Correlation Clustering in Data Streams

Kook Jin Ahn[1] KOOKJIN@CIS.UPENN.EDU
University of Pennsylvania, Philadelphia, PA 19104, USA.

Graham Cormode G.CORMODE@WARWICK.AC.UK
University of Warwick, Coventry CV4 7AL, UK.

Sudipto Guha SUDIPTO@CIS.UPENN.EDU
University of Pennsylvania, Philadelphia, PA 19104, USA.

Andrew McGregor MCGREGOR@CS.UMASS.EDU
University of Massachusetts, Amherst, MA 01003, USA.

Anthony Wirth AWIRTH@UNIMELB.EDU.AU
Department of Computing and Information Systems, The University of Melbourne, Vic 3010, Australia.

Abstract

In this paper, we address the problem of *correlation clustering* in the dynamic data stream model. The stream consists of updates to the edge weights of a graph on n nodes and the goal is to find a node-partition such that the end-points of negative-weight edges are typically in different clusters whereas the end-points of positive-weight edges are typically in the same cluster. We present polynomial-time, $O(n \cdot \mathrm{polylog}\, n)$-space approximation algorithms for natural problems that arise.

We first develop data structures based on linear sketches that allow the "quality" of a given node-partition to be measured. We then combine these data structures with convex programming and sampling techniques to solve the relevant approximation problem. However the standard LP and SDP formulations are not obviously solvable in $O(n \cdot \mathrm{polylog}\, n)$-space. Our work presents space-efficient algorithms for the convex programming required, as well as approaches to reduce the adaptivity of the sampling. Note that the improved space and running-time bounds achieved from streaming algorithms are also useful for offline settings such as MapReduce models.

[1]Currently at Google, Inc. Email:kookjin@google.com

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

1. Introduction

The correlation clustering problem was first formulated as an optimization problem by Bansal et al. (2004). The input is a complete weighted graph G on n nodes, where each pair of nodes uv has weight $w_{uv} \in \mathbb{R}$. A positive-weight edge indicates that u and v should be in the same cluster, whereas a negative-weight edge indicates that u and v should be in different clusters. Given a node-partition $\mathcal{C} = \{C_1, C_2, \ldots\}$, we say edge uv *agrees* with $\mathcal{C}$, denoted by $uv \sim \mathcal{C}$, if the relevant soft constraint is observed. Summing over all edges,

$$\mathsf{agree}(G, \mathcal{C}) := \sum_{uv \sim \mathcal{C}} |w_{uv}| = \sum_{uv} |w_{uv}| - \mathsf{disagree}(G, \mathcal{C})\,.$$

The goal is to find the partition $\mathcal{C}$ that maximizes $\mathsf{agree}(G, \mathcal{C})$ or, equivalently for optimization, minimizes $\mathsf{disagree}(G, \mathcal{C})$. Solving this problem exactly is known to be NP-hard. A large body of work has been devoted to approximating $\mathsf{max\text{-}agree}(G) = \max_{\mathcal{C}} \mathsf{agree}(G, \mathcal{C})$ and $\mathsf{min\text{-}disagree}(G) = \min_{\mathcal{C}} \mathsf{disagree}(G, \mathcal{C})$, along with variants $\mathsf{min\text{-}disagree}_k(G)$ and $\mathsf{max\text{-}agree}_k(G)$, where we consider partitions with at most k clusters. In this paper, we focus on multiplicative approximation results exclusively. If all weights are ± 1, there is a polynomial time approximation scheme PTAS for max-agree (Bansal et al., 2004; Giotis & Guruswami, 2006) and a 2.06-approximation for min-disagree (Chawla et al., 2015). When there is an upper bound k on the number of clusters in $\mathcal{C}$, and all weights are ± 1, Giotis & Guruswami (2006) introduced a PTAS for both problems. Even $k = 2$ is interesting and an efficient local-search approximation was introduced by Coleman et al. (2008).

If the weights are arbitrary, there is a 0.7666-approximation for max-agree (Charikar et al., 2005; Swamy, 2004) and an $O(\log n)$ approximation for min-disagree (Charikar et al., 2005; Demaine et al., 2006). These methods use convex programming: as described, this cannot be implemented in $O(n \operatorname{polylog} n)$ memory even when the input graph is sparse. This annoying feature is well known in practice: Bagon & Galun (2011); Bonchi et al. (2014); Elsner & Schudy (2009) discuss the difficulty of scaling the convex programming approach.

Clustering and Graph Analysis in Data Streams. Given the importance of clustering as a basic tool for analyzing massive data sets, it is unsurprising that a considerable effort has gone into designing clustering algorithms in the relevant computational models. In particular, in the data-stream model we are permitted a limited number of passes (ideally just one) over the data while using only limited memory. This model abstracts the challenges in traditional applications of stream processing such as network monitoring, and also leads to I/O-efficient external-memory algorithms. Naturally, in either context, an algorithm should also be fast, both in terms of the time to process each stream element and in returning the final answer.

Classical clustering problems including k-median (Charikar et al., 2003; Guha et al., 2000), k-means (Ailon et al., 2009), and k-center (Charikar et al., 2004; Guha, 2009; McCutchen & Khuller, 2008) have all been studied in the stream model, as surveyed by Silva et al. (2013). Non-adaptive sampling algorithms for correlation clustering can be implemented in the data stream model and such algorithms were used by Ailon & Karnin (2012) to construct additive approximations. However, the first multiplicative result for correlation clustering was only recently established; Chierichetti et al. (2014) presented a polynomial-time $(3+\epsilon)$-approximation for min-disagree on ± 1-weighted graphs using $O(\epsilon^{-1} \log^2 n)$ passes. Their basic approach yields both a MapReduce and *semi-streaming* algorithm — that is, a streaming algorithm using $\Theta(n \operatorname{polylog} n)$ memory (Feigenbaum et al., 2005). Using space proportional to the number of nodes can be shown to be necessary for solving many natural graph problems in the streaming model including, we show, correlation clustering. McGregor (2014) surveys semi-streaming algorithms and graph sketching.

Computational Model. In the basic graph stream model the input stream consists of a sequence edges and their weights. The available space to process the stream and perform any necessary post-processing is $O(n \operatorname{polylog} n)$ bits. Our results also extend to the *dynamic graph stream model* where the stream consists of both insertions and deletions of edges; the weight of an edge is specified when the edge is inserted and deleted (if it is subsequently deleted).

For simplicity, we assume that all weights are integral. We will consider three types of weighted graphs: (a) *unit weights*, where all $w_{uv} \in \{-1, 1\}$; (b) *bounded weights*, where all weights are non-zero, with absolute value bounded above by a constant. and (c) *arbitrary weights* where we only require that all weights are $O(\operatorname{poly} n)$.

Our Results. For max-agree we provide the following single-pass streaming algorithms, each needing $\tilde{O}(n\epsilon^{-2})$ space: (i) a polynomial-time $(1-\epsilon)$-approximation for bounded weights (Theorem 7), and (ii) a $0.766(1-\epsilon)$ approximation for arbitrary weights in $\tilde{O}(n\epsilon^{-10})$ time (Theorem 16).

