Correlation Clustering in Constantly Many Parallel Rounds

Vincent Cohen-Addad [*1] **Silvio Lattanzi** [*1] **Slobodan Mitrović** [*2] **Ashkan Norouzi-Fard** [*1] **Nikos Parotsidis** [*1]
Jakub Tarnawski [*3]

Abstract

Correlation clustering is a central topic in unsupervised learning, with many applications in ML and data mining. In correlation clustering, one receives as input a signed graph and the goal is to partition it to minimize the number of disagreements. In this work we propose a massively parallel computation (MPC) algorithm for this problem that is considerably faster than prior work. In particular, our algorithm uses machines with memory sublinear in the number of nodes in the graph and returns a constant approximation while running only for a constant number of rounds. To the best of our knowledge, our algorithm is the first that can provably approximate a clustering problem on graphs using only a constant number of MPC rounds in the sublinear memory regime. We complement our analysis with an experimental analysis of our techniques.

1. Introduction

Clustering is a classic problem in machine learning. The goal of clustering is to partition a given set of objects into sets so that objects in the same cluster are similar to each other while objects in different clusters are dissimilar. One of the most studied formulations of this problem is *correlation clustering*. Thanks to its simple and natural formulation, this clustering variant has many applications in finding clustering ensembles (Bonchi et al., 2013), in duplicate detection (Arasu et al., 2009), community detection (Chen et al., 2012), disambiguation tasks (Kalashnikov et al., 2008), and automated labelling (Agrawal et al., 2009; Chakrabarti et al., 2008).

Correlation clustering was first formulated by Bansal et al. (2004). Formally, in this problem we are given as input a weighted graph with n nodes, where positive edges represent similarities between nodes and negative edges represent dissimilarities between them. We are interested in clustering the nodes to minimize the sum of the weights of the negative edges contained inside any cluster plus the sum of positive edges crossing any two clusters. The problem is known to be NP-hard, and much attention has been paid to designing approximation algorithms for the minimization version of the problem, as well as for its complementary version where one is interested in maximizing agreement. In particular, for the most studied version of the problem, where the weights are restricted to be in $\{-1, +1\}$, a polynomial-time approximation scheme is known for the maximization version of the problem (Bansal et al., 2004) and a 2.06-approximation algorithm is known for its minimization version (Chawla et al., 2015). Furthermore, when weights are in $\{-1, +1\}$ and the number of clusters is upper-bounded by k, a polynomial-time approximation scheme is known also for the minimization version of the problem (Giotis & Guruswami, 2005). For arbitrary weights, we know a 0.7666-approximation algorithm for the maximization version of the problem (Charikar et al., 2005; Swamy, 2004) and an $O(\log n)$-approximation for the minimization version of the problem (Demaine et al., 2006).

One main drawback of classic solutions for correlation clustering is that they do not scale very well to very large networks. Thus, as the magnitude of available data grows, it becomes increasingly important to design efficient parallel algorithms for this problem. Unfortunately, obtaining such algorithms is often challenging because classic solutions to graph problems are inherently sequential, e.g., the algorithm is defined iteratively and in an adaptive manner. Concretely, a well-known and widely used algorithm for the unweighted minimization version of the problem requires solving a linear program or running a so-called *Pivot* algorithm (Ailon et al., 2008; Chawla et al., 2015). Designing an efficient parallel linear program solver, if one exists at all, is a major challenge. The Pivot algorithm is extremely elegant and simple: it starts by selecting a node uniformly at random in the graph; then it creates a cluster by clustering together the node with all its positive neighbors; finally the algorithm recurs on the rest of the graph. Interestingly, this simple algorithm returns a 3-approximation to the min-

[*]Equal contribution [1]Google Research, Zürich, Switzerland [2]CSAIL, MIT, Cambridge, MA, USA [3]Microsoft Research, Redmond, WA, USA. Correspondence to: Ashkan Norouzi-Fard <ashkannorouzi@google.com>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

imization version of the problem when the weights are in $\{-1, +1\}$. However, despite its simplicity, it is quite challenging to parallelize this algorithm efficiently. A strong step in this direction was presented by Chierichetti et al. (2014), who show how to approximately parallelize the Pivot algorithm using $O\left(\frac{\log^2 n}{\epsilon}\right)$ parallel rounds to obtain a $(3+\epsilon)-$approximation for the problem. In a subsequent work, Ahn et al. (2015) present a nice result for the semi-streaming setting, which can be adapted to provide a 3-approximation by running $O(\log\log n)$ rounds and using $\tilde{O}(n)$ memory per machine. In another related work, Pan et al. (2015) propose a new algorithm that runs in $O\left(\frac{\log n \log \Delta}{\epsilon}\right)$ rounds (where Δ is the maximum positive degree) and obtain very nice experimental results. A natural important question has thus been: *Is it possible to approximate unweighted minimum disagreement in $o(\log n)$ many rounds with $o(n)$ memory per machine?* In this paper we answer this question affirmatively. Moreover, we design an algorithm that requires only $O(1)$ rounds, thus improving on the existing approaches in regimes of both $\tilde{O}(n)$ and $o(n)$ memory per machine. Next, we discuss the precise model of parallelism that we use in this work.

The MPC model. We design algorithms for the massively parallel computation (MPC) model, which is a theoretical abstraction of real-world parallel systems such as MapReduce (Dean & Ghemawat, 2008), Hadoop (White, 2012), Spark (Zaharia et al., 2010) and Dryad (Isard et al., 2007). The MPC model (Karloff et al., 2010; Goodrich et al., 2011; Beame et al., 2013) is widely used as the de-facto standard theoretical model for large-scale parallel computing.

