Correlation Clustering in MapReduce

Flavio Chierichetti*
Sapienza University
Rome, Italy
flavio@di.uniroma1.it

Nilesh Dalvi†
Trooly, Inc.
Mountain View, CA
nileshdalvi@gmail.com

Ravi Kumar
Google, Inc.
Mountain View, CA
ravi.k53@gmail.com

ABSTRACT

Correlation clustering is a basic primitive in data miner's toolkit with applications ranging from entity matching to social network analysis. The goal in correlation clustering is, given a graph with signed edges, partition the nodes into clusters to minimize the number of disagreements. In this paper we obtain a new algorithm for correlation clustering. Our algorithm is easily implementable in computational models such as MapReduce and streaming, and runs in a small number of rounds. In addition, we show that our algorithm obtains an almost 3-approximation to the optimal correlation clustering. Experiments on huge graphs demonstrate the scalability of our algorithm and its applicability to data mining problems.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications - *Data Mining*

Keywords: Scalable clustering; Signed networks.

1. INTRODUCTION

Correlation clustering is a basic primitive underlying several data management and data mining tasks such as deduplication [5,19,24], identifying communities [15] and network link prediction [12].

The correlation clustering problem is the following: given an undirected signed graph where each edge is labeled either positive or negative, partition the nodes into clusters so that the total number of disagreements is minimized. Here, a disagreement is said to happen if a positive edge becomes an inter-cluster edge or a negative edge becomes an intra-cluster edge. Note that the problem definition does not contain the number of clusters; this is one of the main attractions of correlation clustering, since in many applications it is not possible to specify the number of clusters a priori. Correlation clustering was first introduced by Bansal, Blum, and Chawla [9]. They showed the problem is NP-hard in general and presented a constant-factor approximation for complete graphs. The factor was later improved to three by Ailon, Charikar, and Newman [3]; Gionis, Mannila, and Tsaparas [22] obtained a deterministic three-approximation algorithm for weighted graphs obeying the triangle inequality.

Even though correlation clustering is defined for signed graphs, such graphs arise in a variety of contexts, some obvious and some less obvious. A sign (positive or negative) on an edge in graph can connote different semantics depending on the underlying setting. In case of social networks and trust/opinion networks, a positive sign denotes endorsement whereas a negative sign denotes disapproval; signed networks in social media have been extensively studied recently [29, 30]. If the nodes of the graph are items that are possibly duplicated, a positive edge can signify that the items might be the same and negative edges could mean they are not the same. If the nodes of the graph are vectors, the sign could correspond to the angle between the vectors, denoting how aligned or different they are to each other.

Correlation clustering has been used in a wide range of data mining applications. We outline a few here; see Section 2 for more.

(i) In the entity deduplication problem, the goal is to remove entities that are duplicates of one another. Such duplicates can arise if the entities were obtained through independent sources/feeds. A positive edge between two entities indicates that the entities could be the same and a negative edge indicates that they could be different. Correlation clustering has been used in this setting to cluster this graph and produce entities that are duplicates [5, 19, 24].

(ii) In signed social networks, where positive edge denote friendship and negative edges denote enmity, correlation clustering is a natural way to identify communities [15]. It has also been used as a tool for the link classification problem, i.e., to predict the sign of a missing edge [12], with formal interpretations to the prediction complexity of link classification in a supervised transductive learning framework.

(iii) In machine learning and data mining, the problem of aggregating multiple clusterings and the problem of consensus clustering can be viewed as special cases of the correlation clustering problem [22]; here, the edge weights correspond to the fraction of clusters separating the two nodes.

In many of these applications, the signed graphs exhibit the following two key properties: the positive degree of each node is typically *bounded* and the graph is *complete*, with the understanding that the missing edges are to be interpreted as negative. The randomized algorithm of Ailon et al. [3], which obtains a three-approximation for such graphs, will be the starting point of our work. Their algorithm is tantalizingly simple: pick a *pivot* node uniformly at random, remove it and all its positive neighbors as a cluster, repeat. Despite its simplicity, it is clearly inefficient on massive graphs since it is inherently sequential, possibly removing only a few nodes in each round of processing the graph.

*Part of this work was done while the author was visiting Google. This work was partially supported by a Google Focused Research Award, and by a Google Faculty Award.
†This work was done while the author was at Facebook.

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623743.

Our results. In this work we obtain an efficient scalable algorithm for the correlation clustering problem. In particular, we show how to make the approximation algorithm of [3] round-efficient by picking multiple pivots in *parallel* and repeating. While this idea is natural, we show that one has to be careful in how the pivots must be picked in parallel; a less judicious choice can lead to poor quality solutions. We show two key features of our parallel pivot algorithm. (i) It runs in a logarithmic number of rounds for graphs with a constant number of positive neighbors; for arbitrary graphs, the number of rounds gets further multiplied by the logarithm of the maximum positive degree. (ii) For complete graphs, it outputs a solution that is a close to three-approximation to the optimal correlation clustering solution. Our algorithm is extremely easy to implement in MapReduce, streaming, and message-passing models.

Our proof for the running time is a delicate charging argument that tracks the node with large positive degrees and shows that each of them either disappears or has its degree halved after a certain number of rounds. Our proof for the approximation guarantee proceeds by getting a lower bound on the optimal by considering the dual of a linear program. In contrast, for general graphs, we show that the bounded positive degree (or, for that matter, a bounded degree) assumption does not buy much algorithmic benefit: the problem is as hard as the unbounded degree case.

We then implement our algorithm in a real MapReduce system. We run our algorithm on a graph corresponding to an entity deduplication instance; this graph consists of around 30M nodes and close to a billion edges. The ability to run our algorithm on such a large instance shows its scalability and applicability. We also show that our algorithm achieves performance very close to that of [3], despite being much efficient in terms of its round complexity. We also implement our algorithm in the streaming model and demonstrate its ability to handle graphs with more than 2.5B edges, but only using a very limited amount of main memory.

2. RELATED WORK

The related work falls into three categories: the extensive theoretical work on correlation clustering, the applications of correlation clustering to web mining and machine learning problems, and the theme of efficient, scalable algorithms for web-scale mining.

As discussed earlier, there has been a lot of work on obtaining good approximation algorithms for correlation clustering on complete graphs [3, 9, 22]. The algorithm of Ailon et al. (known as Pivot) will be the starting point for our work. For general graphs, Demaine et al. [18] presented an $O(\log n)$-approximation algorithm; they also showed a logarithmic integrality gap for a natural linear program for the problem. There have been some work on considering natural extensions of the vanilla correlation clustering problem; for example, bipartite correlation clustering was studied by Ailon et al. [2], overlapping correlation clustering was studied by Bonchi, Gionis, and Ukkonen [11], chromatic correlation clustering was studied by Bonchi et al. [10], and subspace correlation clustering was studied by Günnemann et al. [23]. To the best of our knowledge, there has been no result on scalable algorithms with provable bounds for correlation clustering.

Correlation clustering is such a basic primitive that it finds a multitude of applications in web mining applications; in many of these situations, the freedom to not specify the number of clusters to a clustering algorithm is often desirable. Correlation clustering has been extensively used in the general problem of deduplication. For example, a modification of correlation clustering (by including hard constraints) was considered [5] to perform large-scale deduplication of entity references; for an extensive survey on this topic, see [19, 24]. Further applications include disambiguation in people search [25], co-reference resolution [34], and many NLP applications (see the references in [20]). In web search context, correlation clustering has been used to cluster query refinements in web search [36], automatically label query-URL pairs with human-like judgments [1], and segment Web pages [13]. In social network setting, it has been used for link classification in signed networks such as trust and opinion networks [12] and for clustering sparse graphs that arise in social networks [15]. Correlation clustering has also been applied in learning settings such as support vector machines [21] and cluster aggregation [11]. See the surveys [27, 38].