We can show that any algorithm that can test whether min-disagree$(G) = 0$ in a single pass, for arbitrary weights, must store $\Omega(n + |E^-|)$ bits for any $|E^-| \leq \binom{n}{2}$. For unit weights, the lower bound is $\Omega(n)$. We provide a single-pass algorithm that uses $\tilde{O}(n\epsilon^{-2} + |E^-|)$ space and $\tilde{O}((|E^-| + n\epsilon^{-2})^2)$ time and provides an $O(\log |E^-|)$ approximation (Theorem 13). Since Demaine et al. (2006) and Charikar et al. (2005) provide approximation-preserving reductions from the 'minimum multicut' problem to min-disagree with arbitrary weights, it is expected to be difficult to approximate the latter to better than a $\log |E^-|$ factor in polynomial time. For unit weights when min-disagree$(G) \leq t$, we provide a single-pass polytime algorithm that uses $\tilde{O}(n + t\epsilon^{-2})$ space (Theorem 4). We provide a $\tilde{O}(n\epsilon^{-2})$-space PTAS for min-disagree$_2$ for bounded weights (Theorem 10).

We also considering multipass streaming algorithms. For unit weights, we presnt a $O(\log \log n)$-pass algorithm that mimics the an by Ailon et al. (2008), and with high probability provides a 3 approximation (Theorem 20). This improves the result of Chierichetti et al. (2014). For min-disagree$_k(G)$, on unit-weight graphs with $k \geq 3$, we give a $\min(k-1, O(\log \log n))$-pass polynomial-time algorithm using $\tilde{O}(n\epsilon^{-2})$ space (Theorem 21).

Techniques and Roadmap. In Section 2, we present three basic data structures for the agree and disagree query problems where a partition $\mathcal{C}$ is specified at the end of the stream, and the goal is to return an approximation of agree$(G, \mathcal{C})$ or disagree$(G, \mathcal{C})$. They are based on linear sketches and incorporate ideas from recent work on constructing graph sparsifiers via linear sketches (Ahn et al., 2012b). These data structures can be constructed in the semi-streaming model and can be queried in $\tilde{O}(n)$ time. As algorithms rely on relatively simple matrix-vector operations, they can be implemented fairly easily in MapReduce.

In Section 3, we introduce several new ideas in solving the LP and SDP for min-disagree and max-agree. In each case, the convex formulation has allow each candidate solution to be represented, verified, and updated in small space. Finally, we discuss multipass algorithms in Section 4. Proofs are deferred to the full version.

2. Basic Data Structures and Applications

We introduce three basic data structures that can be constructed with a single-pass over the input stream that defines the weighted graph G. Given a query partition $\mathcal{C}$, these data structures return estimates of $\mathsf{agree}(G,\mathcal{C})$ or $\mathsf{disagree}(G,\mathcal{C})$. Solving the correlation clustering optimization problem with these structures directly would require exponential time or $\omega(n\,\mathrm{polylog}\,n)$ space. Instead, we will exploit them carefully to design more efficient solutions. In this section, we present a short application of each data structure. that illustrates their benefit.

2.1. First Data Structure: Bilinear Sketch

Consider a graph G with unit weights ($w_{ij} \in \{-1,1\}$) and a clustering $\mathcal{C}$. Define the matrices M^G and $M^{\mathcal{C}}$ where $M^G_{ij} = \max(0, w_{ij})$ and

$$M^{\mathcal{C}}_{ij} = \begin{cases} 0 & \text{if } i \text{ and } j \text{ are separated in } \mathcal{C} \\ 1 & \text{if } i \text{ and } j \text{ are not separated in } \mathcal{C}\,. \end{cases}$$

Observe that the (squared) matrix distance induced by the Frobenius norm gives exactly $\mathsf{disagree}(G,\mathcal{C}) = \|M^G - M^{\mathcal{C}}\|_F^2 = \sum_{ij}(m^G_{ij} - m^{\mathcal{C}}_{ij})^2$. To efficiently estimate $\|M^G - M^{\mathcal{C}}\|_F^2$ when $\mathcal{C}$ is not known a priori, we can repurpose the bilinear sketch approach of Indyk & McGregor (2008):

1. Let $\alpha \in \{-1,1\}^n$ and $\beta \in \{-1,1\}^n$ be independent random vectors whose entries are 4-wise independent, and in a single pass over the input compute $Y = \sum_{ij\in E^+} \alpha_i\beta_j$.
2. Given query partition $\mathcal{C} = \{C_1, C_2, \ldots\}$, return $X = \left(\sum_\ell \left(\sum_{i\in C_\ell} \alpha_i\right)\left(\sum_{i\in C_\ell}\beta_i\right) - Y\right)^2$.

Following the results of Indyk & McGregor (2008) and Braverman et al. (2010), we have:

Lemma 1. *For each* $\{f_{ij}\}_{i,j\in[n]}$, $\mathbb{E}\left[\left(\sum_{i,j}\alpha_i\beta_j f_{ij}\right)^2\right] = \sum_{i,j} f_{ij}^2$ *and* $\mathbb{V}\left[\left(\sum_{i,j}\alpha_i\beta_j f_{ij}\right)^2\right] \le 9\left(\sum_{i,j} f_{ij}^2\right)^2$.

Applying the above lemma to $f_{ij} = m^G_{ij} - m^{\mathcal{C}}_{ij}$ establishes that $\mathbb{E}[X] = \mathsf{disagree}(G,\mathcal{C})$ and $\mathbb{V}[X] \le 9(\mathsf{disagree}(G,\mathcal{C}))^2$. Hence, running $O(\epsilon^{-2}\log\delta^{-1})$ parallel repetitions of the scheme and averaging the results appropriately[2] yields a $(1\pm\epsilon)$-approximation for $\mathsf{disagree}(G,\mathcal{C})$ with probability at least $1-\delta$.

Theorem 2. *For unit weights, there exists a* $O(\epsilon^{-2}\log\delta^{-1}\log n)$*-space algorithm for the* disagree *query problem. Each positive edge is processed in* $\tilde{O}(\epsilon^{-2})$ *time, while the query time is* $\tilde{O}(\epsilon^{-2}n)$.

[2]Specifically, take the standard approach of partitioning the estimates into $O(\log\delta^{-1})$ groups, each of size $O(\epsilon^{-2})$. With constant probability, the mean of each group is within a $1\pm\epsilon$ factor; we finally return the median of the resulting group estimates.

We note that by setting $\delta = 1/n^n$ in the above theorem, it follows that we may estimate $\mathsf{disagree}(G,\mathcal{C})$ for all partitions $\mathcal{C}$ using $\tilde{O}(\epsilon^{-2}n)$ space. Hence, we can also $(1+\epsilon)$ approximate $\mathsf{min\text{-}disagree}(G)$ given exponential time. While this is near-optimal in terms of the space use, in this paper we focus on polynomial time algorithms.

Application to Cluster Repair. Consider the Cluster Repair problem (Gramm et al., 2005), in which we are promised $\mathsf{min\text{-}disagree}(G) \le t$ for some constant t. As we can narrow down the number of possible clusterings to $\mathrm{poly}(n)$, there is a simple polynomial-time application of the above data structure.

First construct a spanning forest F of G^+ using the $\tilde{O}(n)$-space dynamic graph algorithm (Ahn et al., 2012a). Let $\mathcal{C}_F$ be the node-partition corresponding to the connected components of F. Then, let $F_1, F_2, \ldots$ be all the forests formed by deleting at most t edges from F. Let $\mathcal{C}_{F_i}$ be the node-partition corresponding to the connected components of F_i.