In the MPC model, computation proceeds in synchronous parallel *rounds* over multiple machines. Each machine has memory S. At the beginning of the computation, data is arbitrarily partitioned across the machines. During each round, machines process data locally. At the end of a round, machines exchange messages, with the restriction that each machine is allowed to send messages and receive messages of total size S. The efficiency of an algorithm in this model is measured by the number of rounds it takes for the algorithm to terminate and by the size S of the memory of every machine. In this paper we focus on the most practical and challenging regime, also known as the *sublinear* regime, where each machine has memory $S = O(n^\delta)$ where δ is an arbitrary constant smaller than 1.

Our contribution. Our main contribution is to present a constant-factor approximation algorithm for the minimization problem when the weights are in $\{-1, +1\}$. Our new algorithm runs using only a constant number of rounds in the sublinear regime.

Theorem 1.1. *For any constant $\delta > 0$, there exists an MPC algorithm that, given a signed graph $G = (V, E^+)$, where E^+ denotes the set of edges with weight +1, in $O(1)$ rounds computes a $O(1)$-approximate correlation clustering. Letting $n = |V|$, this algorithm succeeds with probability at least $1 - 1/n$ and requires $O(n^\delta)$ memory per machine. Moreover, the algorithm uses a total memory of $O(|E^+| \cdot \log n)$.*

To the best of our knowledge, this is the first MPC graph clustering algorithm that runs in a constant number of rounds in the sublinear regime. Furthermore, we also show that our algorithms extend to the semi-streaming setting. In particular, in this setting, our algorithm outputs an $O(1)$-approximate correlation clustering in only $O(1)$ passes over the stream. In terms of the number of passes, this significantly improves on the 3-approximate algorithm by (Ahn et al., 2015), which requires $O(\log\log n)$ passes.

Theorem 1.2. *There exists a semi-streaming algorithm that, given a signed graph $G = (V, E^+)$, where E^+ denotes the set of edges with weight +1, in $O(1)$ passes computes a $O(1)$-approximate correlation clustering. Letting $n = |V|$, this algorithm succeeds with probability at least $1 - 1/n$.*

We complement our theoretical results with an empirical analysis showing that our MPC algorithm is significantly faster than previously known algorithms (Chierichetti et al., 2014; Pan et al., 2015). Furthermore, despite its theoretical approximation guarantees being inferior to previous work, in our experiments the quality of the solution is better. We explain this as follows: (1) all clusters returned by our algorithm are guaranteed to be very dense, as opposed to the pivot-based algorithms, where formed clusters might be sparse, and (2) the similarities between our algorithm and existing heuristics for clustering that are known to work very well in practice (Xu et al., 2007).

Techniques and Roadmap. In contrast to previous known parallel algorithms (Chierichetti et al., 2014; Ahn et al., 2015; Pan et al., 2015), our algorithm is not based on a parallel adaptation of the Pivot algorithm. Instead, we study structural properties of correlation clustering. We show that, up to losing a constant factor in the quality of the solution, one can simply focus on clusters consisting of points whose neighborhoods are almost identical (up to a small multiplicative factor); we call such points "in agreement" and then focus on clusters of such points. The next key idea is to trim the input graph so as to only keep edges between some specific points in agreement (intuitively, the points that are in agreement with many of their neighbors), so that clusters of points in agreement correspond to connected components in this trimmed graph. We show how the above operations can be performed in few rounds. Finally, it remains to simply compute the connected components of the trimmed graph to obtain the final clusters. Here we prove an important feature of the trimmed graph: Each connected component has constant diameter. This ensures that this last step can indeed be performed in few parallel rounds.

In Section 3 we present our algorithm, then in Section 3.1 we present its analysis and in Section 3.2 its MPC implementation. In Section 3.3 we show how to extend these ideas to the semi-streaming setting. Finally we present our experimental results in Section 4. All the missing proofs and experiments are presented in the supplementary material.

2. Preliminaries

In this paper, we study the *min-disagree* variant of correlation clustering in the "complete graph case", where we are given a complete graph and each edge is labeled with either -1 or $+1$: E^+ and E^- respectively denote the set of edges labeled $+1$ and -1. The goal is to find a partition $C_1, \ldots, C_t$ of the vertices of the graph that minimizes the following objective:

$$f(C_1, \ldots, C_t) = \sum_{\substack{\{u,v\} \in E^+: \\ u \in C_i, v \in C_j, i \neq j}} 1 + \sum_{\substack{\{u,v\} \in E^-: \\ u,v \in C_i}} 1.$$

Notation. In our analysis and discussion, instead of working with a signed graph (V, E^+, E^-), we work with an unweighted undirected graph $G = (V, E)$, where E refers to E^+. Therefore, by this convention, $E^- = \binom{V}{2} \setminus E$. In addition, when we say that two vertices u and v are neighbors, we mean that $\{u, v\} \in E = E^+$.

We use some standard notation that we briefly recall here. For a vertex $v \in V$, we refer to its neighborhood by $N(v)$ and to its degree by $d(v)$; we further let $N(v, H)$ denote the neighborhood of v in a subgraph H of G. We also consider the degree of a vertex v induced on an arbitrary subset $S \subseteq V$ of nodes and we denote it by $d(v, S)$. We also refer to the hop-distance between two vertices $u, v \in V$ by $\text{dist}^G(u, v)$. We consider the hop distance also in subgraphs $\widetilde{G}$ of G, in which case we denote the distance by $\text{dist}^{\widetilde{G}}(u, v)$. Finally for any two sets R, S, we denote their symmetric difference by $R \triangle S$.

Remark 2.1. *We assume that each vertex has a self-loop "+" edge. Note that this does not affect the cost of clustering, as a self-loop is never cut by a clustering. Note that this assumption implies that $v \in N(v)$.*

3. Algorithm

The starting point of our approach is the notion of *agreement* between vertices. Informally, we say that u and v are in agreement when their neighborhoods significantly overlap. Intuitively, in such scenario, we expect u and v to be treated equally by an algorithm: either u and v are in the same cluster, or both of them form singleton clusters.

