Author/s:
Ahn, KJ;Cormode, G;Guha, S;McGregor, A;Wirth, A

Title:
Correlation Clustering in Data Streams

Date:
2021-07-01

Citation:
Ahn, K. J., Cormode, G., Guha, S., McGregor, A. & Wirth, A. (2021). Correlation Clustering in Data Streams. Algorithmica, 83 (7), pp.1980-2017. https://doi.org/10.1007/s00453-021-00816-9.

Persistent Link:
https://hdl.handle.net/11343/274029

Algorithmica
https://doi.org/10.1007/s00453-021-00816-9

Correlation Clustering in Data Streams

Kook Jin Ahn[1] · Graham Cormode[2] · Sudipto Guha[1] · Andrew McGregor[3] · Anthony Wirth[4]

Received: 9 November 2018 / Accepted: 19 February 2021

Abstract
Clustering is a fundamental tool for analyzing large data sets. A rich body of work has been devoted to designing data-stream algorithms for the relevant optimization problems such as k-center, k-median, and k-means. Such algorithms need to be both time and and space efficient. 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, $O(n \cdot \operatorname{polylog} n)$-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. Unfortunately, the standard LP and SDP formulations are not obviously solvable in $O(n \cdot \operatorname{polylog} n)$-space. Our work presents space-efficient algorithms for the convex programming required, as well as approaches to reduce the adaptivity of the sampling.

Keywords Correlation clustering · Data streams · Linear sketches · Linear programming

1 Introduction

Correlation Clustering is a widely studied model of clustering within graphs where edges are marked as positive or negative. The aim is to choose clusters so that most edges within clusters are positive, while most edges between clusters are negative. The correlation clustering problem was initially proposed for complete unweighted graphs. The motivation for this formulation is an intuitive one: there are many

✉ Graham Cormode
G.Cormode@warwick.ac.uk

Extended author information available on the last page of the article

Published online: 13 March 2021

scenarios where some measure of affinity between entities can be quantified (say, in social networks), and we seek to group entities into clusters of similar objects.

The correlation clustering problem was first formulated as an optimization problem by Bansal, Blum and Chawla [12]. 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. The goal is to find the partition $\mathcal{C}$ that maximizes

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

or, equivalently, that minimizes $\mathsf{disagree}(G, \mathcal{C}) := \sum_{uv} |w_{uv}| - \mathsf{agree}(G, \mathcal{C})$. Solving this problem exactly is known to be NP-hard, but a constant factor approximation algorithm for the minimization problem was one of the first results on this problem [6].

Since it was first studied, a large body of work [6, 12, 17, 21, 30, 48] has been devoted to approximating $\mathsf{max{-}agree}(G) = \max_{\mathcal{C}} \mathsf{agree}(G, \mathcal{C})$ and $\mathsf{min{-}disagree}(G) = \min_{\mathcal{C}} \mathsf{disagree}(G, \mathcal{C})$, along with variants $\mathsf{min{-}disagree}_k(G)$ and $\mathsf{max{-}agree}_k(G)$, where we consider partitions with at most k clusters. In this paper, we focus on multiplicative approximation results. If all weights are ± 1, there is a polynomial time approximation scheme (PTAS) for $\mathsf{max{-}agree}$ [12, 30] and a 2.06-approximation for $\mathsf{min{-}disagree}$ [19]. When there is an upper bound, k, on the number of clusters in $\mathcal{C}$, and all weights are ± 1, Guruswami [30] introduced a PTAS for both problems. Even $k = 2$ is interesting, with an efficient local-search approximation introduced by Coleman, Saunderson and Wirth [21].

If the weights are arbitrary, there is a 0.7666-approximation for $\mathsf{max{-}agree}$ [17, 48] and an $O(\log n)$-approximation for $\mathsf{min{-}disagree}$ [17, 23]. These methods use convex programming: as originally described, this cannot be implemented in $O(n \operatorname{polylog} n)$ space, even when the input graph is sparse. This aspect is well known in practice, and Elsner and Schudy [25], Bagon and Galun [11], and Bonchi, Garcia Sorino and Liberty [14] discuss the difficulty of scaling the convex programming approach.

Applications. When first formulated, correlation clustering was a theoretical and rigorous attempt to answer the co-reference problem in natural language processing [12]. However, that followed a similar formulation, called *cluster editing*, whose genesis lay in clustering gene expression patterns [45]. The clustering aggregation and consensus clustering problems build on correlation clustering [29]. Meanwhile, the inconsistent soft constraints in correlation clustering model problems in duplicate detection [24, 37], record linkage [33] as well as crowdsourcing approaches to entity resolution [49, 50].

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 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 [18, 36], k-means [7], and k-center [16, 34, 42] have all been studied in the data stream model, as surveyed by Silva *et al.*[46]. Non-adaptive sampling algorithms for correlation clustering can be implemented in the data stream model, as applied by Ailon and Karnin [8], to construct *additive* approximations. Chierichetti, Dalvi and Kumar [20] presented the first multiplicative approximation data stream algorithm: a polynomial-time $(3+\varepsilon)$ -approximation for min−disagree on ± 1-weighted graphs using $O(\varepsilon^{-1}\log^2 n)$ passes and *semi-streaming* space — that is, a streaming algorithm using space as a function of n that is $\Theta(n\,\mathrm{polylog}\,n)$ [26]. Pan *et al.*[44] and Bonchi *et al.*[14] discuss faster non-streaming implementations of related ideas but the work of Chierichetti, Dalvi and Kumar [20] remained the state of the art data stream algorithm until our work. Using space roughly proportional to the number of nodes can be shown to be necessary for solving many natural graph problems including, it will turn out, correlation clustering. For a recent survey of semi-streaming algorithms and graph sketching see [43].

1.1 Computational Model

In the basic graph stream model, the input is a sequence of edges and their weights. For semi-streaming algorithms, the available space to process the stream and perform any necessary post-processing is $O(n\,\mathrm{polylog}\,n)$ bits, focusing on the dependence on n, rather than on parameters such as ε. 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 in the range $[-w_*, -1] \cup [1, w_*]$ for some constant $w_* \geq 1$; and (c) *arbitrary weights*, where all weights are in the range $[-w_*, w_*]$ and here $w_* = \mathrm{poly}\,(n)$. We denote the sets of positive-weight and negative-weight edges by E^+ and E^-, respectively, and define $G^+ = (V, E^+)$ and $G^- = (V, E^-)$. Within this streaming model of computation, we are concerned with how much space is required, and how many passes through the input data are needed.

We note that many of our algorithms, such as those based on sparsification [3], can also be implemented in MapReduce.

1.2 Techniques and Results

In Sect. 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 $\mathsf{agree}(G, \mathcal{C})$ or $\mathsf{disagree}(G, \mathcal{C})$. They are based on linear sketches and incorporate ideas from work on constructing graph sparsifiers via linear sketches. These data structures can be constructed in the semi-streaming model and can be queried in $\tilde{O}(n)$ time. As the algorithms rely on relatively simple matrix-vector operations, they can be implemented fairly easily in MapReduce.

In Sects. 3 and 4, we introduce several new ideas for solving the LP and SDP for $\mathsf{min-disagree}$ and $\mathsf{max-agree}$. In each case, the convex formulation must allow each candidate solution to be represented, verified, and updated in small space. But the key point made 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 the future, we expect the space efficiency of solving convex optimization to be increasingly important.

We discuss multipass for algorithms for $\mathsf{min-disagree}$ in Sect. 5. Our results are based on adapting existing algorithms that, if implemented in the data stream model, may appear to take $O(n)$ passes. However, with a more careful analysis we show that $O(\log \log n)$ passes are sufficient. Finally, we present space lower bounds in Sect. 6. These are proved using reductions from communication complexity and establish that many of our algorithms are space-optimal.

In more detail, our results on the different formulations of correlation clustering (whether the weights are unit, bounded or arbitrary; whether to maximize agreements, or minimize disagreements; and whether the number of clusters is fixed) are as follows.

Max-Agree. For $\mathsf{max-agree}$, we provide the following single-pass streaming algorithms, each needing $\tilde{O}(n\mathrm{poly}(k, \varepsilon^{-1}))$ space: (i) a polynomial-time $(1-\varepsilon)$-approximation for bounded weights (Theorem 4), and (ii) an algorithm with approximation factor $0.7666(1-\varepsilon)$ for arbitrary weights in $\tilde{O}(n\varepsilon^{-10})$ time (Theorem 11).

Min-Disagree We show that any constant pass algorithm that can test whether $\mathsf{min-disagree}(G) = 0$ in a single pass, for unit weights, must store $\Omega(n)$ bits (Theorem 15). For arbitrary weights, the lower bound increases to $\Omega(n + |E^-|)$ (Theorem 16) and to $\Omega(n^2)$ in the case the graph of negative edges may be dense (Theorem 14). We provide a single-pass algorithm that uses $s = \tilde{O}(n\varepsilon^{-2} + |E^-|)$ space and $\tilde{O}(s^2)$ time and provides an $O(\log |E^-|)$ approximation (Theorem 9). Since Demaine *et al.*[23] and Charikar *et al.*[17] provided approximation-preserving reductions from the "minimum multicut" problem to $\mathsf{min-disagree}$ 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 $\mathsf{min-disagree}(G) \leq t$, we provide a single-pass polynomial time algorithm that uses $\tilde{O}(n + \varepsilon^{-2}t)$ space (Theorem 2). We provide a $\tilde{O}(n\varepsilon^{-2})$-space PTAS for $\mathsf{min-disagree}_2$ for bounded weights (Theorem 6).

We also consider multiple-pass streaming algorithms. For unit weights, we present an algorithm with $O(\log \log n)$ passes that mimics the algorithm of Ailon *et al.*[6], and provides a 3-approximation in expectation (Theorem 12), improving on the result of Chierichetti *et al.*[20]. For $\mathsf{min-disagree}_k(G)$, on unit-weight graphs with $k \geq 3$, we give a $\min(k-1, O(\log \log n))$-pass polynomial-time algorithm using $\tilde{O}(n\mathrm{poly}(k, \varepsilon^{-1}))$ space (Theorem 13). This result is based on emulating an algorithm by Giotis and Guruswami [30] in the data stream model.

