginning of iteration* t, $\{y_{(u,w,v)}^{(t)}\}$ *is a feasible solution to* DUAL$^{(2)}$. *This holds even for the last iteration* t *where it does not enter the main body of the loop.*

Proof. It suffices to show that $cong_e^{(t)} \leq 1$ for any edge $e \in E^+ \cup E^-$ at the beginning of each iteration t. Since iteration $t-1$ has been executed, and by the condition of the main loop, we have $\Phi(t-1) \leq m^{1/\epsilon}/\exp(\epsilon)$. By Lemma 3.4, we have $\Phi(t) \leq \exp(\epsilon) \cdot \Phi(t-1) \leq \exp(\epsilon) \cdot (m^{1/\epsilon}/\exp(\epsilon)) = m^{1/\epsilon}$ and hence $l_e^{(t)} \leq \Phi(t) \leq m^{1/\epsilon}$. Based on Invariant 1, $l_e^{(t)} = \left(m^{1/\epsilon}\right)^{cong_e^{(t)}}$. Therefore, $cong_e^{(t)} \leq 1$. ∎

3.2 Optimality When we compute our primal solution, we first set $z_e^{(T)} = l_e^{(T)}/\alpha^{(T)}$. Then, we increase z_e for all positive edges by ϵ/m, which in total impose an extra ϵ additive quantity to the primal objective. Let OPT be the optimal objective value for both PRIMAL$^{(2)}$ and DUAL$^{(2)}$. The next lemma gives a bound for $\sum_e z_e^{(T)}$.

LEMMA 3.6. *Suppose that* $2/m \le \epsilon \le 1/10$. *Then,* $\sum_e z_e^{(T)} \le (1 + 15\epsilon) \cdot \mathrm{OPT}$.

Proof. [Proof of Lemma 3.6.] To establish the approximate ratio of the primal solution, it suffices to show that the dual objective $\sum_{p\in\mathcal{P}_2} y_p^{(T)}$ is within a $(1+O(\epsilon))$-factor of the primal objective $\sum_e z_e^{(T)} = \Phi(T)/\alpha^{(T)}$. We first establish the relation between the potential increase $\Phi(t) - \Phi(t-1)$ and the changes to the dual value within iteration $t-1$:

$$\begin{aligned}
\Phi(t) - \Phi(t-1) &= \sum_{(u,w,v)\in S} l^{(t)}_{(u,w,v)}(\exp(\epsilon) - 1) \\
&\le |S| \cdot (\exp(\epsilon) - 1)(1+\epsilon)\alpha^{(t)} \\
&= \left(\frac{\ln m}{\epsilon^2} \sum_{p\in\mathcal{P}_2} \Delta y_p^{(t-1)}\right) \cdot (\exp(\epsilon) - 1)(1+\epsilon)\alpha^{(t)} \\
&\le (1+\epsilon)^2 \frac{\ln m}{\epsilon} \alpha^{(t)} \cdot \sum_{p\in\mathcal{P}_2} \Delta y_p^{(t-1)},
\end{aligned}$$

where $\Delta y_p^{(t-1)} = y_p^{(t)} - y_p^{(t-1)}$ is the flow sent to triangle p at the $(t-1)$th iteration. By rearranging the terms, we obtain:

$$\frac{\Phi(t) - \Phi(t-1)}{\alpha^{(t)}} \le (1+\epsilon)^2 \frac{\ln m}{\epsilon} \cdot \sum_{p\in\mathcal{P}_2} \Delta y_p^{(t-1)}. \tag{1}$$

In Line 17 of Algorithm 1, the algorithm chooses T such that $T \ge T_{min}$ and $\Phi(T)/\alpha^{(T)}$ is minimized, so

$$\sum_e z_e^{(T)} = \frac{\Phi(T)}{\alpha^{(T)}} \le \frac{\Phi(t)}{\alpha^{(t)}} \quad \text{for any } t \ge T_{\min}. \tag{2}$$

On the other hand, by Lemma 3.4 and the fact that Φ is non-decreasing, we have $1 \le \frac{\Phi(t)}{\Phi(t-1)} \le \exp(\epsilon)$. When $0 < \epsilon < 1$, we have $\exp(\epsilon) \le (1+\epsilon)^2$ and

$$\begin{aligned}
\ln\left(\frac{\Phi(t)}{\Phi(t-1)}\right) &\le \frac{\Phi(t)}{\Phi(t-1)} - 1 && (\ln x \le x - 1 \text{ for all } x > 0) \\
&= \frac{\Phi(t)}{\Phi(t-1)} \cdot \frac{\Phi(t) - \Phi(t-1)}{\Phi(t)} \\
&\le \exp(\epsilon) \cdot \frac{\Phi(t) - \Phi(t-1)}{\Phi(t)} \\
&\le (1+\epsilon)^2 \frac{\Phi(t) - \Phi(t-1)}{\Phi(t)}. && (3)
\end{aligned}$$

Therefore, at any iteration t, we can bound $\sum_e z_e^{(T)}$ by

$$\begin{aligned}\sum_e z_e^{(T)} \ln\left(\frac{\Phi(t)}{\Phi(t-1)}\right) &\le \frac{\Phi(t)}{\alpha^{(t)}} \cdot (1+\epsilon)^2 \frac{\Phi(t)-\Phi(t-1)}{\Phi(t)} && \text{(by Equation (3))}\\ &\le (1+\epsilon)^2 \frac{\Phi(t)-\Phi(t-1)}{\alpha^{(t)}}\\ &\le (1+\epsilon)^4 \frac{\ln m}{\epsilon} \cdot \sum_{p\in\mathcal{P}_2} \Delta y_p^{(t-1)}. && \text{(by Equation (1))}\end{aligned}$$

Next, by summing over all $t \ge T_{\min}$, we obtain

$$\sum_e z_e^{(T)} \sum_{t\ge T_{\min}} \ln\left(\frac{\Phi(t)}{\Phi(t-1)}\right) \le (1+\epsilon)^4 \frac{\ln m}{\epsilon} \cdot \left(\sum_{t\ge T_{\min}, p\in\mathcal{P}_2} \Delta y_p^{(t-1)}\right) \tag{4}$$

It is straightforward to see that the summation of the right-hand side telescopes to at most the current dual objective, which is at most OPT. To lower bound the left-hand side, we notice that since $\Phi(T_{\min}-1) < \frac{m^3}{\epsilon}$ and the final $\Phi(T_{last})$ is at least $\frac{m^{1/\epsilon}}{\exp(\epsilon)}$, where T_{last} denotes the last iteration. When $\frac{2}{m} \le \epsilon \le \frac{1}{10}$, we have $\exp(\epsilon)/\epsilon \le m$, thus:

$$\begin{aligned}\sum_{t\ge T_{\min}} \ln\left(\frac{\Phi(t)}{\Phi(t-1)}\right) &= \ln\left(\frac{\Phi(T_{last})}{\Phi(T_{\min})}\right)\\ &\ge \ln\left(\frac{\epsilon\cdot m^{1/\epsilon}}{\exp(\epsilon)\cdot m^3}\right)\\ &= \left(\frac{1}{\epsilon}-3\right)\ln m - \ln\left(\frac{\exp(\epsilon)}{\epsilon}\right)\\ &\ge \left(\frac{1}{\epsilon}-4\right)\ln m. \qquad (5)\end{aligned}$$

Combining all together, when $\frac{2}{m} \le \epsilon \le \frac{1}{10}$, we have

$$\begin{aligned}\sum_e z_e^{(T)} &= \frac{\sum_e z_e^{(T)} \cdot \left(\frac{1}{\epsilon}-4\right)\ln m}{\left(\frac{1}{\epsilon}-4\right)\ln m}\\ &\le \frac{1}{\left(\frac{1}{\epsilon}-4\right)\ln m} \cdot \sum_e z_e^{(T)} \cdot \sum_{t\ge T_{\min}} \ln\left(\frac{\Phi(t)}{\Phi(t-1)}\right) && \text{(by Equation (5))}\\ &\le \frac{1}{\left(\frac{1}{\epsilon}-4\right)\ln m} \cdot (1+\epsilon)^4 \frac{\ln m}{\epsilon} \cdot \left(\sum_{t\ge T_{\min}, p\in\mathcal{P}_2} \Delta y_p^{(t-1)}\right) && \text{(by Equation (4))}\\ &\le \frac{(1+\epsilon)^4}{1-4\epsilon} \cdot \mathrm{OPT}\\ &\le (1+15\epsilon)\cdot \mathrm{OPT} && (\epsilon < 1/10)\end{aligned}$$

□

Using Lemma 3.6, we can show the final output $\{z_e\}$ is a $(1+O(\epsilon))$-approximate solution:

LEMMA 3.7. *Suppose that* $2/m \le \epsilon \le 1/10$. *Algorithm 1 outputs* $\{z_e\}$ *such that* $\sum_{e\in E^+\cup E^-} z_e \le (1+16\epsilon)\cdot \mathrm{OPT}$.

Proof. By assumption, there must be at least one $(+,+,-)$ triangle, so OPT ≥ 1. By Lemma 3.6,

$$\begin{aligned}\sum_{e\in E^+\cup E^-} z_e &\leq \sum_e z_e^{(T)} + m\cdot\frac{\epsilon}{m}\\ &\leq (1+15\epsilon)\mathrm{OPT} + \epsilon\mathrm{OPT}\\ &\leq (1+16\epsilon)\mathrm{OPT}.\end{aligned}$$

□

3.3 Work and Span In this section, we will prove the last piece of Lemma 1.2, the parallel running time of Algorithm 1. To begin with, we establish an upper bound on the number of iterations that share the same value of α.

LEMMA 3.8. *For any fixed α, there will be at most $R = (3/\epsilon)\ln((1+\epsilon)\alpha)$ iterations such that $\alpha^{(t)} = \alpha$.*

Proof. Assume that at some iteration t, we have $\alpha(t) = \alpha$. From Invariant 1, we know that $l^{(t)}_{(u,w,v)} \geq \alpha$ for any triangle $(u,w,v) \in \mathcal{P}_2$. Our objective is to demonstrate that, after $R = (3/\epsilon)\ln((1+\epsilon)\alpha)$ iterations, we have $l^{(t+R)}_{(u,w,v)} \geq (1+\epsilon)\alpha$. Therefore, if α has not been changed, the set of eligible open triangles will be empty and the algorithm has to increase $\alpha^{(t+R)}$.