MapReduce algorithms have been developed for basic graph problems such as minimum spanning trees [26], triangle counting [37], and matching [28, 35]. In the combinatorial optimization setting, MapReduce algorithms for the problems of maximum coverage [16], densest subgraph [7], and k-means clustering [8] were obtained recently. All these algorithms have the flavor that they show how to parallelize an *inherently* sequential algorithm, but still almost preserving the original guarantees. Our work is yet another instance of this theme, though the actual techniques we use are vastly different from that of the above. The scalability of correlation clustering algorithms has been addressed before [5, 6, 20] but none of these proposes provably scalable and provably good algorithms.

3. BACKGROUND

In this section we set up the basic notation and problem definitions. Let $G = (V, E)$ be an undirected graph, with $|V| = n$. Each edge $e \in E$ is labeled either *positive* or *negative*; let E^+ be the set of all positive edges and let $E^- = E \setminus E^+$ be the set of all negative edges. Let $\Gamma(v) = \{w \mid \{v, w\} \in E\}$ be the *neighbors* of $v \in V$, let $\deg(v) = |\Gamma(v)|$ be its *degree*, and let $\Delta = \max_{v \in V} \deg(v)$ be the maximum degree in G. Similar concepts are defined with respect to the labels: for example, let $\Gamma^+(v) = \{w \mid \{v, w\} \in E^+\}$, $\deg^+(v) = |\Gamma^+(v)|$, and $\Delta^+ = \max_{v \in V} \deg^+(v)$. If G is *weighted*, let $w^+ : E \to [0, 1]$ and $w^- : E \to [0, 1]$ be the edge weight functions; here by weighted we mean that each edge has a positive weight and a negative weight and they sum to 1.

DEFINITION 3.1 (CORRELATION CLUSTERING). *Given a labeled graph $G = (V, E)$, find a partition of V that minimizes the sum of positive edges across two clusters and the sum of negative edges inside the clusters.*

In other words, the goal of correlation clustering is to minimize the total number of disagreements. Note that the number of clusters is not specified as part of the objective; this is one of the major attractions of correlation clustering. Also, the above definition naturally extends to weighted graphs, where the weight of an edge is used in charging for the disagreements. As we mentioned in Section 1, we will be stating our results in terms of Δ^+ (i.e, the number of positive edges incident on every node); we will be interested in very efficient algorithms when Δ^+ is bounded.

Correlation clustering is an NP-hard problem [9] and hence there has been work on developing approximation algorithms for it (see Section 2). A clustering is said to be an *α-approximation* if its cost is at most α times the cost of the optimal correlation clustering. (Note that the number of clusters produced by an approximation algorithm need not be the same as the number of optimal clusters.) We focus on an elegant algorithm of Ailon, Charikar, and Newman [3], henceforth referred to as the Pivot algorithm.

The basic idea in Pivot is simple: pick a node uniformly at random; designate all the nodes connected to it by positive edges as a cluster; remove this cluster; and repeat. For future convenience, we state this algorithm in the following manner. The subroutine

Algorithm 1 CreateCluster(V, E^+, v)

1: $C \leftarrow \{v\} \cup \Gamma^+(v)$ *// positive neighbors*
2: $V \leftarrow V \setminus C$
3: $E^+ \leftarrow E^+ \cap \binom{V}{2}$
4: **output** C as a cluster
5: **return** (V, E^+)

CreateCluster implements the task of creating a cluster and updating the graph, once a pivot is chosen. Given this, the Pivot algorithm follows easily. An analysis of this algorithm shows that it

Algorithm 2 Pivot$(G = (V, E^+))$

1: **while** $V \neq \emptyset$ **do**
2: $v \leftarrow$ uniform at random node in V
3: $(V, E^+) \leftarrow$ CreateCluster(V, E^+, v)

obtains a three-approximation when G is a complete graph [3].

Given the simplicity of Pivot, it is somewhat tempting to use it in large-scale applications. A careful scrutiny however indicates that the algorithm makes expensive multiple passes over the input, i.e., every time a new cluster is output. In fact, this behavior can be pathological: for example, if the graph has only negative edges or if the graph is a line, then Pivot makes a linear number of passes. This clearly limits its applicability on massive real-world graphs.

4. ALGORITHMS

In this section we present a fast, scalable algorithm for correlation clustering. This algorithm guarantees a constant-factor approximation when the graph is complete and runs in polylogarithmically many rounds. For general graphs, we show that the problem is as hard to approximate even in the bounded-degree case.

A natural way to obtain a scalable parallel algorithm for correlation clustering would be to parallelize Pivot by picking many pivots in parallel and grow clusters around them (using CreateCluster) and hope that the graph shrinks significantly in each round. To proceed in this direction, we need to stipulate the distribution with which multiple pivots should be picked. One has to be careful in choosing a distribution: we illustrate two plausible approaches and indicate their pitfalls on complete graphs.

Suppose multiple pivots are chosen with probability proportional to their degrees (this is desirable since it will also shrink the graph faster). One can show that with high probablity this algorithm produces a non-constant approximation on a graph composed of $\ln n$ stars of positive edges having $n/\ln n$ nodes each, and such that each star center pair is joined by a negative edge.

On the other hand, suppose multiple pivots are chosen inversely proportional to their degrees (this approach was used by Luby [32] to obtain a parallel algorithm for the maximal independent set problem). In our setting, however, one can verify the following: for a positive-edge clique of $n/2$ nodes, where each node in the clique is connected with a positive edge to a new node and where the rest of the edges are negative, this algorithm would produce a non-constant approximation with high probability. In general, even if a particular way of choosing multiple pivots looks appealing, it is unclear how to preserve the approximation properties of Pivot.

4.1 A parallel pivot algorithm

We present our algorithm called ParallelPivot (see Algorithm 3). As stated before, the idea behind the algorithm is to choose pivots in parallel and grow clusters around them. The pivots are chosen in a delicate manner in a two-step process: in the first step, each node is sampled with probability inversely proportional to the current maximum positive degree; this results in a set of *active* nodes. In the second step, active nodes that are adjacent to each other (line 6) are deactivated; the remaining active nodes are the *pivots* (line 7). If a non-pivot node is adjacent to more than one pivot, then it is assigned to the smallest of them (lines 10–11) according to a uniform random permutation of the nodes (π in the algorithm). Then, as in Pivot, each pivot grows its own cluster (in parallel) using its positive neighbors (line 13). In the next section, we show two properties of this algorithm: (i) unlike Pivot, it runs in logarithmically many rounds and (ii) like Pivot, it still outputs a constant-factor approximation for complete graphs.

Algorithm 3 ParallelPivot$(G = (V, E^+), \epsilon)$

1: $\pi \leftarrow$ a random permutation of V
2: **while** $E^+ \neq \emptyset$ **do**
3: $\Delta^+ \leftarrow \max_{v \in V} \deg^+(v)$ *// current max +ve degree*
4: **for** $v \in V$ **do**
5: $A \leftarrow A \cup \{v\}$ with prob. $p = \epsilon/\Delta^+$ *// active node*
6: $A' \leftarrow A \cap \bigcup_{v \in A} \Gamma^+(v)$ *// adjacent active nodes*
7: $P \leftarrow A \setminus A'$ *// pivots*
8: $B \leftarrow \{v \mid v \in V \setminus P \wedge |\Gamma^+(v) \cap P| \geq 2\}$
// non-pivots with multiple adjacent pivots
9: **for** $v \in B$ **do**
10: $w \leftarrow \arg\min_{w' \in \Gamma^+(v) \cap P} \pi(w')$ *// pick one using π*
11: $E^+ \leftarrow E^+ \setminus \{\{v, w'\} \mid w' \in \Gamma^+(v) \cap (P \setminus \{w\})\}$
// remove positive edges to others
12: **for** $p \in P$ **do**
13: $(V, E^+) \leftarrow$ CreateCluster(V, E^+, p)
14: **for** $v \in V$ **do**
15: **output** $\{v\}$ as a cluster

4.2 Analysis

Number of rounds. We first prove a crucial point about the progress made by the algorithm in each round. A natural approach to track the progress of the algorithm is by lower bounding the number of nodes or edges removed in each round. Unfortunately, such a lower bound would be too weak to show that ParallelPivot finishes in a small number of rounds; we therefore resort to a subtler argument. We track the maximum positive degree of the graph, and show that after logarithmically many rounds of the algorithm, nodes with large positive degree either disappear, or their positive degree shrinks by a constant factor.

