Static to Dynamic Correlation Clustering

Nairen Cao
New York University, NY, USA

Vincent Cohen-Addad
Google Research, New York, NY, USA

Euiwoong Lee
University of Michigan, Ann Arbor, MI, USA

Shi Li
Nanjing University, China

David Rasmussen Lolck
University of Copenhagen, Denmark

Alantha Newman
Université Grenoble Alpes, France

Mikkel Thorup
University of Copenhagen, Denmark

Lukas Vogl
EPFL, Lausanne, Switzerland

Shuyi Yan
University of Copenhagen, Denmark

Hanwen Zhang
University of Copenhagen, Denmark

Abstract

Correlation clustering is a well-studied problem, first proposed by Bansal, Blum, and Chawla [Mach. Learn. '04]. The input is an unweighted, undirected graph. The problem is to cluster the vertices so as to minimize the number of edges between vertices in different clusters and missing edges between vertices inside the same cluster. This problem has a wide application in data mining and machine learning. We introduce a general framework that transforms existing static correlation clustering algorithms into fully-dynamic ones that work against an adaptive adversary.

We show how to apply our framework to known efficient correlation clustering algorithms, starting from the classic 3-approximate Pivot algorithm from Ailon, Charikar and Newman [JACM'08]. Applied to the most recent sublinear 1.485-approximation algorithm from Cao, Cohen-Addad, Lee, Li, Lolck, Newman, Thorup, Vogl, Yan and Zhang [STOC'25] [1], we get an 1.485-approximation fully-dynamic algorithm that works with worst-case constant update time. The original static algorithm gets its approximation factor with constant probability, and we get the same against an adaptive adversary in the sense that for any given update step, not known to our algorithm, our solution is an 1.485-approximation with constant probability when we reach this update.

Most of previous dynamic algorithms, including the celebrated result from Behnezhad, Charikar, Ma and Tan [FOCS'19], had approximation factors around 3 in expectation, and they could only handle an oblivious adversary. A recent algorithm by Braverman, Dharangutte, Pai, Shah, and Wang [AISTATS'25] handles an adaptive adversary, but it has a large unspecified constant approximation ratio. This contrasts with our general transformation, which works with all the best approximation factors known for the static case.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Facility location and clustering

Keywords and phrases Dynamic Algorithms, Correlation Clustering, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.48

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2504.12060` [14]

Funding *Euiwoong Lee*: Supported in part by NSF grant CCF-2236669 and Google.
Shi Li: Affiliated with the School of Computer Science in Nanjing University, and supported by the State Key Laboratory for Novel Software Technology, and the New Cornerstone Science Laboratory.

[1] The conference paper from Cao, Cohen-Addad, Lee, Li, Lolck, Newman, Thorup, Vogl, Yan and Zhang [STOC'25] claimed an approximation factor of 1.437, based on result from Cao, Cohen-Addad, Lee, Li, Newman and Vogl [STOC'24]. However, the STOC'24 paper has a subtle bug, which was fixed in the arXiv version with the correct ratio of 1.485.

David Rasmussen Lolck: Supported by VILLUM Foundation Grant 54451, Basic Algorithms Research Copenhagen (BARC).
Mikkel Thorup: Supported by VILLUM Foundation Grant 54451, Basic Algorithms Research Copenhagen (BARC).
Lukas Vogl: Supported by the Swiss National Science Foundation project 200021-184656 "Randomness in Problem Instances and Randomized Algorithms".
Shuyi Yan: Supported by VILLUM Foundation Grant 54451, Basic Algorithms Research Copenhagen (BARC).
Hanwen Zhang: Supported by VILLUM Foundation Grant 54451, Basic Algorithms Research Copenhagen (BARC), Independent Research Fund Denmark, grant 1054-00032B, and the Carlsberg Foundation, grant CF24-1929.

1 Introduction

Correlation clustering is a classic clustering problem. Given an undirected unweighted graph $G = (V, E)$, our goal is to compute a clustering of the vertices that minimizes the number of edges between vertices in different clusters and missing edges between vertices inside the same cluster.

In this work, we address correlation clustering in the dynamic setting, where the graph G is updated by inserting and deleting edges.[1] Our goal is to maintain a clustering that is good with respect to the correlation clustering problem for the current graph. We present a framework that transforms efficient algorithms for the static correlation clustering problem into a dynamic algorithm that can handle edge updates against an adaptive adversary, at almost no cost to the approximation ratio. However, the efficiency has to be relative to a compressed graph representation that we shall introduce later. We will apply this transformation to all the best static near-linear time algorithms.

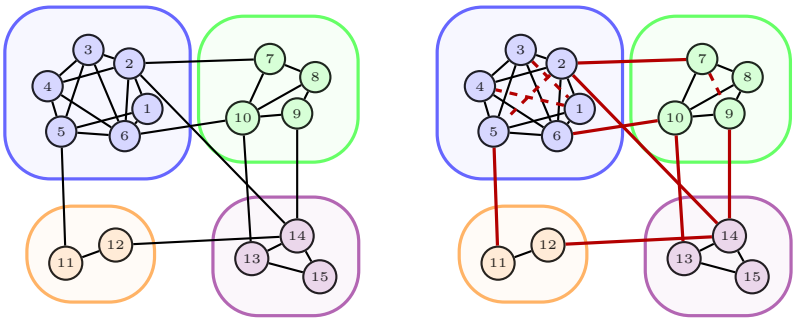


Figure 1 A clustering of a graph and its cost. Red solid lines marks all edges between vertices in different clusters. Red dashed lines mark all the missing edges between vertices in the same cluster. This formulation is equivalent to the signed graph version, where + edges are treated as edges and − edges are treated as non-edges.

[1] In other works, it is usual to describe the input as being a complete graph with edges of weight either $+1$ or -1. Since we are working in a dynamic setting, and care about the running time, it makes more sense to use the convention that $+1$ edges are edges in the input graph and -1 edges are the non-edges.

1.1 Prior work

The correlation clustering problem was first studied by Bansal, Blum, and Chawla in [6]. The model has a number of applications such as clustering ensembles [10], duplicate detection [3], community mining [20], disambiguation tasks [27], and automated labeling [1, 16]. Correlation clustering is of fundamental importance for the machine learning and data mining communities and a large body of work for solving correlation clustering in practice keeps on appearing at flagship conferences in these areas [25, 12, 17, 28, 29, 31].

Polynomial time approximation

The first (large) constant approximation algorithm is due to [6]. The constant was then improved to 4 by [18] who also proved that the problem is APX-hard. Later, Ailon, Charikar, and Newman [2] introduced the crucial idea of pivot-based algorithms, where in each round, the algorithm picks a random unclustered vertex and creates a new cluster consisting of this vertex and some of its neighbors. They gave a combinatorial 3-approximation algorithm and improved the approximation ratio to 2.5 by rounding a standard linear program (LP) which has an integrality gap of at least 2. [19] improved the approximation ratio to 2.06 using a more sophisticated rounding scheme of the same LP.

To bypass the integrality gap of 2, Cohen-Addad, Lee, and Newman [23] used the Sherali-Adams hierarchy and achieved a 1.995-approximation, which was then improved to a 1.73-approximation by combining the pivot-based rounding with the newly-developed set-based rounding [22]. The best approximation algorithm currently known is a 1.485[1]-approximation due to [15] which proposed the Cluster LP to achieve this result. They also showed that obtaining a 25/24-approximation is NP-hard.

Deterministic algorithms

When it comes to deterministic algorithms, the fastest one with (large) constant factor approximation is Algorithm 1 in [21] which has a trivial deterministic implementation is $O(nm)$ time. A deterministic factor $(3+\varepsilon)$- approximation is presented in [26] running in $\tilde{O}(n^3)$ time. One can obtain a deterministic 2.5-approximation in (large) polynomial time by first solving the $O(n^3)$-sized LP with the Ellipsoid method and rounding it with the work of [30]. [2]

Near-linear time

For the purpose of transforming static algorithms into dynamic ones, our best hope are those with near-linear running time, that is $\tilde{O}(m)$-time, where m is the number of edges. We remark that while sublinear time algorithms exist, their sublinear time holds only when $m = \omega(n)$. As we have to deal with the case $m = O(n)$, these algorithms are not different from those with a near-linear running time for our purpose. The best deterministic running time of $O(nm)$ from [21] is thus far too slow.

Among all near-linear time algorithms, the first one is the classical linear time 3-approximate Pivot algorithm of [2]. The first sublinear time algorithm in $O(n \log^2 n)$ time with constant approximation ratio is from [5], which is based on the parallel algorithm of [21].

[2] In the conference version of [12, Proposition 2.1], the authors claim that the existence proof of [5] gives a deterministic $O(m)$ time constant-approximate algorithm. The authors have since retracted their claim in the most recent arXiv version [11].

We also have the more recent local-search based 1.847-approximation of [24] which runs in sublinear time. Recently, in [13], it was shown that the 1.485-approximation algorithm based on the Cluster LP from [15] can be implemented in sublinear time. All of these near-linear time algorithms are Monte Carlo algorithms.

Dynamic clustering with low approximation ratio

The dynamic setting has also been explored. Much of the work here has been on how to maintain a pivot-based clustering under these changes. One result is that of [8], which shows how to maintain a Maximal Independent Set with an update time of $O(\log^2 n \log^2 \Delta)$ where Δ is the maximum degree of any vertex. This can be used to maintain the 3-approximate Pivot algorithm for a fixed permutation of pivots. This was since improved in [25] to maintaining a $(3+\varepsilon)$-approximation with an update time of $O(1/\varepsilon)$. Finally, it has recently been shown that the barrier of 3-approximation indeed can be broken in the dynamic setting in [7]. Here they give a 2.997-approximation algorithm, again based on maintaining the pivot under additions and deletions with a polylogarithmic update time. *All of these pivot-based results only work with an oblivious adversary*: they fix the random order of the pivots in advance, maintaining it for all updates. All of these results only work with an oblivious adversary: they fix the random order of the pivots in advance, maintaining it for all updates.