Assume that $l^{(t)}_{(u,w,v)} \in [\alpha, (1+\epsilon)\alpha)$ for the triangle (u,w,v). Otherwise, since we never decrease the length function, we already have $l^{(t+R)}_{(u,w,v)} \geq (1+\epsilon)\alpha$. If (u,w,v) is ever chosen to an edge-disjoint set S in some iteration i, where $i \in [t, t+R)$, then we must have

$$l^{(i+1)}_{(u,w,v)} \geq \exp(\epsilon)\cdot l^{(t)}_{(u,w,v)} \geq (1+\epsilon)\alpha.$$

On the other hand, if (u,w,v) has not been chosen into S in any iteration, the algorithm must choose at least one edge on (u,w,v) and increase its length by a factor of $\exp(\epsilon)$ after each iteration. By the pigeonhole principle, after $R = (3/\epsilon)\ln((1+\epsilon)\alpha)$ iterations, there exists an edge in the triangle (u,w,v) whose length is increased by a factor of $\exp(\epsilon\cdot R/3)$. Consequently, the contribution of this edge to $l^{(t+R)}_{(u,w,v)}$ satisfies

$$\begin{aligned}l^{(t+R)}_{(u,w,v)} &\geq \exp(\epsilon\cdot R/3)\\ &\geq \exp(\epsilon\cdot\ln((1+\epsilon)\alpha)/\epsilon)\\ &\geq (1+\epsilon)\alpha.\end{aligned}$$

In either case, we can conclude that $l^{(t+R)}_{(u,w,v)} \geq (1+\epsilon)\alpha^{(t)}$ for all $(u,w,v) \in \mathcal{P}_2$. □

To bound the total number of iterations in Algorithm 1, we need to bound the maximum $\alpha^{(t)}$ and the number of different $\alpha^{(t)}$ values. Since our assumption guarantees that $\mathrm{OPT}(\textsc{Primal}^{(2)}) \geq 1$, we know that there will be at least one triangle in $\mathcal{P}_2$. As $\alpha^{(t)}$ is always a lower bound for the shortest triangle at iteration t, the maximum possible value of $\alpha^{(t)}$ is $3\cdot m^{1/\epsilon}$ when the congestion is 1. Based on Lemma 3.8, there will be $O(\epsilon^{-2}\log m)$ iterations for any fixed α.

Moreover, if S is empty, we increase $\alpha^{(t)}$ by a factor of $(1+\epsilon)$. Therefore, there will be at most $O(\log_{1+\epsilon}(m^{1/\epsilon})) = O(\epsilon^{-2}\log m)$ different $\alpha^{(t)}$ values. Combining with Lemma 3.8, we obtain the following lemma:

LEMMA 3.9. *In Algorithm 1, the total number of iterations is $O(\epsilon^{-4}\log^2 m)$.* □

We can estimate the total number of non-zero terms in the output using Lemma 3.9.

LEMMA 3.10. *The output of Algorithm 1, $\{z_e\}$, has $O(\epsilon^{-2} m \log m)$ non-zero values.*

Proof. The output $z_e > 0$ if and only if the congestion of the edge e is not zero. According to Line 9 of the algorithm, each time the algorithm increases the congestion of a negative edge, the congestion of *some two* positive edges must be increased by $\epsilon^2 / \ln m$ each. Since there are m positive edges and the congestion is always at most 1, this implies that at most $\frac{1}{2}\epsilon^{-2} m \ln m$ negative edges can have non-zero congestion. ∎

We have not yet specified how to compute the maximal edge-disjoint set S. In Section 5, we prove Lemma 1.3 by giving a parallel algorithm that finds a maximal edge-disjoint set S.

LEMMA 1.3. *Let $G = (V, E^+ \cup E^-_{>1})$, and $l : \binom{V}{2} \to [1, \infty)$ be a length function with $l(u, v) = 1$ for $uv \in \binom{V}{2} \setminus (E^+ \cup E^{-1}_{>1})$, and $L > 0$ be a length limit. Let $m_f = |E^+ \cup E^-_{>1}|$. Then, there exists a parallel algorithm such that, in $O(m_f^{1.5} \log^3 n)$ work and $O(\log^3 n)$ span, the algorithm returns a maximal edge-disjoint set S of open triangles with length less than L.*

We are now ready to prove the main Lemma 1.2 regarding solving PRIMAL$^{(2)}$. Note that if ϵ is too small (e.g., $\epsilon \le 2/m$ in the last section), as we allow poly$(1/\epsilon)$ = poly(m) span and work, we can simply run a linear program solver (e.g. [20]) to solve (P1) and obtain an approximation ratio of 2.06 by using [17]. Hence, we may assume $\epsilon \ge 2/m$.

LEMMA 1.2. *Given the set E^+ of m positive edges and a parameter $\epsilon > 0$, there exists a parallel algorithm that computes an $(1+\epsilon)$-approximate solution to PRIMAL$^{(2)}$ using $\tilde{O}(\epsilon^{-7} m^{1.5})$ work and $\tilde{O}(\epsilon^{-4})$ span. In addition, the support size of the solution is at most $\tilde{O}(\epsilon^{-2} m)$.*

Proof. By Lemma 3.1 and Lemma 3.7, Algorithm 1 returns a $(1 + O(\epsilon))$-approximate solution $\{z_e\}$ for PRIMAL$^{(2)}$. To implement Algorithm 1 in the parallel setting, note that by Lemma 3.10, there are at most $O(\epsilon^{-2} m \log m)$ negative edges of length greater than 1 throughout the algorithm. By Lemma 1.3 it takes $O(m_f^{1.5} \log^3 n)$ work and $O(\log^3 m_f)$ span to compute a maximal edge-disjoint set S of $\mathcal{P}_2'$, where $m_f = O(\epsilon^{-2} m \log m)$. Therefore, at each iteration, Algorithm 1 takes $O(\epsilon^{-3} m^{1.5} \log^4 n)$ work and $O(\log^3 n)$ span. By Lemma 3.9, there are $O(\epsilon^{-4} \log^2 m)$ iterations, so Algorithm 1 takes $O(\epsilon^{-7} m^{1.5} \log^{6.5} n)$ work and $O(\epsilon^{-4} \log^5 n)$ span in total. Finally, by Lemma 3.10 again, the support size of the returned solution is also $O(\epsilon^{-2} m \log m)$. ∎

4 A 2.4-Approximation Rounding Algorithm

In this section, first, we present a sequential rounding algorithm that achieves a 2.4-approximation ratio and then show how to parallelize it.

Recall that we denote the triangle (u, w, v) by (uw, wv, vu) when we are mapping certain attributes to the edges. Given an assignment $\{x_e\}_{e \in E^+ \cup E^-}$, when we say a triangle (uw, wv, vu) has edge length (a, b, c), we mean $x_{uw} = a, x_{wv} = b$ and $x_{vu} = c$.

For an assignment $\{x_e\}$, we say $\{x_e\}$ satisfies the **triangle inequality**, if for all triangles (uw, wv, vu) with edge length (x_{uw}, x_{vw}, x_{uv}), we have $x_{uw} + x_{wv} \ge x_{vu}$, $x_{wv} + x_{vu} \ge x_{uw}$, and $x_{vu} + x_{uw} \ge x_{wv}$. For an assignment $\{x_e\}$, we say $\{x_e\}$ satisfies the **partial triangle inequality**, if for all $(+, +, -)$-triangles (uw, wv, vu) with edge length (x_{uw}, x_{wv}, x_{vu}), we have $x_{uw} + x_{wv} \ge x_{vu}$.

Our algorithm, shown in Algorithm 2, takes an assignment $\{x_e\}_{e \in E^+ \cup E^-}$ satisfying the partial triangle inequality as the input. To get an assignment satisfying the partial triangle inequality, we first compute a $(1+\epsilon)$-approximate solution for PRIMAL$^{(2)}$ and then set $x_e = z_e$ for all positive edges and $x_e = 1 - z_e$ for all negative edges. A feasible solution in of PRIMAL$^{(2)}$ satisfies that for every $uv \in E^-$, $uw, wv \in E^+$, $z_{vu} + z_{uw} + z_{wv} \ge 1$. This implies $x_{uw} + x_{wv} \ge x_{uv}$, so such $\{x_e\}$ satisfies the partial triangle inequality. Moreover, for all e, since $z_e \in [0, 1]$, we have $x_e \in [0, 1]$.

Algorithm 2 is based on the pivot rounding framework of [3, 17]. The algorithm iteratively selects a random pivot u from the unclustered vertices, forms a cluster by adding each unclustered node v into the

cluster with probability $1 - p_{uv}$, where p_{uv} is defined as

$$p_{uv} = \begin{cases} f^+(x_{uv}) & \text{if } (u,v) \in E^+, \\ f^-(x_{uv}) & \text{if } (u,v) \in E^-, \end{cases}$$

and f^+, f^- are two functions to be determined. Note that in Algorithm 2, we state the step for choosing a random pivot as choosing the first unclustered node from a random permutation (Line 5) for the ease of parallelization in Section 4.2.

The main difference between our algorithm and [17] is that we choose different f^+, f^- functions. This difference arises because in [17] the input satisfies the triangle inequality for all types of triangles, while ours only satisfies the partial triangle inequality.

Algorithm 2 The sequential rounding algorithm.
Input: Graph G and an assignment $\{x_{uv}\}$ satisfying the partial triangle inequality.
Output: A partition of V.