LEMMA 1. *Suppose that $\epsilon < \frac{1}{2}$. Fix a constant $c > 0$ and let $\Delta^+ \geq 1$ be the maximum positive degree before beginning a round of the loop at line 2. Then, for any node v, after $\lceil \frac{8c}{\epsilon} \ln n \rceil$ rounds, with probability at least $1 - n^{-c}$, either v will have been removed from the graph or it will have degree at most $\Delta^+/2$.*

PROOF. Consider the graph at the beginning of an arbitrary round of the algorithm and let $\Delta^+, \deg^+(\cdot), \Gamma^+(\cdot)$ be defined with respect to this graph. The probability that a node is activated in this round is $p = \epsilon/\Delta^+$. Let v be an arbitrary node and consider the event $\xi_v =$ "in this round of the loop exactly one neighbor w of v gets activated and no neighbor of this neighbor w gets activated."

Then,

$$\Pr[\xi_v] = \sum_{w\in\Gamma(v)} \left(p\cdot(1-p)^{|\Gamma^+(w)\cup\Gamma^+(v)\setminus\{w\}|}\right)$$
$$\geq \sum_{w\in\Gamma(v)} \left(p\cdot(1-p)^{\deg^+(v)}\cdot(1-p)^{\deg^+(w)}\right).$$

Now, observe that since $\Delta^+ \geq \max\left(\deg^+(v),\deg^+(w)\right)$, we have $p\cdot\max\left(\deg^+(v),\deg^+(w)\right)\leq\epsilon$. Thus,

$$\deg^+(v)+\deg^+(w)\leq\frac{2\epsilon}{p},$$

and

$$\Pr[\xi_v]\geq\sum_{w\in\Gamma(v)}\left(p\cdot(1-p)^{\frac{2\epsilon}{p}}\right).$$

By calculus, it holds that $(1-x)^{\frac{1}{x}}\geq\frac{1}{4}$, for each $x\in\left(0,\frac{1}{2}\right]$. Since $\epsilon\leq\frac{1}{2}$, we have $p\leq\frac{1}{2}$ and therefore

$$\left((1-p)^{\frac{1}{p}}\right)^{2\epsilon}\geq\left(\frac{1}{4}\right)^{2\epsilon}\geq\frac{1}{4}.$$

We thus obtain,

$$\Pr[\xi_v]\geq\frac{p\cdot\deg^+(v)}{4}.$$

If $\deg^+(v)>\frac{\Delta^+}{2}$, then we have

$$p\cdot\deg^+(v)>\frac{\epsilon}{2},$$

and hence $\Pr[\xi_v]>\epsilon/8$.

Now consider round i and let Δ_i^+ be the maximum positive degree before this round. Consider any node v that had positive degree at least $\Delta_i^+/2$ in round i, and consider any subsequent round $j>i$. If the positive degree of v in round j is at most $\Delta_i^+/2$, then v will have shrunk its positive degree (with respect to round i) by a factor of at least 2. Otherwise, since the maximum positive degree is non-increasing during the execution of the algorithm, we will have that $\Pr[\xi_v]>\epsilon/8$ even at round j. Therefore, after $\left\lceil\frac{8c}{\epsilon}\ln n\right\rceil$ rounds, either v's positive degree shrinks by a factor of at least 2 or the probability that v will end up in a cluster is at least

$$1-\left(1-\frac{\epsilon}{8}\right)^{\frac{8c}{\epsilon}\ln n}\geq 1-n^{-c}.\quad\square$$

From this, we can bound the number of passes the algorithm makes on the input graph.

COROLLARY 2. *Algorithm 3 terminates after at most*

$$O\left(\frac{1}{\epsilon}\log n\cdot\log\Delta^+\right)$$

rounds of its loop, with probability at least $1-1/n^c$, for any constant $c>0$.

Approximation factor. We now proceed to prove the approximation guarantee when G is a complete graph. We start by recalling a definition from [3]. Consider the original graph $G=(V,E)$, and call a set $T\subseteq V$ of three nodes a *bad triangle* if it induces two edges in E^+ and one edge in E^-. Let $\mathcal{T}\subseteq\binom{V}{3}$ be the set of bad triangles in the original graph $G=(V,E)$. Let $S_{v,v'}=\bigcup_{\{v,v'\}\subset T\in\mathcal{T}}T$ be the union of the sets of nodes of the bad triangles that contain both v and v'. We now show that the distribution with which nodes as picked as pivots by ParallelPivot is close to the distribution induced by Pivot.

LEMMA 3. *Suppose that $\epsilon<\frac{1}{7}$. Let $v',v''\in V$ be two nodes in the instance that are contained in at least one bad triangle and let P be the set of pivots chosen in a round of Algorithm 3 at the beginning of which no node in $S_{v',v''}$ was part of a cluster. Then,*

$$\Pr\left[\{v',v''\}\cap P\neq\emptyset\mid S_{v',v''}\cap P\neq\emptyset\right]\leq\frac{2}{|S_{v',v''}|\cdot(1-7\epsilon)}.$$

PROOF. For simplicity, let $S=S_{v',v''}$. Clearly, $|S|\geq 3$. Note that before a pivot is chosen in S, the maximum of the degree of v' and the degree of v'' in the graph induced by S will be at least $\frac{|S|-2}{2}$. Therefore, before a pivot is chosen in S, we have $\Delta^+\geq\frac{|S|-2}{2}\geq\frac{|S|}{6}\implies|S|\leq 6\Delta^+$.

We compute the probability that an arbitrary node $v\in S$ becomes a pivot, given that $S\cap P\neq\emptyset$. By Bayes' rule,

$$\Pr[v\in P\mid S\cap P\neq\emptyset]=\frac{\Pr[v\in P\wedge S\cap P\neq\emptyset]}{\Pr[S\cap P\neq\emptyset]}.$$

Observe that since $v\in S$, the event "$v\in P\wedge S\cap P\neq\emptyset$" is equal to the event "$v\in P$". Moreover, the event "$v\in P$" is a subset of the event "v has been activated"; the latter event has probability p. Therefore,

$$\Pr[v\in P\mid S\cap P\neq\emptyset]\leq\frac{p}{\Pr[S\cap P\neq\emptyset]}.$$

Now, consider the probability of the event "$S\cap P\neq\emptyset$"; it is not smaller than the probability that exactly one node $v\in S$ gets activated and that none of its neighbors outside of S gets activated. Since Δ^+ is an upper bound on the maximum positive degree,

$$\begin{aligned}\Pr[S\cap P\neq\emptyset] &\geq |S|\cdot p\cdot(1-p)^{|S|+\Delta^+}\\ &\geq |S|\cdot p\cdot(1-p)^{7\Delta^+}.\end{aligned}$$

Recall that $(1-x)^y\geq 1-xy$ for each $0<x<1, y\geq 1$. Hence,

$$\Pr[S\cap P\neq\emptyset]\geq|S|\cdot p\cdot\left(1-7\Delta^+p\right)=(1-7\epsilon)\cdot|S|\cdot p.$$

Therefore for any node $v\in S$, we have

$$\Pr[v\in P\mid S\cap P\neq\emptyset]\leq\frac{1}{|S|\cdot(1-7\epsilon)}.$$

The main claim can be then proved by a union bound on the two events given by $v=v'$ and $v=v''$. $\square$

We now prove our main algorithmic result. Our proof follows the one in [3], adapting its single-pivot choice charging argument to our parallel multiple-pivot process.