Table 1 Summary of approximation results in this paper

Section/Theorem	Problem	Weights	Passes	Space Bound	Approximation Factor
§2.1 Thm 1	disagree	Unit	1	$\tilde{O}(\varepsilon^{-2})$	$1+\varepsilon$
§2.1 Thm 2	min−disagree	Unit	1	$\tilde{O}(n+\varepsilon^{-2}t)$	$1+\varepsilon$ if min−disagree$(G)\le t$
§2.2 Thm 4	max−agree	Bounded	1	$O(n\,\mathrm{poly}\,(k,\varepsilon^{-1}))$	$1-\varepsilon$
§2.3 Thm 5	disagree$_2$	Arbitrary	1	$\tilde{O}(n\varepsilon^{-2})$	$1\pm\varepsilon$
§2.3 Thm 6	min−disagree$_2$	Bounded	1	$\tilde{O}(n\,\mathrm{poly}\,(\varepsilon^{-1}))$	$1+\varepsilon$
§3.4 Thm 9	min−disagree	Arbitrary	1	$\tilde{O}(n\varepsilon^{-2}+\lvert E^-\rvert)$	$3(1+\varepsilon)\log\lvert E^-\rvert$
§4 Thm 11	max−agree	Arbitrary	1	$\tilde{O}(n\varepsilon^{-2})$	$0.7666(1-\varepsilon)$
§5.1 Thm 12	min−disagree	Unit	$\log\log n$	$\tilde{O}(n)$	3
§5.2 Thm 13	min−disagree$_k$	Unit	$\log\log n$	$\tilde{O}(n\,\mathrm{poly}\,(k,\varepsilon^{-1}))$	$1+\varepsilon$
§6 Thm 15	min−disagree	Unit	p	$\Omega(n/p)$	Any
§6 Thm 16	min−disagree	Arbitrary	1	$\Omega(n+\lvert E^-\rvert)$	Any
§6 Thm 17	disagree$_3$	Arbitrary	1	$\Omega(n^2)$	Any

We summarize all our results in Table 1 in the order that they appear subsequently. The table shows the various problems (using the notation introduced in Sect. 1), based on how many passes are used, and whether edge weights are unit, bounded by a constant, or arbitrary (see Sect. 1.1).

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 need to exploit them carefully to design more efficient solutions. Meanwhile, in this section, we present a short application of each data structure that illustrates their utility.

2.1 First Data Structure: Bilinear Sketch

Consider a graph G with unit weights ($w_{ij}\in\{-1,1\}$) and a clustering $\mathcal{C}$. Our first data structure allows us to solve the *query problem*, which is, given G and $\mathcal{C}$, to report (an approximation of) $\mathsf{disagree}(G,\mathcal{C})$. Define the matrices M^G and $M^{\mathcal{C}}$ where $M^G_{ij}=\max(0,w_{ij})$ and

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

Note that if $w_{ij}=1$, then

$$(M^G_{ij} - M^{\mathcal{C}}_{ij})^2 = (1 - M^{\mathcal{C}}_{ij})^2 = I[i \text{ and } j \text{ are separated in } \mathcal{C}]$$

whereas, if $w_{ij} = -1$ then

$$(M^G_{ij} - M^{\mathcal{C}}_{ij})^2 = (M^{\mathcal{C}}_{ij})^2 = I[i \text{ and } j \text{ are not separated in } \mathcal{C}] .$$

Hence, 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 and McGregor [38]. The basic sketch is as follows:

1. Let $\alpha \in \{-1, 1\}^n$ and $\beta \in \{-1, 1\}^n$ be independent random vectors whose entries are 4-wise independent; in a single pass over the input, compute

$$Y = \sum_{ij \in E^+} \alpha_i \beta_j.$$

   Specifically, we maintain a counter that is initialized to 0 and for each $ij \in E^+$ in the stream we add $\alpha_i \beta_j$ to the counter and if $ij \in E^+$ is deleted we subtract $\alpha_i \beta_j$ from the counter; the final value of the counter equals Y. Note that α and β can be determined by a hash function that can be stored in $\tilde{O}(1)$ space such that each entry can be constructed in $\tilde{O}(1)$ time.
2. Given query partition $\mathcal{C} = \{C_1, C_2, \ldots\}$, return $X = \left(Y - \sum_\ell \left(\sum_{i \in C_\ell} \alpha_i\right)\left(\sum_{i \in C_\ell} \beta_i\right)\right)^2$.

To analyze the algorithm we will need the following lemma due to Indyk and McGregor [38] and Braverman *et al.*[15].

Lemma 1 *Given n^2 values $\{f_{ij} \in \mathbb{R}\}_{i,j \in [n]}$, we have for each f_{ij}, $\mathbb{E}\left[(\sum_{i,j} \alpha_i \beta_j f_{ij})^2\right] = \sum_{i,j} f_{ij}^2$ and $\mathbb{V}\left[(\sum_{i,j} \alpha_i \beta_j f_{ij})^2\right] \leq 9(\sum_{i,j} f_{ij}^2)^2$.*

The following theorem will be proved by considering an algorithm that computes multiple independent copies of the above sketch and combines the estimates from each.

Theorem 1 *For unit weights, there exists an $O(\varepsilon^{-2} \log(\delta^{-1}) \log(n))$-space algorithm for the* disagree *query problem which succeeds, i.e., returns a $1 + \varepsilon$-approximate solution, with probability $1 - \delta$. Each positive edge is processed in $\tilde{O}(\varepsilon^{-2})$ time, while the query time is $\tilde{O}(\varepsilon^{-2} n)$.*

Proof We first observe that, given Y, the time to compute X is $\tilde{O}(n)$. This follows because for a cluster $C_\ell \in \mathcal{C}$, on n_ℓ nodes, we can compute $\sum_{i \in C_\ell} \alpha_i$ and $\sum_{i \in C_\ell} \beta_i$ in $\tilde{O}(n_\ell)$ time. Hence the total query time is $\tilde{O}(\sum_\ell n_\ell) = \tilde{O}(n)$ as claimed.

We next argue that repeating the above scheme a small number of times in parallel yields a good estimate of $\mathsf{disagree}(G, \mathcal{C})$. To do this, note that

$$X = \left(\sum_{ij \in E^+} \alpha_i \beta_j - \sum_\ell \left(\sum_{i \in C_\ell} \alpha_i \right) \left(\sum_{i \in C_\ell} \beta_i \right) \right)^2 = \left(\sum_{ij} \alpha_i \beta_j (M^G_{ij} - M^{\mathcal{C}}_{ij}) \right)^2 .$$

We then apply Lemma 1 to $f_{ij} = M^G_{ij} - M^{\mathcal{C}}_{ij}$ and deduce that

$$\mathbb{E}[X] = \mathsf{disagree}(G, \mathcal{C}) \quad \text{and} \quad \mathbb{V}[X] \leq 9(\mathsf{disagree}(G, \mathcal{C}))^2 .$$

Hence, running $O(\varepsilon^{-2} \log \delta^{-1})$ parallel repetitions of the scheme and averaging the results appropriately yields a $(1 \pm \varepsilon)$-approximation for $\mathsf{disagree}(G, \mathcal{C})$ with probability at least $1 - \delta$. Following a fairly standard approach, we partition the estimates into $O(\log \delta^{-1})$ groups, each of size $O(\varepsilon^{-2})$ (see [22, Section 1.4.1] for example). We can ensure that with probability at least 2/3, the mean of each group is within a $1 \pm \varepsilon$ factor by an application of the Chebyshev bound; we then argue using the Chernoff bound that the median of the resulting group estimates is a $1 \pm \varepsilon$ approximation with probability at least $1 - \delta$. □

Remark 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 $\tilde{O}(\varepsilon^{-2} n)$ space. Hence, given exponential time, we can also $(1 + \varepsilon)$-approximate $\mathsf{min-disagree}(G)$. While this is near-optimal in terms of space, in this paper we focus on polynomial-time algorithms.

Application to Cluster Repair. Consider the Cluster Repair problem [32], in which, for some constant t, we are promised $\mathsf{min{-}disagree}(G) \leq t$ and want to find the clustering $\operatorname{argmin}_{\mathcal{C}} \mathsf{disagree}(G, \mathcal{C})$.

We first argue that, given a spanning forest F of (V, E^+) we can limit our attention to checking a polynomial number of possible clusterings. The spanning forest F can be constructed in the dynamic graph stream model using an algorithm with space $\tilde{O}(n)$ [4]. Let $\mathcal{C}_F$ be the clustering corresponding to the connected components of E^+. Let $F_1, F_2, \ldots, F_p$ be the forests that can be generated by adding t_1 and then removing t_2 edges from F where $t_1 + t_2 \leq t$. Let $\mathcal{C}_{F_i}$ be the node-partition corresponding to the connected components of F_i.

Lemma 2 *The optimal partition of G is $\mathcal{C}_{F_i}$ for some $1 \leq i \leq p$. Furthermore, $p = O(n^{2t})$.*

Proof Let E^+_* be the set of edges in the optimal clustering that are between nodes in the same cluster and let $E^+_* = (E^+ \cup A) \setminus D$, i.e., A is the set of positive edges that need to be added and D is the set of edges that need to be deleted to transform E^+ into a collection of node-disjoint clusters. Since $\mathsf{min{-}disagree}(G) \leq t$, we know

$|A| + |D| \leq t$. It is possible to transform F into a spanning forest F' of $E^+ \cup A$ by adding at most $|A|$ edges. It is then possible to generate a spanning forest of F'' with the same connected components as $E^+_* = (E^+ \cup A) \setminus D$ by deleting at most $|D|$ edges from F'. Hence, one of the forests F_i considered has the same connected components at E^+_*.