1: **function** SeqRounding($G = (V, E), \{x_{uv}\}$)
2: Draw a permutation π of the vertex set V uniformly at random.
3: $V_0 \leftarrow V, t \leftarrow 0$
4: **while** $|V_t| > 0$ **do**
5: Let the pivot w be the vertex with the smallest $\pi(w)$ in V_t and set $S_t \leftarrow \{w\}$
6: ▷ This step is equivalent to picking the pivot $w \in V_t$ uniformly at random.
7: For each vertex $u \in V_t$, add u to S_t with probability $(1 - p_{uw})$ independently.
8: $V_t \leftarrow V_t \setminus S_t, t \leftarrow t + 1$
9: **end while**
10: **return** $\{S_0, S_1, ..., S_{t-1}\}$
11: **end function**

4.1 Approximation Ratios Let S_w be the cluster of w when w is chosen as a pivot. To obtain an approximate ratio of the algorithm, [17] consider the following terms for a triangle (u, v, w),

$$cost(u, v \mid w) = \begin{cases} Pr[(u \in S_w \text{ and } v \notin S_w) \text{ or } (u \notin S_w \text{ and } v \in S_w) \mid w \text{ is the pivot}] & \text{if } (u,v) \in E^+, \\ Pr[u \in S_w \text{ and } v \in S_w \mid w \text{ is the pivot}] & \text{if } (u,v) \in E^- \end{cases}$$

$$lp(u, v \mid w) = \begin{cases} x_{uv} \cdot Pr[u \in S_w \text{ or } v \in S_w \mid w \text{ is the pivot}] & \text{if } (u,v) \in E^+, \\ (1 - x_{uv}) \cdot Pr[u \in S_w \text{ or } v \in S_w \mid w \text{ is the pivot}] & \text{if } (u,v) \in E^- \end{cases}$$

The term $cost(u, v \mid w)$ can be intuitively understood as the cost of the edge (u, v) for Algorithm 2 when w is selected as the pivot. If w is selected as the pivot and (u, v) is a positive edge, then a disagreement occurs if exactly one of u or v is clustered into S_w. If w is selected as the pivot and (u, v) is a negative edge, then the disagreement cost is incurred if both u and v are clustered into S_w.

On the other hand, the term $lp(u, v \mid w)$ represents the cost of the edge (u, v) for the assignment x_e when w is chosen as the pivot. In a high-level sense, we are trying to charge the actual cost to the objective value of the LP solution, so we will need to make sure that each term in the objective function is charged by at most one triangle throughout the algorithm. Here, we charge the cost contributed by edge uv whenever at least one of u or v is clustered into S_w. The contribution of uv to the object value is either x_{uv} or $(1 - x_{uv})$, depending on whether $(u, v) \in E^+$ or $(u, v) \in E^-$.

It should be noted that the corresponding probabilities can be expressed by p_{uw} and p_{vw}. Once we substitute them into the terms, we obtain the following expressions for $cost(u, v \mid w)$ and $lp(u, v \mid w)$.

$$cost(u, v \mid w) = \begin{cases} p_{uw} + p_{vw} - 2p_{uw}p_{vw} & \text{if } (u,v) \in E^+, \\ (1 - p_{uw}) \cdot (1 - p_{vw}) & \text{if } (u,v) \in E^- \end{cases}$$

$$lp(u, v \mid w) = \begin{cases} x_{uv} \cdot (1 - p_{uw}p_{vw}) & \text{if } (u, v) \in E^+, \\ (1 - x_{uv}) \cdot (1 - p_{uw}p_{vw}) & \text{if } (u, v) \in E^- \end{cases}$$

The analysis considers the cost for a triangle (u, w, v), where each of u, w, and v is chosen as a pivot with equal probability, $ALG(uwv)$ and $LP(uwv)$ represents the cost for triangle (u, w, v) for Algorithm 2 and the assignment $\{x_e\}$, respectively.

$$ALG(uwv) = cost(u, v \mid w) + cost(u, w \mid v) + cost(v, w \mid u)$$
$$LP(uwv) = lp(u, v \mid w) + lp(u, w \mid v) + lp(v, w \mid u)$$

[17] showed if the ratio between $ALG(uwv)$ and $LP(uwv)$ is upper bounded ρ for every triangle (u, w, v), the output of the Algorithm 2 has an approximation ratio of at most ρ in expectation. More precisely,

LEMMA 4.1. ([17]) *Fix a set of functions (f^+, f^-) with $f^+(0) = f^-(0) = 0$. If $ALG(uwv) \le \rho LP(uwv)$ for every $u, w, v \in V$. Let ALG be the disagreement in clustering Algorithm 2 outputs and $LP = \sum_{e \in E^+} x_e + \sum_{e \in E^-}(1 - x_e)$, then*

$$E[ALG] \le \rho \cdot LP$$

The next question is: What is the best choice of functions f^+ and f^- that minimize ρ? [17] analyze four different types of triangles (namely, $(+,+,+), (+,+,-), (+,-,-)$, and $(-,-,-)$) and achieve a value of $\rho = 2.06$ by carefully selecting f^+ and f^-. Note that for $(+,+,-)$ and $(+,+,+)$ triangles, the triangle inequality is necessary in order to obtain such an approximation ratio with respect to the functions they have designed. As we only have the partial triangle inequality for $(+,+,-)$ triangles, we will need to come up with different f^+ and f^- functions.

We will first show how to pick the functions to achieve such a 2.4 approximation ratio when the solution satisfies the partial triangle inequality. Then, we will show that under the framework of [17], the 2.4 factor is the best ratio we can achieve when the solution does not satisfy the triangle inequality for all the triangles, but only the partial triangle inequality for $(+,+,-)$ triangles.

LEMMA 4.2. *Fix (f^+, f^-) as*

$$f^+(x) = \begin{cases} 1.2x & \text{if } x \le \frac{5}{6}, \\ 1 & \text{if } x \ge \frac{5}{6} \end{cases}$$

and $f^-(x) = x$. For any $\{x_e\}$ such that $x_{uw} + x_{wv} \ge x_{uv}$ holds for any $(u, w), (v, w) \in E^+$, $(u, v) \in E^-$, we have $ALG(uwv) \le 2.4 \cdot LP(uwv)$.

To show that our chosen functions $f^+(x)$ and $f^-(x)$ yield a 2.4 approximation ratio, we will conduct a case-by-case analysis based on different types of triangles. It is worth noting that [17] has already established the ratio for $(-,-,-)$ and $(+,-,-)$ triangles even when the solution does not obey the triangle inequality.

LEMMA 4.3. ([17]) *Fix $f^-(x) = x$, we have $ALG(uwv) \le LP(uwv)$ for all $(-,-,-)$ triangles.*

LEMMA 4.4. ([17]) *Fix $f^-(x) = x$, if $f^+(x) \le 2x$ for $x \in [0, 1]$, then we have $ALG(uwv) \le 2LP(uwv)$, for all $(+,-,-)$ triangles.*

Let (a, b, c) denote the edge lengths of a triangle (u, w, v), that is, $x_{uw} = a$, $x_{vw} = b$ and $x_{uv} = c$. Define the function $\mathcal{C}(a, b, c)$ as follows:

$$\mathcal{C}(a, b, c) = ALG(uwv) - 2.4LP(uwv)$$

We begin by showing that for $(+,+,-)$ triangles with edge weights (a, b, c) that satisfies the partial triangle inequality, we have $\mathcal{C}(a, b, c) \le 0$.

LEMMA 4.5. *Given our choice of $f^+(x)$ and $f^-(x)$, for any $(+,+,-)$ triangle with edge weights (a,b,c), where $a+b \geq c$ and $a,b,c \in [0,1]$, we have $\mathcal{C}(a,b,c) \leq 0$.*

Proof. For a $(+,+,-)$ triangle, we have

$$\begin{aligned}\mathcal{C}(a,b,c) =&(1+2c-2cf^+(a)-2cf^+(b)+f^+(a)f^+(b))-\\ &2.4(1+a+b-c-bcf^+(a)-acf^+(b)-(1-c)f^+(a)f^+(b))\end{aligned}$$

Depends on whether $a \geq \frac{5}{6}$ or $b \geq \frac{5}{6}$, we have 3 different cases. When $a,b \in (\frac{5}{6},1]$, we have $f^+(x)=1$ and

$$\begin{aligned}\mathcal{C}(a,b,c) &= (1+2c-2c-2c+1)-2.4(1+a+b-c-bc-ac-1+c)\\ &= 2-2c-2.4a-2.4b+2.4ac+2.4bc\\ &= (2.4a+2.4b-2)c+2-2.4a-2.4b \qquad (2.4a+2.4b-2 \geq 0\ )\\ &\leq 2.4a+2.4b-2+2-2.4a-2.4b \leq 0\end{aligned}$$

Since a and b are asymmetric, the second case is $a \in [0,\frac{5}{6}]$ and $b \in (\frac{5}{6},1]$. We know

$$\begin{aligned}\mathcal{C}(a,b,c) &= (1+2c-2.4ac-2c+1.2a)-2.4(1+a+b-c-1.2abc-ac-1.2a+1.2ac)\\ &= -1.4+1.68a-2.4b-2.88ac+2.88abc\\ &= -1.4-(-1.68a+2.4b)-2.88ac(1-b) \leq 0\end{aligned}$$

The last case is when $a,b \in [0,\frac{5}{6}]$, we have $f^+(x)=1.2x$ and

$$\begin{aligned}\mathcal{C}(a,b,c) &= (1+2c-2.4ac-2.4bc+1.44ab)-2.4(1+a+b-c-0.96abc-1.44ab)\\ &= -1.4-2.4a-2.4b+4.4c+4.896ab-2.4ac-2.4bc+2.304abc\\ &= (4.4-2.4a-2.4b+2.304ab)c-2.4a-2.4b+4.896ab-1.4\end{aligned}$$

Since $4.4-2.4a-2.4b+2.88ab \geq 0$ for $a,b \in [0,\frac{5}{6}]$, $\mathcal{C}(a,b,c)$ will be maximized when $c=min(1,a+b)$. Another point is that $\mathcal{C}(a,b,c)$ is maximized when $a=b$. When $a=b \leq \frac{1}{2}$, we have

$$\begin{aligned}\mathcal{C}(a,b,c) &= (4.4-2.4a-2.4b+2.304ab)c-2.4a-2.4b+4.896ab-1.4\\ &= (4.4-2.4a-2.4a+2.304aa)2a-2.4a-2.4a+4.896aa-1.4\\ &= 4.608a^3-4.704a^2+4a-1.4 = (a-0.5)(4.608a^2-2.4a+2.8) \leq 0\end{aligned}$$

When $a=b \in [\frac{1}{2},\frac{5}{6}]$, we have

$$\begin{aligned}\mathcal{C}(a,b,c) &= (4.4-2.4a-2.4a+2.304aa)-2.4a-2.4a+4.896aa-1.4\\ &= 7.2a^2-9.6a+3 = (a-0.5)(7.2a-6) \leq 0\end{aligned}$$

Combining all cases, we have $\mathcal{C}(a,b,c) \leq 0$ for any $(+,+,-)$ triangle whenever $a+b \geq c$.
∎

The remaining case is the $(+,+,+)$ triangles.