THEOREM 4. *Suppose that $\epsilon<\frac{1}{7}$. Then, Algorithm 3 returns a $\left(1+\frac{2}{1-7\epsilon}\right)$-approximate correlation clustering.*

PROOF. Observe that the total cost of the clustering $\mathcal{C}$ produced by Algorithm 3 can be upper bounded by the number of bad triangles that will be removed from the graph because of the selection of one of their nodes as a pivot. For $T\in\mathcal{T}$, let p_T be the probability that at least one of its nodes is chosen as a pivot while T is a completely part of the graph. Then, the expected cost incurred by the algorithm can be upper bounded by

$$E[\mathcal{C}]\leq\sum_{T\in\mathcal{T}}p_T. \tag{1}$$

For lower bounding the cost of the optimum, we consider the following linear program (LP) relaxation of correlation clustering.

$$\begin{cases}\text{minimize } \sum_{\{v,v'\}\in\binom{V}{2}}x_{v,v'}\\ \text{subject to}\\ x_{v,v'}+x_{v,v''}+x_{v',v''}\geq 1 & \forall\{v,v',v''\}\in\mathcal{T},\\ 0\leq x_{v,v'} & \forall\{v,v'\}\in\binom{V}{2}.\end{cases}$$

Note that in the integral solution, $x_{v,v'} = 1$ if the edge between v and v' is not satisfied, and 0 otherwise. The dual of the above program can be written as follows.

$$\begin{cases} \text{maximize} \sum_{T \in \mathcal{T}} y_T \\ \text{subject to} \\ \sum_{T \in \mathcal{T} \wedge T \supset \{v,v'\}} y_T \leq 1 & \forall \{v, v'\} \in \binom{V}{2}, \\ 0 \leq y_T & \forall T \in \mathcal{T}. \end{cases}$$

The value of any feasible solution of this dual is a lower bound on the optimal value of the LP, which in turn is a lower bound on the value of the optimal correlation clustering. We will show that

$$y_T = \frac{p_T}{1 + \frac{2}{1-7\epsilon}}, \tag{2}$$

is a feasible solution to the dual. Then, (1) and (2) will prove that the algorithm returns a $1 + \frac{2}{1-7\epsilon}$-approximation in expectation.

To show (2), consider any edge $\{v, v'\} \in E$ and for simplicity, let $S = S_{v,v'}$. Recall that a bad triangle will increment the total cost of the algorithm's solution if and only if at least one of its nodes is chosen as a pivot before the triangle disappears. The expected total cost of the $|S| - 2$ bad triangles containing $\{v, v'\}$ can then be upper bounded in the following manner. If any node $v'' \in S \setminus \{v, v'\}$ is chosen before any of $\{v, v'\}$, then a cost of 1 (i.e., the cost of the bad triangle $\{v, v', v''\}$) is incurred and all the bad triangles containing $\{v, v'\}$ will be eliminated; the probability of this event is at most 1. If, on the other hand, at least one of v and v' is chosen as a pivot before any other node in S, then the cost will be $|S| - 2$ (i.e., a cost 1 for each of the $|S| - 2$ triangles). By Lemma 3, the probability of this event is at most $\frac{2}{|S|(1-7\epsilon)}$. Thus, the expected total cost of the $|S| - 2$ bad triangles containing $\{v, v'\}$ is at most

$$\sum_{T \supset \{v,v'\}} p_T \leq 1 \cdot 1 + (|S| - 2) \cdot \frac{2}{|S|(1-7\epsilon)} \leq 1 + \frac{2}{1-7\epsilon},$$

since the expected cost of a bad triangle is p_T. Therefore, if $y_T = p_T/(1 + \frac{2}{1-7\epsilon})$, then all the dual constraints will be satisfied. □

Suppose that we fix an upper bound of $\tau < \frac{1}{7}$ on ϵ. Then, the approximation ratio of our algorithm, as long as $\epsilon \leq \tau$, can be upper bounded by $3 + \frac{14\epsilon}{1-7\tau}$. As ϵ approaches 0, this ratio approaches $3 + 14\epsilon + O(\epsilon^2) = 3 + O(\epsilon)$. Concretely, if $\epsilon \leq \tau = \frac{1}{14}$, then we have that the approximation ratio is at most $3 + 28\epsilon$.

We observe that the total number of bits communicated in a round is at most $O(n \log n)$, which is also the size of the output and is therefore, in a sense, inevitable. We also observe that the analysis can be extended to weighted graphs that satisfy the so-called *probability constraints*, as in [3] (the algorithm still remains the same); we omit the analysis details in this version of the paper.

Tightness of the running time. We show that our running time analysis is near-tight. To this end, we construct the following complete instance $G = (V, E)$ by only specifying the positive edges; the rest of the edges will be negative. Choose some integer $1 \leq k < (1 - \epsilon) \lceil \log_2 n \rceil$, and for each $i = 1, \ldots, k$, create $\lceil n/2^k \rceil$ disjoint stars with 2^i leaves each and add them to G. It is easy to see that $|V| = \Theta(n)$ and $\Delta^+ = 2^k$, and that the number of executions of the algorithm's loop on this instance is $\Theta(\log n \cdot \log \Delta^+)$.

4.3 Realizations in scalable models

MapReduce. Recall that in the MapReduce model of computation [17], the input is partitioned across a set of machines and the computation is split into a map phase and a reduce phase. In the map phase, each machine consumes its portion of the input and outputs a set of key-value tuples. In the reduce phase, all the tuples with the same key are processed by a single machine and output, if any, is emitted. These two phases can be repeated, yielding a multi-round algorithm. Since each phase is expensive from both computational and data access points of view, it is desirable to have MapReduce algorithms with a small number of rounds.

We now illustrate how ParallelPivot can be implemented in a small number of rounds in MapReduce. Clearly, the maximum positive degree of a graph can be computed in one round of MapReduce. Likewise, the task of (randomly) selecting active nodes and computing pivots by removing adjacent active nodes (lines 5–7) can also be computed in one round. Once the pivots are computed, assigning each non-pivot node to a unique pivot neighbor (lines 10–11) can be done with an additional round. Finally, it is easy to observe that CreateCluster itself, with the graph update, can be implemented in yet another round. In Section 5.1 we show the performance of a MapReduce realization of our algorithm.

Streaming. In the streaming model of computation [4], the input is read sequentially by a machine with a limited amount of main memory (typically, sublinear in the size of the input). The goal is design an algorithm that makes a few passes over the input in order to compute the function. In the graph case, input is assumed to be available as a stream of edges. It is easy to check that each of the steps of ParallelPivot can be implemented in the streaming model, where the only amount of memory is to store the pivots and their neighbors in each pass; in Section 5.3 we show the performance of a realization of our algorithm in the streaming model.

Pregel. Pregel is a large-scale graph computing framework that is based on the bulk synchronous message-passing model [33]. In this model, each node of the graph is a computational unit, the computation proceeds in supersteps, and within each superstep, messages can be asynchronously exchanged between the nodes. It is easy to check that the two communication-intensive steps in our algorithm, namely, finding the current maximum degree and computing the pivot nodes from the active nodes can themselves be implemented in a constant number of Pregel supersteps.

4.4 General graphs

In this section we consider the correlation clustering problem on general (i.e., not necessarily complete) graphs. For general graphs, the best known is an $O(\log n)$-approximation algorithm. This algorithm, however, is based on rounding a linear program [18]. Given this, we turn our attention to the case when Δ is bounded (which is stronger than assuming Δ^+ is bounded). Unfortunately, we show that correlation clustering on bounded-degree graphs is as hard as solving correlation clustering on arbitrary graphs, thereby dashing any hopes for an improved centralized algorithm.

THEOREM 5. *There exists a constant c such that if there is a polynomial time α-approximation algorithm for correlation clustering on graphs with $\Delta \leq c$, then there is a polynomial time α-approximation algorithm for correlation clustering on every graph.*

PROOF. We show a reduction from an arbitrary graph G to a bounded-degree graph G'. Let $G = (V, E^+, E^-)$, where $|V| = n$ is sufficiently large. From G, we will construct (in polynomial time) G' with the following properties: there exists a canonical form of the (correlation clustering) solutions of G' such that every solution of G' can be transformed in polynomial time into a canonical solution having equal or smaller cost and there is a polynomial-time computable bijective mapping between canonical solutions of G' and solutions of G that leaves the cost of the solutions unchanged. These properties will complete the proof.