Our algorithms are parametrized by two constants β, λ that will be determined later.

Definition 3.1 (Weak Agreement). *Two vertices u and v are in i-*weak agreement *if $|N(u) \triangle N(v)| < i\beta \cdot \max\{|N(u)|, |N(v)|\}$. If u and v are in 1-weak agreement, we also say that u and v are in* agreement*.*

Having the agreement notion in hand, we provide our approach in Algorithm 1.

Algorithm 1 Correlation-Clustering(G)

```
1: Discard all edges whose endpoints are not in agreement.
   (First compute the set of these edges. Then remove this
   set.)
2: Call a vertex light if it has lost more than a $\lambda$-fraction
   of its neighbors in the previous step. Otherwise call it
   heavy.
3: Discard all edges between two light vertices.
4: Call the current graph $\widetilde{G}$, or the sparsified graph. Compute
   its connected components, and output them as the
   solution.
```

3.1. Analysis

Our analysis consists of two main parts. The first part consists of analyzing properties of $\widetilde{G}$ and, in particular, showing that each connected component of $\widetilde{G}$ has $O(1)$ diameter, with all vertices being in $O(1)$-weak agreement. The second part shows that the number of edges removed in Lines 1-3 of Algorithm 1 is only a constant factor larger than the cost an optimal solution.

3.1.1. Properties of $\widetilde{G}$

Our analysis hinges on several properties of vertices being in weak agreement. We start by stating those properties, whose proofs are deferred to the supplementary material.

Fact 3.2. *Suppose that $\beta < \frac{1}{20}$.*

(1) If u and v are in i-weak agreement, for some $1 \leq i < \frac{1}{\beta}$, then

$$(1 - \beta i) d(u) \leq d(v) \leq \frac{d(u)}{1 - i\beta}.$$

(2) Let $k \in \{2, 3, 4, 5\}$ and $v_1, \ldots, v_k \in V$ be a sequence of vertices such that v_i is in agreement with v_{i+1} for $i = 1, \ldots, k-1$. Then v_1 and v_k are in k-weak agreement.

(3) If u and v are in i-weak agreement, for some $1 \leq i < \frac{1}{\beta}$, then $|N(v) \cap N(u)| \geq (1 - i\beta) d(v)$.

By building on these claims, we are able to show that $\widetilde{G}$ has a very convenient structure: each of its connected components has diameter of only at most 4; and every two vertices (one of them being heavy) in a connected component of $\widetilde{G}$ are in 4-weak agreement. More formally, we have:

Lemma 3.3. *Suppose that $5\beta + 2\lambda < 1$. Let CC be a connected component of $\widetilde{G}$. Then, for every $u, v \in CC$:*

(a) if u and v are heavy, then $\mathrm{dist}^{\widetilde{G}}(u, v) \leq 2$,

(b) $\mathrm{dist}^{\widetilde{G}}(u, v) \leq 4$,

(c) $\mathrm{dist}^{G}(u, v) \leq 2$,

(d) if u or v is heavy, then u and v are in 4-weak agreement.

We now illustrate how to apply Lemma 3.3 to show further helpful properties of connected components of $\widetilde{G}$. First, observe that a non-trivial connected component CC of $\widetilde{G}$ (i.e., one consisting of at least two vertices) has at least one heavy vertex. (As a reminder, heavy vertices are defined on Line 2 of Algorithm 1.) Indeed, any edge in $\widetilde{G}$ has at least one heavy endpoint, as assured by Line 3 of Algorithm 1. Let x be such a heavy vertex. Then, by Property (d) of Lemma 3.3 we have that *every* other vertex in CC shares a large number of neighbors with x. One can turn this property into a claim stating that all vertices in CC have induced degree inside CC very close to $|CC|$. Formally:

Lemma 3.4. *Let CC be a connected component of $\widetilde{G}$ such that $|CC| \geq 2$. Then, for each vertex $u \in CC$ we have that*

$$d(u, CC) \geq (1 - 8\beta - \lambda)|CC|.$$

(Note that $d(u, CC)$ in Lemma 3.4 is defined with respect to the edges appearing in G.) Building on Lemma 3.4 we can now show that it is not beneficial to split a connected component into smaller clusters. Intuitively, this is the case as each vertex in a connected component CC has degree almost $|CC|$, while splitting CC into at least two clusters would force the smallest cluster (that has size at most $|CC|/2$) to cut too many "+" edges, while in CC it has relatively few "-" edges.

Lemma 3.5. *Let CC be a connected component in $\widetilde{G}$. Assume that $8\beta + \lambda \leq 1/4$. Then, the cost of keeping CC as a cluster in G is no larger than the cost of splitting CC into two or more clusters.*

Lemma 3.5 implies the following key insight (proved in the supplementary material).

Lemma 3.6. *Let G' be a* non-complete[1] *graph obtained from G by removing any "+" edge $\{u, v\}$ (i.e., changing it into a "neutral" edge) where u and v belong to different connected components of $\widetilde{G}$. Then, our algorithm outputs a solution that is optimal for the instance G'.*

3.1.2. Approximation Guarantee

In our analysis we will consider a fixed optimal solution (of instance G), denoted by $\mathcal{O}$, whose cost is denoted by OPT. Recall that our algorithm returns a clustering that is optimal for G' (Lemma 3.6). Therefore to bound the approximation ratio of our solution we need to bound the cost in G of an optimal clustering for G'. To do so, it is enough to bound the number of "+" edges in G that are absent from G' – and every such edge has been deleted by our algorithm. We have the following two lemmas. The main intuition behind their proofs, which can be found in the supplementary material, is that when two vertices are not in agreement, or when a vertex is light, then there are many edges (or non-edges) in the 1-hop or 2-hop vicinity that $\mathcal{O}$ pays for. We can charge the deleted edges to them.