Dynamic against adaptive adversary

For all the above pivot-based dynamic algorithms, an *adaptive adversary* could easily learn the order of the pivots, and then make them perform very badly afterward by constructing an input graph that is bad for this pivot order. Recently, [12] presented a dynamic large constant factor approximation algorithm supporting updates in $O(\log^2 n)$ amortized time against an adaptive adversary. It is based on the non-pivot-based $O(n \log^2 n)$ time randomized algorithm from [5], but at the cost of a large approximation factor. They also mention "*It is unclear how the pivot-based algorithms could be made to work in the adversarially robust setting*"[12, 2nd page]. In particular, this concerns the classic 3-approximate Pivot algorithms.

The general issue of getting a Monte Carlo randomized dynamic algorithm to work against an adaptive adversary is addressed in [9]. They show a generic transformation from the oblivious setting to the adaptive setting using tools from differential privacy, but the transformation has an extra polynomial factor on the update time even in the few cases studied in their paper. In this paper, however, we are aiming for update times that are polylogarithmic or even constant.

Clustering for dynamic streams

A recent result from [4] shows that for a fully-dynamic graph, we can maintain a linear sketch of sublinear $\tilde{O}(n)$ size from which we can derive a correlation clustering in randomized polynomial time. The basic idea is that they desparsify the sketch into a graph G' that is similar to the current graph G in the sense that any correlation clustering will have almost the same cost in G and G'. They can therefore apply any polynomial time correlation clustering algorithm to G' and get an almost as good correlation clustering for G. This result follows the streaming tradition and is very interesting from an information theoretic perspective. However, spending polynomial time whenever we want a clustering of the current graph is not good from our perspective of dynamic graph algorithms, where we want to maintain a concrete clustering in constant or logarithmic time per edge update.

The interesting aspect from [4] is that their sketch is of $\tilde{O}(n)$ size. A fundamental issue from a fully-dynamic perspective is that a randomized sketch of sublinear size appears to sacrifice the ability to handle a non-oblivious adaptive adversary. The issue is that if we use a randomized sketch of the current graph, and if we after each update reveal a clustering based on the random choices in the sketch, then the adversary can learn about these random choices and then make updates that destroy the quality of the sketch. This is not so much an issue for the streaming scenario in [4] where they just maintain the sketch, and only in the end construct the clustering in polynomial time. However, it is a major issue if we want to maintain a clustering throughout the updates against an adaptive adversary, and this is the setting we are considering in this paper.

1.2 Our results and techniques

This paper is a conference version of the full version [14], which contains the complete details. The full version continues from Section 4.

In this paper, we introduce a general framework that transforms efficient static algorithms for the static correlation clustering problem into dynamic algorithms that work against an adaptive adversary. However, the efficiency has to relate to a compressed graph representation that we shall introduce later. In particular, this will work for the pivot-based algorithms that, as mentioned above, have so far failed against adaptive adversaries.

We apply our framework to all near-linear time correlation clustering algorithms with low approximation ratio mentioned before: the classical 3-approximate Pivot algorithm from [2], the 1.847-approximation local search algorithm from [24] and the most recent 1.485-approximation Cluster LP algorithm from [13]. The latter gives us our main fully-dynamic result:

▶ **Theorem 1.** *For any $\delta \in (0,1)$ we can maintain clustering for a fully-dynamic graph in $O(\log 1/\delta)$ worst-case time per edge update. Against an adaptive adversary, for each i the maintained clustering is a 1.485-approximation with probability at least $1-\delta$ at update i.*

Another way to explain Theorem 1 is that the adversary can choose any i hidden to the algorithm before the updates start and adaptively choose the first i updates.

A particular situation of Theorem 1 is when $\delta = \frac{1}{\text{poly}(n)}$. Then we get polynomially small error probability with logarithmic update time. This matches the setting of [12].

Some modifications we make to the static algorithms are of independent interest. Currently, the best implementations of the algorithms from [24, 13] are sublinear in $\tilde{O}(n)$ time, but this includes logarithmic factor in the running time even for constant error probability. As a result, the algorithm does not run in $O(m)$ time if $m = o(n \log n)$.

We show that these algorithms can be implemented in strictly linear time, that is, $O(m)$ time. We further show that the static algorithm from [24] can be implemented in $O(m)$ time, yielding a 1.847-approximation with probability at least $1 - \exp(-\sqrt{m}/\log^3 m)$. For such an exponentially low error probability, we would normally have used $\Theta(\sqrt{m}/\log^3 m)$ repetitions, leading to a corresponding polynomial blow-up in time. The same exponentially low error probability can be achieved for the computation of the Cluster LP in [13], but the LP rounding brings the error probability up to an arbitrarily small constant if we want $O(m)$ time.

In the rest of this subsection we will introduce the framework and later sketch how the different static algorithms can be modified to work within it.

Graph representation and compression via correlation clustering

We will now be more precise about how we want to represent a graph $G = (V, E)$. Instead of storing the edges E, we will store a clustering $\mathcal{C}$ together with the set D of *violated pairs*, that is, unordered vertex pairs (u, v) such that either $(u, v) \in E$ but u and v are in different clusters in $\mathcal{C}$, or $(u, v) \notin E$ but u and v are in the same cluster in $\mathcal{C}$. The correlation clustering cost of $\mathcal{C}$ is exactly $|D|$. We shall refer to set D as the *violation* of the clustering $\mathcal{C}$, and to $(\mathcal{C}, D)$ as the *cluster representation*, noting that it uniquely defines the edge set E. We say that the cluster representation $(\mathcal{C}, D)$ is c-approximate if the clustering is c-approximate, that is, if $|D|$ is at most c times the minimum correlation clustering cost. In addition, it is simple to reconstruct E from $(\mathcal{C}, D)$.

We can think of the cluster representation as a graph compression: It is never worse than the actual graph representation by more than the $O(n)$ space required to store $\mathcal{C}$, since we can always choose the clustering to be a singleton cluster for each vertex. Then D is just the set E of edges in the graph. In the other extreme, if we have all vertices forming a single cluster, then the violation D is exactly the set of non-edges $\binom{V}{2} \setminus E$. However, our representation could be much better in both of these extremes if the clustering $\mathcal{C}$ has small cost. We can in fact bound the space required by $O(n + |D|)$, from which we get these properties. It is worth mentioning that one can ensure $O(n)$ additional space compared to the normal graph representation, with two global counters on the size of D and E. However, this is not necessary to achieve our main results.

Fully-dynamic framework

To get a fully dynamic correlation clustering algorithm, assume that we have a static algorithm that takes as input an arbitrary cluster representation $(\mathcal{C}, D)$ and outputs a c-approximate cluster representation $(\mathcal{C}', D')$ in $O(t|D|)$ time for some parameter t. Since D may be much smaller than the edge set E, we can think of using $(\mathcal{C}, D)$ as a warm start. We also note that $|D|$ might be much smaller than the clustering $\mathcal{C}$, which is of size $\Theta(n)$. Therefore, the static algorithm can only produce a clustering $\mathcal{C}'$ with limited modification of the input clustering $\mathcal{C}$.

Now, in correlation clustering, each edge update will only change the cost of a given clustering by 1. Thus, if at some point, we have a good clustering $\mathcal{C}$ with violation D, then the same clustering will remain good for $O(\varepsilon|D|)$ edge updates. More precisely, we will show that if $\mathcal{C}$ was a c-approximation, then $\mathcal{C}$ will remain a $(1 + \varepsilon)c$-approximation after $\mu|D|$ edge updates, where $\mu \leq \frac{\varepsilon}{2(1+\varepsilon)c}$. Therefore, it suffices to reconstruct a new c-approximate clustering after $\mu|D|$ edge updates.

In the period between reconstructions, we will just store the sequence U of $\mu|D|$ edge updates (so the violation is not updated explicitly as that would involve hashing). For a fixed clustering $\mathcal{C}$, we will show how to update our violation D according to the edge updates in U deterministically in $O(|D| + |U|) = O(|D|)$ time. Next, we apply the assumed static algorithm to our updated cluster representation $(\mathcal{C}, D)$ to reconstruct a new c-approximate cluster representation $(\mathcal{C}', D')$ in $O(t|D|)$ time. The new clustering $\mathcal{C}'$ will again remain good for the next $\mu|D'|$ edge updates.

The result is a $(1 + \varepsilon)c$-approximate fully-dynamic algorithm with $O(t/\mu)$ amortized update time. With standard background rebuilding, we can de-amortize and get $O(t/\mu)$ worst-case update time. Summing up, we will show:

▶ **Theorem 2.** *Suppose we have a static correlation clustering algorithm that given any cluster representation $(\mathcal{C}, D)$, produces a c-approximate cluster representation $(\mathcal{C}', D')$ in $O(t|D|)$ time. Then we have a fully-dynamic algorithm that maintains a $(1+\varepsilon)c$-approximate correlation clustering in worst-case $O(t/\mu)$ update time per edge, where $\mu \leq \frac{\varepsilon}{2(1+\varepsilon)c}$.*

Tricky randomization

The above approach works perfectly if our algorithm for the static case is deterministic, but all the known near-linear time static constant approximate algorithms are Monte Carlo algorithms. They only produce a c-approximations in expectation or with some error probability bounded by $p < 1$.