The key gadget we use in our proof is a constant-degree regular expander graph $H = (W, F)$ with the following properties: (i) $|W| = 2n$, (ii) $\forall v \in W, \deg(v) = 2|F|/|W| = O(1)$, and (iii) for each set $S \subseteq W$ such that $|S| \leq |W|/2$, it holds that the cut $(S, W \setminus S)$ has at least $2|S|$ edges. Such graphs exist and can be constructed in polynomial time [31]. The idea then is to "tensor" each node in G with H so that the resulting graph will have constant degree and if the tensoring is done carefully, then the correlation clustering solutions will be preserved as well.

The target graph $G' = (V', E'^{+}, E'^{-})$ will be made up of n copies of H, one for each node in V. Each edge in each of these copies will be labeled positive. These copies will be connected with each other through what we call *instance* edges. Fix $v_i \in V$ and consider v_i's copy of H denoted $H_i = (W_i, F_i)$. Since $|W_i| = 2n$, it is possible to find an injective function from $\Gamma^{+}(v_i) \cup \Gamma^{-}(v_i)$ to W_i; we use $w_{i,j}$ to denote the node in W_i corresponding to the (v_i's side of) the edge $\{v_i, v_j\}$ in $E^{+} \cup E^{-}$. If $\{v_i, v_j\}$ is a positive (resp., negative) edge, then we add a positive (resp., negative) instance edge between $w_{i,j}$ and $w_{j,i}$. This completes the description of the instance G'. Note that no node in V' will be hit by more than one instance edge and therefore Δ' will still be a constant.

Next, we show correctness. We say that a clustering of G' is *canonical* if no W_i gets partitioned among two or more clusters. Observe that a canonical clustering of G' will have cost equal to the number of instance edges that are incorrectly clustered. More precisely, if $\{v_i, v_j\} \in E^{+}$, then a canonical clustering of G' will pay 1 for that edge if it places W_i and W_j in two different clusters. Analogously, if $\{v_i, v_j\} \in E^{-}$, then a canonical clustering of G' will pay 1 for that edge if it places W_i and W_j in the same cluster. Therefore, if we produce a clustering of G from a canonical clustering of G' by putting v_i and v_j together if and only if W_i and W_j are together in the canonical clustering, then the cost of the clustering of G will equal the cost of the canonical clustering of G'. Conversely, to go from a clustering of G to a canonical clustering of G' with the same cost, it suffices to cluster the sets W_i and W_j together in G' if and only if v_i and v_j are clustered together in G.

Therefore, we only need to show that at least one optimal clustering of G' is canonical. We show this by methodically transforming a non-canonical solution to a canonical solution of smaller or equal cost in polynomial time. Indeed, suppose in the current non-canonical solution, W_i is partitioned between $k \geq 2$ clusters into mutually disjoint parts $P_1, \ldots, P_k$, with $|P_1| \leq \cdots \leq |P_k|$ and $|P_1| + \cdots + |P_k| = |W_i|$. Necessarily, we have $|P_{k-1}| \leq n/2$ and therefore the number of non-instance (positive) edges that are cut by the original clustering is at least $2 \cdot |W_i \setminus P_k|/2$; this follows from the expansion property (iii) of H_i. If t_k was the number of instance edges hitting P_k that were misplaced by the original clustering, we then have that the contribution[1] of W_i to the cost function is at least $2 \cdot |W_i \setminus P_k|/2 + t_k/2$. On the other hand if for each $i = 1, \ldots, k-1$, we disconnect the nodes in P_i from their current cluster and we put them in P_k's cluster, then we will have that all the non-instance edges of W_i will be correctly clustered. Therefore, the cost of the set $W_i \setminus P_k$ will be at most the cost of its instance edges, i.e., at most $|W_i \setminus P_k|$. The cost of P_k will also be limited to the cost of its instance edges, i.e., $t_k/2$. Therefore, the total cost will be at most $|W_i \setminus P_k| + t_k/2$. Therefore, merging all the P_i's together will not increase the cost of the clustering. By repeating this merging operation, we will end up with a canonical clustering of cost not larger than that of the original clustering. □

[1]The contribution of a set of nodes to the cost function is defined as follows: a misplaced edge fully contained in the set has cost 1, whereas a misplaced edge half-contained in the set has cost $\frac{1}{2}$.

In light of this negative result, to obtain a scalable algorithm in the general case, one has to resort to heuristics. A few possibilities are:

(i) One could run Algorithm 3 on general graphs, hoping that it will produce a reasonably good solution. Note however that the algorithm is still efficient, i.e., runs in $O(\log n \cdot \log \Delta^{+})$ rounds.

(ii) Instead of minimizing the number of disagreements in the correlation clustering objective (Definition 3.1), one could instead maximize the number of agreements. One way to do that would be to find a maximal matching M of (V, E^{+}) and take each matched edge in M as a cluster and the remaining nodes as singletons. Note that this heuristic can be implemented scalable using the methods in [28]. The heuristic does produce an $O(\Delta)$ approximation for the maximization version[2]. Indeed, since no negative edges are clustered, no mistakes are made on the edges in E^{-}. Since $M \subseteq E^{+}$ is a maximal matching, $|M| \geq \frac{|E^{+}|}{2\Delta - 1}$ for otherwise there must exist an edge in E^{+} whose endpoints are not covered by M. Thus, the proposed clustering correctly joins at least an $\Omega\left(\frac{1}{\Delta}\right)$ fraction of the edges in E^{+}.

(iii) One could try to adapt the LP rounding algorithm of [18]. To do so, though, one would need to (1) solve the linear program, and (2) round its fractional solution to an integral one. It is unclear whether these two steps can be carried out efficiently in a MapReduce-like framework.

5. EXPERIMENTS

We implemented our techniques in MapReduce and streaming, and evaluated it on the following datasets. The goal our experiments will be to show that (i) the solution quality of ParallelPivot is very close to that of Pivot and (ii) ParallelPivot runs in a very small number of rounds.

Datasets. For our first dataset, we use the structured dataset embedded in HTML pages that was extracted by the Web Data Commons project (`webdatacommons.org/`). The project extracts all Microformat, Microdata, and RDFa data from the Common Crawl web corpus (`commoncrawl.org/`), the largest and most up-to-data web corpus that is currently available to the public, and provide the extracted data for download in the form of RDF-quads. From this dataset, we constructed all entities that correspond to *places*, where a place is defined as any entity that has a `name`, either a street address or geo co-ordinates, and optionally `phone` and `website`. Appendix 6 describes the mapping that we used from the RDF types to our schema, which can be used to reproduce our dataset. We extracted around 35M places from Web Data Commons. Our goal is to resolve duplicates in this dataset. Given a pair of entities, we use the *TF-IDF* similarity between their fields to determine if they are duplicates. Since we cannot compute the similarity between all pairs, we use an n-gram index [14] on the fields to generate candidate pairs, and run our similarity condition on all candidates to generate duplicate edges. The resulting dataset has around 900M edges. We call this dataset WDC.

The second dataset that we use is the Cora dataset (`people.cs.umass.edu/~mccallum/data.html`). The Cora dataset consists of 1,878 citations to real papers. This dataset has been hand-clustered into groups referring to the same paper. While this is a much smaller dataset, the availability of ground truth enables us to study the accuracy of our algorithm, as well as compare it with Pivot. On this dataset, we trained a similarity classifier, and ran it on all pairs of node. The classifier resulted in 68,882 positive edges. We denote this dataset by CORA.

[2]We recall that there exists a trivial 2-approximation for the maximization version: return the best of the all-in-one-cluster, and the each-in-its-own-cluster, partitions.