To bound p, we proceed as follows. There are less than n^{2t_1} different forests that can result from adding at most t_1 edges to F. For each, there are at most n^{t_2} forests that can be generated by deleting at most t_2 edges from the, at most $n-1$, edges in F'. Hence, $p < \sum_{t_1,t_2 : 0 \leq t_1+t_2 \leq t} n^{2t_1+t_2} < t^2 n^{2t}$. □

The procedure is then to take advantage of this bounded number of partitions by computing each $\mathcal{C}_{F_i}$ in turn, and estimating $\mathsf{disagree}(G, \mathcal{C}_{F_i})$. We report the $\mathcal{C}_{F_i}$ that minimizes the (estimated) repair cost. Consequently, setting $\delta = 1/(p\,\mathrm{poly}\,(n))$ in Theorem 1 yields the following theorem.

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

2.2 Second Data Structure: Sparsification

The next data structure is based on graph sparsification and works for arbitrarily weighted graphs. A sparsification of graph G is a weighted graph H such that the weight of every cut in H is within a $1 + \varepsilon$ factor of the weight of the corresponding cut in G. A celebrated result of Benczúr and Karger [13] shows that it is always possible to ensure the the number of edges in H is $\tilde{O}(n\varepsilon^{-2})$. A subsequent result shows that this can be constructed in the dynamic graph stream model.

Theorem 3 ([5, 31]) *There is a single-pass algorithm that returns a sparsification using space $\tilde{O}(n\varepsilon^{-2})$ and time $\tilde{O}(m)$.*

The next lemma establishes that a graph sparsifier can be used to approximate agree and $\mathsf{disagree}$ of a clustering.

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

$$\mathsf{agree}(G, \mathcal{C}) = (1 \pm \varepsilon/2)\mathsf{agree}(H, \mathcal{C}) \pm \varepsilon w(E^+)/2$$

and

$$\mathsf{disagree}(G, \mathcal{C}) = (1 \pm \varepsilon/2)\mathsf{disagree}(H, \mathcal{C}) \pm \varepsilon w(E^-)/2\,.$$

Furthermore, $\mathsf{max{-}agree}(G) = (1 \pm \varepsilon)\mathsf{max{-}agree}(H)$.

Proof The proofs for agree and disagree are symmetric, so we restrict our attention to agree. Let $\varepsilon' = \varepsilon/6$. The weight of edges in E^- that are cut is estimated within a $1+\varepsilon'$ factor in the sparsifier. For an arbitrary cluster $C \in \mathcal{C}$, and letting $w'(\cdot)$ represent the weight in the sparsifier,

$$\begin{aligned} w(uv \in E^+ : u, v \in C) &= w(uv \in E^+ : u \in C, v \in V) - w(uv \in E^+ : u \in C, v \notin C) \\ &= \sum_{u \in C} w(uv \in E^+ : v \in V) - \sum_{u \in C} w(uv \in E^+ : v \notin C) \\ &= (1 \pm \varepsilon') \sum_{u \in C} w'(uv \in E^+ : v \in V) \\ &\quad - (1 \pm \varepsilon') \sum_{u \in C} w'(uv \in E^+ : v \notin C) \\ &= w'(uv \in E^+ : u, v \in C) \pm 2\varepsilon' w'(uv \in E^+ : u \in C, v \in V), \end{aligned}$$

where the third line follows because, for each $u \in \mathcal{C}$, the weights of cuts $(\{u\}, V \setminus \{u\})$ and $(C, V \setminus C)$ are approximately preserved. The final line simply combines and rewrites the two error terms, since

$$\begin{aligned} \varepsilon' \sum_{u \in C} w'(uv \in E^+ : v \notin C) \leq & \varepsilon' \sum_{u \in C} w'(uv \in E^+ : v \in V) \\ = & \varepsilon' w'(uv \in E^+ : u \in C, v \in V). \end{aligned}$$

Summing over all clusters $C \in \mathcal{C}$, the total additive error is

$$2\varepsilon' w'(E^+) \leq 2\varepsilon'(1+\varepsilon') w(E^+) \leq \varepsilon w(E^+)/2\,,$$

(assuming $\varepsilon \leq 1$), as required.

The last part of the lemma follows because $w(E^+) \leq \mathsf{max-agree}(G)$, as can be seen by considering the trivial all-in-one-cluster partition. □

Application to max−agree *with Bounded Weights.* In Sect. 3, based on the sparsification construction, we develop a poly (n)-time streaming algorithm that returns a 0.7666-approximation for max−agree when G has arbitrary weights. However, in the case of unit weights, a RAM-model PTAS for max−agree is known [12, 30]. It would be unfortunate if, by approximating the unit-weight graph by a weighted sparsification, we lost the ability to return a $1 \pm \varepsilon$ approximation in polynomial time.

We resolve this by emulating an algorithm by Giotis and Guruswami [30] for $\mathsf{max-agree}_k$ using a single pass over the stream[1]. Their algorithm is as follows:

1. Let $\{V^i\}_{i \in [m]}$ be an arbitrary node-partition, where $m = \lceil 4/\varepsilon \rceil$ and $\lfloor n/m \rfloor \leq |V^i| \leq \lceil n/m \rceil$.
2. For each $j \in [m]$, let S^j be a random sample of $r = \text{ poly}\,(1/\varepsilon, k, \log 1/\delta)$ nodes in $V \setminus V_j$.
3. For all possible k-partitions of each of $S^1, \ldots, S^m$:

[1] Note $\mathsf{max-agree}_k(G) \geq (1-\varepsilon)\mathsf{max-agree}(G)$ for $k = O(1/\varepsilon)$ [12].

- For each j, let $\{S_i^j\}_{i\in k}$ be the partition of S^j.
- Compute and record the cost of the clustering in which each $v \in V^j$ is assigned to the ith cluster defined by the (fixed) S_i^j as

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

4. For all the clusterings generated, return the clustering $\mathcal{C}$ that maximizes $\mathsf{agree}(G,\mathcal{C})$.

Giotis and Guruswami [30] prove that the above algorithm achieves a $1+\varepsilon$ approximation factor with high probability if all weights are $\{-1,+1\}$. We explain in Section A that their analysis actually extends to the case of bounded weights. The more important observation is that we can simulate this algorithm in conjunction with a graph sparsifier. Specifically, the sets $V_1,\ldots,V_m$ and $S_1,\ldots,S_m$ can be determined before the stream is observed. To emulate step 3, we just need to collect the rnm edges incident on each S_i during the stream. If we simultaneously construct a sparsifier during the stream we can evaluate all of the possible clusterings that arise. With r and m as set in the algorithm above, the space needed is $rmn = O(n\mathrm{poly}(k,\varepsilon))$. Focusing on n, rather than k or ε, and recalling that a semi-streaming algorithm is one that uses $O(n\mathrm{polylog}n)$ space, this leads to the following theorem.

Theorem 4 *For bounded-weight inputs, there exists a polynomial-time semi-streaming algorithm that, within* $\tilde{O}(n\,\mathrm{poly}\,(k,1/\varepsilon))$ *space, with high probability,* $(1-\varepsilon)$*-approximates* $\mathsf{max{-}agree}(G)$.

2.3 Third Data Structure: Node-Based Sketch

In this section, we develop a data structure that supports queries to $\mathsf{disagree}(G,\mathcal{C})$ 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 4 *For a two-partition* $\mathcal{C}=\{C_1,C_2\}$, $\mathsf{disagree}(G,\mathcal{C}) = \|\sum_{\ell\in C_1} a^\ell - \sum_{\ell\in C_2} a^\ell\|_1$.

Proof The result follows immediately from consideration of the different possible values for the $\{i,j\}$th coordinate of the vector $\sum_{\ell\in C_1} a^\ell - \sum_{\ell\in C_2} a^\ell$. The sum can be expanded as

$$\left|\left(\sum_{\ell\in C_1} a^\ell - \sum_{\ell\in C_2} a^\ell\right)_{ij}\right| = \begin{cases} |w_{ij}/2 - w_{ij}/2| & \text{if } ij \in E^- \text{ and } i,j \text{ in different clusters} \\ |w_{ij}/2 + w_{ij}/2| & \text{if } ij \in E^- \text{ and } i,j \text{ in the same cluster} \\ |w_{ij}/2 + w_{ij}/2| & \text{if } ij \in E^+ \text{ and } i,j \text{ in different clusters} \\ |w_{ij}/2 - w_{ij}/2| & \text{if } ij \in E^+ \text{ and } i,j \text{ in the same cluster} \end{cases}.$$

Hence $\left|\left(\sum_{\ell\in C_1} a^\ell - \sum_{\ell\in C_2} a^\ell\right)_{ij}\right| = |w_{ij}|$ if and only if the edge is a disagreement.

□

We apply the ℓ_1-sketching result of Kane, Nelson and Woodruff [40] to compute a random linear sketch of each a^i.

Theorem 5 *For arbitrary weights, and for query partitions that contain two clusters there exists an $O(\varepsilon^{-2} n \log \delta^{-1} \log n)$-space algorithm which provides a $1 \pm \varepsilon$ approximation to a* $\mathsf{disagree}_2$ *query with probability at least $1 - \delta$. The query time is $O(\varepsilon^{-2} n \log \delta^{-1} \log n)$.*

Unfortunately, for queries $\mathcal{C}$ where $|\mathcal{C}| > 2$, $\Omega(n^2)$ space is necessary, as shown in Sect. 6.

Application to $\mathsf{min{-}disagree}_2(G)$ *with Bounded Weights.* We apply the above node-based sketch in conjunction with another algorithm by Giotis and Guruswami [30], this time for $\mathsf{min-disagree}_2$. Their algorithm for general k is as follows:

1. Sample $r = \text{poly}(1/\varepsilon, k) \cdot \log n$ nodes S and for every possible k-partition $\{S_i\}_{i\in[k]}$ of S:
   (a) Consider the clustering where $v \in V \setminus S$ is assigned to the ith cluster where

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

2. For all the clusterings generated, return the clustering $\mathcal{C}$ that minimizes $\mathsf{disagree}(G, \mathcal{C})$.

As with the max-agreement case, Giotis and Gurusawmi [30] prove that the above algorithm achieves a $1 + \varepsilon$ approximation factor with high probability if all weights are $\{-1, +1\}$. We explain in Section A that their analysis actually extends to the case of bounded weights. Again note we can easily emulate this algorithm for $k = 2$ in the data stream model in conjunction with the third data structure. 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. Focusing on n, rather than k or ε, the space cost is $\tilde{O}(n)$, and hence the algorithm is semi-streaming.