LEMMA 4.6. *Given our choice of $f^+(x)$ and $f^-(x)$, for any $(+,+,+)$ triangle with edge weights (a,b,c) and $a,b,c \in [0,1]$, we have $\mathcal{C}(a,b,c) \leq 0$.*

Proof. Consider a $(+,+,+)$ triangle with edge lengths (a,b,c). We have:

$$\begin{aligned}\mathcal{C}(a,b,c) =&2\big(f^+(a)+f^+(b)+f^+(c)-f^+(a)f^+(b)-f^+(b)f^+(c)-f^+(a)f^+(c)\big)-\\ &2.4\big(a+b+c-cf^+(a)f^+(b)-af^+(b)f^+(c)-bf^+(a)f^+(c)\big)\end{aligned}$$

Since $\mathcal{C}(a,b,c)$ is symmetric, we can assume without loss of generality that $a \geq b \geq c$. We consider two cases:

Case 1: At least one of a, b, c is greater than $\frac{5}{6}$. Since $a \geq b \geq c$, $a \geq \frac{5}{6}$. We have:

$$\begin{aligned}\mathcal{C}(a,b,c) &= 2\big(1 - f^+(b)f^+(c)\big) - 2.4\big(a + b + c - cf^+(b) - af^+(b)f^+(c) - bf^+(c)\big)\\ &= -(2.4a - 2)(1 - f^+(b)f^+(c)) - 2.4(b - bf^+(c)) - 2.4(c - cf^+(b)) \leq 0\end{aligned}$$

Case 2: $a, b, c \in [0, \frac{5}{6}]$. In this case, we have:

$$\begin{aligned}\mathcal{C}(a,b,c) &= 2\big(1.2a + 1.2b + 1.2c - 1.44ab - 1.44ac - 1.44bc\big) - 2.4\big(a + b + c - 4.32abc\big)\\ &= 2.88(3.6abc - ab - bc - ac)\\ &= 2.88(ab(1.2c - 1) + bc(1.2a - 1) + ac(1.2b - 1)) \leq 0\end{aligned}$$

Combining all two cases, we have shown that $\mathcal{C}(a,b,c) \leq 0$ for $(+,+,+)$ triangles. ∎

Now, we show that the 2.4 approximation ratio is the best we can obtain when the solution does not satisfy the triangle inequality for all the triangles, but only the partial triangle inequality for $(+,+,-)$ triangles.

LEMMA 4.7. *For any (f^+, f^-) with $f^+(0) = f^-(0) = 0$, there exists a graph G and an assignment $\{x_e\}$ that satisfies the partial triangle inequality such that there is a triangle (u,w,v) in G with $ALG(uwv) \geq 2.4LP(uwv)$.*

Proof. To establish the lower bound, let G be a graph containig a $(-,-,-)$ triangle with edge weights $(1,1,1)$, a $(+,+,-)$ triangle with edge lengths $(0.5, 0.5, 1)$, and a $(+,+,+)$ triangle with edge weights $(0,0,\frac{1}{2})$.

Consider the $(-,-,-)$ triangle with edge weights $(1,1,1)$. Note that $LP(uwv) = 0$. If $f^-(1) < 1$, then $ALG(uwv) > 0$, which makes $ALG(uwv)/LP(uwv)$ unbounded. Therefore, we may assume $f^-(1) = 1$.

Next, we examine the $(+,+,-)$ triangle with edge weights $(0.5, 0.5, 1)$. Here, we have

$$LP(uwv) = 2 \cdot \frac{1}{2}\left(1 - f^+\left(\frac{1}{2}\right) \cdot f^-(1)\right) = 1 - f^+\left(\frac{1}{2}\right)$$

On the other hand, we have

$$\begin{aligned}ALG(uwv) &= 2\left(f^+\left(\frac{1}{2}\right) + f^-(1) - 2f^+\left(\frac{1}{2}\right) \cdot f^-(1)\right) + \left(1 - f^+\left(\frac{1}{2}\right)\right)^2\\ &= \left(1 - f^+\left(\frac{1}{2}\right)\right) \cdot \left(3 - f^+\left(\frac{1}{2}\right)\right)\end{aligned}$$

Hence, we find that $ALG(uwv) = (3 - f^+\left(\frac{1}{2}\right)) \cdot LP(uwv)$.

Consider now the $(+,+,+)$ triangle with edge weights $(0,0,\frac{1}{2})$. Here, we have $LP(uwv) = \frac{1}{2}$ and $ALG(uwv) = 2f^+\left(\frac{1}{2}\right)$. Thus, we obtain $ALG(uwv) = 4f^+\left(\frac{1}{2}\right) LP(uwv)$.

Combining the above two equations, we obtain the inequality

$$ALG(uwv) \geq \min\left(3 - f^+\left(\frac{1}{2}\right), 4f^+\left(\frac{1}{2}\right)\right) LP(uwv)$$

When $3 - f^+\left(\frac{1}{2}\right) = 4f^+\left(\frac{1}{2}\right)$, we obtain $f^+\left(\frac{1}{2}\right) = 0.6$ and $ALG(uwv) \geq 2.4LP(uwv)$. ∎

4.2 The Parallel Rounding Algorithm In Algorithm 2, after a pivot u is chosen, each unclustered node v tries to join the cluster of u with probability $1 - p_{uv}$. To parallelize Algorithm 2, first, we consider an equivalent sequential algorithm, Algorithm 3. In this algorithm, instead of revealing the randomness of all the edges incident to u after the pivot u is chosen, we reveal all such randomness at the beginning of the algorithm, before we started to perform any pivoting steps. This can be thought as first constructing an instance $G' = (V, E'^+, E'^-)$ where each edge uv is labelled as $+$ with probability $1 - p_{uv}$ and labelled as $-$ with probability p_{uv}. Running the standard Pivot algorithm of [3] on G' will produce exactly the same output as if we run Algorithm 2 directly, if we use the same randomness for p_{uv} in both algorithms. In sum, we can pre-round the assignments $\{x_e\}$ into an instance G' and then the remaining step is to perform the Pivot algorithm.

Algorithm 3 The sequential Pivot algorithm with pre-rounding.
Input: Graph G and an assignment $\{x_{uv}\}$ satisfying the partial triangle inequality.
Output: A partition of V.

function SeqPreRounding($G = (V, E), \{x_{uv}\}$)
  for (u, v) such that $p_{uv} < 1$ **do**
    add (u, v) to E'^+ with probability $1 - p_{uv}$
  end for
  Draw a permutation π of the vertex set V uniformly at random.
  $V_0 \leftarrow V, t \leftarrow 0$
  while $|V_t| > 0$ **do**
    Let the pivot w be the vertex with the smallest $\pi(w)$ in V_t and set $S_t \leftarrow \{w\}$
    For $u \in V_t$ such that $(u, w) \in E'^+$, add u to S_t
    $V_t \leftarrow V_t \setminus S_t, t \leftarrow t + 1$
  end while
  return $\{S_0, S_1, ..., S_{t-1}\}$
end function

To parallelize Algorithm 3, note that the Pivot algorithm of [3] is known to be implementable efficiently in the parallel setting [10, 29]. The observation was that we can perform multiple steps of Algorithm 3 in one parallel round as follows. Vertices whose π-values are local minimum serve as the pivots. All non-pivot nodes then join the neighboring pivot with the smallest π-value. For completeness, we give the description of our parallel algorithm in Algorithm 4.

Note that the pivots chosen throughout the algorithm are exactly the vertices that comprise the greedy maximal independent set (MIS) induced by the permutation π in $G'^+ = (V, E'^+)$. [29] showed such a process terminates $O(\log n)$ rounds. We can see that Algorithm 4 produces exactly the same output as Algorithm 3 if they are coupled with the same random permutation π and the same randomness for the probability $\{p_{uv}\}$.

Finally, it is important to note that for our chosen functions f^+ and f^-, if $x_{uv} = 1$ then $p_{uv} = 1$. This implies we can ignore the edge uv as it will never be added to E'^+. Therefore, Algorithm 4 takes $\tilde{O}(m_f)$ work and $\tilde{O}(1)$ span, where $m_f = |\{e \in E^- \mid x_e < 1\} \cup E^+|$ is the number of positive edges plus the number of negative edges such that $x_e < 1$.

Combining all together, we have the following lemma:

Lemma 4.8. (A restatement of Lemma 1.1) *Given a graph $G^+ = (V, E^+)$ and an assignment $\{x_e\}_{e \in E^+ \cup E^-}$ satisfying the partial triangle inequality. Let $LP = \sum_{e \in E^+} x_e + \sum_{e \in E^-} (1 - x_e)$ and $m_f = |\{e \in E^- \mid x_e < 1\} \cup E^+|$. Algorithm 4 outputs a clustering that is upper-bounded by $2.4 \cdot LP$ in $\tilde{O}(m_f)$ work and $\tilde{O}(1)$ span.*

By Lemma 1.3 and Lemma 4.8, we prove our main theorem as follows:

Proof. [Proof of Theorem 1.1] First, we use Algorithm 1 to compute a $(1 + \epsilon)$-approximate solution $\{z_e\}$ of Primal$^{(2)}$. We then set $x_e = z_e$ for all positive edges and $x_e = 1 - z_e$ for all negative edges. Let

Algorithm 4 The parallel rounding algorithm.
Input: Graph G and an assignment $\{x_{uv}\}$ satisfying the partial triangle inequality.
Output: A partition of V, $\mathcal{S}$.

function ParallelRounding($G = (V, E), \{x_{uv}\}$)
  for (u, v) such that $p_{uv} < 1$ **do**
    add (u, v) to E' with probability $1 - p_{uv}$
  end for
  $G' \leftarrow (V, E')$
  Draw a permutation π of the vertex set V uniformly at random.
  while $|V'| \geq 1$ **do**
    Let $W = \{u \in G' \mid \pi(u) < \pi(v)$ for every $(u, v) \in G'\}$.
    ▷ W is the set of vertices in G' with no earlier neighbors.
    For $w \in W$, set $S(w) \leftarrow \{w\}$
    for $v \in V(G') \setminus W$ **do**
      Let $w = \arg\min_{u \in W, (u,v) \in G'} \pi(u)$
      $S(w) \leftarrow S(w) \cup \{v\}$
      ▷ every non-pivot vertex v joins the adjacent pivot with the smallest $\pi(\cdot)$-value, if it exists.
    end for
    $G' \leftarrow G' \setminus \bigcup_{w \in W} S(w)$
    $\mathcal{S} \leftarrow \mathcal{S} \cup (\bigcup_{w \in W} \{S(w)\})$
  end while
  return $\mathcal{S}$
end function

$LP = \sum_{e \in E^+} x_e + \sum_{e \in E^-} (1 - x_e)$. We have $LP = \sum_{e \in E^+ \cup E^-} z_e \leq (1 + \epsilon) \cdot \text{OPT}(\text{Primal}^{(2)})$. By Lemma 1.3, this step takes $\tilde{O}(\epsilon^{-7} m)$ work and $\tilde{O}(\epsilon^{-4})$ span.