Lemma 3.7. *The number of edges deleted in Line 1 of our algorithm that are not cut in $\mathcal{O}$ is at most $\frac{2}{\beta} \cdot \mathrm{OPT}$.*

Lemma 3.8. *The number of edges deleted in Line 3 of our algorithm that are not cut in $\mathcal{O}$ is at most $\left(\frac{1}{\beta} + \frac{1}{\lambda} + \frac{1}{\beta\lambda}\right) \cdot \mathrm{OPT}$.*

Lemmas 3.6, 3.7, and 3.8 together imply that Algorithm 1 is a constant-factor approximation:

Theorem 3.9. *Algorithm 1 is a constant-factor approximation.*

Proof. Let G' be the (non-complete) graph as defined in Lemma 3.6. Observe that the clusters that our algorithm outputs are exactly the connected components of G'. Let $D = E^+(G) \setminus E^+(G')$ be the set of edges in G that go between different connected components of G' (equivalently, of $\widetilde{G}$). Further, recall that $\mathcal{O}$ is a fixed optimal solution for instance G.

The main idea of our proof is to look at the costs of $\mathcal{O}$ and of our solution in the instance G', for which our solution is optimal. The cost of any solution differs between the two instances G and G' by at most $|D|$, which is at most the number of edges deleted by our algorithm. So, we can pay $|D|$ to move from G' to G. On the other hand, any solution is no more expensive in G' than it is in G. That is, for any solution X we have

$$\mathrm{cost}_{G'}(X) \leq \mathrm{cost}_G(X) \leq |D| + \mathrm{cost}_{G'}(X)\,.$$

Denote the solution returned by Algorithm 1 by OUR. Lemma 3.6 states that it is optimal for G'. That is, $\mathrm{cost}_{G'}(\mathrm{OUR}) \leq \mathrm{cost}_{G'}(\mathcal{O})$. Thus we have

$$\begin{aligned}\mathrm{cost}_G(\mathrm{OUR}) &\leq |D| + \mathrm{cost}_{G'}(\mathrm{OUR})\\ &\leq |D| + \mathrm{cost}_{G'}(\mathcal{O})\\ &\leq |D| + \mathrm{cost}_{G}(\mathcal{O})\\ &= |D| + \mathrm{OPT}\,.\end{aligned}$$

Finally, note that $|D|$ is at most the number of edges deleted by our algorithm (since any edge of G that goes

[1] We remark that everywhere else in the paper, correlation clustering instances are always complete graphs.

between different connected components of $\widetilde{G}$ must necessarily have been deleted by our algorithm). The latter can be upper-bounded, using Lemmas 3.7 and 3.8, by $\mathrm{OPT} + \frac{2}{\beta} \cdot \mathrm{OPT} + \left(\frac{1}{\beta} + \frac{1}{\lambda} + \frac{1}{\beta\lambda}\right) \cdot \mathrm{OPT}$. In total, we get a $\left(2 + \frac{3}{\beta} + \frac{1}{\lambda} + \frac{1}{\beta\lambda}\right)$-approximation. □

We note that in our analysis we do not optimize for a constant; nevertheless we now present a precise upper bound on the approximation ratio by providing a setting for the constants β and λ. We also note that despite the large theoretical approximation ratio, our algorithm works very well in practice.

Recall that Lemma 3.3 requires that $5\beta + 2\lambda < 1$, and Lemma 3.5 requires $8\beta + \lambda \leq \frac{1}{4}$, the latter condition being stronger. Also, Fact 3.2 requires $\beta < \frac{1}{20}$ (which is also implied by the above). Thus we can set, e.g., $\beta = \lambda = \frac{1}{36}$. Then the above proof of Theorem 3.9 gives a 1442-approximation guarantee. A more optimized setting of constants is $\beta \approx 0.0176$ and $\lambda \approx 0.1085$, which gives an approximation ratio ≈ 701.

Finally, the following is proved in the supplementary material:

Remark 3.10. *For fixed values of β and λ, the above analysis is tight, in the sense that the term $\frac{1}{\beta\lambda}$ is necessary.*

3.2. MPC Implementation of Algorithm 1

In this section we prove Theorem 1.1. The proof is divided into two parts: discussing the MPC implementation and proving the approximation ratio of the final algorithm. There are two main steps that we need to implement in the MPC model: for each edge $\{u, v\}$, we need to compute whether u and v are in an agreement (needed for Line 1); and to compute the connected components of $\widetilde{G}$ (Line 4). We separately describe how to implement these tasks. The approximation analysis is given in Section 3.2.3.

3.2.1. Computing Agreement

Let $e = \{u, v\}$ be an edge in G. To test whether u and v are in agreement, we need to compute how large $N(v) \triangle N(u)$ (or how large $N(u) \cap N(v)$) is (see Definition 3.1). However, it is not clear how to find $|N(v) \triangle N(u)|$ exactly for each edge $\{u, v\} \in E$ while using total memory of $\widetilde{\mathcal{O}}(|E|)$. So, instead, we will approximate $|N(v) \triangle N(u)|$ and use this approximation to decide whether u and v are in agreement. In particular, u and v will sample a small fraction of their neighbors, i.e., of size $O((\log n)/\beta)$, and then these samples will be used to approximate the similarity of their neighbourhoods. We now describe this procedure in more detail.