Theorem 6 *For bounded-weight inputs, there exists a polynomial-time semi-streaming algorithm that, within space* $\tilde{O}(n \operatorname{poly}(1/\varepsilon))$*, with high probability* $(1+\varepsilon)$*-approximates* $\mathsf{min{-}disagree}_2(G)$.

3 Convex Programming in Small Space: min−disagree

In this section, we present a linear programming-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 required to solve convex programs has always been an issue, a relatively recent consideration is the restriction to small space [2]. In this presentation, we pursue the Multiplicative Weight Update technique and its derivatives. This method has a rich history across many different communities [9], and has been extended to semi-definite programs [10]. In this section, we focus on linear programs in the context of min−disagree; we postpone the discussion of SDPs to Sect. 4.

In all multiplicative weight 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. The LP system is

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

where $\mathbf{A} \in \mathbb{R}_+^{N \times M}$, $\mathbf{c}, \mathbf{y} \in \mathbb{R}_+^M$, and $\mathbf{b} \in \mathbb{R}_+^N$. To solve the MWM-LP approximately, the multiplicative-weight update algorithm proceeds iteratively. In each iteration, given the *current* solution, $\mathbf{y}$, the procedure maintains a set of multipliers (one for each constraint) and computes a new *candidate* solution $\mathbf{y}'$ which (approximately) satisfies the linear combination of the inequalities, as defined in Theorem 7.

Theorem 7 ([9, Theorem 3.3]) *Suppose that,* $\delta \leq \frac{1}{2}$ *and in each iteration* t*, given a vector of non-negative multipliers* $\mathbf{u}(t)$*, a procedure (termed Oracle) provides a candidate* $\mathbf{y}'(t)$ *satisfying three* ***admissibility*** *conditions (where* ρ, ℓ *are parameters that describe the Oracle's guarantees*),

(i) $\mathbf{c}^T\mathbf{y}'(t) \geq \alpha$;
(ii) $\mathbf{u}(t)^T\mathbf{A}\mathbf{y}'(t) - \mathbf{u}(t)^T\mathbf{b} \leq \delta \sum_i \mathbf{u}_i(t)$; and
(iii) $-\rho \leq -\ell \leq \mathbf{A}_i\mathbf{y}'(t) - \mathbf{b}_i \leq \rho$, for all $1 \leq i \leq n$.

We set $\mathbf{u}(t+1)_i = (1 + \delta(\mathbf{A}_i\mathbf{y}'(t) - \mathbf{b}_i)/\rho)\mathbf{u}(t)_i$*. Assuming we start with* $\mathbf{u}(0) = \mathbf{1}$*, after* $T = O(\rho\ell\delta^{-2} \ln M)$ *iterations the average vector,* $\mathbf{y} = \sum_t \mathbf{y}'(t)/T$*, satisfies* $\mathbf{A}_i\mathbf{y} - \mathbf{b}_i \leq 4\delta$*, for all* i.

The computation of the new candidate depends on the specific LP being solved. The parameter ρ is called the *width*, and controls the speed of convergence. The parameter ℓ is bounded by ρ, but a better bound on ℓ allows a better convergence estimate. A small-width Oracle is typically a key component of an efficient solution, for example, to minimize running times, number of rounds, and

so forth. However, the width parameter is inherently tied to the specific formulation chosen. Consider the standard LP relaxation for min−disagree, where variable x_{ij} indicates edge ij being cut:

$$\begin{array}{ll} \min \sum_{ij\in E^+} w_{ij}x_{ij} + \sum_{ij\in E^-} |w_{ij}|(1-x_{ij}) & \\ x_{ij} + x_{j\ell} \geq x_{i\ell} & \text{for all } i,j,\ell \\ x_{ij} \geq 0 & \text{for all } i,j \end{array} .$$

The triangle constraints state that if we cut one side of a triangle, we must also cut at least one of the other two sides. The size of the formulation is in $\Theta(n^3)$, where n is the size of the vertex set, irrespective of the number of nonzero entries in $E^+ \cup E^-$. In what follows, we will make use of sparsifications of the edge sets in order to reduce the size of problems. However, note that for this LP formulation, since the size is always $\Theta(n^3)$, an edge sparsification would 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.

The first hurdle is the storage requirement. We cannot store all the edges/variables which can be $\Omega(n^2)$. This is avoided by using a sparsifier and invoking (the last part of) Lemma 3. Let H^+ be the sparsification of E^+ with $m' = |H^+|$. For edge $sq \in H^+$ let w^h_{sq} denote its weight after sparsification. For each pair $ij \in E^-$ and some set of edges E', let $P_{ij}(E')$ denote the set of all paths between i and j involving edges only in the set E'. Consider the following LP for min−disagree, similar to that of Wirth [51], but in this sparsified setting:

$$\begin{array}{lll} & \min & \sum_{ij\in E^-} |w_{ij}|z_{ij} + \sum_{sq\in H^+} w^h_{sq}\, x_{sq} \\ \forall p \in P_{ij}(H^+), ij \in E^- & & z_{ij} + \sum_{sq\in p} x_{sq} \geq 1 \\ \forall ij \in E^-, sq \in H^+ & & z_{ij}, x_{sq} \geq 0 \end{array} \qquad \text{(LP1)}$$

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. Therefore, the relevant variable in the objective function is 1 whenever the assignment to an edge disagrees with the input.

By Lemma 3, the objective value of LP1 is at most $(1+\varepsilon)$ times the optimum value of min−disagree. However, LP1 now has exponential size, and it is unclear how we can maintain the multipliers and update them in small space. To overcome this major hurdle, we follow the approach below.

3.1 A Dual Primal Approach

Consider a primal minimization problem, for example, min−disagree, in the canonical form:

$$\text{Primal LP:} \begin{cases} \min \ \mathbf{b}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \ \ \mathbf{A}^{\mathrm{T}}\mathbf{x} \geq \mathbf{c}, \quad \mathbf{x} \geq \mathbf{0}\,. \end{cases}$$

The dual of the above problem for a guess, α of the optimum solution (to the Primal) becomes

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

which is the same as the decision version of MWM-LP as described earlier. We apply Theorem 7 to the Dual LP, however we still want a *solution* to the Primal LP. Note that despite *approximately* solving the Dual LP, we do not have a Primal solution. Even if we had some *optimal* solution to the Dual LP, we might still require a lot of space or time to find a Primal solution, though we could at least rely on complementary slackness conditions. Unfortunately, similar general conditions do not exist for approximately optimum (or feasible) solutions. To circumvent this issue:

(a) We apply the multiplicative-weight framework to the Dual LP and try to find an approximately feasible solution $\mathbf{y}$ such that $\mathbf{c}^T\mathbf{y} \geq (1 - O(\delta))\alpha$ and $\mathbf{A}\mathbf{y} \leq \mathbf{b}, \mathbf{y} \geq 0$.

(b) The Oracle is modified to provide a $\mathbf{y}$, subject to conditions (i)–(iii) of Theorem 7, or an $\mathbf{x}$ that, for some $f \geq 1$, satisfies

$$\mathbf{b}^T\mathbf{x} \leq f \cdot \alpha, \quad \mathbf{A}^T\mathbf{x} \geq \mathbf{c}, \quad \mathbf{x} \geq 0\,.$$

Intuitively, the Oracle is asked to either make progress towards finding a feasible dual solution or provide an f-approximate primal solution in a single step.

(c) If the Oracle returns an $\mathbf{x}$ then we know that $\mathbf{c}^T\mathbf{y} > (\mathbf{b}^T\mathbf{x})/f$ is not satisfiable. We can then consider smaller values of α, say $\alpha \leftarrow \alpha/(1+\delta)$. We eventually find a sufficiently small α that the Dual LP is (approximately feasible) and we have a $\mathbf{x}$ satisfying

$$\mathbf{b}^T\mathbf{x} \leq f \cdot (1+\delta)\alpha, \quad \mathbf{A}^T\mathbf{x} \geq \mathbf{c}, \quad \mathbf{x} \geq 0\,.$$

Note that computations for larger α continue to remain valid for smaller α.

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 (that has exponentially many constraints) in a single step, has also benefited solving Maximum Matching in small space [3]. However in Ahn and Guha [3], the constraint matrix was unchanging across iterations (the objective function value did vary) – here we will have the constraint matrix vary across iterations (along with the value of the objective function). Clearly, such a result will not apply for arbitary constraint matrices and the correct choice of a formulation is key.

One key insight is that the dual, in this case (and as a parallel with matching) has exponentially many variables, but fewer constraints. Such a constraint matrix is easier to satisfy approximately in a few iterations because there are many more *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. Other examples of this same phenomenon are the numerous dynamic connectivity/sparsification results due to Guha *et al.*[35], where the algorithm repeatedly finds edges in cuts (dual of connectivity) to demonstrate connectivity. In that example, the $O(\log n)$ seemingly adaptive iterations collapse into a single iteration.

Parts of the three steps, that is, (a)–(c) outlined above, have been used to speed up running times of SDP-based approximation algorithms [10]. In such cases, there was no increase to the number of constraints nor consideration of non-standard formulations. It is often thought, and as explicitly discussed by Arora and Kale [10], that 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, in this paper, we align both sets of techniques and use them interchangeably.

The remainder of Sect. 3 is organized as follows. We first provide a generic Oracle construction algorithm for MWM-LP, in Sect. 3.2. As a first example, we then apply this algorithm on the *multicut* problem in Sect. 3.3—the multicut problem is inherently related to min−disagree for arbitrary weights [17, 23]. We then show how to combine all the ideas to solve min−disagree in Sect. 3.4.

3.2 From Rounding Algorithms to Oracles

Recall the formulation MWM-LP, and Theorem 7. Algorithm 1 takes an f-approximation for the Primal LP and produces an Oracle for MWM-LP. This is a generic transformation that satisfies conditions (i) and (ii) of Theorem 7 for any problem whose dual matches MWM-LP. As a consequence of the transformation the analysis need only focus on condition (iii) as discussed in the statement of Theorem 7. The main steps correspond to (1) producing a (possibly) infeasible primal solution; (2) attempting to round that (possibly) infeasible primal solution; (3) deciding the success/failure of the rounding step and identifying a set of violated constraints in case of failure; and (4) the indices of the violated constraints supply the coordinates of an admissible dual candidate $\mathbf{y}$ (as defined by Theorem 7). Each one of these steps has associated costs that depend on the problem formulation in terms of the number of variables, constraints (dual variables) and the choice of the rounding algorithm. We revisit these costs in the specific context of correlation shortly.