Lemma 3. *The optimal partition of* G *is a refinement of* $\mathcal{C}_F$ *and a coarsening of some* $\mathcal{C}_{F_i}$ *and there are at most* $O\left((n(t+1))^{t+1}\right)$ *such partitions.*

Therefore, setting $\delta = O\left((n(t+1))^{-(t+1)}\right)$ in Theorem 2 yields the following theorem.

Theorem 4. *For a unit-weight graph* G *with* $\mathsf{min\text{-}disagree}(G) \le t$*, there exists a polynomial-time data-stream algorithm using* $\tilde{O}(n + t\epsilon^{-2})$ *space that with high probability* $(1+\epsilon)$ *approximates* $\mathsf{min\text{-}disagree}(G)$.

2.2. Second Data Structure: Sparsification

A sparsification of graph G is a weighted graph H such that the weight of every cut in H is within a $1+\epsilon$ factor of the weight of the corresponding cut in G. A celebrated result of Benczúr & Karger (1996) shows that the size of H is at most $\tilde{O}(n\epsilon^{-2})$. A recent result shows that this can be constructed in the dynamic graph stream model.

Theorem 5. *Ahn et al. (2012b). There is a single-pass semi-streaming algorithm that returns a sparsification using space* $\tilde{O}(n\epsilon^{-2})$ *and time* $\tilde{O}(m)$.

The algorithm can also be implemented in MapReduce (Ahn & Guha, 2015). The next lemma is straightforward.

Lemma 6. *Let* H^+ *and* H^- *be sparsifications of* $G^+ = (V, E^+)$ *and* $G^- = (V, E^-)$ *such that all cuts are preserved within factor* $(1\pm\epsilon/3)$*, and let* $H = H^+ \cup H^-$*. For every clustering* $\mathcal{C}$*,* $\mathsf{agree}(G,\mathcal{C}) = (1\pm\epsilon)\mathsf{agree}(H,\mathcal{C}) \pm \epsilon w(E^+)$ *and* $\mathsf{disagree}(G,\mathcal{C}) = (1\pm\epsilon)\mathsf{disagree}(H,\mathcal{C}) \pm \epsilon w(E^-)$*. Note that* $\mathsf{disagree}(G,\mathcal{C}) = (1\pm\epsilon)\mathsf{disagree}(H^+ \cup E^-, \mathcal{C})$.

Application to max-agree with Unit Weights. It would be unfortunate if, by approximating the unit-weight graph by a weighted sparsification, we lost the ability to return a $1 \pm \epsilon$ approximation in polynomial time. We resolve this as follows. We emulate part of an algorithm by Giotis & Guruswami (2006) for max-agree$_k$ using a single pass over the stream[3]. In their algorithm, the nodes are partitioned into $m = O(1/\epsilon)$ groups $V_1, V_2, \ldots, V_m$, each of size $O(\epsilon n)$, and for each V_i we draw a sample of $r = \text{poly}(1/\epsilon, k, \log 1/\delta)$ nodes S_i from $V \setminus V_i$. Using the weights on edges between each S_i and V_i, we consider all possible partitions of S_i. The required sampling can be performed simultaneously with the construction of the sparsifier. Then, at the end of the stream, the possible partitions are generated and we use the graph sparsifier to find the best of these partitions.

Theorem 7. *For bounded-weight inputs, there exists a poly-time semi-streaming algorithm that with high probability $(1+\epsilon)$-approximates* max-agree(G).

2.3. Third Data Structure: Node-Based Sketch

In this section we develop a data structure that supports queries to disagree$(G, \mathcal{C})$ for arbitrarily *weighted* graphs when $\mathcal{C}$ is restricted to be a 2-partition. For each node i, define the vector, $a^i \in \mathbb{R}^{\binom{n}{2}}$, indexed over the $\binom{[n]}{2}$ edges, where the only non-zero entries are:

$$a^i_{ij} = \begin{cases} w_{ij}/2 & \text{if } ij \in E^- \\ w_{ij}/2 & \text{if } ij \in E^+, i < j \\ -w_{ij}/2 & \text{if } ij \in E^+, i > j \end{cases}$$

Lemma 8. *For a two-partition $\mathcal{C} = \{C_1, C_2\}$,* disagree$(G, \mathcal{C}) = \| \sum_{\ell \in C_1} a^\ell - \sum_{\ell \in C_2} a^\ell \|_1$.

Hence, we use the ℓ_1-sketching result of Kane et al. (2010) to compute a random linear sketch of each a^i.

Theorem 9. *For arbitrary weights, and for query partitions that contain two clusters, to solve the* disagree *query problem, there exists an $O(\epsilon^{-2} n \log \delta^{-1} \log n)$-space algorithm. The query time is $O(\epsilon^{-2} n \log \delta^{-1} \log n)$.*

Unfortunately, for queries $\mathcal{C}$ where $|\mathcal{C}| > 2$, $\Omega(n^2)$ space is necessary.

Application to min-disagree$_2(G)$ with Bounded Weights. We apply the above node-based sketch in conjunction with another algorithm by Giotis & Guruswami (2006), this time for min-disagree$_2$. It samples $r = \text{poly}(1/\epsilon) \cdot \log n$ nodes S and, using the weights of the edges incident on S, generates 2^{m-1} possible partitions; we generalize this to the bounded weights case. The sampling of S and its incident edges can be performed using one pass and $O(nr \log n)$ space. We then find the best of these possible partitions in post-processing using the above node-based sketches.

Theorem 10. *For bounded-weight inputs, there exists a polynomial-time semi-streaming algorithm that with high probability $(1+\epsilon)$-approximates* min-disagree$_2(G)$.

3. Convex Programming in Small Space

In this section we discuss an SDP-based algorithm for max-agree and an LP-based algorithm for min-disagree. At a high level, progress arises from new ideas and modifications needed to implement convex programs in small space. While the time to solve convex programs has always been an issue, additionally restricting to small space is relatively recent (Ahn & Guha, 2013). In this paper, we follow the Multiplicative Weight Update method and its derivatives. This method has a rich history across many different communities (Arora et al., 2012), and has been extended to SDPs (Arora & Kale, 2007).

In all these approaches, the optimization problem is first reduced to a decision variant, involving a "guess" α of the objective value; we show later how to instantiate this guess. For LPs, we seek to solve the system:

$$\text{MWM LP: } \begin{cases} & \mathbf{c}^{\mathrm{T}}\mathbf{y} \geq \alpha \\ \text{s.t} & \mathbf{A}\mathbf{y} \leq \mathbf{b}, \quad \mathbf{y} \geq \mathbf{0} \end{cases}$$

where $\mathbf{A} \in \mathbb{R}_+^{n \times m}$, $\mathbf{c}, \mathbf{y} \in \mathbb{R}_+^m$, and $\mathbf{b} \in \mathbb{R}_+^n$. For SDPs, consider the following definition:

Definition 1. *For matrices $\mathbf{X}, \mathbf{Z}$, let $\mathbf{X} \circ \mathbf{Z}$ denote $\sum_{i,j} \mathbf{X}_{ij}\mathbf{Z}_{ij}$, let $\mathbf{X} \succeq \mathbf{0}$ denote that $\mathbf{X}$ is positive semidefinite, and let $\mathbf{X} \succeq \mathbf{Z}$ denote $\mathbf{X} - \mathbf{Z} \succeq \mathbf{0}$.*

A semidefinite decision problem in canonical form is:

$$\text{MWM SDP: } \begin{cases} & \mathbf{C} \circ \mathbf{X} \geq \alpha, \\ \text{s.t} & \mathbf{F}_j \circ \mathbf{X} \leq g_j, \ \forall 1 \leq j \leq q, \quad \mathbf{X} \succeq \mathbf{0} \end{cases}$$

where $\mathbf{C}, \mathbf{X} \in \mathbb{R}^{n \times n}$ and $\mathbf{g} \in \mathbb{R}_+^q$. Denote the set of the feasible solutions by $\mathcal{X}$. Typically we are interested in the Cholesky decomposition of $\mathbf{X}$, a set of n vectors $\{\mathbf{x}_i\}$ such that $\mathbf{X}_{ij} = \mathbf{x}_i^T \mathbf{x}_j$.