As the first step, we test whether $d(u)$ and $d(v)$ are within a factor $1 - \beta$. If they are not, then by Fact 3.2 (1) u and v are not in agreement and hence we immediately remove the edge $\{u, v\}$ from G. Next, each vertex v creates two vertex-samples. To do so, for each j smaller or equal than $O((\log n)/\beta)$ we define the set $S(j)$ as a subset of nodes obtained by sampling every node in the graph independently with probability $\min\left\{\frac{a \log n}{\beta \cdot j}, 1\right\}$, where a is a constant to be fixed later. Then we define $S(v, j)$ for every node v as $S(v, j) = S(j) \cap N(v)$ and j_v to be the largest power of $1/(1 - \beta)$ smaller or equal than $d(v)$. Then, each vertex v keeps $S(v, j_v)$ and $S(v, j_v/(1 - \beta))$. Note that by construction, for any two vertices v and u, we either have that $w \in S(v, j)$ and $w \in S(u, j)$, or $w \notin S(v, j)$ and $w \notin S(u, j)$. To implement this, each vertex w will independently in parallel flip a coin to decide whether for a given j it should be sampled or not.

Once we obtain the two samples, v sends the samples together with information about its degree to each of its incident edges.[2] After that, every edge $\{u, v\}$ holds: $S(v, j_v)$, $S(v, j_v/(1 - \beta))$, $S(u, j_u)$, and $S(u, j_u/(1 - \beta))$. Without loss of generality assume $d(u) \geq d(v)$. Since we have that $d(v)/(1 - \beta) \geq d(u) \geq d(v)$, then $j_v = j_u$ or $j_v/(1 - \beta) = j_u$. For the sake of brevity, let $j = j_u$. We now use $S(v, j)$ and $S(u, j)$ to estimate $|N(v) \triangle N(u)|$.

Define a random variable $X_{u,v}$ as

$$X_{u,v} \stackrel{\text{def}}{=} |S(v, j) \triangle S(u, j)|. \tag{1}$$

In case $\frac{a \cdot \log n}{\beta \cdot j} \geq 1$, we have $X_{u,v} = |N(v) \triangle N(u)|$, which means we directly get the *exact* value of $|N(v) \triangle N(u)|$. So assume that $\frac{a \cdot \log n}{\beta \cdot j} < 1$. By linearity of expectation we have

$$\mathbb{E}\left[X_{u,v}\right] = \frac{a \cdot \log n}{\beta \cdot j} |N(v) \triangle N(u)|.$$

Hence, if v and u are in agreement, we have

$$\mathbb{E}\left[X_{u,v}\right] \leq \frac{a \cdot \log n}{\beta \cdot j} \beta d(u) = \frac{a \cdot \log n}{j} d(u).$$

Based on this, our algorithm for deciding whether u and v are in agreement is given as Algorithm 2.

We now show that with high probability for every two vertices u and v: if the algorithm returns “Yes”, then u and v are in an agreement; and, if u and v are in 0.8-weak agreement, then the algorithm returns “Yes”.

Lemma 3.11. *For any constant $\delta > 0$, there exists an MPC algorithm that, given a signed graph $G = (V, E^+)$, in $O(1)$ rounds for all pairs of vertices $\{u, v\} \in E^+$ outputs “Yes”*

[2] We refer the reader to (Goodrich et al., 2011) and Section 6 of (Czumaj et al., 2019) for details on how to collect these samples on each edge in $O(1)$ MPC rounds.

Algorithm 2 Agreement(u, v)

```
if $d(u)$ and $d(v)$ are not within factor $1 - \beta$ then
    Return "No"
end if
Let $\tau \overset{\text{def}}{=} \frac{a \cdot \log n}{j} \cdot \max\{d(u), d(v)\}$
if $X_{u,v} \leq 0.9 \cdot \tau$ then
    Return "Yes"
end if
Return "No"
```

if u and v are in 0.8-weak agreement, and outputs "No" if u and v are not in agreement. Letting $n = |V|$, this algorithm succeeds with probability $1 - 1/n$, uses n^δ memory per machine, and uses a total memory of $\tilde{O}(|E^+|)$.

Our proof of Lemma 3.11, in which we set the value of a, is provided in the supplementary material.

3.2.2. Computing connected components

We now turn to explaining how to compute connected components in $\widetilde{G}$. Recall that, by Lemma 3.3, each connected component of $\widetilde{G}$ has diameter at most 4. We leverage this fact to design a simple algorithm that in $O(1)$ rounds marks each connected component with a unique id, as follows.

Algorithm 3 Connected-Components

```
Each vertex $v$ holds an $id_v^i$, $i = 0 \ldots 4$. Let $id_v^0 = v$.
for $i = 1 \ldots 4$ do
    For each $v$, we let $id_v^i = \max_{w \in N(v)} id_w^{i-1}$
end for
Return as a connected component all vertices $w$ that
    have the same $id_w^4$.
```

Let CC be a connected component of $\widetilde{G}$, and let $v^\star$ be the vertex of CC with the largest label (largest id^0). Correctness of Algorithm 3 follows by simply noting that at the end of iteration i all the vertices x at distance at most i from $v^\star$ will have $id_x^i = v^\star$. Since CC has diameter at most 4, it means all the vertices of CC will have the same id^4.

3.2.3. Approximation Analysis

Note that the approximation ratio is affected only by the fact that our algorithm now *estimates* agreement using Algorithm 2 as opposed to computing it exactly. That is, our MPC algorithm might return that two vertices are not in agreement while in fact they are. Nonetheless, it happens only for vertices which are *not* in 0.8-weak agreement, i.e., for vertices that are close to not being in agreement. This might only cause our algorithm to delete more edges; and the only part of our analysis that suffers from this are the approximation guarantees of Section 3.1.2. This can be easily fixed by replacing β by $0.8 \cdot \beta$. Then, Theorem 3.9 implies that using Algorithm 2 to test agreement between vertices still obtains an $O(1)$-approximation.

Now we are ready to prove our main Theorem.