Naively, the above should be fine, even against an adaptive adversary, because our current clustering is good if the last rebuild was a c-approximation as in the deterministic case, and this happened with probability at least $1 - p$. The next rebuild is done using its own independent randomness, so an adaptive adversary cannot do anything to break a guarantee that holds for any input cluster representation, e.g., that it is a c-approximation with probability at least $1 - p$. Nevertheless, there is an issue since the new cluster representation $(\mathcal{C}', D')$ will be kept for $\mu|D'|$ updates. Hence, if it is bad in the sense that D' is large, then it will survive for longer time. We will present a concrete example about this effect, showing that we get a logarithmic factor more chances of failing at a particular update. In fact, this does not even depend on the adversary being adaptive. The same construction can be done by an oblivious adversary.

Error probability against adaptive adversaries

It is a bit subtle what we mean by saying that the dynamic algorithm is correct with some probability against an adaptive adversary, since this adversary knows our current clustering and can see exactly how bad it is compared with the optimum. It is therefore never a probabilistic statement whether the current clustering is good to the adversary. However, we can study statements of this kind: for any given update step i, what is the probability P_i that we have a bad clustering just after update i. An adaptive adversary (1) can choose i in advance, (2) knows our current clustering at any time, and (3) can adaptively pick the first i updates so as to maximize P_i. Yet we want to bound P_i.

The above definition of P_i may seem a bit cryptic, but this is important when we want constant error probability with constant update time. More specifically, with a long update sequence, we can create and remove a linear number of different constant sized graphs. Our dynamic algorithm will cluster all of them independently, so if it fails with constant probability, then we expect it to fail on a constant fraction of them, and the adaptive adversary will see when it fails. Nevertheless we promise that it is correct just after a predetermined update i with constant probability P_i. If we increase the update time to $O(\log n)$, then we can make the error probability so small that we do not expect any errors over a polynomially long update sequence, and then we do not need the special definition of P_i.

Fully-dynamic framework for Monte Carlo algorithms

What we first show is that if the static algorithm gives a c-approximation with probability at least $1 - p$, then our fully dynamic algorithm gives an $(1 + \varepsilon)c$-approximation with error probability $P_i = O(p \log n)$.

We can avoid losing the factor $\log n$ if the static algorithm for input clustering $(\mathcal{C}, D)$, $|D| \geq 1$, produces a c-approximation with probability at least $1 - q(|D|)$ where q falls at least inversely in $|D|$. Then P_i is bounded by $O(q(1))$.

Technically, the assumption of $|D| \geq 1$ means that we avoid dividing by 0. However, if our input cluster representation had $|D| = 0$, then it would be a zero-cost optimal solution, and then we would not involve the static algorithm to try computing a better solution.

Finally, we have a combined result. Suppose the static algorithm for input clustering $(\mathcal{C}, D)$, $|D| \geq 1$ produces a c-approximation with probability at least $1 - p$ and an $O(c)$ approximation with probability at least $1 - q(|D|)$ where again q falls at least inversely in $|D|$. Then P_i is bounded by $O(p + q(1))$.

We shall see how all these bounds play together with existing static algorithms, but first we summarize them in the theorem below. In this theorem, we consider randomized Monte-Carlo algorithms that *aim* at certain targets that they may *fail* to achieve.

▶ **Theorem 3.** *Suppose we have a static correlation clustering algorithm that, given any cluster representation $(\mathcal{C}, D)$ with $|D| \geq 1$, aims to produce a c-approximate cluster representation $(\mathcal{C}', D')$ in $O(t|D|)$ time. Then we have a fully-dynamic algorithm that aims to maintain a $(1+\varepsilon)c$-approximate correlation clustering in worst-case $O(t/\mu)$ time per update where $\mu = \min\left\{\frac{\varepsilon}{2(1+\varepsilon)c}, 1/6\right\}$.*

- **(a)** *Suppose that with probability at most p, the static algorithm* fails *to produce a c-approximate clustering. Then, against an adaptive adversary, for any fixed i unknown to the algorithm, the clustering maintained by the fully-dynamic algorithm fails to be $(1+\varepsilon)c$-approximate at update i with probability $P_i = O(p \log n)$.*
- **(b)** *Suppose that with probability at most $q(|D|)$, which falls at least inversely with $|D| \geq 1$, the static algorithm* fails *to produce a c-approximate clustering. Then, against an adaptive adversary, for any fixed i unknown to the algorithm the clustering maintained by the fully-dynamic algorithm fails to be $(1+\varepsilon)c$-approximate at update i with probability $P_i = O(q(1))$.*

By combining the two cases above, we can get a best of both worlds statement, namely that we can get the approximation ratio of Theorem 3(a) case while avoiding the logarithmic blow-up in the probability, by using the properties of Theorem 3(b).

▶ **Theorem 4.** *Let $\mathcal{A}$ and $\hat{\mathcal{A}}$ be two static correlation clustering algorithms that take as input any cluster representation $(\mathcal{C}, D)$ with $|D| \geq 1$ and output a cluster representation $(\mathcal{C}', D')$, both in $O(t|D|)$ time. Furthermore, let the probability of $\mathcal{A}$ failing to produce a c-approximate solution be bounded by $p = 1 - \Omega(1)$ and the probability of $\hat{\mathcal{A}}$ failing to produce a $\hat{c}$ approximate solution be bounded by $q(|D|)$, where q falls at least inversely with $|D|$.*

Then we have a fully-dynamic algorithm that aims to maintain a $(1+\varepsilon)c$-approximate correlation clustering in worst-case $O(t/\mu)$ time per update where $\mu = \min\left\{\frac{\varepsilon}{2(1+\varepsilon)c}, \frac{1}{6}, \frac{1}{2\hat{c}}\right\}$. Furthermore, against an adaptive adversary, for any fixed i unknown to the algorithm, the clustering maintained by the fully-dynamic algorithm fails to be $(1+\varepsilon)c$-approximate at update i with probability $P_i = O(p + q(1))$.

It should be mentioned that while this theorem does give results against an adaptive adversary, this does not mean that it can be strengthened against an oblivious adversary. We are in fact later going to show a matching lower bound for the probability of failure, which contains a strategy that can be implemented both by an adaptive and an oblivious adversary.

It is also worth noting that the probability P_i in the statement is not independent for different values of i. You only get independence between two updates i and j if between processing them the clustering is rebuild. The frequency of this is however dependent on the cost of the clustering, a value that is very easy to influence by the adversary.

One may wonder if one could get the error probabilities to work simultaneously for a sequence of T updates without resorting to a union bound, multiplying the error probability by T. As a base state, consider an arbitrarily large graph consisting of disjoint cliques, including

singleton vertices. The optimal correlation clustering cost is zero, so any approximation algorithm has to agree on the clustering into cliques. With s updates, the adversary can create a nontrivial instance of correlation clustering over some of the vertices. And with s more updates, the adversary can change the graph back to the base state. Therefore, starting from the base state, in T updates, the adversary can create $T/(2s)$ nontrivial and independent instances to challenge the algorithm, so getting global error bound better than a union bound is not likely.

Consequence if a deterministic constant approximate algorithm is found

Our result will be improved if there exists a deterministic $O(|D|)$ time algorithm $\hat{\mathcal{A}}$ with constant approximation ratio. According to Theorem 2, this would give a deterministic dynamic algorithm that maintains a clustering of $(1+\varepsilon)c$ approximation at every update. In addition, the existence would also imply that the dynamic algorithm would inherit any expected approximation ratio from algorithm $\mathcal{A}$, instead of only achieving this approximation ratio with constant probability, since we can get a deterministic upper bound on the cost anytime we rebuild the clustering. This would also give a worst-case guarantee on the quality of the clustering even when $\mathcal{A}$ always fails.

▶ **Theorem 5.** *Let $\mathcal{A}$ and $\hat{\mathcal{A}}$ be two static correlation clustering algorithms that take as input any cluster representation $(\mathcal{C}, D)$ with $|D| \geq 1$ and output a cluster representation $(\mathcal{C}', D')$, both in $O(t|D|)$ time. Furthermore, let the probability of $\mathcal{A}$ failing to produce a c-approximate solution be bounded by $p = 1 - \Omega(1)$ and $\hat{\mathcal{A}}$ be a deterministic $\hat{c}$-approximation algorithm.*

Then we have a fully-dynamic algorithm that aims to maintain a $(1+\varepsilon)c$-approximate correlation clustering in worst-case $O(t/\mu)$ time per update where $\mu = \min\left\{\frac{\varepsilon}{2(1+\varepsilon)c}, \frac{1}{6}, \frac{1}{2\hat{c}}\right\}$. Furthermore, against an adaptive adversary, for any fixed i unknown to the algorithm, the clustering maintained by the fully-dynamic algorithm is an $(1+\varepsilon)c$-approximation in expectation at update i.

Existing static algorithms modified for our dynamic framework

We will now discuss how the known near-linear static algorithms can be modified and used in our dynamic framework.

First, as a general standard note. Suppose we have a Monte-Carlo algorithm that, given a cluster representation $(\mathcal{C}, D)$, aims to produce a c-approximate cluster representation $(\mathcal{C}', D')$ and fails with probability at most p. We switch to the new clustering $\mathcal{C}'$ only if it has lower cost, that is, if $|D'| < |D|$. This means that if we make k iterations of the algorithm, then the probability that we do not end up with a c-approximation drops to p^k.