To solve the LP or the SDP, the multiplicative-weight update algorithm proceeds iteratively. In each iteration, given the current solution $\mathbf{y}, \mathbf{X}$ the meta-algorithm either decides that the current candidate is (approximately) feasible or computes a new current solution.

Equivalently, we can describe the process as maintaining a set of multipliers (one for each constraint) and computing a new candidate solution $\mathbf{y}', \mathbf{X}'$ which (approximately) satisfies the linear combination of the inequalities. The new current solution is a linear combination of the previous solution and the new candidate.

[3] Note max-agree$_k(G) \geq (1-\epsilon)$max-agree(G) for $k = O(1/\epsilon)$ (Bansal et al., 2004).

These two views are consistent: typically, the multiplier of a constraint is a exponential of the violation (suitably scaled) induced by the current solution, and the current solution is typically the running average (suitably scaled) of the candidates produced by the different iterations. Consider the following two theorems.

Theorem 11 (Steurer (2010)). *Let $\mathbf{D}$ be a fixed diagonal matrix with positive entries and assume $\mathcal{X}$ be nonempty. Suppose there is an Oracle that for each positive semidefinite $\mathbf{X}$ either (a) tests and declares $\mathbf{X}$ to be approximately feasible — for all $1 \le i \le q$, we have $\mathbf{F}_i \circ \mathbf{X} \le g_i + \delta$, or (b) provides a real symmetric matrix $\mathbf{A}$ and a scalar b satisfying (i) $\mathbf{A} \circ \mathbf{X} \le b - \delta$ and for all $\mathbf{X}' \in \mathcal{X}$, $\mathbf{A} \circ \mathbf{X}' \ge b$ and (ii) $\rho\mathbf{D} \succeq \mathbf{A} - b\mathbf{D} \succeq -\rho\mathbf{D}$, then a multiplicative-weight-style algorithm produces an approximately feasible $\mathbf{X}$, in fact its Cholesky decomposition, in $T = O(\rho^2\delta^{-2} \ln n)$ iterations.*

Theorem 12 (Arora et al. (2012)). *Suppose that, given a set of non-negative multipliers $\mathbf{u}(t)$ in iteration t, an Oracle provides a $\mathbf{y}(t)$ satisfying: (i) $\mathbf{c}^T\mathbf{y}(t) \ge \alpha$, (ii) $\mathbf{u}(t)^T\mathbf{A}\mathbf{y}(t) - \mathbf{u}(t)^T\mathbf{b} \le \delta \sum_i \mathbf{u}_i(t)$, and (iii) $-\rho \le -\ell \le \mathbf{A}_i\mathbf{y}(t) - \mathbf{b}_i \le \rho$ for all $1 \le i \le n$. After $T = O(\rho\ell\delta^{-2} \ln n)$ iterations of the multiplicative-weight update algorithm, the average vector, $\mathbf{y} = \frac{1}{T}\sum_t \mathbf{y}(t)$, satisfies $\mathbf{A}_i\mathbf{y} - \mathbf{b}_i \le 4\delta$ for all i.*

The above two theorems are expressed differently, each corresponding to a different view of the multiplicative-weight update method. The first view corresponds to the "state" that needs to be maintained in each iteration; the second corresponds to a more operational view, where multiple constraints are reduced to a single linear constraint which is (hopefully) easier to solve. From the perspective of space efficiency, both views must be followed. In both contexts, the parameter ρ is called the "width", and controls the speed of convergence. As is explicitly stated in the two cited papers, and as is widely recognized, the construction of a small-width Oracle is the key component of an effective solution. However, the width parameter is inherently tied to the specific formulation chosen.

Consider the standard LP relaxation for min-disagree, where x_{ij} corresponds to edge ij being cut.

$$\begin{aligned} \min \sum_{ij \in E^+} w_{ij}x_{ij} + \sum_{ij \in E^-} |w_{ij}|(1 - x_{ij}) \\ x_{ij} + x_{j\ell} \ge x_{i\ell} \qquad \forall i, j, \ell \\ x_{ij} \ge 0 \qquad \forall i, j \end{aligned}$$

The constraints simply enforce that if we cut one side of a triangle, we must also cut at least one of the other two sides. Note that the size of formulation is $\Theta(n^3)$ irrespective of the number of nonzero entries in E^+, E^-. We will use the sparsification of E^+, but that does not in any way change the size of the above linear program. To achieve $\tilde{O(n)}$ space, we need new formulations and new algorithms to solve them.

Let H^+ be the sparsification of E^+ with $m' = |H^+|$. For edge $sq \in H^+$ let w^h_{sq} denote the weight after the sparsification. For $ij \in E^-$, let $P_{ij}(E')$ denote the set of all paths using the edges of edge set E'. Consider the following LP for min-disagree, similar to that by Wirth (2004), but in this sparsified setting:

$$\begin{aligned} \min \sum_{ij \in E^-} |w_{ij}|z_{ij} + \sum_{sq \in H^+} w^h_{sq}x_{sq} \\ z_{ij} + \sum_{sq \in p} x_{sq} \ge 1 \quad \forall p \in P_{ij}(H^+), ij \in E^- \\ z_{ij}, x_{sq} \ge 0 \qquad \forall ij \in E^-, sq \in H^+ \end{aligned} \tag{LP1}$$

The intuition of an integral 0/1 solution is that $z_{ij} = 1$ for all edges $ij \in E^-$ that are not cut, and $x_{sq} = 1$ for all $sq \in H^+$ that are cut, by the intended clustering. Although LP1 now has exponential size, we will apply the multiplicative weights framework (Theorem 12) to the dual of LP1. The oracle we provide has the twist that if it fails to find $\mathbf{y}(t)$ with the necessary properties, then it provides a feasible f-approximation of the primal (in this case LP1). This idea, of applying the multiplicative-weight update method to a formulation with exponentially many variables (the dual), and modifying the Oracle to provide a solution to the primal (with exponentially many constraints) in a single step, has also benefited solving maximum matching in small space (Ahn & Guha, 2015). The existence of multiple such examples demonstrates that starting from the dual, a "dual-primal method", helps solve convex programs in small space. One key insight is that the dual, with exponentially many variables and few constraints, is easier to solve in a few iterations because there are many degrees of freedom. This reduces the adaptive nature of the solution, and therefore we can make a lot of progress in satisfying many of the primal constraints in parallel. In contrast, the classic application of primal-dual techniques in approximation algorithms tries to construct a solution of the dual efficiently: in that context, in polynomial time. It is often thought, as is explicitly mentioned by Arora & Kale (2007), that the primal-dual approximation algorithms use a different set of techniques from the primal-dual approach of multiplicative-weight update methods. By switching the dual and the primal, we align both sets of techniques and use them interchangeably.