Algorithm 1 From a rounding algorithm to an Oracle.

1: Transform vector $\mathbf{u}(t)$ (a vector of weights for the constraints of **Dual LP**) into a vector of scaled primal variables $\mathbf{x}$, thus: $x_i = \alpha u(t)_i / \sum_i b_i u(t)_i$.
2: Perform a rounding algorithm for the Primal LP with $\mathbf{x}$ as the input fractional solution (as described in (b) previously). Either there is a subset of violated constraints in the Primal LP or (if no violated constraint exists) there is a solution with objective value at most $f \cdot \alpha$, where f is the approximation factor for the rounding algorithm. In case no violated constraint exists, return $\mathbf{x}$.
3: Let $S = \{i_1, i_2, \ldots, i_k\}$ be (the indexation of) the set of violated constraints in the Primal LP and let $\Delta = \sum_{i \in S} c_i$.
4: Let $y_i = \alpha / \Delta$ for $i \in S$, and let $y_i = 0$ otherwise. Return $\mathbf{y}$. Note the two return types are different based on whether progress was made in the primal or dual direction.

The following lemma shows how to satisfy the first two conditions of Theorem 7; the width parameter has to be bounded separately for a particular problem.

Lemma 5 *If $c_j > 0$ for each Primal constraint, and $\sum_i u(t)_i > 0$, then Algorithm 1 returns a candidate $\mathbf{y}$ that satisfies conditions (i) and (ii) of Theorem 7.*

Proof By construction, $\mathbf{c}^{\mathrm{T}}\mathbf{y} = \alpha$, addressing condition (i). So it remains to prove that $\mathbf{u}(t)^T\mathbf{A}\mathbf{y} - \mathbf{u}(t)^T\mathbf{b} \leq 0$. Since $\mathbf{u}(t)$ is a scaled version of $\mathbf{x}$,

$$\begin{aligned}
\frac{1}{\sum_i u(t)_i}\left(\mathbf{u}(t)^T\mathbf{A}\mathbf{y} - \mathbf{u}(t)^T\mathbf{b}\right) &= \frac{1}{\sum_i x_i}\sum_i x_i(\mathbf{A}_i\mathbf{y} - b_i) \\
&= \frac{1}{\sum_i x_i}\left(\sum_i x_i\mathbf{A}_i\mathbf{y} - \sum_i x_i b_i\right) \\
&= \frac{1}{\sum_i x_i}\left(\sum_j y_j(\mathbf{A}_j^T\mathbf{x}) - \sum_i x_i b_i\right) \\
&\leq \frac{1}{\sum_i x_i}\left(\sum_j y_j c_j - \sum_i x_i b_i\right) = 0
\end{aligned}$$

The inequality in the second line follows from y_j only being positive if the corresponding Primal LP constraint is violated. Finally, by construction, $\sum_j y_j c_j = \alpha$ and $\sum_i b_i x_i = \alpha$; since we also assumed that $\sum_i u(t)_i > 0$, the lemma follows. □

3.3 Streaming Multicut Problem

The Minimum Multicut problem is defined as follows. Given a weighted undirected graph and κ pairs of vertices (s_i, t_i), for $i = 1, \ldots, \kappa$, the goal is to remove the lowest weight subset of edges such that every i, s_i is disconnected from t_i.

In the streaming context, suppose that the weights of the edges are in the range $[1, W]$ and the edges are ordered in an arbitrary order defining a dynamic data stream (with both insertions and deletions). We present a $O(\log \kappa)$

-approximation algorithm for the multicut problem that uses $\tilde{O}(n\varepsilon^{-2}\log W + \kappa)$ space and $\tilde{O}(n^2\varepsilon^{-7}\log^2 W)$ time excluding the time to construct a sparsifier. The $\tilde{O}(n^2)$ term dominates the time required for sparsifier construction. The relevant papers have more details regarding streaming sparsifiers [35, 41]. The algorithm is defined in terms of a parameter, δ, which will eventually be set to $O(\varepsilon)$.

MC1 Sparsify the graph defined by the dynamic data stream, preserving all cuts, and thus the optimum multicut, within a $1 \pm \delta$ factor. Let E' be the edges in the sparsification and $|E'| = m'$, where $m' = O(n\delta^{-2}\log W)$, from the results of Ahn *et al.*[5]. Let (w_{jq}) refer to weights after the sparsification.

MC2 Given an edge set $E'' \subseteq E'$, let $P'(i, E'')$ be the set of all s_i–t_i paths in the edge set E''. The LP that captures Multicut is best viewed as relaxation of a 0/1 assignment. Variable x_{jq} is an indicator of whether edge (j, q) is in the multicut. If we interpret x_{jq} as an assignment of lengths, then for all $i \in [\kappa]$, all $p \in P'(i, E')$ have length at least 1. The relaxation is therefore:

$$\begin{array}{ll} \alpha^* = \min \sum_{(j,q)\in E'} w_{jq}x_{jq} & \\ \text{s.t. } \sum_{(j,q)\in p} x_{jq} \geq 1 & \text{for all } i \in [\kappa], p \in P'(i, E') \\ x_{jq} \geq 0 & \forall (j, q) \in E' \end{array} \tag{LP2}$$

MC3 Compute an initial upper bound $\alpha_0 \in [(1 + 4\delta)\alpha^*, (1 + 4\delta)n^2\alpha^*]$ (see Lemma 6).

MC4 Following the dual-primal approach in Algorithm 1, as α decreases (note the initial α_0 being high, we cannot hope to even approximately satisfy the dual), we consider the (slightly modified) dual

$$\begin{array}{ll} \sum_p y_p \geq \alpha & \\ \frac{1}{w_{jq}} \sum_{p:(j,q)\in p} y_p \leq 1 & \text{for all } (j, q) \in E' \\ y_p \geq 0 & \text{for all } i \in [\kappa], p \in P'(i, E') \end{array} \tag{LP3}$$

More specifically, we consider the following variation: given α, let $E'(\alpha)$ be the set of edges of weight at least $\delta\alpha/m'$, and we seek:

$$\begin{array}{ll} \sum_p y_p \geq \alpha & \\ \frac{1}{w_{jq}} \sum_{p:(j,q)\in p} y_p \leq 1 & \text{for all } (j, q) \in E'(\alpha) \\ y_p \geq 0 & \text{for all } i \in [\kappa], p \in P'(i, E'(\alpha)) \end{array} \tag{LP4}$$

MC5 Run the Oracle provided in Algorithm 2.

MC6 If an $\mathbf{x}$ is received, set $\alpha \leftarrow \alpha/(1 + \delta)$ as in (c) in Sect. 3.1. This step occurs at least once (Lemma 7). Note that reducing α corresponds to *adding constraints as well as variables* to LP4 due to new edges in $E'(\alpha/(1 + \delta)) - E'(\alpha)$. Set $u_{i'}(t + 1) = (1 - \delta/\rho)^t$ for each new constraint i' added, assuming that the Oracle in step (MC5) has been run a total of t times thus far. Lemma 8 shows that this transformation provides a $\mathbf{u}$ and a collection $\mathbf{y}(t)$ as if the multiplicative weight algorithm for LP4 was run for the current value of $\alpha = \alpha_1$.

MC7 If the number of iterations required by Theorem 7 have been completed, then average the **y** returned. This ensures that we obtain an approximately feasible solution for LP4. This corresponds to a proof of (near) optimality. We return the **x** returned corresponding to the previous value of α (which was $\alpha(1+\delta)$) as the solution. This is an $f(1+O(\delta))$ approximation (Lemma 7). If we have not completed the number of iterations, we return to (MC5).

Lemma 6 *Consider introducing the edges of E' from the largest weight to smallest. Let w be the weight of the first edge whose introduction connects some pair (s_i, t_i). Set $\alpha_0 = (1+4\delta)n^2 w$. Then $\alpha_0 \in [(1+4\delta)\alpha^*, (1+4\delta)n^2\alpha\ *]$.*

Proof Note w is a lower bound on α^*; moreover, if we delete the edge with weight w and all subsequent edges in the ordering we have a feasible multicut solution. Therefore $\alpha^* \leq n^2 w$. The lemma follows. □

Naively, this edge-addition process runs in $\tilde{O}(m'\kappa)$ time, since the connectivity needs to be checked for every pair. However, we can introduce the edges in groups, corresponding to weights in $(2^{z-1}, 2^z]$, as z decreases; we check connectivity after introducing each group. This algorithm runs in time $\tilde{O}(m' + \kappa \log W)$ and approximates w, i.e., overestimates w by a factor of at most 2, since we have a geometric sequence of group weights. The initial value of α can thus be set to $(1+4\delta)2^z n^2$.

Lemma 7 *α is decreased, as in (MC6), at least once. The solution returned in (MC7) is an $f(1+O(\delta))$ approximation to α^*.*

Proof Using Theorem 7 once we are in (MC7) multiplying the average of the y_p by $1/(1+4\delta)$ gives a feasible solution for LP4 for the edge set $E(\alpha)$. Moreover, for all paths p, containing any edge in $E' - E'(\alpha)$, we have $y_p = 0$. Therefore this new solution is a feasible solution of LP3. Therefore $\alpha/(1+4\delta) \leq \alpha^*$ once we reach the required number of iterations in (MC7). This proves that we must decrease α at least once, because α_0 is larger than $(1+4\delta)\alpha^*$ (Lemma 6).