The first algorithm we consider is the classical 3-approximate Pivot algorithm from [2]. It takes $O(m)$ time on a given graph, and we show how to implement it in $O(|D|)$ time for a given input clustering $(\mathcal{C}, D)$. This ends up requiring weighted sampling of vertices instead of the normal uniform one. In addition, this also has the consequence of showing how we can speed up the running time of the Pivot algorithm with a "hot start". The approximation factor of the Pivot algorithm is in expectation, but making $O(\log\log n)$ iterations, we get a $(3+o(1))$-approximation with failure probability $o(1/\log n)$. Thus, we have:

▶ **Theorem 6.** *We have a static correlation clustering algorithm that given any cluster representation $(\mathcal{C}, D)$, in $O(|D|)$ time produces an expected 3-approximate cluster representation $(\mathcal{C}', D')$. Repeating $O(\log\log n)$ times, we get a $(3+o(1))$-approximate solution with probability at least $1 - o(1/\log n)$.*

Plugging Theorem 6 into Theorem 3(a) using a sufficiently small $\varepsilon > 0$, we get:

▶ **Corollary 7.** *For any $\varepsilon > 0$, we have a fully-dynamic algorithm that aims to maintain a $(3+\varepsilon)$-approximate correlation clustering in $O(\frac{1}{\varepsilon}\log\log n)$ worst-case time per edge update. Against an adaptive adversary, for any i the clustering fails to be $(3+\varepsilon)$-approximate at update i with probability $P_i = o(1)$. The failure probability can be reduced to any δ if we spend $O(\frac{1}{\varepsilon}(\log\log n + \log 1/\delta))$ worst-case time per edge update.*

The main advantage of Corollary 7 over the previous pivot-based dynamic algorithms with approximation factors around 3 is that it works against an adaptive adversary. Compared to [12], by setting $\delta = 1/\text{poly}(n)$, Corollary 7 improves both the approximation ratio from a large constant to $3+\epsilon$ and the update time from amortized $O(\log^2 n)$ to worst-case $O(\log n)$, with the same error probability.

Next, we consider the 1.847-approximate local search algorithm from [24]. As discussed earlier, it uses $\Omega(n \log n)$ time even if we just want constant error probability, but this is not $O(m)$ time if $m = o(n \log n)$. In this paper, we show an $O(m)$ time implementation of the local search algorithm with exponentially small error probability of $\exp(-\sqrt{m}/\log^3 m)$. The previous static algorithm tries to find clusters with roughly the same probability regardless of their sizes. However, we notice that each small cluster only has a small random contribution to the total cost, so their total contribution is strongly concentrated and we do not need to find all of them. On the other hand, for large clusters, we can afford to spend more time and make sure to find all of them with high probability. Our full implementation is based on a smooth sampling distribution according to the cluster sizes. Moreover, our implementation can also be made efficient in cluster representation, as stated below.

▶ **Theorem 8.** *We have a static correlation clustering algorithm that given any cluster representation $(\mathcal{C}, D)$, in $O(|D|)$ time produces a cluster representation $(\mathcal{C}', D')$ that is below 1.847-approximate with probability at least $1 - \exp(-\sqrt{|D|}/\log^3 |D|)$.*

The next corollary follows from this special case of Theorem 8 when $\mathcal{C}$ is the set of singletons and $D = E$.

▶ **Corollary 9.** *There exists a 1.847-approximate $O(m)$ time static correlation clustering algorithm in the normal graph representation with error probability at most $\exp(-\sqrt{m}/\log^3 m)$.*

Since the error probability in Theorem 8 falls more than inversely in $|D|$, we can apply Theorem 3(b). The approximation factor 1.847 in Theorem 8 is already rounded up by a small constant, so using a sufficiently small $\varepsilon > 0$ in Theorem 3, we get:

▶ **Corollary 10.** *We have a fully-dynamic algorithm that aims to maintain a 1.847-approximate correlation clustering in constant worst-case time per edge update. Against an adaptive adversary, for any fixed i unknown to the algorithm, the clustering is 1.847-approximate with constant probability at update i. The failure probability can be reduced to any δ if we spend $O(\log 1/\delta)$ worst-case time per edge update.*

Finally, we want to apply our dynamic framework to the current best 1.485-approximation algorithm from [15] which was implemented in sublinear time in [13]. We will need all the techniques mentioned above. The algorithm has two parts. First it solves the Cluster LP from [15], second it rounds the solution.

The first part aims for a fractional solution to the Cluster LP that is $(1+\varepsilon)$-approximate relative to the optimal integral solution. This part uses multiplicative weight updates and finds a nearly optimal fractional solution to the aggregated constraint with the above local

search. We face a situation parallel to the one we faced with the local search from [24]. The algorithm is sublinear but not linear if $m = o(n \log n)$ and it only works with constant probability. Using the techniques we used to make the local search from [24] work for Theorem 8, we can take any cluster representation $(\mathcal{C}, D)$ and solve the Cluster LP in $O(|D|)$ time, yielding a fractional solution that is $(1+\varepsilon)$-approximate relative to the optimal integral solution with probability at least $1 - \exp(-\sqrt{|D|}/\log^3 |D|)$.

In [15, 13], the fractional solution is rounded using pivot-techniques. The rounding gets a 1.485-approximate solution with constant probability. Given a cluster representation $(\mathcal{C}, D)$, we want to implement the rounding in $O(|D|)$. Here we can employ some of the techniques we used for the classical 3-approximate pivot in Theorem 6. However, all the rounding algorithms only work with constant failure probability.

▶ **Theorem 11.** *We have a static algorithm that given any cluster representation $(\mathcal{C}, D)$, in $O(|D|)$ time solves the Cluster LP. For any given constant $\varepsilon > 0$, the fractional solution is within a factor $(1 + \varepsilon)$ of the optimal integral solution with probability at least $1 - \exp(-\sqrt{|D|}/\log^3 |D|)$.*

The above fractional solution can be rounded to an integral cluster representation in $O(|D|)$ time. The solution is below 1.485-approximate with constant probability.

The next corollary follows from this special case of Theorem 11 when $\mathcal{C}$ is the set of singletons and $D = E$.

▶ **Corollary 12.** *There exists a 1.485-approximate $O(m)$ time static correlation clustering algorithm in the normal graph representation with constant error probability.*

Because of the constant error probability from the rounding, we do not currently benefit from the exponentially low error probability for finding the fractional solution. However, it will be important if we one day find rounding algorithms working with higher probability.

We will now plug both Theorem 8 and Theorem 11 into Theorem 4. More precisely, we use Theorem 8 to get an $O(1)$-approximate solution with exponentially high probability, and Theorem 11 to get a below 1.485-approximate cluster representation with constant probability. Applying Theorem 4 with a sufficiently small $\varepsilon > 0$, we get our main result in Theorem 1.

2 Dynamic Framework

In most part of the paper, we will work on a fixed graph $G = (V, E)$ represented by $(\mathcal{C}, D)$, a pair of clustering and the symmetric difference between the edges E and the edges of $\mathcal{C}$. This graph will be receiving edge updates in the form of inserting new edges into E or removing existing edges from E. For each vertex $v \in V$, let $d(v)$ be the degree of v, $N(v)$ be the set of neighbors of v together with v itself.

Given a clustering $\mathcal{C}$, let $C(v)$ be the cluster of v in $\mathcal{C}$. Let $\mathcal{E}(\mathcal{C})$ be the set of pairs (u, v) such that u and v are in the same cluster in $\mathcal{C}$. Let $\text{cost}(\mathcal{C})$ be the cost for using this clustering, by definition $\text{cost}(\mathcal{C}) = |E \triangle \mathcal{E}(\mathcal{C})|$, where $A \triangle B$ denotes the symmetric difference between the sets A and B. We will use these notations throughout the paper.

For the dynamic algorithm, we will follow the approach that for every certain number of updates we will recompute the full clustering. For the purpose of this, between the recomputations we are going to maintain the cluster representation of the current graph $G = (V, E)$:

- $\mathcal{C}$: A clustering of the graph G.
- D: The edges from the symmetric difference $\mathcal{E}(\mathcal{C}) \triangle E$, also referred to as the *current violation*.

Then $|D|$ is the cost of the current clustering $\mathcal{C}$. In addition, we shall assume the following trivial representation of the clustering $\mathcal{C}$:

- an identifier for each cluster.
- a doubly-linked list of the vertices in each cluster that can be accessed from the identifier.
- labeling each vertex with its cluster.

With this representation, we can easily, in constant time, ask if two vertices are in the same cluster or move vertices between clusters. Also, for a given vertex u, we can also list the vertices in the same cluster in linear time.

One of the insights for our dynamic framework is that existing static c-approximate correlation clustering algorithms can be transformed to work efficiently if we give them an arbitrary cluster representation $(\mathcal{C}, D)$ as input instead of the classic graph representation $G = (V, E)$. With this input we will transform them so that they produce a new c-approximate correlation clustering $\mathcal{C}'$ in $O(|D|)$ time. We will then use this algorithm to recompute our clustering before the updates has changed the cost of our clustering too much.

Since $\mathcal{C}'$ is then going to be the clustering we use as the initial clustering the next time we recompute, as long as we compute the clustering using an $O(1)$-approximate clustering for $\mathcal{C}'$ then $|D'| = |E \triangle \mathcal{C}'| = O(\text{cost}(\textsc{Opt}))$. This means that after $O(\epsilon|D|)$ updates the cost has at most changed by a multiplicative factor of $1 + \varepsilon$, while the recomputation uses $O(|D|)$ time.

Algorithm 1 Dynamify$(\mathcal{A}, \varepsilon)$: $\mathcal{A}$ takes a cluster presentation $(\mathcal{C}, D)$ and modifies it to a c-approximate cluster representation $(\mathcal{C}', D')$.

```
1:  C ← V, D ← ∅, r ← 0, μ ← ε/(2(1+ε)c)
2:  procedure FLIP(u, v)                          ▷ (u, v) is the edge being updated.
3:      if (u, v) ∈ D then
4:          D ← D \ {(u, v)}
5:      else
6:          D ← D ∪ {(u, v)}
7:      r ← r − 1
8:      if r ≤ 0 then
9:          (C', D') ← A(C, D)
10:         if |D'| ≤ |D| then
11:             (C, D) ← (C', D')
12:         r ← μ|D|
13: end procedure
```

It should be noted that we initially are going to present the algorithm as if the updates are computed in an amortized fashion. It is however not too problematic to deamortize the algorithm by computing the algorithm $\mathcal{A}$ in the background of the following r updates. The details of this computation can be found in Section 2.3.