Theorem 1.1. *For any constant $\delta > 0$, there exists an MPC algorithm that, given a signed graph $G = (V, E^+)$, where E^+ denotes the set of edges with weight +1, in $O(1)$ rounds computes a $O(1)$-approximate correlation clustering. Letting $n = |V|$, this algorithm succeeds with probability at least $1 - 1/n$ and requires $O(n^\delta)$ memory per machine. Moreover, the algorithm uses a total memory of $O(|E^+| \cdot \log n)$.*

Proof. The bounds on the round complexity and memory usage follow directly from the reasoning in Sections 3.2.1 and 3.2.2 and by noticing that step 2 (determining which vertices are light) can be easily implemented in $O(1)$ MPC rounds.

The approximation guarantees follow because even if we delete some additional edges from $\widetilde{G}$ that are in agreement but not in 0.8-weak agreement, we still obtain a constant-factor approximation as noted above. □

3.3. Semi-streaming Implementation

We now discuss how to implement our algorithm in the multi-pass semi-streaming setting, and effectively prove Theorem 1.2. In the classic streaming setting, edges of an input graph arrive one by one as a stream. For an n-vertex graph, an algorithm in this setting is allowed to use $O(\text{poly} \log n)$ memory. The semi-streaming setting is a relaxation of the streaming setting, in which an algorithm is allowed to use $O(n \,\text{poly} \log n)$ memory. We now describe how to implement each of our algorithms in the semi-streaming setting while making multiple passes over the stream. We remark that the order of edges presented in different passes can differ.

To implement Algorithm 2, we first fix $O(\log n)$ random bits for each vertex v and each relevant j (recall that there are $O((\log n)/\beta)$ such j values) needed to decide whether v belongs to $S(w, j_w)$, for some $w \in N(v)$.[3] This is the same as we did in Section 3.2.1. Next, we make a single pass over the stream and collect $S(v, j_v)$ and $S(v, j_v/(1 - \beta))$ for each v. After this, we are equipped with all we need to compute whether two endpoints of a given edge are in agreement or not.

Next, we make another pass and mark light vertices, where the notion of a light vertex is defined in Algorithm 1. Note that bookkeeping which vertices are light requires only $O(n)$ space.

[3] As a reminder, j_v is the largest power of $1/(1 - \beta)$ not greater than $d(v)$.

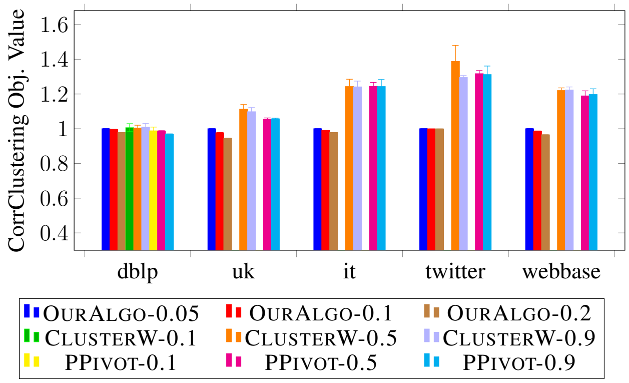

Figure 1. The correlation clustering objective values for the different algorithms and configurations that we consider. The objective value of all the algorithms is normalized by dividing by the objective value of OurAlgo0.05 for the respective dataset.

After these steps, in our memory we have (1) a mark for whether each vertex is light or not, and (2) a way to test whether two vertices are in agreement or not without the need to use any information from the stream. This implies that now, whenever an edge arrives on the stream, we can immediately decide whether it belongs to $\widetilde{G}$ or not. Hence, we have all the information needed to proceed to implementing Algorithm 3.

To implement Algorithm 3, we make 4 passes over the stream. In the i-th pass, for each edge $\{u, v\}$ on the stream that *belongs* to $\widetilde{G}$ we update $id_v^i = \max\{id_v^{i-1}, id_u^{i-1}\}$ and, similarly for u, $id_u^i = \max\{id_v^{i-1}, id_u^{i-1}\}$. Since $\widetilde{G}$ has diameter at most 4, this suffices to output the desired clusters of $\widetilde{G}$.

This concludes our implementation of the semi-streaming algorithm.

4. Empirical Evaluation

Graph	# vertices	# edges
dblp-2011	986,324	6,707,236
uk-2005	39,459,925	921,345,078
it-2004	41,291,594	1,135,718,909
twitter-2010	41,652,230	1,468,365,182
webbase-2001	118,142,155	1,019,903,190

Table 1. The datasets used in our experiments.

Datasets. To empirically analyze our algorithm compared to state-of-the-art parallel algorithms for correlation clustering, we considered a collection of two social networks and three web graphs. All our datasets were obtained from The Laboratory for Web Algorithmics[4] (Boldi & Vigna, 2004; Boldi et al., 2011; 2004), and some of their statistics are summarized in Table 1. The *dblp-2011* dataset is the DBLP co-authorship network from 2011, *uk-2005* is a 2005 crawl of the .uk domain, *it-2004* a 2004 crawl of the .it domain, *twitter-2010* a 2010 crawl of twitter, and *webbase-2001* is a 2001 crawl by the WebBase crawler. We converted all datasets to be undirected and removed parallel edges. The correlation clustering instance is formed by considering all present edges as "+" edges and all missing edges as "-" edges.

Algorithms and parameters. In our experiments we consider three algorithms: our algorithm from Section 3 (we refer to it as OurAlgo), as well as the ClusterWild (ClusterW, in short) algorithm from Pan et al. (2015) and the ParallelPivot (PPivot, in short) from Chierichetti et al. (2014). ClusterW and PPivot admit a parameter ϵ, which affects the number of parallel rounds required to perform the computation, depending on the structure of the input graph. For PPivot, ϵ also slightly affects the theoretical approximation guarantees (i.e., the approximation is $(3+\epsilon)$). We adopt the setting of ϵ from Pan et al. (2015), and use $\epsilon \in \{0.1, 0.5, 0.9\}$ for both algorithms. Our algorithm has two parameters λ, β which affect the approximation of the algorithm (see Lemma 3.7 and Lemma 3.8), but the number of rounds is independent of these parameters and is a fixed constant. For simplicity, we set $\lambda = \beta \in \{0.05, 0.1, 0.2\}$. To refer to an algorithm with a specific parameter, we append the parameter value to the algorithm name, e.g., we say OurAlgo-0.05.