After running Algorithm 4, we obtain a clustering whose cost is at most $2.4 \cdot LP = (2.4 + O(\epsilon)) \cdot \text{OPT}(\text{Primal}^{(2)}) \leq (2.4 + O(\epsilon)) \cdot \text{OPT}(\text{Primal}_I)$, where $\text{OPT}(\text{Primal}_I)$ denotes the optimal value of the correlation clustering problem. Since $m_f = \tilde{O}(\epsilon^{-2} m)$, by Lemma 4.8, this step takes $\tilde{O}(\epsilon^{-2} m)$ work and $\tilde{O}(1)$ span.

The total running cost is $\tilde{O}(\epsilon^{-7} m)$ work and $\tilde{O}(\epsilon^{-4})$ span, which is dominated by the cost for solving $\text{Primal}^{(2)}$. ∎

5 Maximal Edge-Disjoint Eligible Open Triangles

Recall that we are given a graph $G = (V, E^+ \cup E^-_{>1})$, where E^+ is the set of positive edges and $E^-_{>1}$ is the set of negative edges with non-zero flow (and thus, with length greater than 1). We let $m_f = |E^+ \cup E^-_{>1}|$ and $m = |E^+|$. Note that we will always have $m_f = O(m \log m/\epsilon^2)$ as implied by Lemma 3.10. We are also given a length function $l : \binom{V}{2} \to [1, \infty)$, with $l(u, v) = 1$ for each $(u, v) \in \binom{V}{2} \setminus (E^+ \cup E^-_{>1})$. Throughout this section, we say that an open triangle (u, w, v) is *eligible* if $(w, u), (w, v) \in E^+$, $(u, v) \notin E^+$, and $l(w, u) + l(w, v) + l(u, v) < (1 + \epsilon)\alpha$. Our task is to compute a maximal edge-disjoint set S of $\mathcal{P}'_2$, where $\mathcal{P}'_2 := \{(u, w, v) | \ (u, w), (w, v) \in E^+, (u, v) \notin E^+, l(u, w, v) < (1 + \epsilon)\alpha\}$ is the set of eligible open triangles.

In this section, we will prove Lemma 1.3 by giving a parallel combinatorial algorithm for finding a maximal edge-disjoint eligible open triangles. The algorithm uses $O(m_f^{1.5} \log^3 n)$ work and $O(\log^3 n)$ span. Additionally, we will show that if Boolean matrix multiplication does not have a truly subcubic time combinatorial algorithm, our algorithm is nearly work-optimal.

5.1 Parallel maximal edge-disjoint eligible open triangles

We first show a natural sequential algorithm for maximal edge-disjoint eligible open triangles and why it is difficult to parallelize it.

The Conflict Graph Approach. One may think of constructing a *conflict graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{P}_2'$ is the set of all eligible $(+,+,-)$ triangles and $e \in \mathcal{E}$ means there is a common edge between two eligible triangles. Then once we run an efficient parallel greedy MIS algorithm [10, 29], we can get maximal edge-disjoint eligible open triangles. However, a very subtle point is that despite there being $O(m^{1.5})$ $(+,+,+)$ triangles on the graph G, there could be as many as eligible $\Theta(mn)$ $(+,+,-)$ triangles, which is $\Theta(n^3)$ when the graph is dense. Thus, simulating the greedy MIS algorithm directly takes up to $\tilde{O}(mn)$ total work in the worst case.

A Sequential Greedy MIS Algorithm in $O(m_f^{1.5})$ Time. It is certainly not necessary to enumerate all $O(mn)$ eligible $(+,+,-)$ triangles at once before invoking the MIS computation on the conflict graph $\mathcal{G}$. Consider the following sequential algorithm: for each positive edge $(w,u) \in E^+$, the algorithm considers each incident positive edges (w,v) in the order of non-decreasing length. Sorting the edges is beneficial because once $l(w,u) + l(w,v) + 1 \geq (1+\epsilon)\alpha$ we are certain that all eligible triangles involving (w,u) were explored. Hence, it suffices to consider only the edges (w,v) such that $l(w,u) + l(w,v) + 1 < (1+\epsilon)\alpha$.

Upon considering an edge (w,v), the algorithm checks if the triangle (u,w,v) is eligible. If (u,w,v) is an eligible $(+,+,-)$ triangle, the algorithm simply adds this triangle to the MIS and removes both edges (w,u) and (w,v) from E^+. Otherwise, we have found an *unwanted* triangle: it could be a $(+,+,+)$ triangle, a $(+,+,-)$ triangle whose negative edge has a large length, or a $(+,+,-)$ triangle whose negative edge has already been added some triangle in the MIS.

In the end of the algorithm, we are able to deduce that for each positive edge $(w,u) \in E^+$, either it belongs to some eligible triangles in the returned MIS, or all triangles involving the edge (w,u) are now unwanted. To analyze the runtime of the sequential algorithm, it suffices to bound the number of triangles that are once considered throughout the execution. We notice that the number of once-considered triangles is at most the number of unwanted triangles plus the size of the returned MIS.

To bound the total number of unwanted triangles, we observe that "the third edge" of the inspected unwanted triangle is either a positive edge, a negative edge that is already taken into the MIS, or a negative edge that has non-zero congestion. Such negative edges are not many! There can only be at most $O(m_f)$ of them. Therefore, with the following folklore result, we are able to bound the number of explored unwanted triangles (we call them *alive* triangles) and thus obtain a $O(m_f^{1.5})$ time sequential algorithm.

LEMMA 5.1. (FOLKLORE) *Let $X \subseteq \binom{V}{2}$ be an arbitrary set of edges. Then there are at most $|X|^{1.5}$ triangles using only edges in X.* □

COROLLARY 5.1. *At any moment, the number of triangles that are* explored *but* unwanted *is at most $O(m_f^{1.5})$.* □

Challenges to Parallelization. There are some challenges one has to overcome when parallelizing the above sequential algorithm. The first idea would be to use separate processors for each positive edge (with a direction) $(w,u) \in E^+$. Imagine that each positive edge has a list of positive edges (w,v) to be explored. We say that an edge with a direction $(w,u) \in E^+$ is *active* if there are still unexplored triangles (u,w,v) for (w,u) where $(w,v) \in E^+$. To simulate the sequential algorithm, each processor on behalf of a positive edge attempts to explore an eligible triangle. However, there could be some issues when multiple eligible triangles are found at the same time.

For example, if a parallel algorithm discovers $O(m)$ eligible triangles (one for each positive edge) at the same time, there could be multiple eligible triangles sharing the same edge. In this case, only a few triangles can be added to the MIS and the others become either unwanted (if the negative edge is used up) or destroyed (if a positive edge is used up). This creates a long dependency chain and we will have no guarantee that this algorithm terminates in polylogarithmic time. To mitigate this situation, it seems that for each positive edge $(w,u) \in E^+$, the algorithm has to consider more than one triangle at a time and run a parallel MIS on a larger set of eligible triangles. However, if the algorithm discovers too many eligible triangles at a time, the total work may become too large and exceed $\Omega(m_f^{1.5})$.

5.1.1 The parallel algorithm for maximal edge-disjoint eligible open triangles

The Trick of Doubling or Reset. Fortunately, we can apply a *doubling trick* to the above approach. The parallel algorithm now executes in *rounds*, and there is a global "exploration rate parameter" r, initially set to be $1/2$. In each round, every positive edge explores a bunch of r new triangles. Next, the algorithm collects all eligible triangles $\mathcal{C}$ and runs a parallel maximal independent set (MIS) algorithm on the conflict graph $\mathcal{G}[\mathcal{C}]$. The eligible triangles added to the MIS are then removed from the graph. Finally, depending on how many triangles are still alive (explored but unwanted) on the graph, the algorithm either *doubles* the parameter r or *resets* r to be 1.

Intuitively, in each round, the algorithm explores a set of eligible $(+,+,-)$ triangles as long as its size is within a constant fraction of currently alive triangles. Since the total number of alive triangles is at most $O(m_f^{1.5})$ by Corollary 5.1, the total work of the algorithm can then be $\tilde{O}(R \cdot m_f^{1.5})$, where R is the total number of rounds (we will show in Lemma 5.3 that $R = O(\log^2 n)$).

The algorithm is summarized in Algorithm 5. To describe our algorithm in greater detail, we introduce the following notations and highlight the main idea.

Active Arcs A^i. For each positive edge $(w,u) \in E^+$, there are two ways to form an eligible open triangle — either attaching another positive edge incident to w or incident to u. Moreover, it suffices to consider the eligible triangles where (w,u) is the shorter positive edge. Our parallel algorithm considers these two types of triangles separately. In particular, at each round i, the algorithm maintains an *active arc set* A^i that contains all *arcs* (w,u) such that there are still some positive edges incident to w not being explored yet from the viewpoint of (w,u). An arc becomes inactive if all eligible triangles are explored or the associated edge is removed from E^+.

Alive Triangles $alive(w,u)$**.** As we mentioned before, an *alive* triangle is an explored triangle but unwanted (either ineligible or some edge that already belongs to a triangle in the output set). Given an active arc $(w,u) \in A^i$, we define the set $alive(w,u)$ to be the triangles (u,w,v) that has been explored so far, such that $l(w,u) < l(w,v)$ (or $ID(u) \le ID(v)$ if $l(w,u) = l(w,v)$) and $(w,v) \in E^+$ *has not been removed yet*.