We are going to present this algorithm as being able to flip the occurrence of an edge. This is the method FLIP in Algorithm 1. This is mostly to simplify the implementation since inserting and deleting edges are essentially the same operation, especially since we are working in a model where what matters is whether the edge currently is contained in the symmetric difference.

Regarding maintaining the set D when performing the FLIP operations in Algorithm 1, we have to be a bit careful. While this would be trivial using hashing, this would add an additional layer of complexity to handle the probability of the hashing not running in the

expected time. Furthermore, in contrast to the clustering algorithm, we are not allowed to fail at this task for any update, since it could have adverse consequences for updates in the far future.

For each edge, we are interested in whether or not it is present in the symmetric difference. The reason we have to be careful is that we only have $O\left(\varepsilon^{-1}\right)$ time for each update, and we do not want this to increase the amount of space we have to use. We could do this trivially by using a 2d array of size $|V|^2$ for each possible edge. This would however increase the memory footprint of the whole algorithm to $O(|V|^2)$ from $O(|E|)$. We are instead going to perform this type of operation in an amortized way, by only calculating the symmetric difference D right before we recompute with the algorithm $\mathcal{A}$. We keep an empty array of size $|V|$. Then for each vertex, we keep track of all updates that changes the neighborhood of this vertex, storing their occurrence each time they are made. Right before recomputing the clustering with $\mathcal{A}$, we then for each vertex v incident to either an update or an edge in D go through all the updates incident to v and determine the final state of each of them. From this we can update all neighbors that could potentially be part of the symmetric difference, by flipping the corresponding entries in the empty array of size $|V|$ for each update and each entry in D, each in $O(1)$ time. This means that we can compute the symmetric difference after r updates in time $O(r + |D|)$.

In the rest of this section, we are going to assume that the clustering algorithm $\mathcal{A}$ is deterministic. This is to simplify the description of our algorithm, since adding the randomness adds an additional layer of complications. We will later expand on this.

▶ **Theorem 13** (Formal version of Theorem 2). *Let $\mu \leq \frac{\varepsilon}{2(1+\varepsilon)c}$. Let $\mathcal{A}$ be a static correlation clustering algorithm that as input takes as a cluster representation $(\mathcal{C}, D)$ and in $O(t|D|)$ time produces a c-approximate cluster representation $(\mathcal{C}', D')$. Then* Dynamify$(\mathcal{A}, \varepsilon)$ *from Algorithm 1, is a fully-dynamic algorithm on G that in $O\left(\mu^{-1}t\right)$ worst-case time per edge update maintains a $((1+\varepsilon)c)$-approximate clustering.*

To show this, we primarily need to make the observation that by not updating anything for $\mu|D|$ updates, we only increase the approximation ratio by a multiplicative ratio of $(1+\varepsilon)$. This enables us to batch the updates and perform them only when the cost will have differed significantly.

▶ **Lemma 14.** *Let $\mathcal{C}$ be a c-approximation of the optimal clustering of the graph $G = (V, E)$ with the symmetric difference $\mathcal{E}(\mathcal{C})\triangle E$, and let $\mu \leq \frac{\varepsilon}{2(1+\varepsilon)c}$. Let $G' = (V', E')$ be a graph such that $|E\triangle E'| \leq \mu|\mathcal{E}(\mathcal{C})\triangle E|$. Then $\mathcal{C}$ is an $((1+\varepsilon)c)$-approximation for the graph G'.*

Proof. Let $\mathcal{C}^*$ be an optimal clustering for G and let $\mathcal{C}'^*$ be an optimal clustering for G'. Then we have that $|\mathcal{E}(\mathcal{C})\triangle E| \leq c|\mathcal{E}(\mathcal{C}^*)\triangle E|$. Furthermore, $\mathcal{C}^*$ gives a lower bound of the optimal solution in G' as $|\mathcal{E}(\mathcal{C}^*)\triangle E| - |E\triangle E'| \leq |\mathcal{E}(\mathcal{C}'^*)\triangle E'|$. Combining we have that

$$\begin{aligned}
|E\triangle E'| &\leq \mu|\mathcal{E}(\mathcal{C})\triangle E| \\
&\leq \frac{\varepsilon}{2(1+\varepsilon)}|\mathcal{E}(\mathcal{C}^*)\triangle E| \\
&\leq \frac{\varepsilon}{2(1+\varepsilon)}(|\mathcal{E}(\mathcal{C}^*)\triangle E'| + |E\triangle E'|) \\
&\leq \frac{\varepsilon}{2(1+\varepsilon)}(|\mathcal{E}(\mathcal{C}'^*)\triangle E'| + 2|E\triangle E'|).
\end{aligned}$$

Isolating for $|E \triangle E'|$, we get $|E \triangle E'| \leq \frac{\varepsilon}{2}|\mathcal{E}(\mathcal{C}'^*) \triangle E'|$. Using this inequality we get

$$\begin{aligned}|\mathcal{E}(\mathcal{C}) \triangle E'| &\leq |\mathcal{E}(\mathcal{C}) \triangle E| + |E \triangle E'| \\ &\leq c|\mathcal{E}(\mathcal{C}^*) \triangle E| + |E \triangle E'| \\ &\leq c|\mathcal{E}(\mathcal{C}'^*) \triangle E'| + 2c|E \triangle E'| \\ &\leq (1+\varepsilon)c|\mathcal{E}(\mathcal{C}'^*) \triangle E'|.\end{aligned}$$

◀

Proof of Theorem 13. By the previous discussion, we can maintain the symmetric difference in an amortized fashion just before it is handed over to $\mathcal{A}$.

For the running time, consider the time at which we recompute the clustering. Let $\hat{D}$ be the violation that was computed the previous time that the cluster was recomputed and D the true violation just before a recomputation. Then it holds that $|D| \leq (1+\mu)|\hat{D}|$ from the fact that each update can increase the violation by at most 1. Thus, we observe that whenever we run $\mathcal{A}(\mathcal{C}, D)$, we have it is at least $\frac{\mu}{1+\mu}|D|$ updates since we last recomputed with $\mathcal{A}$. As $\mathcal{A}$ has running time $O(t|D|)$ and $\mu \leq 1$ by definition, the amortized running time is $O(\mu^{-1}t)$ per update.

Finally, to show the approximation ratio, for any update let E be the edge set at the previous time when the cluster representation $(\mathcal{C}, D)$ was recomputed, and E' the current edge set. Then, since each update at most changes the set of edges by 1 element and that it is at most $\mu|D|$ updates since the last recomputation, we have $|E \triangle E'| \leq \mu|D|$. As $\mathcal{C}$ is c-approximate, by Lemma 14 Dynamify$(\mathcal{A}, \varepsilon)$ maintains an $((1+\varepsilon)c)$-approximate clustering. ◀

Finally, we want to mention that it is possible to maintain the cost of the solution at all times doing the computation, as long as the user is well-behaved, that is the user never attempts to delete an edge that does not exist nor insert an edge that already exists. To achieve this, we are further going to assume that the algorithm is told for each update/flip whether the update is an edge insertion or an edge deletion. In this case, you can simply check whether the two vertices being updated currently are in the same cluster, and then depending on this and whether the update is an insertion or a deletion directly compute the new cost.

2.1 Dynamic blow-up of error probabilities

As we mentioned early, the error probability of a Monte Carlo algorithm can blow up logarithmaticly through our dynamic framework. We will show this effect with a concrete example in this subsection. To show how this blow-up can behave, we are going to work with a specific hypothetical approximation algorithm $\mathcal{A}$, which on cluster representation $(\mathcal{C}, D)$ with probability $1-p$ outputs the optimal clustering and with probability p does nothing and outputs its own input $(\mathcal{C}, D)$ for a constant p. We note that this is not only a problem against an adaptive adversary, but that since the following construction is fully deterministic and specified ahead of time, an oblivious adversary could also implement the same updates.

We imagine that the initial state of the dynamic algorithm is that G consists of $n/3$ disjoint 2-paths. Over the next $2n/3$ updates, we remove both edges in each of the 2-paths in the following updates one by one. We are going to look at the probability that the clustering is not updated after the $(2n/3-1)$'th update, since this is the first time the optimal clustering hits cost 0. Failure to recompute at this update would imply no approximation, and therefore that our algorithm has failed badly.

The issue now arises in the case when we at any point overestimate the actual cost by more than a factor $2\mu^{-1}$. If this happens, r is going to be assigned a value larger than the number of updates before we hit cost 0, implying that we are going to fail badly at the $(2n/3-1)$'th update. This is guaranteed to happen if $k = \mu^{-1}\ln(2\mu^{-1}) = O(1)$ recomputations in a row fail to compute an good approximation. If we were to look at the probability of encountering k failures in a row, then this is given by p^k.

The total number of recomputations is at least $\log n/\log(\mu^{-1}) = \Omega(\log n)$. So we also have at least $r = \Omega(\log n)$ opportunities to achieve k fails in a row. Thus the probability of *never* having k fails in a row is therefore bounded by $(1-p^k)^r < \exp(-rp^k) = \frac{1}{n^{\Omega(p^k)}}$. Since p is a constant, the probability for the algorithm to succeed is polynomially small.