Interestingly, the algorithm of Steurer (2010) can be viewed as a dual-primal algorithm. This algorithm collects separating hyperplanes to solve the dual of the SDP: on failure to provide such a hyperplane, the algorithm provides a primal feasible $\mathbf{X}$. It is therefore unsurprising that the candidate $\mathbf{X}$ generated by Steurer's algorithm is an exponential of the (suitably scaled) averages of the hyperplanes (A, b): this would be the case if we were applying the multiplicative-weight update paradigm to the dual of the SDP in canonical form. Of course, to prove that such paradigms (Ahn &

Guha, 2015; Steurer, 2010) work requires some careful effort, and even after that the construction of the Oracle is never obvious. But the key point, reiterated here, is that the formulation plays an outsized role in terms of space efficiency, *both from the perspective of the state required to compute and the operational perspective of efficiently updating that state.* In future, we expect the space efficiency of solving convex optimization to be increasingly important.

3.1. min-disagree with Arbitrary Weights

In this section, we sketch the following theorem:

Theorem 13. *There is a $3(1+\epsilon)\log|E^-|$-approximation algorithm for* min-disagree *that requires $\tilde{O}((n\epsilon^{-2} + |E^-|)^2\epsilon^{-3})$ time, $\tilde{O}(n\epsilon^{-2}+|E^-|)$ space, and a single pass.*

We apply Theorem 12 (the multiplicative-weight update framework) to the dual of LP1, but omit the constraints in the dual corresponding to small-weight edges. For each $\alpha \geq 0$, let $H^+(\alpha), E^-(\alpha)$ be the set of edges in H^+, E^-, respectively, with weight at least $\delta\alpha/(m'+|E^-|)$. Note that ignoring edges not in $E^-(\alpha), H^+(\alpha)$ will only decrease $\sum_p y_p$ by at most $2\delta\alpha$. Consider:

$$\begin{aligned}
&\sum_p y_p \geq (1-2\delta)\alpha \\
&\frac{1}{|w_{ij}|}\sum_{p\in P_{ij}(H^+(\alpha))} y_p \leq 1 \qquad \forall ij \in E^-(\alpha) \\
&\frac{1}{w^h_{sq}}\sum_{p\in\mathbf{P}:sq\in p} y_p \leq 1 \qquad \forall sq \in H^+(\alpha) \\
&y_p \geq 0 \qquad \forall p \in \mathbf{P}(\alpha) \qquad \text{(LP2)}
\end{aligned}$$

where $\mathbf{P}(\alpha) = \bigcup_{ij\in E^-(\alpha)} P_{ij}(H^+(\alpha))$.

We attempt to find an approximate feasible solution to LP2 for a large value of α. If the Oracle fails to make progress then it provides a solution to LP1 of value $f\cdot\alpha$. In that case we set $\alpha \leftarrow \alpha/(1+\delta)$ and try the Oracle again. Note that if we lower α then the Oracle invocations for larger values of α continue to remain valid; if $\alpha_1 \leq \alpha_2$, then $P_{ij}(H^+(\alpha_1)) \supseteq P_{ij}(H^+(\alpha_2))$. Eventually we lower α sufficiently that we have a feasible solution to LP2. But then we have a solution to LP1 returned by the Oracle of value $f\alpha(1+\delta)$, corresponding to the previous value of α. The feasibility of LP2 shows that $\alpha(1-2\delta)$ is a lower bound for the optimum solution of LP1.

The Oracle is provided in Algorithm 1 and relies on the following lemma:

Lemma 14. *Let $\kappa = |E^-|$, $Z = \sum_{uv\in H^+(\alpha)} x_{uv}w^h_{uv}$. Let*

$$\mathrm{vol}(B(u,r)) = \frac{Z}{\kappa} + \sum_{\substack{vv'\in H^+(\alpha)\\ v,v'\in B(u,r)}} x_{vv'}w^h_{vv'} + \sum_{\substack{vv'\in H^+(\alpha)\\ d^x(u,v)\leq r<d^x(u,v')}} (r-d^x(u,v))w^h_{vv'}\,.$$

Algorithm 1 Oracle for LP2

1: Given multipliers u^t_{sq} for $sq \in H^+(\alpha)$ and v^t_{ij} for $ij \in E^-(\alpha)$, define $Q_u = \sum_{sq\in H^+(\alpha)} w^h_{sq}u^t_{sq}$ and $Q_v = \sum_{ij\in E^-(\alpha)} |w_{ij}|v^t_{ij}$.
2: Let $x_{sq} = \alpha u^t_{sq}/(Q_u+Q_v)$, $z_{ij} = \alpha v^t_{ij}/(Q_u+Q_v)$.
3: Treating the x_{sq} as edge lengths, let $d^x(\cdot,\cdot)$ be the shortest path metric. Define the *ball* of radius r centered at ζ:

$$B(\zeta, r) = \{\zeta' \mid d^x(\zeta,\zeta') \leq r\}$$

and the weight of the edges of cut by the ball:

$$\mathrm{cut}(B(\zeta,r)) = \sum_{\substack{\zeta'\zeta''\in H^+(\alpha)\\ d^x(\zeta,\zeta')\leq r<d^x(\zeta,\zeta'')}} w^h_{\zeta'\zeta''}$$

4: Find a collection of balls $B(\zeta_1,r_1), B(\zeta_2,r_2),\ldots$ such that (i) each radius at most $1/3$, (ii) every endpoint of an edge in $E^-(\alpha)$ belongs to some ball, and (iii) $\sum_g \mathrm{cut}(B(\zeta_g,r_g)) \leq 3\alpha Q_u/(Q_u+Q_v)\cdot\ln(|E^-|+1)$. The existence of such balls follows from Lemma 14.
5: **if** there exists $ij \in E^-(\alpha)$ with i,j in the same ball and $z_{ij} < 1/3$. **then**
6: Find the corresponding path p between i and j. Since the length of this path is at most $2/3$ and $z_{ij} < 1/3$, the corresponding constraint is violated. Return $y_p = \alpha$ and $y_{p'} = 0$ for all other paths for edges in E^-.
7: **else**
8: Return the union of cuts defined by the balls.

Suppose that, for a node ζ, the radius r of its ball is increased until $\mathrm{cut}(B(\zeta,r)) \leq C\mathrm{vol}(B(\zeta,r))$. *If $C = 3\ln(\kappa+1)$, the ball stops growing before the radius becomes $1/3$. We start this process setting ζ_1 to be an arbitrary endpoint of an edge in E^-, and let the stopping radius be r_1. We remove $B(\zeta_1,r_1)$ and continue the process on the remainder of the graph. The collection of $B(\zeta_1,r_1), B(\zeta_2,r_2),\ldots$ satisfy the condition that each radius is at most $1/3$ and $\sum_g$ cut$(B(\zeta_g,r_g)) \leq CZ$.*

The above lemma is essentially applying the result of Garg et al. (1993) to $H^+(\alpha)$ with the terminal pairs defined by E^-. The proof follows from the fact that $\mathrm{cut}(B(\zeta,r))$ is the derivative of $\mathrm{vol}(B(\zeta,r))$ w.r.t. r, and the volume cannot increase by more than a factor of $\kappa+1$. For nonnegative x_{sq}, standard shortest-path algorithms lead to a running time of $\tilde{O}(m')$.