The third dataset that we use is an undirected version of a Twitter follower graph (`an.kaist.ac.kr/traces/WWW2010.html`). This graph consists of more than 41M nodes and 2.5B edges, with the maximum degree of 2.9M. In addition to being larger, this dataset has a higher average degree, a very high maximum degree, and a skewed degree distribution. We denote this dataset by TWITTER. As before, we treat the follower edges as positive and the missing edges as negative.

5.1 Results on WDC

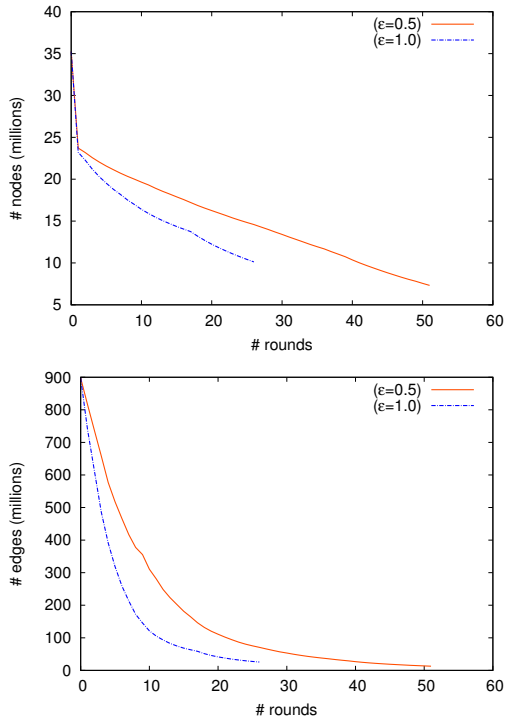

Figure 1: Number of nodes and edges remaining after each round on the WDC dataset ($\epsilon = 0.5, 1$).

We ran 50 rounds of ParallelPivot on the WDC dataset with a setting of $\epsilon = 0.5$, and 25 rounds with a setting of $\epsilon = 1$. After each round, we plotted the number of remaining nodes, the number of remaining edges, the maximum and the average degree, the number of active nodes and the number of pivots.

Figure 1 shows the plots of number of remaining nodes and edges respectively. We see that the nodes decay almost linearly with the number of rounds. The large drop in the first round is due to the fact that a large number of nodes had degree 0 (they did not generate any candidate), and hence got removed in the first round. The number of edges show a more dramatic decay. The edge roughly halve in size after every 6 rounds for $\epsilon = 0.5$ and 3 rounds for $\epsilon = 1$. For $\epsilon = 0.5$, after around 30-40 rounds, the graph is small enough to fit in the memory of singe machines, at which point we can simply use the Pivot algorithm to cluster the remaining graph. For $\epsilon = 1$, we need around 15-20 rounds.

Figure 2 plots Δ^+ (the maximum positive degree) of the graph after each round. The initial graph has $\Delta^+ = 1,400$, but it falls rapidly with each round. It also plots the average degree of the graph, which itself falls rapidly. This indicates that the graph becomes progressively sparser.

The number of active nodes chosen at each round is directly proportional to the number of total remaining nodes and inversely proportional to Δ^+, both of which decrease over time. But Δ^+ decays faster, resulting is more active nodes chosen at each round. Figure 3 shows this plot. It also plots the number of pivots (number of active nodes not adjacent to any other active node). We see that most active nodes end up being pivots. As Figure 3 shows, this ratio remains flat at around 98% throughout the entire run for $\epsilon = 0.5$. A ratio of 100% would mean that all the pivots are truly random, and the algorithm would be equivalent in accuracy to the Pivot algorithm. Thus, a high ratio of 98% indicates that we are close to Pivot accuracy. For $\epsilon = 1$, the ratio is still very high, between 95% and 98%. Note that the theoretical guarantee of ParallelPivot is a $3 + O(\epsilon)$, which could be quite larger (for constant ϵ) than the 3-approximation returned by Pivot. Still, Figure 3 shows that, in practice, the two algorithms seem to be on par. Since Pivot cannot be run on such a large dataset, we cannot provide a direct comparison of accuracy on this dataset.

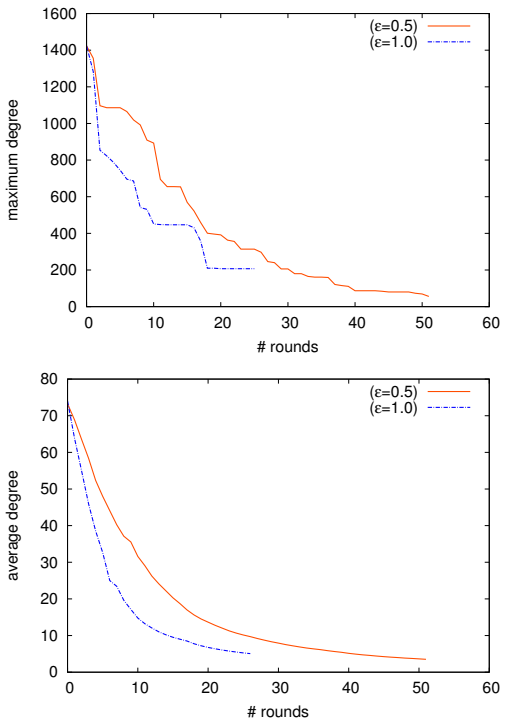

Figure 2: Maximum and average degree after each round on the WDC dataset ($\epsilon = 0.5, 1$).

5.2 Results on CORA

We use CORA to evaluate the accuracy of our techniques. Even for this small dataset, running the optimal clustering is infeasible. Thus, we cannot directly evaluate the accuracy of ParallelPivot with respect to the optimum. Instead, we evaluate its accuracy with respect to the golden truth, and compare it with Pivot. For each of the two algorithms, we define two notions of accuracy. The first is the number of disagreements with respect to the golden truth, which is the sum of the false positives and false negatives. The second is the F-measure of each algorithm, which is the harmonic mean of the precision and the recall with respect to the golden truth.

Figures 4 plot the two measures of accuracy for ParallelPivot, for ϵ ranging from 0.1 to 0.9, and compare it with Pivot. For each ϵ, we ran ParallelPivot 100 times and took the average. The shaded regions in each plot represent the error bars consisting of one stan-

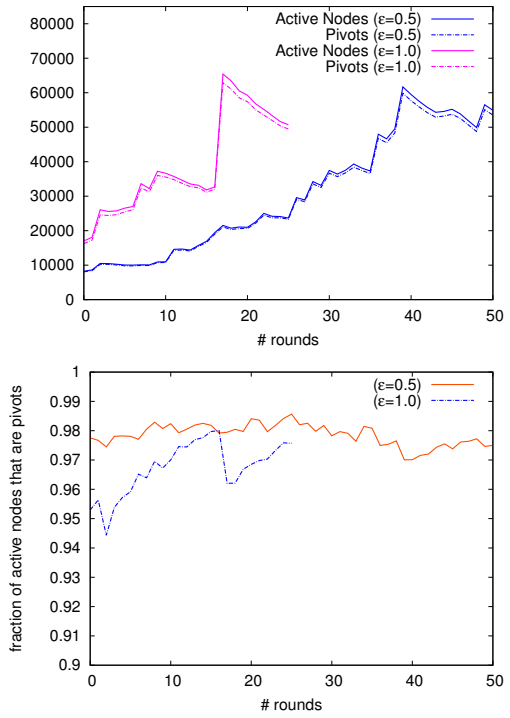

Figure 3: Number and fraction of active nodes chosen as pivots at each round on the WDC dataset ($\epsilon = 0.5, 1$).