Implementation details. In all our experiments the vertices are randomly partitioned among machines (we note that no algorithm requires a fixed partitioning of the input vertices onto machines). We made a fair effort to implement all algorithms equally well, and we did not use any tricks or special data structures. For simplicity, we assume that the entire neighborhood of each vertex fits on a single machine (this is not required by any algorithm). Removing this assumption would increase the number of rounds of all algorithms by a constant factor and most likely would not significantly affect their relative running times.

Setup and methodology. We used 10 machines across all experiments (except for Section 4.1); this is enough for the machines to collectively fit the input graph in memory. We repeated all experiments 3 times, and we report relative average running times (wall-clock time), as a ratio of each measurement compared to the minimum average running time observed across our experiments. We did not use a dedicated system for our experiments. Executions that were

[4] http://law.di.unimi.it/datasets.php

	OurAlgo			ClusterW			PPivot		
param. / Dataset	0.05	0.1	0.2	0.1	0.5	0.9	0.1	0.5	0.9
dblp	1.0**x**	1.1**x**	1.0**x**	244.7**x**	41.2**x**	18.8**x**	1083.6**x**	119.5**x**	42.7**x**
uk	5.5**x**	6.5**x**	10.7**x**	-	445.5**x**	213.1**x**	-	490.8**x**	217.4**x**
it	10.5**x**	14.8**x**	12.4**x**	-	475.7**x**	290.8**x**	-	762.8**x**	274.9**x**
twitter	8.8**x**	15.5**x**	13.9**x**	-	837.5**x**	300.2**x**	-	730.2**x**	392.8**x**
webbase	13.0**x**	13.5**x**	14.6**x**	-	835.1**x**	436.8**x**	-	789.3**x**	458.1**x**

Table 2. Average running times for the algorithms (with different parameters) that we consider. All times are reported relative to the execution time of OurAlgo-0.05 on the dataset dblp, which is approximately 21 seconds. We use 10 machines.

running for an unreasonable amount of time (more than 72 hours) were stopped, and we report no data for such executions; these occurred only for ClusterW-0.1 and PPivot-0.1. We excluded the time that it takes to load the input graph into the memory, as this is unavoidable and uniform across all algorithms.

Results on quality. Figure 1 summarizes the results of our experiments in terms of solution quality, that is, the correlation clustering objective value of the solution computed by the algorithms that we consider. OurAlgo consistently produces better solutions compared to the two competitor algorithms ClusterW and PPivot. In particular, for all datasets but dblp, ClusterW and PPivot produce solutions whose numbers of disagreements are more than 10% to 30% higher compared to the best solution produced by OurAlgo. For dblp, our OurAlgo is very comparable but slightly better than the baselines.

In terms of variance in the quality of the produced clustering between the different runs, OurAlgo has negligible variance, which is natural given that the only source of randomness comes from identifying pairs of vertices that are in agreement. On the other hand, the behavior of ClusterW and PPivot is not as stable, in terms of the quality of the produced solution, as demonstrated by the standard deviation illustrated in Figure 1.

Moreover, Figure 1 shows that the behavior of OurAlgo is not very sensitive to the choice of the parameters λ, β, as for all settings of these parameters OurAlgo produces solutions that are significantly better compared to the state-of-the-art parallel algorithms for correlation clustering. Recall that the parameter ϵ in ClusterW does not affect the solution quality, while it only slightly affects the theoretical guarantees of PPivot. In our experiments we did not observe any correlation between the choice of ϵ and the quality of the solution produced by PPivot.

Performance results. We summarize the average running times of the different algorithms in Table 2. For each algorithm, we report the ratio of its average running time to the average running time of OurAlgo-0.05 on the dblp dataset, which is the fastest average running time we observed throughout our experiments, equal to roughly 21 seconds. It is evident that OurAlgo (independently of its parameters) is consistently over an order of magnitude faster compared to the state-of-the-art parallel algorithms ClusterW and PPivot, and in several cases the gap increases to two orders of magnitude.

While the choice of λ, β in OurAlgo has no effect on the number of rounds performed by OurAlgo, one can observe some deviations between the different parameter choices, which is likely due to time-specific system workload. Nonetheless, for each algorithm its maximum running time across all runs is within a factor at most 2 of its average running time. While the same can be said for ClusterW and PPivot, throughout our experiments we did not observe any case where an execution of either of ClusterW or PPivot performed within a factor 10 of any execution of OurAlgo, even for the smallest instance dblp, where the running times are expected to be the closest. On the other hand, the choice for the parameter ϵ affects the running time of ClusterW and PPivot and requires proper tuning depending on the structure of the input graph (in our graphs, the choice of $\epsilon = 0.9$ always results in significantly faster performance compared to other choices). The executions of ClusterW and PPivot with $\varepsilon = 0.1$, on all datasets except dblp, were stopped as they did not terminate within a reasonable amount of time, and thus are not reported. In the supplementary material we also provide additional analysis on the scalability of our algorithm, confirming these findings.

Furthermore, in the supplementary material we present various statistics regarding the performance and solutions produced by the algorithms, e.g., the number of MPC rounds and the distribution of cluster sizes.