The solution **x** corresponds to $f\alpha(1+\delta)$. Since α is bounded above by $\alpha^*(1+4\delta)$, the second part of the lemma follows as well. □

Algorithm 2 Oracle for LP4

1: Given weights u^t_{jq}, for $(j,q) \in E'(\alpha)$, define $x_{jq} = \alpha u^t_{jq} / \sum_{(j,q)\in E'(\alpha)} w_{jq} u^t_{jq}$.
2: Define the shortest path metric $d^x(\cdot,\cdot)$ with the x_{jq} representing edge lengths. Define $B(\zeta, r) = \{\zeta' \mid d^x(\zeta,\zeta') \le r\}$, which corresponds to a family of balls/regions centered at ζ, each of radius r. Let $cut(B(\zeta,r))$ be the total weight of edges in $E'(\alpha)$ that are cut by $B(\zeta,r)$, i.e.,

$$cut(B(\zeta,r)) = \sum_{(\zeta',\zeta'')\in E'(\alpha); d^x(\zeta,\zeta') \le r < d^x(\zeta,\zeta'')} w_{\zeta'\zeta''}.$$

3: Find a collection of regions $B(\zeta_1, r_1), \ldots, B(\zeta_g, r_g), \ldots$ such that every $r_g \le \frac{1}{3}$ and each s_i (from the given instance of the multicut problem) belongs to some region, and $\sum_g cut(B(u_g, r_g)) \le 3\alpha \ln(\kappa+1)$. Lemma 9 shows us how to find $\{\zeta_j\}$.
4: **if** for some i both s_i and t_i belong to the same region **then**
5: Find the corresponding path p, which is of length at most $2/3$, which violates the constraint. Return $y_p = \alpha$. Implicitly return $y_{p'} = 0$ for all other paths that involve the s_i–t_i pair.
6: **else**
7: Return the union of the cuts defined by the balls (this corresponds to $\mathbf{x}$). The edges in $E'(\alpha)$ contribute at most $3\alpha\ln(\kappa+1)$. The edges in $E' - E'(\alpha)$ contribute at most $\delta\alpha$. The total is $(3\ln(\kappa+1)+\delta)\alpha$. Note that the return types are different as outlined in the dual-primal framework in (a)–(c) earlier.

Corollary 1 *We decrease α at most $O(\delta^{-1}\log n)$ times in step MC6.*

Proof If we decrease α then at some point line (7) of Algorithm 2 provides a solution $\ll \alpha^*$, which is infeasible. Note that the solution would have value $f\alpha$. But this has to be at least α^*. Thus α cannot decrease arbitrarily. Combined with the upper bound in Lemma 6, the result follows. □

Lemma 8 *Algorithm* 2 *returns an admissible* $\mathbf{y}$ *(defined in Theorem* 7*) for* LP4 *with (width) $\rho = m'/\delta$ and $\ell = 1$. Moreover the set of assignments of y_p (over the different iterations) that were admissible for $\alpha = \alpha_2$ remains admissible if α is lowered to $\alpha_1 < \alpha_2$ and* $\mathbf{u}$ *updated as described in (MC6).*

Proof Using Lemma 5, Algorithm 2 returns a $\mathbf{y}$ which satisfies conditions (i) and (ii) of Theorem 7. By construction, in Algorithm 2 $y_p = \alpha$ and only one y_p has a non-zero value. Since we removed all the edges of weight less than $\delta\alpha/m'$, the width parameter is bounded by $\alpha m'/(\delta\alpha) = m'/\delta$. Observe that $\ell = 1$.

If $\alpha_1 < \alpha_2$, then $E'(\alpha_1) \supseteq E'(\alpha_2)$, and therefore $P(i, E'(\alpha_1)) \supseteq P(i, E'(\alpha_2))$. Therefore, for the formulation LP4, we are adding new variables corresponding to new variables (paths) as well as new constraints corresponding to the newly added edges. We can interpret the $\mathbf{y}$ for α_2 to have 0 values for the new variables. This would immediately satisfy (i). This would satisfy (iii) for the old constraints as well. Condition (iii) is satisfied for the newly introduced constraints because the old paths p with $y_p > 0$ for α_2 did not contain an edge in $E'(\alpha_1)$. Thus $\mathbf{A}_i\mathbf{y}(t) = 0$ for the new constraints and $\mathbf{b} = \mathbf{1}$ and $-\rho \le -1 \le \rho$.

For (ii), $\mathbf{u}(t)^T\mathbf{A}\mathbf{y}(t) - \mathbf{u}(t)^T\mathbf{b} \le \delta \sum_i \mathbf{u}(t)$, the first term in the left hand side remains unchanged. The left hand side decreases for every new constraint, and the right hand side increases for every new constraint. □

The next lemma arises from a result of Garg *et al.*[28]; in this context, $Z = \alpha$ and defines the set $\{\zeta_j\}$ in Step 3 of Algorithm 2.

Lemma 9 ([28]) *Let* $Z = \sum_{(u,v)} x_{uv} w_{uv}$. *For* $r \geq 0$, *let* $B(u,r) = \{v \mid d^x(u,v) \leq r\}$ *where* d^x *is the shortest path distance based on the values* x_{uv}. *Let* $vol(B(u,r))$ *be*

$$\frac{Z}{\kappa} + \sum_{\substack{(v,v') \\ v,v' \in B(u,r)}} x_{vv'} w_{vv'} + \sum_{\substack{(v,v') \\ v \in B(u,r), v' \notin B(u,r)}} (r - d^x(u,v)) w_{vv'}$$

Suppose that for a node 0ζ, *the radius* r *of the ball around* ζ *is increased until* $cut(B(\zeta,r)) \leq C \cdot vol(B(\zeta,r))$. *If* $C = 3\ln(\kappa+1)$, *the ball stops growing before the radius becomes* 1/3. *We start this process for* $\zeta_1 = s_1$. *Repeatedly, if some* s_j *is not in a ball, then we remove* $B(\zeta_i, r_i)$ *(all edges inside and those being cut) and continue the process with* $\zeta_{i+1} = s_j$, *on the remainder of the graph. The collection of* $B(\zeta_1, r_1), \ldots, B(\zeta_g, r_g), \ldots$ *satisfy the condition that* $r_g \leq 1/3$ *for all* g *and* $\sum_g cut(B(\zeta_g, r_g)) \leq CZ$.

The proof follows from the fact that $cut(B(\zeta,r))$ is the derivative of $vol(B(\zeta,r))$ as r increases and the volume cannot increase by more than a factor of $\kappa+1$, because it is at least Z/k and cannot exceed $Z/k + Z$. For nonnegative x_{jq} the above algorithm runs in time $\tilde{O}(m')$ using standard shortest-path algorithms.

Using Theorem 7, the total number of iterations needed in MC7, for a particular α is $O(\rho\delta^{-2}\log N) = O(m'\delta^{-3}\log n)$, since the number of constraints $N = O(n^2)$ and $\rho \leq m'/\delta$. This dominates the $O(\frac{1}{\delta}\cdot\log n)$ times we decrease α.

Observe that the algorithm repeatedly constructs a set of balls with non-negative weights; which can be performed in $O(m'\log n)$ time. In each of these balls with $\tilde{m}$ edges, we can find the shortest path in $O(\tilde{m}\log n)$ time (to find the violated pair s_i–t_i). Summed over the balls, each iteration can be performed in $O(m'\log n)$ time. Coupled with the approximation introduced by a sparsifier, setting $\delta = O(\varepsilon)$ we get:

Theorem 8 *There exists a single-pass* $O(\log\kappa)$*-approximation algorithm for the multicut problem in the dynamic semi-streaming model that runs in* $\tilde{O}(n^2\varepsilon^{-7}\log^2 W)$ *time and* $\tilde{O}(n\varepsilon^{-2}\log W + \kappa)$ *space.*

3.4 min−disagree with Arbitrary Weights

In this section, we prove the following theorem:

Theorem 9 *There is a* $3(1+\varepsilon)\log|E^-|$*-approximation algorithm for* min−disagree *that requires* $\tilde{O}((n\varepsilon^{-2} + |E^-|)^2\varepsilon^{-3})$ *time,* $\tilde{O}(n\varepsilon^{-2} + |E^-|)$ *space, and a single pass.*

Consider the dual of LP1, where $\mathbf{P} = \cup_{ij\in E^-} P_{ij}(H^+)$.

$$\begin{aligned}
&\max \sum_p y_p \\
&\frac{1}{|w_{ij}|} \sum_{p \in P_{ij}(H^+)} y_p \le 1 \qquad \forall ij \in E^- \\
&\frac{1}{w^h_{sq}} \sum_{p \in \mathbf{P}: sq \in p} y_p \le 1 \qquad \forall sq \in H^+ \\
&y_p \ge 0 \qquad\qquad\qquad\qquad \forall p \in \mathbf{P}
\end{aligned} \tag{LP5}$$

Recall that LP1 was based on the fact that each path between the two endpoints of a negative edge had to be of a certain length (or else there is a separation). The dual of that formulation corresponds to assigning weights to those paths and trying to "pack" paths such that the total amount of weight (across different paths) does not exceed the cost (in the primal formulation) of cutting the edge. Note that the dual formulation in this case corresponds to a lower bound of the primal minimization problem – the optimal solution of this packing problem will satisfy some of the constraints with equality (complementary slackness) and those will precisely correspond to the edges having nonzero value in an optimum primal formulation in LP1. To reiterate, the overall idea is to continually increase this lower bound using the multiplicative weights approach and Algorithm 1 — or fail and have a feasible primal solution. We apply Theorem 7 (the multiplicative-weight update framework) to the dual of LP1, but omit the constraints in the dual corresponding to small-weight edges, exactly along the lines of MC1–MC7. For each $\alpha \ge 0$, let $H^+(\alpha), E^-(\alpha)$ be the set of edges in H^+, E^-, respectively, with weight at least $\delta\alpha/(m' + |E^-|)$. Consider now the decision version of LP5:

$$\begin{aligned}
&\sum_p y_p \ge \alpha \\
&\frac{1}{|w_{ij}|} \sum_{p \in P_{ij}(H^+(\alpha))} y_p \le 1 \qquad \forall ij \in E^-(\alpha) \\
&\frac{1}{w^h_{sq}} \sum_{p \in \mathbf{P}: sq \in p} y_p \le 1 \qquad \forall sq \in H^+(\alpha) \\
&y_p \ge 0 \qquad\qquad\qquad\qquad \forall p \in \mathbf{P}(\alpha)
\end{aligned} \tag{LP6}$$

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

We attempt to find an approximate feasible solution to LP6 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 $\alpha_1 \le \alpha_2$, then $P_{ij}(H^+(\alpha_1)) \supseteq P_{ij}(H^+(\alpha_2))$ exactly along the lines of Lemma 8.