Since we removed all small edge weights, we may bound the width of the above oracle as follows :

Lemma 15. *$\rho = (m' + |E^-|)/\delta$, $\ell = 1$ for Algorithm 1.*

In fact, we prove a general oracle construction that provides small width for every problem targeted by Theorem 12. The total weight of positive edges cut by the solution returned in line 8 of Algorithm 1 is at most $3\alpha Q_u/(Q_u+Q_v)\cdot\ln(|E^-|+1)$. Each negative edge that is not cut corresponds to setting $z_{ij} = 1$ but $z_{ij} \geq 1/3$; hence the cost of these edges is $\frac{3\alpha Q_v}{Q_u+Q_v}$. Finally, the cost of the edges in neither $E^-(\alpha)$ nor $H^+(\alpha)$ is at most $2\delta\alpha$. The overall solution has cost $(3\ln(|E^-|+1)+2\delta)\alpha$.

Finally, we show how to initialize α. Divide the edges of H^+ according to weight, in intervals $(2^{z-1}, 2^z]$, as we decrease z. For each group z, we find the largest weight edge $ij \in E^-$, call this weight $g(z)$, such that i and j are connected by H^+-edges of group z or higher. Observe that $g(z)$ is an increasing function of z. Let the smallest z such that $g(z) \geq 2^z$ be z_0. Then it follows that the optimum solution is at least 2^{z_0-1}. Again, $2^{z_0}n^2$ serves as an initial value of α, which is a $O(n^2)$ approximation to the optimum solution.

3.2. max-agree with Arbitrary Weights

In this section, we sketch the following theorem:

Theorem 16. *There is a $0.7666(1-\epsilon)$-approximation algorithm for* max-agree(G) *that uses $\tilde{O}(n\epsilon^{-2})$ space, $\tilde{O}(m + n\epsilon^{-10})$ time and a single pass.*

We use Lemma 6 and edge set $H = H^+ \cup H^-$. Let w^h_{ij} correspond to the weight of an edge $ij \in H$. Our SDP for max-agree is:

$$\sum_{ij\in H^+} w^h_{ij}\mathbf{X}_{ij} + \sum_{ij\in H^-} \frac{|w^h_{ij}|(\mathbf{X}_{ii} + \mathbf{X}_{ij} - 2\mathbf{X}_{ij})}{2} \geq \alpha$$

$$\begin{aligned} \mathbf{X}_{ii} &\leq 1 && \forall i \in V \\ -\mathbf{X}_{ii} &\leq -1 && \forall i \in V \\ -\mathbf{X}_{ij} &\leq 0 && \forall i,j \in V \\ \mathbf{X} &\succeq \mathbf{0} \end{aligned} \tag{SDP}$$

If two vertices, i and j, are in the same cluster, their corresponding vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ will coincide, so $\mathbf{X}_{ij} = 1$; on the other hand, if they are in different clusters, their vectors should be orthogonal, so $\mathbf{X}_{ij} = 0$. Observe that under the restriction $\mathbf{X}_{ii} = \mathbf{X}_{jj} = 1$, the contribution of an $ij \in H^-$ is $\mathbf{X}_{ii} + \mathbf{X}_{ij} - 2\mathbf{X}_{ij} = (1 - \mathbf{X}_{ij})$, as intended. However, this formulation helps prove that the width is small.

Definition 2. *Define $d_i = \sum_{j:ij\in H} |w^h_{ij}|$ and $\sum_i d_i = 2W$. Let $\mathbf{D}$ be the diagonal matrix with $\mathbf{D}_{ii} = d_i/2W$.*

A random partition of the graph provides a trivial $1/2$-approximation for maximizing agreements. Letting W be the total weight of edges in H, the sparsified graph, we perform binary search for $\alpha \in [W/2, W]$, and stop when the interval is of size δW. This increases the running time by a $O(\log \delta^{-1})$ factor.

The diagonal matrix $\mathbf{D}$ specified in Definition 2 sets up the update algorithm of Steurer (2010). The choice of $\mathbf{D}$ will be critical to our algorithm: typically, this $\mathbf{D}$ determines the "path" taken by the SDP solver, since $\mathbf{D}$ alters the projection to density matrices. Summarizing, Theorem 16 follows from the Oracle provided in Algorithm 2. The final solution only guarantees $\mathbf{x}_i \cdot \mathbf{x}_j \geq -\delta$. Even though the standard rounding algorithm assumes $\mathbf{X}_{ij} \geq 0$, the fractional solution with $\mathbf{X}_{ij} \geq -\delta$ can be rounded efficiently. Ensuring $\mathbf{x}_i \cdot \mathbf{x}_j \geq 0$ appears to be difficult (or to require a substantially different oracle).

Algorithm 2 Oracle for SDP.

1: For the separating hyperplane, we only describe non-zero entries in $\mathbf{A}$. Recall that we have a candidate $\mathbf{X}$ where $\mathbf{X}_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j$.
2: Let $S_1 = \{i : \|\mathbf{x}_i\|^2 \geq 1 + \delta\}$, $\Delta_1 = \sum_{i\in S_1} d_i$.
3: Let $S_2 = \{i : \|\mathbf{x}_i\|^2 \leq 1 - \delta\}$, $\Delta_2 = \sum_{i\in S_2} d_i$.
4: Let $S_3 = \{ij : \mathbf{x}_i \cdot \mathbf{x}_j < -\delta\}$, $\Delta_3 = \sum_{ij\in S_3} |w_{ij}|$.
5: **if** $\Delta_1 \geq \delta\alpha$ **then**
6: Let $\mathbf{A}_{ii} = -d_i/\Delta_1$ for $i \in S_1$ and $b = -1$.
7: Return $(\mathbf{A}, b)$.
8: **else if** $\Delta_2 \geq \delta\alpha$ **then**
9: Let $\mathbf{A}_{ii} = d_i/\Delta_2$ for $i \in S_2$ and $b = 1$.
10: Return $(\mathbf{A}, b)$.
11: **else if** $\Delta_3 \geq \delta\alpha$ **then**
12: Let $\mathbf{A}_{ij} = w^h_{ij}/\Delta_3$ for $ij \in S_3$ and $b = 0$.
13: Return $(\mathbf{A}, b)$.
14: **else**
15: Ignore all nodes in S_1 and S_2 and all edges in S_3. Let $\mathbf{C}'$ be the matrix that corresponds to the objective function of the modified graph G'.
16: **if** $\mathbf{C}' \circ \mathbf{X} < (1 - 4\delta)\alpha$ **then**
17: Let $\mathbf{A} = \mathbf{C}'/\alpha$ and $b = 1 - 3\delta$. Return $(\mathbf{A}, b)$.
18: **else**
19: Round $\mathbf{X}$, and return the rounded solution.