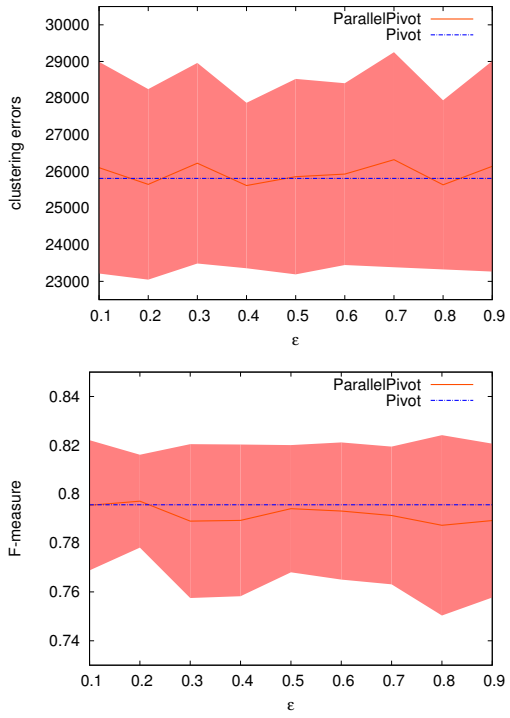

Figure 4: The number of clustering disagreements and the F-measure as a function of ϵ on CORA.

dard deviation. We see that for the entire range of ϵ, both the disagreements and F-measure of ParallelPivot are on par with Pivot.

Figure 5 shows the number of rounds that ParallelPivot takes, as a function of ϵ. Note that the ground truth has 190 clusters, so Pivot, that processes one cluster as a time, will require 190 rounds. ParallelPivot starts with 140 rounds for $\epsilon = 0.1$, but only requires around 25 rounds when ϵ is close to 1. Note that our algorithm is not defined for $\epsilon > 1$, since the probability ϵ/Δ^+ of choosing active nodes can become greater than 1. While our guarantee of $(3 + \epsilon)$ approximation for ParallelPivot only holds for $\epsilon < \frac{1}{7} \simeq 0.14$, Figures 4 and 5 together show that ParallelPivot is at par with Pivot over the entire range of ϵ, and the number of rounds significantly go down when ϵ is close to 1.

5.3 Results on TWITTER

We use TWITTER to demonstrate ParallelPivot in the streaming model. Since the number of edges in this dataset is more than 2.5B, which makes it hard to store the graph entirely on a machine with a small amount of main memory. The algorithm ran for 140 rounds.

Figure 6 shows the number of nodes and edges after each round of ParallelPivot. Perhaps because of the skewed nature of the degree distribution, the behaviors are different: the number of nodes decrease slowly in the beginning and rapidly towards the end, whereas the number of edges has the opposite behavior. Figure 7 shows the average and maximum degrees in each round. The behavior is similar to the number of edges, though the maximum degree drops rapidly; this enables more pivots to be chosen in subsequent rounds of ParallelPivot, which in turn contributes to an increased drop in the number of nodes as seen in Figure 6.

Figure 8 shows the number and fraction of active nodes chosen as pivots in each round of the algorithm. The number of active nodes (and pivots) is unimodal in nature, peaking at around the 100th round. Note that this fully determines the memory required by the streaming algorithm. Using the average degree from Figure 7, we see that the the memory used by the algorithm is upper bounded by the space require to store a graph of around 200K nodes of average degree around 2, which is insignificant (especially, when compared to the size of the original graph).

6. CONCLUSIONS

In this paper we considered the problem of how to efficiently perform correlation clustering on massive graphs. We obtained a simple algorithm that runs in a provably small number of rounds and obtains a constant-factor approximation to correlation clustering. Since correlation clustering is a basic web mining problem, our algorithm opens up the possibility of it being applied to very large instance. We illustrate this by running our algorithm on a real MapReduce and streaming systems on huge real-world instance;

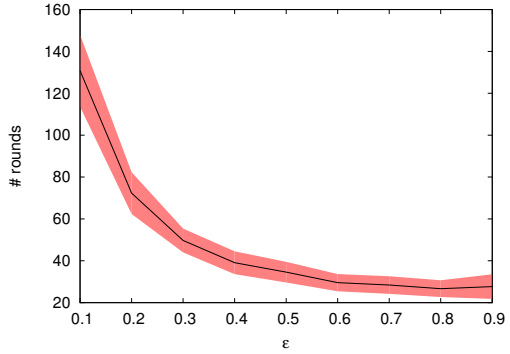

Figure 5: The number of rounds as a function of ϵ on CORA.

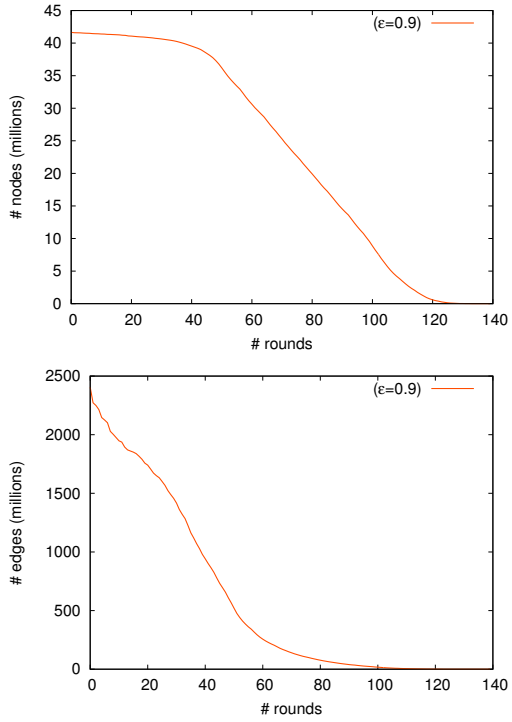

Figure 6: Number of nodes and edges remaining after each round on the TWITTER dataset.

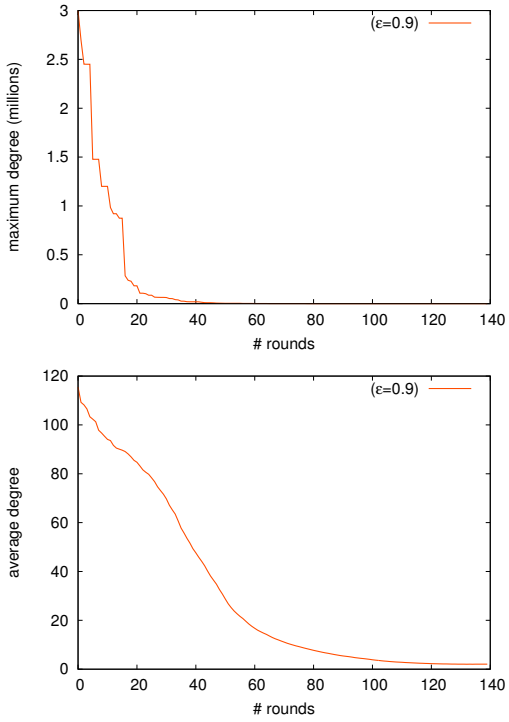

Figure 7: Maximum and average degree after each round on the TWITTER dataset.

this is one of the largest instances where a correlation clustering algorithm has been run. It will be interesting to consider variants of correlation clustering have been used in data mining applications to see if such variants also admit an efficient parallel algorithm.

7. REFERENCES

[1] R. Agrawal, A. Halverson, K. Kenthapadi, N. Mishra, and P. Tsaparas. Generating labels from clicks. In *WSDM*, pages 172–181, 2009.

[2] N. Ailon, N. Avigdor-Elgrabli, E. Liberty, and A. van Zuylen. Improved approximation algorithms for bipartite correlation clustering. *SICOMP*, 41(5):1110–1121, 2012.

[3] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *JACM*, 55(5), 2008.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *JCSS*, 58(1):137–147, 1999.

[5] A. Arasu, C. Ré, and D. Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE*, pages 952–963, 2009.

[6] S. Bagon and M. Galun. Large scale correlation clustering optimization. *CoRR*, abs/1112.2903, 2011.

[7] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and MapReduce. In *VLDB*, pages 454–465, 2012.

[8] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *PVLDB*, 5(7):622–633, 2012.

[9] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[10] F. Bonchi, A. Gionis, F. Gullo, and A. Ukkonen. Chromatic correlation clustering. In *KDD*, pages 1321–1329, 2012.