2.2 Bounding the dynamic error probabilities

The pure cases

We will now prove Theorem 3, analyzing how the failure probability of the static algorithm affects the failure probability of the dynamic algorithm.

We are going to consider any update i^*, and we want to bound the probability that our current clustering $\mathcal{K}$ is not $(1+\varepsilon)c$-approximate when we get to this update. In particular, this requires that the last rebuilt cluster representation $(\mathcal{C}', D')$ was not c-approximate.

Taking a step back, suppose we want to do a rebuild before some update i and the input for this rebuild is the cluster representation $(\mathcal{C}, D)$. We then get a new cluster representation $(\mathcal{C}', D')$ with $|D'| \leq |D|$ which we will keep for $\mu|D'|$ updates. We say that a rebuild happening at update i with input $(\mathcal{C}, D)$ is *risky with respect to* i^* if $i^* - i \leq \mu|D|$, for if we do not get a $|D'| < |D|$, then we will not get a better cluster representation before update i^*. Conversely, if it is not risky, then we know we will get another rebuild before we get to update j. Thus, if our clustering is not $(1+\varepsilon)c$-approximate at update j, then some risky rebuild must have failed. We are going to upper bound the probability that any risky rebuild fails. To do this we are going to bound the number of risky rebuilds that can affect a specific update.

▶ **Lemma 15.** *Let $\mu < 1/6$. Suppose we want to do a rebuild at update i with input representation $(\mathcal{C}, D)$ and that the rebuild is risky with respect to i^*. Then we will perform at most 3 rebuilds with input violations above $|D|/2$ between update i and i^*.*

Proof. If update i is risky then $i^* - i \leq \mu|D|$. Suppose the rebuild performs a rebuild that produces the cluster representation $(\mathcal{C}', D')$ such that $|D'| \geq |D|/3$. Then the number of updates until the next rebuild would be at least $\mu|D|/3$. We can therefore have at most 3 such rebuilds we pass before we pass update i^* and this includes a risky rebuild at update i. Suppose on the other hand that we from some update $j \geq i$ do a rebuild leading to an output violation D' is of size less than $|D|/3$. We have at most $\mu|D| < |D|/6$ updates between $j \geq i$ and i^* due to the requirement of μ. Since every update can at most increase the violation by 1, we conclude that between update j and i^*, the violation is always of size below $|D|/2$. We conclude there can at most exists 3 rebuilds between update i and update i^* with an input violation greater than $|D|/2$. ◀

It should be noted that the statement of Lemma 15 is completely independent of how each rebuild is done or whether it succeeds or not. It only uses that the output violation is never bigger than the input violation. Since we have an upper bound of m for the maximum number of violations, we have at most $O(\log m) = O(\log n)$ risky rebuilds, and so if for any risky rebuild the probability that that rebuild fails is bounded by p, then $O(p\log n)$

bounds the probability that any risky rebuild fails, hence the probability that we do not have a $(1+\varepsilon)c$-approximate clustering when we get to update j. This completes the proof of Theorem 3(a).

We now want to understand what happens if the failure probability with input cluster representation $(\mathcal{C}, D)$ is $q(|D|)$ for some non-increasing function q. Let $Q(d)$ be the maximal probability that some risky rebuild fails, starting from some risky rebuild with input violation at most d. Clearly this is an increasing function with $Q(d) = 0$ if $d < 1$. Stating from a risky rebuild with input size d, it follows from Lemma 15, that we can have at most 3 risky rebuilds before we get one with input violation size at most $d/2$. Therefore we have the recurrence $Q(d) \leq 3q(d/2) + Q(d/2)$. If q is a function falling inversely in d, then the sum is geometrically increasing, and then we conclude that $Q(d) = O(q(1))$. This completes the proof of Theorem 3(b).

The mixed case

We will now prove Theorem 4. In this case we are working with two static algorithms.

- $\mathcal{A}$ that is c-approximate with probability p and
- $\hat{\mathcal{A}}$ that is $\hat{c}$-approximate with probability $q(d)$ where q falls inversely with the input violation size d.

Every time we do a rebuild, we apply first $\hat{\mathcal{A}}$, followed by $\mathcal{A}$. We are furthermore always going to keep the best solution after each application of both algorithms, so the combined algorithm $\mathcal{A} \circ \hat{\mathcal{A}}$ gives the best of both worlds (in fact, $\mathcal{A} \circ \hat{\mathcal{A}}$ fails to be $\hat{c}$-approximate with probability at most $p \cdot q(d)$, but we will not exploit that).

We are again going to fix an update i^* and we want to show that the dynamic algorithm is $(1+\varepsilon)c$-approximate at i^* with probability $O(p + q(1))$. We define μ small enough that if we have a c-approximate clustering with violation size d, then it remains $(1+\varepsilon)c$-approximate for μd updates. We will also define μ smaller than $1/6$ and $1/(2\hat{c})$. Picking the minimum of these three options is still a constant. As in the proof for Theorem 3, we say that a rebuild at update i is at risk with respect to update i^* if we have input cluster representation $(\mathcal{C}, D)$ and $i^* - i \leq \mu|D|$. We then apply $\hat{\mathcal{A}}$ producing a cluster representation $(\hat{\mathcal{C}}, \hat{D})$ that will be used as input for $\mathcal{A}$. Next, we only say $\mathcal{A}$ is at risk with respect to i^* if $\hat{\mathcal{C}}$ is $\hat{c}$-approximate and $i^* - i \leq \mu|D^*|$. We then apply $\hat{\mathcal{A}}$ to $(\hat{\mathcal{C}}, \hat{D})$ producing the final output $(\mathcal{C}', D')$ of the rebuild with $\mathcal{A} \circ \hat{\mathcal{A}}$.

With the above definitions, suppose we get to update i^* with a clustering that is not $(1+\varepsilon)c$-approximate. Let the last rebuild be at update i. Then $\mathcal{A} \circ \hat{\mathcal{A}}$ must have failed producing a c-approximate clustering $\mathcal{C}'$. Moreover, the whole rebuild must have been at risk, for otherwise, there would be another rebuild before update i^*.

If $\hat{\mathcal{A}}$ succeeded, then $\mathcal{A}$ must still have been at risk, for otherwise we again conclude that there would be another rebuild before update j. Finally, $\mathcal{A}$ must have failed producing a c-approximate clustering $\mathcal{C}'$.

Recall that $\hat{\mathcal{A}}$ fails producing a $\hat{c}$-approximate clustering with probability $q(d)$ where q falls inversely with the input violation size d. As in our previous analysis, we would like to conclude that the probability that we ever fail a risky application of $\hat{\mathcal{A}}$ which fails is $O(q(1))$. However, we need to revisit the argument to check that we are not cheating because the full rebuild also applies $\mathcal{A}$.

Consider a risky rebuild at update i. With input violations size d, risky means that $i^* - i \leq \mu d$. We will now apply Lemma 15 which doesn't care how the rebuild is done, so it also applies when now rebuild with $\mathcal{A} \circ \hat{\mathcal{A}}$. The lemma states that we can perform at most 3 rebuilds with input violations above $d/2$ between update i and i^*.

As before, we now define $Q(d)$ as the maximal probability that $\mathcal{A}^*$ ever fails on on a risky rebuild starting from input violation size at most d. Then $Q(d) \leq 3q(d/2)+Q(d/2) = O(q(1))$. We will now assume that $\hat{\mathcal{A}}$ never fails a risky rebuild. The last rebuild must then have included a risky application of $\mathcal{A}$ which failed. We will bound the probability of this event by $O(p)$. For this we use the following lemma.

▶ **Lemma 16.** *Let $\mu \leq 1/(2\hat{c})$. Suppose we do a rebuild at update i with a $\hat{c}$-approximate input representation $(\hat{\mathcal{C}}, \hat{D})$ and that the rebuild is risky with respect to i^*. Then we will perform at most $2\hat{c}$ rebuilds between update i and i^*.*

Proof. Let $(\mathcal{C}_i^*, D_i^*)$ be an optimal clustering at update i. Each update can decrease the violation by at most one, so $|D_{i^*}^*| \geq |D_i^*| - (i^* - i) \geq |\hat{D}|/\hat{c} - \mu|\hat{D}| \geq |\hat{D}|/(2\hat{c})$, where the last inequality used that $\mu \leq 1/(2\hat{c})$. For every rebuild between update i and update i^* producing an output representation $(\mathcal{C}', D')$, we have $|D'| \geq |D_i^*| \geq |\hat{D}|/(2\hat{c})$, and so we have at least $\mu|\hat{D}|/(2\hat{c})$ updates between any two rebuilds between update i and i^*. Thus the total number of rebuilds can be at most $\frac{(i^*-i)}{\mu|\hat{D}|/(2\hat{c})} \leq \frac{\mu|\hat{D}|}{\mu|\hat{D}|/(2\hat{c})} = 2\hat{c}$. ◀

Using Lemma 16, we conclude that from the first risky rebuild with $\mathcal{A}$, there can be at $2\hat{c}$ times rebuilds, hence the probability that $\mathcal{A}$ makes a risky failure with respect to i^* is bounded by $2\hat{c}p = O(p)$.

Summing up, we conclude that the probability that $\hat{\mathcal{A}}$ or $\mathcal{A}$ ever fail a risky update with respect to i^* is bounded by $O(p + q(1))$. This also bounds the probability that our dynamic clustering is not $(1+\varepsilon)c$-approximate when we get to update i^*, completing the proof of Theorem 4.

2.3 Deamortization

In this section, we are going to explore how to de-amortize the algorithm so that each update runs in worst case $O(\mu^{-1}t)$ instead of just amortized $O(\mu^{-1}t)$. For this, we are mostly going to be using standard techniques. The general idea is that whenever we determine that we are going to recompute the clustering, we start doing this by making $O(\mu^{-1}t)$ work for each update on the recomputation.