4.1. Speedup Evaluation

In this section we study the parallelism of OurAlgo. We use a fixed parameter $\lambda = \beta = 0.05$, as the choice of this parameter does not significantly affect the running time of the algorithm; indeed, when repeating the experiments for different parameter settings, we observed a very similar picture to the one we report below. To measure speed-up,

#machines Dataset	1	2	4	8	16
it	1**x** (±0.303)	2.392**x** (±0.032)	3.114**x** (±0.119)	4.823**x** (±0.266)	5.445**x** (±0.790)
twitter	1**x** (±0.149)	4.451**x** (±0.0151)	4.968**x** (±0.0803)	7.270**x** (±0.138)	5.479**x** (±0.338)
webbase	1**x** (±0.0618)	5.280**x** (±0.225)	4.441**x** (±0.166)	12.161**x** (±0.0110)	11.306**x** (±0.047)

Table 3. Average speedup achieved by OURALGO-0.05, for an increasing number of machines. The standard deviation of the running time, as a fraction of the running time, is presented in parentheses.

we start from 1 machine and we double the number of machines at each step, that is, we consider 1, 2, 4, 8, and 16 machines. Each reported running time is the average time of three repetitions of the algorithm, presented relative to the average running time of OURALGO-0.05 with 1 machine. Our results are summarized in Table 3.

Across all datasets, we observe a trend of near-linear speedup as the number of machines grows from 1 to 8. There is no significant speedup in the transition from 8 to 16 (in fact, in two out of the three cases we see worse running times when using 16 machines), and this is likely because we reach a tipping point where the cost of communication between the machines is higher compared to the benefit gained by parallelism, for the specific datasets that we consider. Moreover, the speedup achieved across the three datasets is not uniform, and this is due to the fact that 1 machine might be more appropriate for some datasets but not enough for other datasets; indeed, the highest speedup is achieved for the webbase dataset, which the largest among the graphs that we consider.

Although we observe small inconsistencies in the overall picture of our experiment, which is due to high variance in the observed running times (recall that we do not use a dedicated system for our experiments), one can observe a clear trend highlighting a near-linear speedup as the number of machines increases.

Conclusions and Future Work

We present a new parallel algorithm for correlation clustering and we prove both theoretically and experimentally that our algorithm is extremely fast and returns high-quality solutions. Interesting open problems are to improve the approximation guarantees of our algorithm and to establish a more formal connection between our results and well-known similar heuristics (Xu et al., 2007). Another direction would be to design an MPC algorithm in the sublinear regime for the *weighted* version of the problem.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. S. Mitrović was supported by the Swiss NSF grant No. P400P2_191122/1, MIT-IBM Watson AI Lab and research collaboration agreement No. W1771646, NSF award CCF-1733808, and FinTech@CSAIL.

References

Agrawal, R., Halverson, A., Kenthapadi, K., Mishra, N., and Tsaparas, P. Generating labels from clicks. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pp. 172–181, 2009.

Ahn, K., Cormode, G., Guha, S., McGregor, A., and Wirth, A. Correlation clustering in data streams. In *International Conference on Machine Learning*, pp. 2237–2246. PMLR, 2015.

Ailon, N., Charikar, M., and Newman, A. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):1–27, 2008.

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

Bansal, N., Blum, A., and Chawla, S. Correlation clustering. *Machine learning*, 56(1):89–113, 2004.

Beame, P., Koutris, P., and Suciu, D. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pp. 273–284. ACM, 2013.

Boldi, P. and Vigna, S. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pp. 595–601, Manhattan, USA, 2004. ACM Press.

Boldi, P., Codenotti, B., Santini, M., and Vigna, S. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

Boldi, P., Rosa, M., Santini, M., and Vigna, S. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M. P., Bertino, E., and Kumar, R. (eds.), *Proceedings of the 20th international conference on World Wide Web*, pp. 587–596. ACM Press, 2011.

Bonchi, F., Gionis, A., and Ukkonen, A. Overlapping correlation clustering. *Knowledge and information systems*, 35(1):1–32, 2013.

Chakrabarti, D., Kumar, R., and Punera, K. A graph-theoretic approach to webpage segmentation. In *Proceedings of the 17th international conference on World Wide Web*, pp. 377–386, 2008.

Charikar, M., Guruswami, V., and Wirth, A. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.

Chawla, S., Makarychev, K., Schramm, T., and Yaroslavtsev, G. Near optimal lp rounding algorithm for correlation-clustering on complete and complete k-partite graphs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 219–228, 2015.

Chen, Y., Sanghavi, S., and Xu, H. Clustering sparse graphs. In *Proceedings of the 25th International Conference on Neural Information Processing Systems-Volume 2*, pp. 2204–2212, 2012.

Chierichetti, F., Dalvi, N., and Kumar, R. Correlation clustering in mapreduce. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 641–650, 2014.

Czumaj, A., Łacki, J., Madry, A., Mitrovic, S., Onak, K., and Sankowski, P. Round compression for parallel matching algorithms. *SIAM Journal on Computing*, 49(5): STOC18–1, 2019.

Dean, J. and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Demaine, E. D., Emanuel, D., Fiat, A., and Immorlica, N. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006.

Giotis, I. and Guruswami, V. Correlation clustering with a fixed number of clusters. *arXiv preprint cs/0504023*, 2005.

Goodrich, M. T., Sitchinava, N., and Zhang, Q. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pp. 374–383. Springer, 2011.

Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41, pp. 59–72. ACM, 2007.

Kalashnikov, D. V., Chen, Z., Mehrotra, S., and Nuray-Turan, R. Web people search via connection analysis. *IEEE Transactions on Knowledge and Data Engineering*, 20(11):1550–1565, 2008.

Karloff, H., Suri, S., and Vassilvitskii, S. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pp. 938–948. SIAM, 2010.

Pan, X., Papailiopoulos, D. S., Oymak, S., Recht, B., Ramchandran, K., and Jordan, M. I. Parallel correlation clustering on big graphs. In *NIPS*, 2015.

Swamy, C. Correlation clustering: maximizing agreements via semidefinite programming. In *SODA*, volume 4, pp. 526–527. Citeseer, 2004.

White, T. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

Xu, X., Yuruk, N., Feng, Z., and Schweiger, T. A. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 824–833, 2007.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.