Lemma 17. *Algorithm 2 is δ-separating: for all returned $(\mathbf{A}, b)$, $\mathbf{A} \circ \mathbf{X} \leq b - \delta$ and $\forall \mathbf{X}' \in \mathcal{X}$, $\mathbf{A} \circ \mathbf{X}' \geq b$ where $\mathcal{X}$ is the feasible space of SDP.*

Lemma 18. *For $\rho = O(1/\delta)$, Algorithm 2 is ρ-bounded: $\rho\mathbf{D} \succeq \mathbf{A} - b\mathbf{D} \succeq -\rho\mathbf{D}$*

The update procedure (Steurer, 2010) maintains (and defines) the candidate vector $\mathbf{X}$ implicitly. In particular it uses matrices of dimension $n \times d$, in which every entry is a (scaled) Gaussian random variable. The algorithm also uses a precision parameter (degree of the polynomial approximation to represent matrix exponentials) r. Assuming that T_M is the time for a multiplication between a returned $\mathbf{A}$ and some vector, the update process computes the tth $\mathbf{X}$ in time

$O(t \cdot r \cdot d \cdot T_M)$, a quadratic dependence on t in total. We will ensure that any returned $\mathbf{A}$ has at most m' nonzero entries, and therefore $T_M = O(m')$. The algorithm requires space that is sufficient to represent a linear combination of the matrices $\mathbf{A}$ which are returned in the different iterations. We can bound $\rho = O(1/\delta)$, and therefore the total number of iterations is $\tilde{O}(\delta^{-4})$. For our purposes, in max-agree we will have $d = O(\delta^{-2} \log n)$, $r = O(\log(\delta^{-1})$, and $T_M = O(m')$, giving us a $\tilde{O}(n\delta^{-10})$ time and $\tilde{O}(n\delta^{-2})$ space algorithm. However, unlike the general $\mathbf{X}$ used in Steurer's approach, in our oracle the $\mathbf{X}$ is used in a very specific way. This leaves open the question of determining the exact space-versus-running-time tradeoff.

4. Multipass Algorithms

4.1. min-disagree with Unit Weights

Consider the 3-approximation algorithm for min-disagree on unit-weight graphs due to Ailon et al. (2008).

1: Let $v_1, \ldots, v_n$ be a uniformly random ordering of V. Let $U \leftarrow V$ be the set of "uncovered" nodes.
2: **for** $i = 1$ to n **do**
3: **if** $v_i \in U$ **then**
4: Define $C_i \leftarrow \{v_i\} \cup \{v_j \in U : v_i v_j \in E^+\}$ and let $U \leftarrow U \setminus C_i$. We say v_i is "chosen".
5: **else**
6: $C_i \leftarrow \emptyset$.
7: **Return** the collection of non-empty sets C_i.

It may appear that emulating the above algorithm in the data stream model requires $\Omega(n)$ passes, since determining whether v_i should be chosen may depend on whether v_j is chosen for each $j < i$. However, we will show that $O(\log \log n)$-passes suffice. This improves upon a result by Chierichetti et al. (2014), who developed a modification of the algorithm that used $O(\epsilon^{-1} \log^2 n)$ streaming passes and returned a $(3+\epsilon)$-approximation, rather than a 3-approximation. Our improvement is based on the following:

Lemma 19. *Let U_t be the set of uncovered nodes after iteration t of the above algorithm, and let*

$$F^+_{t,t'} = \{v_i v_j \in E^+, i, j \in U_t, t < i, j \leq t'\} .$$

With high probability, $|F^+_{t,t'}| \leq 5 \cdot \ln n \cdot t'^2/t$.

Semi-Streaming Algorithm. Our semi-streaming algorithm proceeds as follows. For $j \geq 1$, let $t_j = (2n)^{1-1/2^j}$: during the $(2j-1)$-th pass, we store all edges in $F^+_{t_{j-1},t_j}$, and during the $(2j)$-th pass we determine U_{t_j}. After the $(2j)$-th pass we have simulated the first t_j iterations of Ailon et al.'s algorithm. Since $t_j \geq n$ for $j = 1 + \log \log n$, our algorithm terminates after $O(\log \log n)$ passes.

Theorem 20. *On a unit-weight graph, there exists a $O(\log \log n)$-pass semi-streaming algorithm that returns with high probability a 3-approximation to* min-disagree.

4.2. min-disagree$_k$ with Unit Weights

Our result in this section is based the following algorithm of Giotis & Guruswami (2006) that returns a $(1+\epsilon)$-approximation for min-disagree$_k$ on unit-weight graphs. The algorithm samples $m = \text{poly}(1/\epsilon, k) \cdot \log n$ nodes S and for every possible k-partition $\{S_i\}_{i \in [k]}$ of S computes the cost of the clustering where $v \in V \setminus S$ is assigned to the ith cluster if

$$i = \operatorname*{argmax}_j \left(\sum_{s \in S_j : sv \in E^+} w_{sv} + \sum_{s \notin S_j : sv \in E^-} |w_{sv}| \right) .$$

Let $\mathcal{C}'$ be the best clustering found. If all clusters in $\mathcal{C}'$ have at least $n/(2k)$ nodes, return $\mathcal{C}'$. Otherwise, fix all the clusters of size at least $n/(2k)$ and recurse on the set of nodes in "small" clusters. To emulate each recursive step in one pass, we simply choose S at the start of the stream and then collect all incident edges on S. We then use the disagree oracle developed in Section 2.1 to find the best possible partitions during post-processing. To design an $O(\log \log n)$-pass algorithm, we proceed as follows. In the i-th pass, suppose we have k' clusters still to determine and let V_{i-1} be the set of nodes that have not yet been included in some (large) cluster. We pick k' random sets of samples $S_1, \ldots, S_{k'}$ in parallel from V_{i-1} each of size $N_i = mn^{2^{i-1}/p}$, for $p = \log n$. Since $N_i \geq n$ for $i \geq 1 + \log \log n$, the algorithm must terminate in $O(\log \log n)$ passes. We prove that in pass i the number of clusters drops by $n^{2^{i-1}/p}$ from the start of the last pass, and inductively, that $|V_i| = n/n^{(2^i-1)/p}$. The space needed by our algorithm for round i is therefore $O(k' N_i |V_i|) = O(kmn^{1+1/p}) = \tilde{O}(kmn)$.

Theorem 21. *There exists a $\min(k-1, \log \log n)$-pass semi-streaming algorithm that $(1+\epsilon)$-approximates* min-disagree$_k$ *with high probability.*

Acknowledgements

Graham Cormode is supported by a Royal Society Wolfson Research Merit Award and the Yahoo Faculty Research Engagement Program. Sudipto Guha is supported by NSF award CCF-1117216. Andrew McGregor is supported by NSF awards CCF-0953754. CCF-1320719, IIS-1251110 and a Google Faculty Research Award. Anthony Wirth is supported by ARC Future Fellowship FT120100307.

References

Ahn, Kook Jin and Guha, Sudipto. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation (ICALP 2011 Special Issue)*, 222:59–79, 2013.

Ahn, Kook Jin and Guha, Sudipto. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In *Symposium on Parallelism in Algorithms and Architectures (SPAA). Also in CORR, arXiv 1307.4359*, 2015. URL http://dx.doi.org/10.1145/2755573.2755586.

Ahn, Kook Jin, Guha, Sudipto, and McGregor, Andrew. Analyzing graph structure via linear measurements. In *Symposium on Discrete Algorithms: SODA*, pp. 459–467, 2012a. URL http://portal.acm.org/citation.cfm?id=2095156&CFID=63838676&CFTOKEN=79617016.

Ahn, Kook Jin, Guha, Sudipto, and McGregor, Andrew. Graph sketches: sparsification, spanners, and subgraphs. In *Principles of Database Systems: PODS*, pp. 5–14, 2012b. URL http://doi.acm.org/10.1145/2213556.2213560.

Ailon, Nir and Karnin, Zohar Shay. A note on: No need to choose: How to get both a PTAS and sublinear query complexity. *CoRR*, abs/1204.6588, 2012.