In the process of describing the amortization, we describe the algorithm as running over a number of epochs. Each epoch consists of a recomputation of the clustering, and the cleaning and comparison of our current available two clusterings and swapping in the best of the two.

The deamortization is essentially going to work by computing $O\left(\mu^{-1}t\right)$ steps of the static algorithm for each update. This means that we are going to have a pipeline-like structure, where whenever an update arrives in some epoch i, we are only going to start computing the clustering using it in the next epoch $i+1$. Then only from the epoch after that one, $i+2$, are we going to use the result of the computations to answer queries. This delay will at most result in a multiplicative error of $(1+O(\varepsilon))$ to the approximation factor for each epoch we delay between receiving the update to an edge and the time in which it is used to answer the received queries.

We have to be a bit careful with this deamortization approach. The reason is that we need to swap out the old clustering for a new one in $O\left(\mu^{-1}t\right)$ time in the worst case. This is non-trivial, since the cluster representation $(\mathcal{C}, D)$ has size $\Omega(n)$ which might be much larger than $|D|$, in which case we do not have the time to make a copy and swap it in. So we have to make sure that we do not copy more than what is necessary.

Swapping in a new solution

To swap in a new solution, for each vertex we maintain two entries, the possible ids of clusters that a particular vertex can belong to. These correspond to the clustering the vertex belongs to in the old clustering and the one in the new clustering. We furthermore have a global flag that indicates whether to use the old clustering or the new one. The way we use this information is that if two options exist for the cluster a vertex can belong to, then the global flag decides. Otherwise, the vertex belongs to the only cluster that is written down. This means that we can swap in the new solution by flipping the global flag.

After we have swapped in the new solution, we spend time proportional to the number of updates for the old computation to clean up the vertices that have two cluster ids written down. After that, the new solution has become the old one, so we finally flip the global flag to say use the old solution if it exists, to be ready for the next round of computations.

3 Dynamic Pivot

In this section, we will show how to implement the Pivot algorithm [2] in $O(|D|)$ time in cluster representation $(\mathcal{C}, D)$, as a warm-up for our full algorithm. Before the detail, we first show some helpful properties of the cluster representation (See Figure 2).

▶ **Definition 17** (Active). *A vertex v is* active *if it is incident to some edges in D, otherwise it is* inactive. *A cluster in $\mathcal{C}$ is* active *if it contains an active vertex, otherwise it is* inactive.

▶ **Definition 18** (Core). *For any active cluster C, the set of inactive vertices in C is called the* core *of C, denoted by* $\text{core}(C)$.

In the next lemma, we will see that all inactive vertices can be treated in groups, and the only challenging part is on the active vertices.

▶ **Lemma 19.** *For any graph represented by $(\mathcal{C}, D)$, the following two statements hold.*

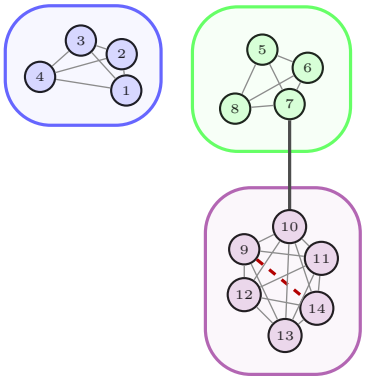
1. *Every inactive cluster C is a perfect clique in the graph, with no exterior edges incident to its vertices.*
2. *For any active cluster C, all the vertices in* $\text{core}(C)$ *have the same neighborhood, which equals to C.*

The proof of the lemma is straightforward. Therefore, any locally good clustering has to include all the inactive clusters. Moreover, it has to put all vertices in a core together and never put two cores in the same cluster. From this fact, we can contract each core into a weighted vertex. When creating a new clustering, we keep the label of the clusters to each core the same, and only move active vertices around. After contraction, we only need to deal with a graph of $O(|D|)$ vertices, which are the cores and active vertices.

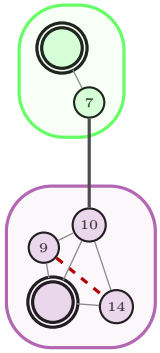
Recall that the Pivot algorithm proceeds by selecting a vertex v uniformly at random, then creates a new cluster $N(v)$ and recurses on the remaining graph. We will simulate the Pivot algorithm directly on the contracted graph in $O(|D|)$ time. The pseudocode is presented in Algorithm 2.

▶ **Theorem 20.** *Pivot$(\mathcal{C}, D)$ from Algorithm 2 runs in $O(|D|)$ time and has the same output distribution as Pivot(V, E) from [2] for the same graph.*

Proof. We will first show the correctness of this algorithm. Each group of vertices contracted to the same one, are neighbors to each other and have the same outgoing neighbors, so they will always be put in the same cluster. Therefore, we can treat them as a single one with a higher probability to be sampled. Note that all clusters in $\mathcal{C} \setminus \mathcal{S}$ are cliques without any



(a) Input representation $(\mathcal{C}, D)$ of graph G.



(b) V' after removing inactive vertices and adding virtual vertices.

Figure 2 Outline of preprocessing done for Pivot$(\mathcal{C}, D)$.

Algorithm 2 Pivot$(\mathcal{C}, D)$.

1: Let V' be the set of active vertices and $\mathcal{S}$ be the set of active clusters.
2: For each active cluster C, add a virtual vertex c for core(C) to V'.
3: For each vertex $v \in V'$, set its weight w_v to be the number of vertices it represents.
4: For each active cluster C, construct a list $L(C)$ of all the vertices in V' from C.
5: For each vertex $v \in V'$, let $N^+(v)$ be the list of neighbors of v out of $C(v)$, and $N^-(v)$ be the list of non-neighbors of v in $C(v)$.
6: $\mathcal{C}' \leftarrow \mathcal{C} \setminus \mathcal{S}$ ▷ We can do this in-place in $O(|\mathcal{S}|)$ time
7: **while** $V' \neq \emptyset$ **do**
8: Pick a vertex $v \in V'$ with probability proportional to w_v
9: $T \leftarrow (N^+(v) \cap V') \cup (L(C(v)) \setminus N^-(v))$
10: $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{T\}$, $V' \leftarrow V' \setminus T$, $L(C(v)) \leftarrow L(C(v)) \cap N^-(v)$
11: Unpack $\mathcal{C}'$ and return the corresponding clustering for the original graph

other edges, so the Pivot algorithm will always put them together. When we start line 16, $N^+(v)$ is the set of neighbors of v out of $C(v)$, $N^-(v)$ is the set of non-neighbors of v in $C(v)$. While running the loop from line 7 to line 10, $L(C)$ keeps track of the remaining vertices in V' associated with cluster C. So T is the set of remaining neighbors of v in V', therefore it has the same output distribution as the Pivot algorithm in [2].

Now we are going to analyze the running time of this algorithm. We have $|\mathcal{S}| \leq 2|D|$ and $|V'| \leq 2|D| + |\mathcal{S}| \leq 4|D|$, so it only takes $O(|D|)$ time before the start of line 7. By Lemma 21, line 8 can be implemented in total time $O(|V'|)$ by weighted sampling without replacement. Finally, note that T in line 9 can be computed in $O(|T| + |N^+(v)| + |N^-(v)|)$ time, and the total size of T is at most $|V'|$, and the total size of $N^+(v) \cup N^-(v)$ is at most $2|D|$, so the running time of the full algorithm is $O(|D|)$. ◀

The next lemma shows that line 8 in total can be implemented in $O(|V'|)$ time.

▶ **Lemma 21** ([26] Lemma 28). *Let A be a set of initially n objects with associated integer weights $\{w_a\}_{a \in A}$. There is a data structure supporting the following operations on A:*

- *SAMPLE(): Samples and removes an element $a \in A$, where $a \in A$ is selected with probability $w_a / \sum_{a' \in A} w_{a'}$.*
- *REMOVE(a): Removes a from A.*

The total time to initialize the data structure and to run the previous operations until A is empty is bounded by $O(n)$, with high probability $1 - \frac{1}{n^c}$, for any constant $c > 0$.

It remains to show how to compute the new symmetric difference D' with the new clustering C'. Here we use the fact that the new clustering $\mathcal{C}'$ is constructed in-place from $\mathcal{C}$, therefore, we get a log of modifications from $\mathcal{C}$ to $\mathcal{C}'$ for free. Let L denote the set of vertices that moved to different clusters from $\mathcal{C}$ to $\mathcal{C}'$. We will compute $D' = E \triangle \mathcal{E}(\mathcal{C}') = \mathcal{E}(\mathcal{C}) \triangle \mathcal{E}(\mathcal{C}') \triangle D$. Since our objective is to find a clustering of small symmetric difference, we can stop and report the original clustering if $\mathcal{C}'$ is worse than $\mathcal{C}$. The pseudocode is given in Algorithm 3.

Algorithm 3 SymmetricDifference$(\mathcal{C}, D, \mathcal{C}', L)$.