Eventually we lower α sufficiently that we have a feasible solution to LP6, and we can claim Theorem 9 exactly along the lines of Theorem 8. The Oracle is provided in Algorithm 3 and relies on the following lemma:

Lemma 10 *Let $\kappa = |E^-|$, $Z = \sum_{uv \in H^+(\alpha)} x_{uv} w^h_{uv}$. Using the definition of $d^x()$ and $B()$ as in Lemma 9, let*

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

Suppose that, for a node ζ, the radius r of its ball is increased until $cut(B(\zeta,r)) \le C vol(B(\zeta,r))$. 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 $\sum_g \text{cut}(B(\zeta_g, r_g)) \le CZ$.

Algorithm 3 Oracle for LP6

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 $B(\zeta,r) = \{\zeta' \mid d^x(\zeta,\zeta') \le r\}$ and the weight of the edges of cut by the ball:

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

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) $\sum_g \text{cut}(B(\zeta_g,r_g)) \le 3\alpha Q_u/(Q_u+Q_v) \cdot \ln(|E^-|+1)$. The existence of such balls follows from Lemma 10.
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 and all edges in $H^+ - H^+(\alpha)$.

The above lemma is essentially the same as Lemma 9, applied to the terminal pairs defined by the endpoints of each edge in E^-. Again, for nonnegative x_{sq}, standard shortest-path algorithms lead to a running time of $\tilde{O}(m')$. We bound the width of the above oracle as follows :

Lemma 11 $\rho = (m' + |E^-|)/\delta, \ell = 1$ for Algorithm 3.

Proof Note that the weights are least $\delta\alpha/(m' + |E^-|)$ in the set of edges $H^+(\alpha), E^-(\alpha)$. The admissible candidate (Step 6 of Algorithm 3) corresponds to assigning weight α to a single path (and 0 weight to all other paths). Therefore the left hand side of any edge in formulation LP6 is at most $\alpha/(\delta\alpha/(m' + |E^-|))$ which

is the upper bound on ρ. The $\ell = 1$ arises since each of the constraints in formulation LP6 has 1 in the right hand side and the left hand side is always nonnegative (based on the assignment proposed in step 6 of Algorithm 3). □

The total weight of positive edges cut by the solution returned in line 8 of Algorithm 3 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 along the lines of Lemma 6. 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 an $O(n^2)$ approximation to the optimum solution.

4 Convex Programming in Small Space: max−agree

In this section we discuss an SDP-based algorithm for max−agree. We will build upon our intuition in Sect. 3 where we developed a linear program based algorithm for min−disagree. However several steps, such as switching of primals and duals, will not be necessary because we will use a modified version of the multiplicative weight update algorithm for SDPs as described by Steurer [47]. As will become clear, the switch of primals and duals is already achieved in the internal working of Steurer's technique [47]. Consider:

Definition 1 For matrices $\mathbf{X}, \mathbf{Z}$, let $\mathbf{X} \circ \mathbf{Z}$ denote the Frobenius product, $\sum_{i,j} \mathbf{X}_{ij}\mathbf{Z}_{ij}$, 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 $\mathbf{X}_{ij} = \mathbf{x}_i^T \mathbf{x}_j$. Consider the following theorem:

Theorem 10 ([47]) *Let* $\mathbf{D}$ *be a fixed diagonal matrix with positive entries, and assume* $\mathcal{X}$ *is nonempty. Suppose there is an Oracle with parameters* ρ *and* δ*, so that for each positive semidefinite* $\mathbf{X}$ *either (a) tests and declares* $\mathbf{X}$ *to be approximately feasible — for all* $1 \leq i \leq q$*, we have* $\mathbf{F}_i \circ \mathbf{X} \leq g_i + \delta$*, or (b) provides a real symmetric matrix* $\mathbf{A}$ *and a scalar* b *satisfying (i)* $\mathbf{A} \circ \mathbf{X} \leq b - \delta$ *and for all* $\mathbf{X}' \in \mathcal{X}$*,* $\mathbf{A} \circ \mathbf{X}' \geq b$ *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.*

The above theorem does not explicitly discuss maintaining a set of multipliers. But interestingly, the algorithm due to Steurer [47] that proves Theorem 10 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 $\mathbf{X}$. The candidate $\mathbf{X}$ generated by the 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! Therefore, along with maximum matching [3] and min−disagree (Sect. 3) we have another example where switching the primal and the dual formulations helps. However in all of these cases, we need to prove that that we can produce a feasible primal solution in a space efficient manner, when the Oracle (for the dual) cannot produce a candidate.

We now prove the following theorem:

Theorem 11 *There is a* $0.7666(1-\varepsilon)$*-approximation algorithm for* max−agree(G) *that uses* $\tilde{O}(n\varepsilon^{-2})$ *space,* $\tilde{O}(m + n\varepsilon^{-10})$ *time and a single pass.*

We use Lemma 3 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:

$$\begin{aligned} \sum_{ij\in H^+} w^h_{ij}\mathbf{X}_{ij} + \sum_{ij\in H^-} \frac{|w^h_{ij}|(\mathbf{X}_{ii}+\mathbf{X}_{jj}-2\mathbf{X}_{ij})}{2} &\geq \alpha \\ \mathbf{X}_{ii} &\leq 1 \quad \forall i \in V \\ -\mathbf{X}_{ii} &\leq -1 \quad \forall i \in V \\ -\mathbf{X}_{ij} &\leq 0 \quad \forall i,j \in V \\ \mathbf{X} &\succeq \mathbf{0} \end{aligned} \tag{SDP}$$

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 $\frac{1}{2}(\mathbf{X}_{ii} + \mathbf{X}_{jj} - 2\mathbf{X}_{ij}) = (1 - \mathbf{X}_{ij})$, 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, for some suitably small user chosen δ. 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 [47]. 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 11 follows from the Oracle provided in Algorithm 4. 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 4 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 12 *Algorithm* 4 *satisfies criterion (i) of Theorem* 10, *i.e., 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.

Proof For line 7, $\mathbf{A} \circ \mathbf{X} \leq \sum_{i \in S_1} -d_i(1 + \delta)/\Delta_1 = -1 - \delta$, since $\|x_i\|^2 \geq 1 + \delta$ for all $i \in S_1$. On the other hand, for a feasible $\mathbf{X}'$, $\|x'_i\|^2 = 1$ for all i. Hence $\mathbf{A} \circ \mathbf{X}' = \sum_{i \in S_1} -d_i/\Delta_1 = -1$. This proves that the oracle is δ-separating when it returns from line 7. For lines 10 and 13, the proof is almost identical.

For line 17, we do not use the violated constraints; instead we use $\mathbf{C}'$ to construct $\mathbf{A}$, and show that $\mathbf{C}' \circ \mathbf{X}' \geq (1 - 3\delta)\alpha$. We start from the fact that $\mathbf{C} \circ \mathbf{X}' \geq \alpha$, since $\mathbf{X}'$ is feasible for SDP. By removing all nodes in S_1, we remove all edges incident on the removed nodes. The total weight of removed edges is bounded by Δ_1, which is this case is less than $\delta\alpha$. Similarly, we lose at most $\delta\alpha$ for each of S_2 and S_3. Hence, the difference between $\mathbf{C}' \circ \mathbf{X}'$ and $\mathbf{C} \circ \mathbf{X}'$ is bounded by $3\delta\alpha$, and so $\mathbf{C}' \circ \mathbf{X}' \geq (1 - 3\delta)\alpha$ which implies $\mathbf{A} \circ \mathbf{X}' \geq 1 - 3\delta$. Therefore we have δ separation because $\mathbf{A} \circ \mathbf{X} = \mathbf{C}' \circ \mathbf{X}/\alpha < 1 - 4\delta$. □

Lemma 13 *Algorithm 4 satisfies criterion (ii) of Theorem 10, i.e., $\rho\mathbf{D} \succeq \mathbf{A} - b\mathbf{D} \succeq -\rho\mathbf{D}$ for some $\rho = O(1/\delta)$.*

Proof Since $|b| \leq 1$ it suffices to show that for every positive semidefinite $\mathbf{Y}$, $|\mathbf{A} \circ \mathbf{Y}| = \rho\mathbf{D} \circ \mathbf{Y}$. For line 7, the proof is straightforward. To start, $\mathbf{A}$ is a diagonal matrix where $|\mathbf{A}_{ii}| = d_i/\Delta_1 \leq d_i/(\delta\alpha)$. On the other hand, $\mathbf{D}_{ii} = d_i/2W$, while $\alpha \geq W/2$, so we have $|\mathbf{A}_{ii}| = O(1/\delta)\mathbf{D}_{ii}$ which proves that $|\mathbf{A} \circ \mathbf{Y}| = O(1/\delta)\mathbf{D} \circ \mathbf{Y}$. The proof is identical for line 10.