Ailon, Nir, Charikar, Moses, and Newman, Alantha. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5), 2008. URL http://doi.acm.org/10.1145/1411509.1411513.

Ailon, Nir, Jaiswal, Ragesh, and Monteleoni, Claire. Streaming k-means approximation. In *Conference on Neural Information Processing Systems: NIPS*, pp. 10–18, 2009. URL http://books.nips.cc/papers/files/nips22/NIPS2009_1085.pdf.

Arora, Sanjeev and Kale, Satyen. A combinatorial, primal-dual approach to semidefinite programs. In *ACM Symposium on Theory of Computing: STOC*, pp. 227–236, 2007. URL http://doi.acm.org/10.1145/1250790.1250823.

Arora, Sanjeev, Hazan, Elad, and Kale, Satyen. The multiplicative weights update method: a meta algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. URL http://www.theoryofcomputing.org/articles/v008a006.

Bagon, S. and Galun, M. Large scale correlation clustering optimization. *arXiv:1112.2903v1*, 2011.

Bansal, Nikhil, Blum, Avrim, and Chawla, Shuchi. Correlation clustering. *Machine Learning*, 56(1–3):89–113, 2004. URL http://dx.doi.org/10.1023/B:MACH.0000033116.57574.95.

Benczúr, András A. and Karger, David R. Approximating $s-t$ minimum cuts in $\tilde{O}(n^2)$ time. In *Symposium on Theory of Computing: STOC*, pp. 47–55, 1996.

Bonchi, Francesco, Garcia-Soriano, David, and Liberty, Edo. Correlation clustering: From theory to practice. In *International Conference on Knowledge Discovery and Data Mining: KDD*, pp. 1972–1972, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2956-9. URL http://doi.acm.org/10.1145/2623330.2630808.

Braverman, Vladimir, Chung, Kai-Min, Liu, Zhenming, Mitzenmacher, Michael, and Ostrovsky, Rafail. AMS without 4-wise independence on product domains. In *International Symposium on Theoretical Aspects of Computer Science: STACS*, pp. 119–130, 2010. URL http://dx.doi.org/10.4230/LIPIcs.STACS.2010.2449.

Charikar, Moses, O'Callaghan, Liadan, and Panigrahy, Rina. Better streaming algorithms for clustering problems. In *Symposium on Theory of Computing: STOC*, pp. 30–39, 2003. URL http://doi.acm.org/10.1145/780542.780548.

Charikar, Moses, Chekuri, Chandra, Feder, Tomás, and Motwani, Rajeev. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. URL http://dx.doi.org/10.1137/S0097539702418498.

Charikar, Moses, Guruswami, Venkatesan, and Wirth, Anthony. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005. URL http://dx.doi.org/10.1016/j.jcss.2004.10.012.

Chawla, Shuchi, Makarychev, Konstantin, Schramm, Tselil, and Yaroslavtsev, Grigory. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In *Symposium on Theory of Computing: STOC*, 2015.

Chierichetti, Flavio, Dalvi, Nilesh N., and Kumar, Ravi. Correlation clustering in mapreduce. In *International Conference on Knowledge Discovery and Data Mining: KDD*, pp. 641–650, 2014. URL http://doi.acm.org/10.1145/2623330.2623743.

Coleman, Tom, Saunderson, James, and Wirth, Anthony. A local-search 2-approximation for 2-correlation-clustering. In *European Symposium on Algorithms: ESA*, pp. 308–319, 2008. URL http://dx.doi.org/10.1007/978-3-540-87744-8_26.

Demaine, Erik D., Emanuel, Dotan, Fiat, Amos, and Immorlica, Nicole. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2–3):172–187, 2006. URL http://dx.doi.org/10.1016/j.tcs.2006.05.008.

Elsner, Micha and Schudy, Warren. Bounding and comparing methods for correlation clustering beyond ilp. In *Workshop on Integer Linear Programming for Natural Langauge Processing: ILP*, pp. 19–27, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-35-0. URL http://dl.acm.org/citation.cfm?id=1611638.1611641.

Feigenbaum, Joan, Kannan, Sampath, McGregor, Andrew, Suri, Siddharth, and Zhang, Jian. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2–3):207–216, 2005. URL http://dx.doi.org/10.1016/j.tcs.2005.09.013.

Garg, Naveen, Vazirani, Vijay V., and Yannakakis, Mihalis. Approximate max-flow min-(multi)cut theorems and their applications. In *ACM Symposium on Theory of Computing: STOC*, pp. 698–707, 1993. URL http://doi.acm.org/10.1145/167088.167266.

Giotis, Ioannis and Guruswami, Venkatesan. Correlation clustering with a fixed number of clusters. *Theory of Computing*, 2(1): 249–266, 2006. URL http://dx.doi.org/10.4086/toc.2006.v002a013.

Gramm, Jens, Guo, Jiong, Hüffner, Falk, and Niedermeier, Rolf. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory Comput. Syst.*, 38(4):373–392, 2005. URL http://dx.doi.org/10.1007/s00224-004-1178-y.

Guha, Sudipto. Tight results for clustering and summarizing data streams. In *International Conference on Database Theory: ICDT*, pp. 268–275, 2009. URL http://doi.acm.org/10.1145/1514894.1514926.

Guha, Sudipto, Mishra, Nina, Motwani, Rajeev, and O'Callaghan, Liadan. Clustering data streams. In *IEEE Foundations of Computer Science: FOCS*, pp. 359–366, 2000. URL http://doi.ieeecomputersociety.org/10.1109/SFCS.2000.892124.

Indyk, Piotr and McGregor, Andrew. Declaring independence via the sketching of sketches. In *Symposium on Discrete Algorithms: SODA*, pp. 737–745, 2008. URL http://dl.acm.org/citation.cfm?id=1347082.1347163.

Kane, Daniel M., Nelson, Jelani, and Woodruff, David P. On the exact space complexity of sketching and streaming small norms. In *ACM-SIAM Symposium on Discrete Algorithms: SODA*, pp. 1161–1178, 2010. URL http://dx.doi.org/10.1137/1.9781611973075.93.

McCutchen, Richard Matthew and Khuller, Samir. Streaming algorithms for k-center clustering with outliers and with anonymity. *International Workshop on Approximation Algorithms for Combinatorial Optimization: APPROX*, pp. 165–178, 2008. doi: 10.1007/978-3-540-85363-3_14.

McGregor, Andrew. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014. URL http://doi.acm.org/10.1145/2627692.2627694.

Silva, Jonathan A., Faria, Elaine R., Barros, Rodrigo C., Hruschka, Eduardo R., Carvalho, André C. P. L. F. de, and Gama, João. Data stream clustering: A survey. *ACM Comput. Surv.*, 46 (1):13:1–13:31, July 2013. ISSN 0360-0300. URL http://doi.acm.org/10.1145/2522968.2522981.

Steurer, David. Fast sdp algorithms for constraint satisfaction problems. In *Symposium on Discrete Algorithms: SODA*, pp. 684–697, 2010.

Swamy, Chaitanya. Correlation clustering: maximizing agreements via semidefinite programming. In *Symposium on Discrete Algorithms: SODA*, pp. 526–527, 2004. URL http://doi.acm.org/10.1145/982792.982866.

Wirth, Anthony Ian. *Approximation Algorithms for Clustering*. PhD thesis, Princeton University, 2004.