1: Compute $D' \leftarrow D \cap (\mathcal{E}(\mathcal{C}) \triangle \mathcal{E}(\mathcal{C}') \triangle D)$ by a scan of D
2: **for** each $v \in L$ **do**
3: Let $C(v)$ and $C'(v)$ be the cluster of v in $\mathcal{C}$ and $\mathcal{C}'$ respectively.
4: Compute $X = C(v) \triangle C'(v)$ and $Y = C(v) \cap C'(v)$ in $O(|C(v)| + |C'(v)|)$ time
5: **for** each $x \in X$ and $y \in Y$ **do**
6: Insert (x, y) to D' if $(x, y) \notin D$
7: $L \leftarrow L \setminus Y$
8: **return** $(\mathcal{C}', D')$

▶ **Lemma 22.** *Algorithm 3 outputs $D' = E \triangle \mathcal{E}(\mathcal{C}')$ in $O(|D| + |D'|)$ time.*

Proof. The correctness of the algorithm is straightforward, and we only focus on its running time. In line 4, we know $v \in C(v) \cap C'(v)$, and $C(v) \neq C'(v)$ as v is moved between clusters. So we have $|X| > 0$ and $|Y| > 0$ and $|X| + |Y| = |C(v)| + |C'(v)|$. Therefore, we will detect $|X| \cdot |Y| = \Omega(|C(v)| + |C'(v)|)$ pairs $(x, y) \in \mathcal{E}(\mathcal{C}) \triangle \mathcal{E}(\mathcal{C}')$ in the next for-loop. Since every pair in $\mathcal{E}(\mathcal{C}) \triangle \mathcal{E}(\mathcal{C}')$ will be checked at most twice, the total running time is $O(|\mathcal{E}(\mathcal{C}) \triangle \mathcal{E}(\mathcal{C}')| + |D|) = O(|D' \triangle D| + |D|) = O(|D| + |D'|)$. ◀

We can directly translate the algorithm into another with running time $O(|D|)$ that output the best of the $(\mathcal{C}, D)$ and $(\mathcal{C}', D)$.

▶ **Corollary 23.** *Given a graph $(\mathcal{C}, D)$, a clustering $\mathcal{C}'$ and the list L of moved vertices, we can output the best of the two clustering $(\mathcal{C}, D)$ and $(\mathcal{C}', D')$ in $O(|D|)$ time.*

Proof. We can simulate Algorithm 3 for at most $c|D|$ steps, where c is a sufficiently large constant determined by Algorithm 3. If it terminates, we return the best of $(\mathcal{C}, D)$ and $(\mathcal{C}', D')$; if it does not stop, we are sure that $|D'| > |D|$, and we can return $(\mathcal{C}, D)$. ◀

As a remark, by Theorem 3 (a), if at each rebuild we repeat Algorithm 2 for $O(\log \log n)$ time and take the best clustering, we can turn Pivot$(\mathcal{C}, D)$ into a $(3 + \varepsilon)$-approximate dynamic correlation clustering algorithm with $O(\log \log n)$ update time and error probability of $O(1/\text{poly}(\log n))$.

References

1 Rakesh Agrawal, Alan Halverson, Krishnaram Kenthapadi, Nina Mishra, and Panayiotis Tsaparas. Generating labels from clicks. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 172–181, 2009. doi:10.1145/1498759.1498824.

2 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5):1–27, 2008. doi:10.1145/1411509.1411513.

3 Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE)*, pages 952–963, 2009. doi:10.1109/ICDE.2009.43.

4 Sepehr Assadi, Sanjeev Khanna, and Aaron Putterman. Correlation clustering and (de)sparsification: Graph sketches can match classical algorithms. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 417–428. ACM, 2025. doi:10.1145/3717823.3718194.

5 Sepehr Assadi and Chen Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. In *Proceedings of the 13th Conference on Innovations in Theoretical Computer Science (ITCS)*, volume 215 of *LIPIcs*, pages 10:1–10:20, 2022. doi:10.4230/LIPIcs.ITCS.2022.10.

6 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56(1):89–113, 2004. doi:10.1023/B:MACH.0000033116.57574.95.

7 Soheil Behnezhad, Moses Charikar, Vincent Cohen-Addad, Alma Ghafari, and Weiyun Ma. Fully dynamic correlation clustering: Breaking 3-approximation, 2024. doi:10.48550/arXiv.2404.06797.

8 Soheil Behnezhad, Mahsa Derakhshan, Mohammad Taghi Hajiaghayi, Clifford Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 382–405, 2019. URL: https://api.semanticscholar.org/CorpusID:202539552.

9 Amos Beimel, Haim Kaplan, Yishay Mansour, Kobbi Nissim, Thatchaphol Saranurak, and Uri Stemmer. Dynamic algorithms against an adaptive adversary: generic constructions and lower bounds. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1671–1684, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520064.

10 Francesco Bonchi, Aristides Gionis, and Antti Ukkonen. Overlapping correlation clustering. *Knowledge and Information Systems*, 35(1):1–32, 2013. doi:10.1007/S10115-012-0522-9.

11 Vladimir Braverman, Prathamesh Dharangutte, Shreyas Pai, Vihan Shah, and Chen Wang. Fully dynamic adversarially robust correlation clustering in polylogarithmic update time. *arXiv preprint arXiv:2411.09979*, 2024. doi:10.48550/arXiv.2411.09979.

12 Vladimir Braverman, Prathamesh Dharangutte, Shreyas Pai, Vihan Shah, and Chen Wang. Fully dynamic adversarially robust correlation clustering in polylogarithmic update time. In Yingzhen Li, Stephan Mandt, Shipra Agrawal, and Mohammad Emtiyaz Khan, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2025, Mai Khao, Thailand, 3-5 May 2025*, volume 258 of *Proceedings of Machine Learning Research*, pages 1477–1485. PMLR, 2025. URL: https://proceedings.mlr.press/v258/braverman25a.html.

13 Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, David Rasmussen Lolck, Alantha Newman, Mikkel Thorup, Lukas Vogl, Shuyi Yan, and Hanwen Zhang. Solving the correlation cluster LP in sublinear time. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1154–1165, 2025. The approximation factor of 1.437 was incorrect and fixed to 1.485 in the arXiv version. doi:10.1145/3717823.3718181.

14 Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, David Rasmussen Lolck, Alantha Newman, Mikkel Thorup, Lukas Vogl, Shuyi Yan, and Hanwen Zhang. Static to dynamic correlation clustering, 2025. doi:10.48550/arXiv.2504.12060.

15 Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, Alantha Newman, and Lukas Vogl. Understanding the cluster linear program for correlation clustering. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1605–1616, 2024. The approximation factor of 1.437 was incorrect and fixed to 1.485 in the arXiv version. doi:10.1145/3618260.3649749.

16 Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. A graph-theoretic approach to webpage segmentation. In *Proceedings of the 17th International conference on World Wide Web (WWW)*, pages 377–386, 2008. doi:10.1145/1367497.1367549.

17 Sayak Chakrabarty and Konstantin Makarychev. Single-pass pivot algorithm for correlation clustering. keep it simple! In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. doi:10.48550/arXiv.2305.13560.

18 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005. doi:10.1016/J.JCSS.2004.10.012.

19 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 47th annual ACM Symposium on Theory of Computing (STOC)*, pages 219–228, 2015. doi:10.1145/2746539.2746604.

20 Yudong Chen, Sujay Sanghavi, and Huan Xu. Clustering sparse graphs. In *Advances in Neural Information Processing Systems (Neurips)*, pages 2204–2212, 2012.

21 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)*, pages 2069–2078, 2021. URL: http://proceedings.mlr.press/v139/cohen-addad21b.html.

22 Vincent Cohen-Addad, Euiwoong Lee, Shi Li, and Alantha Newman. Handling correlated rounding error via preclustering: A 1.73-approximation for correlation clustering. In *Proceedings of 64th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1082–1104, 2023. doi:10.1109/FOCS57990.2023.00065.

23 Vincent Cohen-Addad, Euiwoong Lee, and Alantha Newman. Correlation clustering with Sherali-Adams. In *Proceedings of 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 651–661, 2022. doi:10.1109/FOCS54457.2022.00068.

24 Vincent Cohen-Addad, David Rasmussen Lolck, Marcin Pilipczuk, Mikkel Thorup, Shuyi Yan, and Hanwen Zhang. Combinatorial correlation clustering. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1617–1628, 2024. doi:10.1145/3618260.3649712.

25 Mina Dalirrooyfard, Konstantin Makarychev, and Slobodan Mitrović. Pruned pivot: Correlation clustering algorithm for dynamic, parallel, and local computation models, 2024. doi:10.48550/arXiv.2402.15668.

26 Nick Fischer, Evangelos Kipouridis, Jonas Klausen, and Mikkel Thorup. A faster algorithm for constrained correlation clustering. *arXiv preprint arXiv:2501.03154*, 2025. Conference version to appear at STACS'25. doi:10.48550/arXiv.2501.03154.

27 Dmitri V. Kalashnikov, Zhaoqi Chen, Sharad Mehrotra, and Rabia Nuray-Turan. Web people search via connection analysis. *IEEE Transactions on Knowledge and Data Engineering*, 20(11):1550–1565, 2008. doi:10.1109/TKDE.2008.78.

28 Xinghao Pan, Dimitris S. Papailiopoulos, Samet Oymak, Benjamin Recht, Kannan Ramchandran, and Michael I. Jordan. Parallel correlation clustering on big graphs. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 82–90, 2015. URL: https://proceedings.neurips.cc/paper/2015/hash/b53b3a3d6ab90ce0268229151c9bde11-Abstract.html.

29 Jessica Shi, Laxman Dhulipala, David Eisenstat, Jakub Lacki, and Vahab S. Mirrokni. Scalable community detection via parallel correlation clustering. *Proc. VLDB Endow.*, 14(11):2305–2313, 2021. doi:10.14778/3476249.3476282.

30 Anke van Zuylen and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Math. Oper. Res.*, 34(3):594–620, 2009. doi:10.1287/MOOR.1090.0385.

31 Nate Veldt. Correlation clustering via strong triadic closure labeling: Fast approximation algorithms and practical lower bounds. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 22060–22083. PMLR, 17–23 July 2022. URL: `https://proceedings.mlr.press/v162/veldt22a.html`.