Sorted Neighbor Lists $N_w^{\text{static}}, N_w^i$**.** For all active arcs leaving w, the lists of positive edges to be explored are all incident to w. Hence, it would be convenient for the algorithm to sort the neighbors $N(w)$ by its incident edge length $l(w,v)$ in increasing order, and breaking ties by vertex ID. The sorted list does not change frequently, as there will be many active arcs accessing it via the exploration. We keep the sorted list as N_w^{static} and only update the list in each "reset". In particular, in a round with exploration rate r, each active edge $(w,u) \in A^i$ explores the neighbors in $N_w^{\text{static}}[\text{idx}_{\text{static}}(w,u)+r, \ldots, \text{idx}_{\text{static}}(w,u)+2r-1]$, where $\text{idx}_{\text{static}}(w,u)$ is the index of u appearing in N_w^{static}. We remark that (1) it is possible for an active arc to explore a positive edge that has just been removed in the previous round, and (2) after the "reset", the same ineligible triangle may be explored (and become alive) again, but it will not cause a problem for us.

Moreover, to support fast calculation to $|alive(w,u)|$ at the beginning of each round, the algorithm maintains the latest *sorted list of neighbors* N_w^i for each round i and each vertex w.

Counting Alive Triangles Faster with $cur(w,u)$**.** To compute $|alive(w,u)|$, the algorithm memoizes the quantity $cur(w,u)$ which is defined to be the edge length $l(w,v)$ of the last explored triangle (and the vertex ID v for tie-breaking). Since the algorithm maintains the updated sorted list N_w^i in each round, it suffices to perform a binary search on N_w^i to obtain the exact count $|alive(w,u)| = (\text{idx}_{cur}(w,u) - \text{idx}_{self}(w,u))$, where $\text{idx}_{cur}(w,u)$ is the index of the last explored neighbor (who has not been deleted yet) from the active arc (w,u) in the list N_w^i, and $\text{idx}_{self}(w,u)$ is the index of the vertex u itself in N_w^i.

We remark that there are definitely more efficient parallel data structures (i.e., $O(1)$ span) that maintain the number of alive triangles. However, the bottleneck to our algorithm is the MIS part which already has an $O(\log m_f) = O(\log n)$ span. Thus, we choose a simpler $O(\log n)$ span implementation for ease of understanding.

Resetting the Exploration Rate. If the count of alive triangles becomes too small, say less than $\frac{1}{4}|A^i| \cdot r$, the algorithm can not afford to explore $2r$ more triangles and hence "resets" the exploration. Specifically, the algorithm sets $r \leftarrow 1$, updates the sorted neighboring list N_w^{static}, and resets the search progress $cur(w,u)$ for every active arc. Notice that two things are not being reset — the active arcs A^i is

Algorithm 5 Find any maximal edge-disjoint set of eligible $(+,+,-)$ triangles.

function MaximalEligibleTriangle(G)
  $\mathcal{S} \leftarrow \emptyset$; $i \leftarrow 0$; $r \leftarrow 1/2$.
  For each $w \in V$, obtain two lists of sorted neighbors $N_w^{\text{static}}, N_w^0 \leftarrow N(w)$.
  For each positive edge, add its two directions to the active set $A^0 \leftarrow E^+$.
  For each edge with direction $(w,u) \in E^+$, set $cur(w,u) = (l(w,u), u)$.
  while $A^i \neq \emptyset$ **do**
  ▷ *Part 1: resetting or doubling the exploration rate.*
    for each active arc $(w,u) \in A^i$ **parallel do**
      Locate indices $\text{idx}_{cur}(w,u)$ and $\text{idx}_{self}(w,u)$ in N_w^i using binary search.
      Obtain the count $|alive(w,u)| = \text{idx}_{cur}(w,u) - \text{idx}_{self}(w,u)$.
    end for
    if $\sum_{(w,u)\in A^i} |alive(w,u)| < \frac{1}{4}|A^i| \cdot r$ **then**
      "Reset": $r \leftarrow 1$.
      For each active arc $(w,u) \in E^+$ set $cur(w,u) \leftarrow (l(w,u), u)$.
      Update the list of sorted neighbors $N_w^{\text{static}} \leftarrow N_w^i$.
    else
      "Double": $r \leftarrow 2r$.
    end if
  ▷ *Part 2: collecting eligible triangles.*
    $\mathcal{C} \leftarrow \emptyset$.
    for each $(w,u) \in A^i$, $v \in N_w^{\text{static}}[\text{idx}_{\text{static}}(w,u) + r, \ldots, \text{idx}_{\text{static}}(w,u) + 2r - 1]$ **do**
      if (u,w,v) is eligible and $(u,w),(v,w),(u,v)$ are not marked **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{(u,w,v)\}$
      end if
    end for
    Update $cur(w,u) \leftarrow (l(w,v'), v')$ where v' is the last inspected neighbor for $(w,u) \in E^+$.
  ▷ *Part 3: obtaining an MIS from conflict graph.*
    Find an MIS $S \subseteq \mathcal{C}$ in the conflict graph $\mathcal{G}[\mathcal{C}]$. ▷ See Lemma 5.5.
    for $(u,w,v) \in S$ **do**
      Remove edges (u,w) and (v,w) from E^+ and **mark** all edges $(u,w),(v,w)$, and (u,v).
    end for
    $\mathcal{S} \leftarrow \mathcal{S} \cup S$.
  ▷ *Part 4: update the set of neighbors and active edges.*
    Recompute the list of sorted neighbors $N_w^{i+1} \leftarrow N(w)$. ▷ Some edges were removed!
    $A^{i+1} \leftarrow \emptyset$. ▷ Update A^{i+1} from A^i.
    for $(w,u) \in A^i$ and (w,u) is not marked **do**
      if $v := N_w^{\text{static}}[\text{idx}_{\text{static}}(w,u) + 2r]$ exists and $l(w,u) + l(w,v) + 1 < (1+\epsilon)\alpha$ **then**
        $A^{i+1} \leftarrow A^{i+1} \cup \{(w,u)\}$.
      end if
    end for
    $i \leftarrow i + 1$. ▷ Increment the round number.
  end while
  return $\mathcal{S}$
end function

still decreasing, and the output set $\mathcal{S}$ is still increasing in size. In particular, we can show that the number of active arcs, compared with the number of active arcs at the last "reset" round, must be reduced by a constant fraction (Lemma 5.3), leading to a polylogarithmic span.

We summarize our algorithm in Algorithm 5. Now, we are ready for the analysis.

5.1.2 Correctness

LEMMA 5.2. (CORRECTNESS) *Let $\mathcal{S}$ be the output of Algorithm 5. Then, $\mathcal{S}$ is an MIS of the conflict graph $\mathcal{G}$.*

Proof. It suffices to show that all eligible triangles are explored, which is indeed guaranteed by Line 6, since at the end of the algorithm $A^i = \emptyset$, which implies that all arcs become inactive. ∎

5.1.3 Work and Span

Span Analysis. In Algorithm 5, each outermost loop can be identified as a "double" round or a "reset" round. It is clear that between any two "reset" rounds there can be at most $\log \Delta = O(\log n)$ "double" rounds. Hence, it suffices to show that the total number of "reset" rounds is at most $O(\log m)$. Let the potential Φ to be number of active arcs (i.e., $|A^i|$) throughout the execution. The following lemma states that between any two consecutive "reset" rounds, the potential Φ is dropped by at least a constant fraction.

LEMMA 5.3. *Consider a round that is a "reset" round. Let Φ' be the number of active arcs at the beginning of the previous "reset" round, and let Φ be the number of active arcs at the beginning of this round. Then, $\Phi' - \Phi \geq \frac{1}{16}\Phi$. That is, at least $1/17 \approx 5.88\%$ of the active neighbors were removed during the previous round.*

Proof. Let r be the exploration rate at the beginning of this "reset" round. If no positive edge was removed and no active arc exhausted their exploration since the last "reset", then, the number of alive triangles, by definition, will be exactly $\Phi'(2r-1)$. Since this is a "reset" round and $\Phi'(2r-1) > \frac{1}{4}\Phi r$, there must be some alive triangles being destroyed since the last "reset" round.

There are only two types of events that can cause an alive triangle to disappear:

- Type 1 Event: an active arc finishes its exploration and became inactive.
- Type 2 Event: a positive edge is removed and marked in some round.

We will later show that for each of these events, at most $8r-4$ alive triangles are removed. Let E_1 (resp. E_2) be the number of type 1 (resp. type 2) events that happened since the last "reset" round. Note that this round is a "reset" round, so we have $\sum_{(w,u)\in A^i} |alive(w,u)| < \frac{1}{4}\Phi r$ and

$$\begin{aligned} E_1 + E_2 &\geq \frac{\Phi'(2r-1) - \frac{1}{4}\Phi r}{8r-4} \\ &\geq \frac{\frac{7}{4}r - 1}{8r-4}\Phi && (\Phi' \geq \Phi) \\ &\geq \frac{3}{16}\Phi. && (r \geq 1) \end{aligned}$$

On the other hand, note that type 1 and type 2 events are also the only two reasons for the potential to decrease. Each type 1 event reduces the potential by 1 since one active arc is removed. According to Line 27 of the algorithm, the number of type 2 event is exactly twice as many as the size of the triangles in an MIS $S \subseteq \mathcal{C}$ across all the rounds since the last "reset". Since these triangles are edge-disjoint, and at the time a triangle is added to $\mathcal{C}$ one of the positive edges must be active. Hence, we conclude that

$$\Phi' - \Phi \geq \max\{E_1, \frac{1}{2}E_2\} \geq \frac{1}{16}\Phi. \qquad \text{(Equality holds whenever } E_1 = \frac{1}{2}E_2.\text{)}$$

We now aim to show that both type 1 and type 2 events contribute to the reduction of potential by at most $8r-4$ alive triangles. Let us first consider a type 1 event, in which an active arc finishes its exploration

and is removed from A^i. Note that $alive(w,u) \le 2r-1$ for any $(w,u) \in A^i$, and being inactive does not affect the counting of other active arc's alive triangles. Thus, the number of alive triangles is reduced by at most $2r-1$.

Next, let us consider a type 2 event, where the algorithm removes $(x,y) \in E^+$ and adds it to S. We claim that such an edge destroys at most $8r-4$ alive triangles. First, suppose that both (x,y) and (y,x) are active. In this case, consider the sets $alive(x,y)$ and $alive(y,x)$, each of which contains at most $2r-1$ triangles. Thus, removing (x,y) destroys at most $4r-2$ alive triangles from $alive(x,y)$ and $alive(y,x)$.