[11] F. Bonchi, A. Gionis, and A. Ukkonen. Overlapping correlation clustering. *Knowl. Inf. Syst.*, 35(1):1–32, 2013.

[12] N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella. A correlation clustering approach to link classification in signed networks. *JMLR*, 23:34.1–34.20, 2012.

[13] D. Chakrabarti, R. Kumar, and K. Punera. A graph-theoretic approach to webpage segmentation. In *WWW*, pages 377–386, 2008.

[14] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, 2006.

[15] Y. Chen, S. Sanghavi, and H. Xu. Clustering sparse graphs. In *NIPS*, pages 2213–2221, 2012.

[16] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in Map-Reduce. In *WWW*, pages 231–240, 2010.

[17] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *C. ACM*, 51(1):107–113, 2008.

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

[19] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.

[20] M. Elsner and W. Schudy. Bounding and comparing methods for correlation clustering beyond ILP. In *NAACL-HLT Workshop on ILP-NLP*, 2009.

[21] T. Finley and T. Joachims. Supervised clustering with support vector machines. In *ICML*, pages 217–224, 2005.

[22] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *TKDD*, 1(1), 2007.

[23] S. Günnemann, I. Färber, K. Virochsiri, and T. Seidl. Subspace correlation clustering: Finding locally correlated dimensions in subspace projections of the data. In *KDD*, pages 352–360, 2012.

[24] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *VLDB*, 2(1):1282–1293, 2009.

[25] D. V. Kalashnikov, Z. Chen, S. Mehrotra, and R. Nuray-Turan. Web people search via connection analysis. *TKDE*, 20(11):1550–1565, 2008.

[26] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948, 2010.

[27] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *TKDD*, 3(1):1–58, 2009.

[28] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: A method for solving graph problems in MapReduce. In *SPAA*, pages 85–94, 2011.

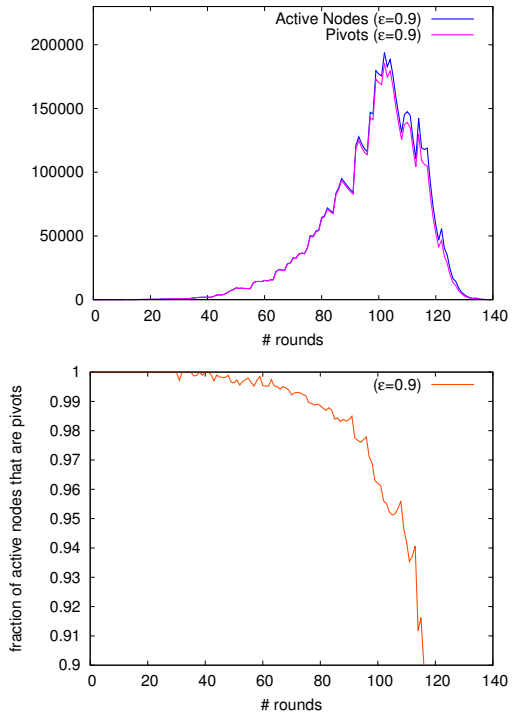

Figure 8: Number and fraction of active nodes chosen as pivots in each round on the TWITTER dataset.

[29] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Predicting positive and negative links in online social networks. In *WWW*, pages 641–650, 2010.

[30] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Signed networks in social media. In *CHI*, pages 1361–1370, 2010.

[31] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[32] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *STOC*, pages 1–10, 1985.

[33] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.

[34] A. McCallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *IIWeb*, pages 79–84, 2003.

[35] G. D. F. Morales, A. Gionis, and M. Sozio. Social content matching in MapReduce. In *PVLDB*, pages 460–469, 2011.

[36] E. Sadikov, J. Madhavan, L. Wang, and A. Y. Halevy. Clustering query refinements by user intent. In *WWW*, pages 841–850, 2010.

[37] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pages 607–614, 2011.

[38] A. Zimek. Correlation clustering. *SIGKDD Explor. Newsl.*, pages 53–54, 2009.

APPENDIX

Web Commons Data

The schema for our places data consists of `name`, `address`, `street`, `locality`, `region`, `country`, `zip`, `category`, `description`, `phone`, `geo`, and `url`. The following table shows all the RDF types in Web Data Commons that we mapped to each of our attribute. Not all attributes are present in all entities.

name
http://rdf.data-vocabulary.org/#name
http://rdf.data-vocabulary.org/name
http://schema.org/LocalBusiness/name
http://data-vocabulary.org/Organization/title
http://www.data-vocabulary.org/Organization/name
http://www.w3.org/2006/vcard/ns#given-name
http://data-vocabulary.org/Organization/name
http://www.w3.org/2006/vcard/ns#n
http://rdf.data-vocabulary.org/#nickname
http://www.w3.org/2006/vcard/ns#additional-name
http://www.w3.org/2006/vcard/ns#nickname
http://rdf.data-vocabulary.org/#title
http://rdf.data-vocabulary.org/title
address
http://data-vocabulary.org/Organization/address
http://rdf.data-vocabulary.org/#addr
http://rdf.data-vocabulary.org/#address
http://rdf.data-vocabulary.org/address
http://schema.org/LocalBusiness/address
http://www.data-vocabulary.org/Organization/address
http://www.w3.org/2006/vcard/ns#adr
street
http://data-vocabulary.org/Organization/street-address
http://rdf.data-vocabulary.org/#street-address
http://rdf.data-vocabulary.org/#street
http://rdf.data-vocabulary.org/street-address
http://rdf.data-vocabulary.org/streetAddress
http://schema.org/LocalBusiness/streetAddress
http://www.w3.org/2006/vcard/ns#street-address
region
http://data-vocabulary.org/Organization/region
http://rdf.data-vocabulary.org/#region
http://rdf.data-vocabulary.org/addressRegion
http://rdf.data-vocabulary.org/region
http://www.w3.org/2006/vcard/ns#region
zip
http://data-vocabulary.org/Organization/postal-code
http://rdf.data-vocabulary.org/#postal-code
http://rdf.data-vocabulary.org/postal-code
http://rdf.data-vocabulary.org/postalCode
http://schema.org/LocalBusiness/postalCode
http://www.w3.org/2006/vcard/ns#postal-code
country
http://data-vocabulary.org/Organization/country-name
http://rdf.data-vocabulary.org/#country-name
http://www.w3.org/2006/vcard/ns#country-name
category
http://rdf.data-vocabulary.org/#category
http://www.w3.org/2006/vcard/ns#category
phone
http://data-vocabulary.org/Organization/tel
http://rdf.data-vocabulary.org/#tel
http://rdf.data-vocabulary.org/tel
http://schema.org/LocalBusiness/telephone
http://www.data-vocabulary.org/Organization/tel
http://www.w3.org/2006/vcard/ns#tel
http://www.w3.org/2006/vcard/ns#workTel
http://schema.org/LocalBusiness/faxNumber
http://www.w3.org/2006/vcard/ns#fax
geo
http://data-vocabulary.org/Organization/geo
http://rdf.data-vocabulary.org/#geo
http://rdf.data-vocabulary.org/geo
http://schema.org/LocalBusiness/geo
http://www.w3.org/2006/vcard/ns#geo
http://rdf.data-vocabulary.org/#latitude
http://www.data-vocabulary.org/Organization/longitude
http://www.w3.org/2006/vcard/ns#longitude
http://rdf.data-vocabulary.org/#longitude
http://www.data-vocabulary.org/Organization/latitude
http://www.w3.org/2006/vcard/ns#latitude
url
http://rdf.data-vocabulary.org/url
http://data-vocabulary.org/Organization/url
http://rdf.data-vocabulary.org/#url
http://rdf.data-vocabulary.org/url
http://schema.org/LocalBusiness/url
http://www.data-vocabulary.org/Organization/url
http://www.w3.org/2006/vcard/ns#url
http://rdf.data-vocabulary.org/acquireURL