Furthermore, the removal of (x,y) affects the alive triangles that use the (x,y) as the longer positive edge, that is there might be some v' such that $(x,y) \in alive(x,v')$ or $(y,x) \in alive(y,v')$. However, there can be at most $2^0+2^1+\cdots+r = 2r-1$ of active arcs (x,v') such that $(x,y) \in alive(x,v')$. Specifically, these active arcs must satisfy $v' \in N_x^{\text{static}}[\text{idx}_{\text{static}}(x,y)-(2r-1), \ldots, \text{idx}_{\text{static}}(x,y)-1]$. By symmetry, there can be at most $2r-1$ active arcs (y,u') that ever consider (u',y,x) as an alive triangle. Hence, there will be at most $8r-4$ alive triangles being destroyed for each type 2 event. ∎

Work Analysis. Now we focus on the total work for Algorithm 5. For Part 1, performing a binary search for each active arc incurs $O(m \log n)$ work per round. Computing the number of alive triangles incurs $O(m)$ work per round. Updating the list of sorted neighbors incurs $O(m \log m)$ work per round. For Part 4, computing N_w^i takes $O(m \log m)$ work per round and updating A^{i+1} takes $O(m)$ work per round. For Part 2 and Part 3, we first bound the size of collected eligible triangles in each round.

LEMMA 5.4. *Within each round, there are at most $O(m_f^{1.5})$ eligible $(+,+,-)$ triangles in $\mathcal{C}$. Moreover, the total work that computes $\mathcal{C}$ in a round is also $O(m_f^{1.5})$.*

Proof. We first observe that in any round, $\sum_{w,u} |alive(w,u)| = O(m_f^{1.5})$ according to Corollary 5.1. If in a round the parameter r is reset, it is clear that $|\mathcal{C}| = O(m)$. Otherwise, according to Line 11, we must have $\frac{1}{4}|A^i| \cdot r \le \sum_{w,u} |alive(w,u)| = O(m_f^{1.5})$. Since each active arc explores at most $2r$ new triangles, at most $|A^i| \cdot (2r) = O(m_f^{1.5})$ triangles are added to $\mathcal{C}$ and the work for Part 2 is also $O(m_f^{1.5})$. ∎

LEMMA 5.5. *Given a non-empty collection $\mathcal{C}$ of triangles, there exists a parallel algorithm that returns an MIS on $\mathcal{G}[\mathcal{C}]$ in $O(|\mathcal{C}| \log |\mathcal{C}|)$ work and $O(\log |\mathcal{C}|)$ span with high probability.*

Proof. We use the parallel greedy MIS algorithm, which works as follows. the algorithm selects an ordering of the vertices π, uniformly at random. In each round, all local minima are added to the independent set and removed from the graph, along with their neighbors. Here, local minima refer to all vertices that appear before their neighbors in the ordering. [29] showed that this algorithm terminates in $O(\log n)$ rounds with high probability where n refers to the number of vertices in the graph.

In our case, we cannot explicitly construct $\mathcal{G}[\mathcal{C}]$ since it contains $O(m_f^{1.5})$ vertices and up to $O(m_f^3)$ edges. However, we can still compute the local minima of $\mathcal{C}$ without constructing $\mathcal{G}[\mathcal{C}]$. Our key observation is that any two triangles of $\mathcal{C}$ are neighbors in $\mathcal{G}[\mathcal{C}]$ if and only if they have a common edge in G. Thus, we compute the local minima of edges of triangles instead of accessing edges in $\mathcal{G}[\mathcal{C}]$. Specifically, let $local(e) = \min_{e \in (u,w,v),(u,w,v) \in \mathcal{C}} \pi(u,w,v)$. Then, a triangle (u,w,v) is a local minimum if and only if $\pi(u,w,v) = local(u,w) = local(w,v) = local(u,v)$. Once we obtain the local minima of triangles in $\mathcal{C}$, we can remove those nodes and their neighbors by removing all triangles whose edges intersect with those local minima of triangles. Note that each triangle in $\mathcal{C}$ contains 3 edges, so the total amount of work is $O(|\mathcal{C}|)$ and the span is $O(\log |\mathcal{C}|)$. ∎

Proof. [Proof of Lemma 1.3] By Lemma 5.3, Lemma 5.4, and Lemma 5.5, we conclude that the algorithm runs in $O(m_f^{1.5} \log^3 n)$ work and $O(\log^3 n)$ span. ∎

5.2 A Randomized Reduction from Triangle Detection One may wonder whether there exists a faster sequential algorithm for the maximal edge-disjoint eligible open triangles problem. We show that, unfortunately, any combinatorial algorithm for this problem must take $\Omega(m_f^{1.5-\delta})$ time for some $\delta > 0$,

unless the Boolean matrix multiplication problem has a truly cubic combinatorial algorithm. The conjecture on the non-existence of such an algorithm has been commonly used to establish lower bounds in fine-grained complexity theory (see [1]). Unless such a conjecture is refuted, our algorithm is optimal up to a polylog n factor.

Instead of solving the maximal set of triangles, we consider a relaxed version of the problem, the *eligible open triangle detection problem*:

- Input: An undirected graph $G = (V, E^+ \cup E^-_{>1})$ with a length function $l : \binom{V}{2} \to [1, \infty]$ where $l(u, v) = 1$ for every edge $uv \in \binom{V}{2} \setminus (E^+ \cup E^-_{>1})$, a non-negative real number L. Let $n = |V|$ and $m_f = |E^+ \cup E^-_{>1}|$.
- Output: Output "Yes", if there is a triangle (u, w, v) such that $(u, w), (w, v) \in E^+, (u, v) \notin E^+$, and $l(u, v) + l(v, w) + l(v, u) < L$. Output "No", if no such triangle exists.

Why do we consider the relaxed problem? Algorithm 1 requires finding a maximal set of eligible open triangles. If there is an algorithm that finds the maximal set of open eligible triangles in $O(m_f^{1.5-\delta})$ time for some constant $\delta > 0$, it can clearly be used to solve the eligible open triangle detection problem also in $O(m_f^{1.5-\delta})$ time.

More generally, the eligible open triangle detection problem captures the key difficulty of our steepest descent algorithm. One may argue that it may not necessarily need to find a maximal set of eligible open triangles but *some* open triangles, or solve some even easier tasks. However, note that any algorithm that follows the framework of Algorithm 1 must send some flow through the (nearly) most beneficial triangle. Thus, it has to find at least one open triangle with some bounded length.

Triangle Detection via Eligible Open Triangle Detection Given a graph $\hat{G} = (\hat{V}, \hat{E})$, the (original) triangle detection problem is to determine whether there exists a $(u, v, w) \in \hat{G}$ such that $uv, vw, wu \in \hat{E}$. Williams and Williams [43] proved the following hardness result for the triangle detection problem:

THEOREM 5.1. ([43]) *The following all have truly subcubic (i.e. $O(n^{3-\delta})$ for some constant $\delta > 0$) combinatorial algorithms, or none of them do:*

- *Boolean matrix multiplication (BMM).*
- *Detecting if a graph has a triangle.*

Now we will show a reduction from the triangle detection problem to the eligible open triangle detection problem, with an overhead of $O(n^2)$, as stated by the following lemma:

LEMMA 5.6. *Suppose that there is an algorithm $\mathcal{A}$ that solves the eligible open triangle detection problem in $T(n)$ time, there is an algorithm that detects if a graph contains a triangle in $\tilde{O}(T(n) + n^2)$ time, with high probability.*

Proof. To detect if a graph $\hat{G} = (\hat{V}, \hat{E})$ has a triangle, we use Algorithm 6. The algorithm consists of $O(\log n)$ repetitions. In each repetition, it adds edges $e \in \hat{E}$ to E^+ with probability $1/2$ and sets the edge weight to 1. If there is a triangle (u, w, v) in G, then with probability at least $(3/8)$, there will be a triangle such that exactly two of the edges are in E^+ and the length of the triangle is 3, since a non-edge in G has length 1.

For each $uv \in \binom{\hat{V}}{2} \setminus \hat{E}$, we add it to $E^-_{>1}$ with length 5. So any triangle containing edges from $E^-_{>1}$ has length at least 5. If there is no triangle in $\hat{G}$, then any open triangle in G has a length of at least 5. Algorithm $\mathcal{A}$ sets L to 4 and so it will always output "No".

Since we repeat such a process for $O(\log n)$ times, if there is a triangle in $\hat{G}$, then with high probability, Algorithm 6 outputs "Yes". If $\hat{G}$ contains no triangle, then Algorithm 6 outputs "No". The running time is $\tilde{O}(n^2 + T(n))$ because, in each iteration, it takes $O(n^2)$ time to create the graph and $T(n)$ time to invoke the algorithm $\mathcal{A}$. ∎

Algorithm 6 Triangle detection via eligible open triangle detection.
Input: A graph $\hat{G} = (\hat{V}, \hat{E})$.
Output: "Yes" if there is a triangle $(u, w, v) \in G$. Otherwise, output "No" .
Oracle: $\mathcal{A}(G, L)$, which returns "Yes" if there exists an eligible open triangle of length less than L in G, and "No" otherwise.

function TRIANGLEDETECTION($\hat{G} = (\hat{V}, \hat{E})$)
  loop $O(\log n)$ times
    $V \leftarrow \hat{V}$, $E^+ \leftarrow \emptyset, E^-_{>1} \leftarrow \emptyset$.
    For $uv \in \binom{\hat{V}}{2} \setminus \hat{E}$, add uv to $E^-_{>1}$, set $l(u, v) = 5$.
    For $e \in \hat{E}$, with $1/2$ probability, add e to E^+ and set $l(e) = 1$.
    $r \leftarrow \mathcal{A}(G = (V, E^+ \cup E^-_{>1}, l), L = 4)$.
    If r is "Yes", **return** "Yes".
  end loop
  return "No".
end function

COROLLARY 5.2. *For any constant $0 < \delta < 1$, no combinatorial algorithm can solve the maximal edge-disjoint eligible open triangles problem in $O(m_f^{1.5-\delta})$, unless there is a combinatorial algorithm solving Boolean matrix multiplication in truly subcubic time.*

Proof. Suppose to the contrary there is an algorithm that solves the maximal edge-disjoint eligible open triangles problem in $O(m_f^{1.5-\delta})$ time. Then by our previous argument, it can be used to solve the eligible open triangle detection problem in $O(m_f^{1.5-\delta})$ time. Since $m_f \le n^2$, this algorithm runs in $O(n^{3-2\delta})$ time. By Lemma 5.6, the triangle detection problem can be solved in $\tilde{O}(T(n) + n^2) = \tilde{O}(n^{3-2\delta} + n^2)$ time. This in turns implies there is a truly subcubic time algorithm for Boolean matrix multiplication by Theorem 5.1. A contradiction occurs. ∎

Note that we only show that combinatorial algorithms are unlikely to beat $O(m_f^{1.5-\delta})$, which excludes, for example, algebraic algorithms. However, we argue that even if we allow all types of algorithms, it is still unlikely that we can obtain an algorithm that is closer to linear time in m_f. Note that, the current best algorithm for triangle detection takes $O(\min(n^\omega, m^{\frac{2\omega}{\omega+1}}))$ time [9], where $\omega \sim 2.37$ is the current best exponent of matrix multiplication. If we have any algorithm that solves the maximal edge-disjoint eligible open triangles problem in $O(m_f^{\omega/2-\delta}) = O(m_f^{1.185-\delta})$ time for any constant $\delta > 0$, by Lemma 5.6, it would imply an algorithm for the triangle detection that runs in $O(n^{\omega-2\delta})$ time. Such a result would be a major improvement for the triangle detection problem.

References

[1] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, 2014. Arxiv full version: *CoRR*, abs/1402.0054, 2014. 3, 27

[2] Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. *Algorithmica*, 83(7):1980–2017, 2021. 1, 2

[3] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. 1, 2, 3, 5, 7, 15, 20

[4] A. Arasu, C. Ré, and D. Suciu. Large-scale deduplication with constraints using dedupalog. In *2009 IEEE 25th International Conference on Data Engineering*, pages 952–963, 2009. 1

[5] Sepehr Assadi and Chen Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. In *13th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 215, pages 10:1–10:20, 2022. 1, 2

[6] Baruch Awerbuch, Rohit Khandekar, and Satish Rao. Distributed algorithms for multicommodity flow problems via approximate steepest descent framework. *ACM Trans. Algorithms*, 9(1):3:1–3:14, December 2012. 1, 4, 6, 8

[7] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004. 1

[8] S. Behnezhad, M. Charikar, W. Ma, and L. Tan. Almost 3-approximate correlation clustering in constant rounds. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 720–731, 2022. 1, 2, 3

[9] Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I 41*, pages 223–234. Springer, 2014. 7, 28

[10] Guy E Blelloch, Jeremy T Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 308–317, 2012. 1, 2, 20, 22

[11] Francesco Bonchi, David Garcia-Soriano, and Edo Liberty. Correlation clustering: From theory to practice. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1972–1972, 2014. 1

[12] Mark Bun, Marek Elias, and Janardhan Kulkarni. Differentially private correlation clustering. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1136–1146, 2021. 3

[13] Mélanie Cambus, Davin Choo, Havu Miikonen, and Jara Uitto. Massively parallel correlation clustering in bounded arboricity graphs. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing (DISC)*, volume 209 of *LIPIcs*, pages 15:1–15:18, 2021. 1, 2

[14] Mélanie Cambus, Fabian Kuhn, Etna Lindy, Shreyas Pai, and Jara Uitto. A single-pass semi-streaming algorithm for $(3+\varepsilon)$-approximate correlation clustering. In *Proc. 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2024. to appear. 1, 2

[15] Moses Charikar, Neha Gupta, and Roy Schwartz. Local guarantees in graph cuts and clustering. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 136–147, 2017. 3

[16] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360 – 383, 2005. 1, 3, 4, 7

[17] Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 219–228, 2015. 1, 3, 5, 6, 15, 16, 17

[18] Flavio Chierichetti, Nilesh Dalvi, and Ravi Kumar. Correlation clustering in mapreduce. In *Proc. 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 641–650, 2014. 1, 2

[19] Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 938–942, 2019. 3

[20] Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *J. ACM*, 68(1), jan 2021. 15

[21] Vincent Cohen-Addad, Chenglin Fan, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub M Tarnawski. Near-optimal correlation clustering with privacy. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 33702–33715, 2022. 3

[22] Vincent Cohen-Addad, Silvio Lattanzi, Andreas Maggiori, and Nikos Parotsidis. Online and consistent correlation clustering. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 4157–4179, 2022. 3

[23] Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078. PMLR, 2021. 1, 2

[24] Vincent Cohen-Addad, Euiwoong Lee, Shi Li, and Alantha Newman. Handling correlated rounding error via preclustering: A 1.73-approximation for correlation clustering. In *64rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2023. to appear. 1

[25] Vincent Cohen-Addad, Euiwoong Lee, and Alantha Newman. Correlation clustering with sherali-adams. In *63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 651–661. IEEE, 2022. 1, 5

[26] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2):172 – 187, 2006. Approximation and Online Algorithms. 3

[27] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007. 1

[28] Micha Elsner and Warren Schudy. Bounding and comparing methods for correlation clustering beyond ILP. In *Proceedings of the Workshop on Integer Linear Programming for Natural Language Processing*, pages 19–27, Boulder, Colorado, 2009. Association for Computational Linguistics. 1

[29] Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy MIS. *ACM Trans. Algorithms*, 16(1):6:1–6:13, 2020. 1, 2, 7, 20, 22, 26, 31

[30] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007. 8

[31] Jafar Jafarov, Sanchit Kalhan, Konstantin Makarychev, and Yury Makarychev. Local correlation clustering with asymmetric classification errors. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 4677–4686, 2021. 3

[32] Sanchit Kalhan, Konstantin Makarychev, and Timothy Zhou. Correlation clustering with local objectives. In *Advances in Neural Information Processing Systems (NuerIPS)*, volume 32, 2019. 3

[33] Silvio Lattanzi, Benjamin Moseley, Sergei Vassilvitskii, Yuyan Wang, and Rudy Zhou. Robust online correlation clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 4688–4698, 2021. 3

[34] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}$(sqrt(rank)) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 424–433, 2014. 3

[35] Daogao Liu. Better private algorithms for correlation clustering. In *Proceedings of Thirty Fifth Conference on Learning Theory*, volume 178 of *Proceedings of Machine Learning Research*, pages 5391–5412, 2022. 3

[36] Claire Mathieu, Ocan Sankur, and Warren Schudy. Online correlation clustering. In *27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 5 of *LIPIcs*, pages 573–584, 2010. 3

[37] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994. 3

[38] Béla Neuendorf. *On Strong Triadic Closure with Edge Insertion*. Bachelor thesis, Philipps-Universität Marburg, May 2020. Available at https://www.uni-marburg.de/de/fb12/arbeitsgruppen/algorith/paper/bachelor-bela-neuendorf-on-strong-triadic-closure.pdf. 5

[39] Gregory J. Puleo and Olgica Milenkovic. Correlation clustering and biclustering with locally bounded errors. *IEEE Trans. Inf. Theory*, 64(6):4105–4119, 2018. 3

[40] Stavros Sintos and Panayiotis Tsaparas. Using strong triadic closure to characterize ties in social networks. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1466–1475, 2014. 5

[41] Chaitanya Swamy. Correlation clustering: Maximizing agreements via semidefinite programming. In *Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 526–527, 2004. 3

[42] Nate Veldt. Correlation clustering via strong triadic closure labeling: Fast approximation algorithms and practical lower bounds. In *International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 22060–22083. PMLR, 2022. 5

[43] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 645–654. IEEE, 2010. 3, 7, 27

A Implementation in MPC Model with $\tilde{O}(\epsilon^{-3}m^{1.5})$ Total Space

We note that our algorithms (Algorithm 1, Algorithm 4, and Algorithm 5) work well also in the Massively Parallel Computation (MPC) model, with a small overhead in rounds. Let $\delta \in (0,1)$ be a constant. In the MPC model, each machine has $O(n^{\delta})$ local storage, and we are allowing $\tilde{O}(\epsilon^{-3}m^{1.5}/n^{\delta})$ machines to run at the same time. The rest of the section devotes to show that our algorithms runs in the MPC model with only a multiplicative $O(1/\delta)$ factor of overhead in rounds.

Algorithm 1 in MPC. There are only three places (line 5, 7, and 17) that need attention. In Line 5, computing the sum of all edge lengths requires knowing the congestion of each edge (and the number of non-zero congestion negative edges). Since the algorithm explicitly stores the $l_{uv}^{(t)}$ values for all non-zero

congested edges (by Lemma 3.10 there are at most $O(\epsilon^{-2} m \log m)$ of them), Line 5 can be simulated in

$$O(\log_{n^\delta} \epsilon^{-2} m \log m) = O(1/\delta)$$

rounds. Notice that we assume $\epsilon \geq 2/m$ so the overhead does not have additional $\text{poly}(1/\epsilon)$ factors.

In Line 7, see the analysis to Algorithm 5 below. Notice that we do not explicitly compute $\mathcal{P}_2'$.

In Line 17, by Lemma 3.9 we know that there are at most $O(\epsilon^{-4} \log^2 m)$ iterations, so all $\Phi(t)$ and $\alpha^{(t)}$ values can be stored in one machine. Thus only $O(1)$ rounds are needed to implement Line 17.

In case where ϵ is super small, book keeping all the histories of $\{y^{(t)}_{(u,w,v)}\}$ and $\{l^{(t)}_e\}$ for all t may be expensive. To save the total amount of space, we do not need to store all the history — it suffices to keep the values $\{z^{(T)}_e\}$ and update them whenever T gets updated.

Algorithm 4 in MPC. In Line 6, drawing a permutation can be simulated by choosing uniformly at random real number in $[0, 1]$ for each vertex, this takes $O(1)$ rounds in MPC. Furthermore, it takes $O(1/\delta)$ rounds to implement Line 9 and Line 12 since for each vertex the algorithm has to gather the smallest values among its neighbors. All other lines have a constant overhead when implemented in MPC.

Algorithm 5 in MPC. Sorting the neighbors (Line 3 and 30) takes $O(1/\delta)$ rounds in MPC. Simulating a binary search (Line 8) takes $O(1/\delta)$ rounds. Counting alive triangles (Line 11) takes $O(1/\delta)$ rounds. Inspect and collect the set $\mathcal{C}$ of eligible triangles (Line 19) takes $O(1)$ rounds. Finally, computing an MIS from the conflict graph $\mathcal{G}[\mathcal{C}]$ in Line 25 takes $O((1/\delta) \log n)$ rounds by simulating [29], which has an additional $O(1/\delta)$ factor comparing to the PRAM implementation.