For lines 13 and 17, consider the decomposition of $\mathbf{Y}$, i.e., $\{\mathbf{y}_i\}$ such that $\mathbf{Y}_{ij} = \mathbf{y}_i \cdot \mathbf{y}_j$. We use the fact that $\mathbf{y}_i \cdot \mathbf{y}_j \leq \|\mathbf{y}_i\|^2 + \|\mathbf{y}_j\|^2$ for every pair of vectors $\mathbf{y}_i$ and $\mathbf{y}_j$. Therefore for $\mathbf{Y}_{ij} = \mathbf{y}_i \cdot \mathbf{y}_j$, we have at line 13,

$$\begin{aligned}
|\mathbf{A} \circ \mathbf{Y}| &= \sum_{ij \in S_3} \frac{|w^h_{ij}|}{\Delta_3} \mathbf{Y}_{ij} \leq \sum_{ij \in S_3} \frac{|w^h_{ij}|}{\Delta_3} (\|\mathbf{y}_i\|^2 + \|\mathbf{y}_j\|^2) \\
&= \frac{1}{\Delta_3} \sum_i \|y_i\|^2 \sum_{j : ij \in S_3} |w^h_{ij}| \leq \frac{1}{\Delta_3} \sum_i d_i \|y_i\|^2 \\
&= \frac{1}{\Delta_3} \sum_i 2W\mathbf{D}_{ii}\mathbf{Y}_{ii} = \frac{2W}{\Delta_3} \mathbf{D} \circ \mathbf{Y},
\end{aligned}$$

which implies $|\mathbf{A} \circ \mathbf{Y}| \leq O(1/\delta)\mathbf{D} \circ \mathbf{Y}$ given $\alpha \geq W/2$ and $\Delta_3 \geq \delta\alpha$. For line 17, let $H^+|_{G'}, H^-|_{G'}$ denote H^+, H^- as modified by line 15, then

$$\begin{aligned}
\mathbf{A} \circ \mathbf{Y} &= \frac{1}{\alpha} \mathbf{C}' \circ \mathbf{Y} = \frac{1}{2\alpha} \sum_{ij \in H^+|_{G'}} 2w^h_{ij} \mathbf{Y}_{ij} + \frac{1}{2\alpha} \sum_{ij \in H^-|_{G'}} |w^h_{ij}| (\mathbf{Y}_{ii} + \mathbf{Y}_{jj} - 2\mathbf{Y}_{ij}) \\
&\leq \frac{1}{2\alpha} \sum_{ij \in G'} 2|w^h_{ij}| (\mathbf{Y}_{ii} + \mathbf{Y}_{jj}) \leq \frac{1}{\alpha} \sum_i d_i \mathbf{Y}_{ii} = \frac{2W}{\alpha} \mathbf{D} \circ \mathbf{Y}
\end{aligned}$$

which implies that $\mathbf{A} \circ \mathbf{Y} = O(1)\mathbf{D} \circ \mathbf{Y}$. Summarizing, Algorithm 4 is $O(1/\delta)$ -bounded. □

Lemmas 12 and 13, in conjunction with Theorem 10 prove Theorem 11. The update procedure [47] 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.

Rounding the Fractional Solution: Note that the solution of the SDP found above is only approximately feasible. Since the known rounding algorithms can not be applied in a black box fashion, the following lemma proves the correctness of the rounding algorithm.

Lemma 14 *If Algorithm* 4 *returns a clustering solution, it has at least* $0.7666(1 - O(\delta))\alpha$ *agreements.*

Proof We show that the rounding algorithm returns a clustering with at least $0.7666(1 - O(\delta))\mathbf{C}' \circ \mathbf{X}$ agreements. Combined with the fact that $\mathbf{C}' \circ \mathbf{X} > (1 - 4\delta)\alpha$ (line 19), we obtain the desired result.

We use the rounding algorithm of Swamy [48] (see also [27]), with caveats. The analysis in [48] starts from a completely feasible solution of SDP, namely $-\mathbf{X}_{ij} \leq 0$ and the analysis appears to depend on this non-negativity. Likewise, the analysis of Swamy [48] requires that $\mathbf{X}_{ii} = 1$. So while the same algorithm is used, a new analysis is required. The algorithm is as follows: we consider the Cholesky decomposition of the matrix which gives us vectors $\{x_i\}$ such that $\mathbf{X}_{ij} = x_i \cdot x_j$. We rescale every $\{x_i\}$ to have length 1. We now run the algorithm of Swamy [48] (which refers to an analysis from Frieze and Jerrum [27] for a different problem). The analysis has three steps:

(1) Changes introduced due to eliminating $\mathbf{X}_{ii} \neq 1$.
(2) We then fix edges (i,j) $-2\delta \leq \mathbf{X}_{ij} \leq 0$ by changing the weight of the edge in the objective function to 0. These could be a holdover from the approximately feasible solution which have become more violated due to the scaling in step 1.
(3) We now consider the analysis in prior work [27, 48].

For step (1), since the SDP deals with $\mathbf{C}'$ instead of $\mathbf{C}$, we can ignore all nodes and edges in S_1, S_2, and S_3. Our first step is to rescale the vectors in $\mathbf{X}$ to be unit vectors. Since all vectors that are not ignored (not in S_1 nor S_2) have length between $1 - O(\delta)$ and $1 + O(\delta)$ (since we take the square root), this only changes the objective value by $O(\delta w_{ij})$ for each edge. Hence the total decrease in the objective function is bounded by $O(\delta W) = O(\delta\alpha)$.

For step (2), we then change the objective value of edges (i, j) with $-2\delta < \mathbf{X}_{ij} < 0$ by changing their weight function in the objective function to 0. This step decreases the objective value by at most $2\delta|w_{ij}|$ for each negative edge (and does not decrease the objective for the positive edges). Again, the objective value decreases by at most $O(\delta\alpha)$.

For step (3) we observe that the rounding algorithm [27, 48] obtains a 0.7666 approximation factor based on an analysis over pairs of vertices that satisfy the constraint $x_i \cdot x_j \geq 0$. Note that the analysis is irrelevant for the other pairs because their weight is 0 due to steps (1) and (2). Therefore, we obtain a clustering that has at least $0.7666(1 - O(\delta))\mathbf{C}' \circ \mathbf{X} - O(\delta\alpha)$ agreements. □

5 Multipass Algorithms

In this section, we present $O(\log \log n)$-pass algorithms for min−disagree on unit weight graphs: these apply to both a fixed and unrestricted number of clusters.

In each pass over the data, the algorithm is presented with the same input, although not necessarily in the same order.

5.1 min−disagree with Unit Weights

Consider the 3-approximation algorithm for min−disagree on unit-weight graphs due to Ailon *et al.*[6].

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.*[20], who developed a modification of the algorithm that used $O(\varepsilon^{-1} \log^2 n)$ streaming passes and returned a $(3 + \varepsilon)$-approximation, rather than a 3-approximation. Our improvement is based on the following lemma:

Lemma 15 *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, $|F_{t,t'}| \leq 5 \cdot \ln n \cdot t'^2/t$.

Proof Note that the bound holds vacuously for $t \leq 10 \ln n$ so in the rest of the proof we will assume $t \geq 10 \ln n$. Fix the set of t' elements in the random permutation and consider the induced graph H on these t' elements. Pick an arbitrary node v in H. We will consider the random process that picks each of the first t entries of the random permutation by picking a node in H uniformly at random without replacement. We will argue that at the end of these t steps, with probability at least $1 - 1/n^{10}$, either v is covered or at most $\alpha t'/t$ neighbors of v in H are uncovered where $\alpha = 10 \ln n$. Hence, by the union bound, all uncovered nodes have at most $\alpha t'/t$ uncovered

neighbors and hence the number of edges in H whose both endpoints are uncovered after the first t steps is at most $(\alpha t'/t) \cdot t'/2$. The lemma follows because $F_{t,t'}$ is exactly the number of edges in H whose both endpoints are uncovered after the first t steps.

To show that after t steps, either v is covered or it has at most $\alpha t'/t$ uncovered neighbors we proceed as follows. Let B_i be the event that after the ith iteration, v is not covered and it has at least $\alpha t'/t$ uncovered neighbors. Then, since $B_{i+1} \subset B_i$ for each i,

$$\begin{aligned}\Pr\big(v \text{ is covered or it has at most } \alpha t'/t \text{ uncovered neighbors}\big) &= 1 - \Pr(B_r)\\ &= 1\\ &\quad - \Pr(B_r \cap B_{r-1} \cap \cdots \cap B_1)\\ &= 1 - p_r p_{r-1} \cdots p_1\end{aligned}$$

where $p_i = \Pr\big(B_i \mid B_1 \cap B_2 \cap \cdots \cap B_{i-1}\big)$. Note that

$$\begin{aligned}p_i \le 1 - \Pr\big(v \text{ gets covered at step } i \mid B_1 \cap B_2 \cap \cdots \cap B_{i-1}\big) &\le 1 - \frac{\alpha t'/t + 1}{t' - (i-1)}\\ &< 1 - \alpha/t\,,\end{aligned}$$

and hence,

$$\begin{aligned}\Pr\big(v \text{ is covered or it has at most } \alpha t'/t \text{ uncovered neighbors}\big) &\ge 1 - (1 - \alpha/t)^t\\ &\ge 1 - \exp(-\alpha)\\ &= 1 - 1/n^{10}\,,\end{aligned}$$

as required. □

Semi-Streaming Algorithm. As a warm-up, first consider the following two-pass streaming algorithm that emulates Ailon et al.'s algorithm using $O(n^{1.5}\log^2 n)$ space:

1. *First pass:* Collect all edges in E^+ incident on $\{v_i\}_{i\in[\sqrt{n}]}$. This allows us to simulate the first $\sqrt{n}$ iterations of the algorithm.
2. *Second pass:* Collect all edges in $F_{\sqrt{n},n}$. This allows us to simulate the remaining $n - \sqrt{n}$ iterations.

The space bound follows since each pass requires storing only $O(n^{1.5}\log n)$ edges with high probability. requires storing at most $n^{1.5}$ edges and, with high probability, the second pass requires storing $|F_{\sqrt{n},n}| = O(n^{1.5}\log n)$ edges.

Our semi-streaming algorithm proceeds as follows.

- For $j \ge 1$, 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} where $t_0 = 0$, 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 the algorithm of Ailon *et al.*[6]'s algorithm. Since $t_j \geq n$ for $j = 1 + \log\log n$, our algorithm terminates after $O(\log\log n)$ passes.

Theorem 12 *On a unit-weight graph, there exists a $O(\log\log n)$-pass semi-streaming algorithm that, within space $O(n\log n)$, returns with high probability a 3-approximation to* min−disagree.

Proof In the first pass, we need to store at most $t_1^2 = ((2n)^{1-1/2})^2 = 2n$ edges. For the odd-numbered passes after the first pass, by Lemma 15, the space is at most

$$5 \cdot \ln n \cdot t_j^2 / t_{j-1} = 5 \cdot \ln n \cdot (2n)^{2-2/2^j} / (2n)^{1-1/2^{j-1}} = 5 \cdot \ln n \cdot 2n = O(n\log n)\,,$$

with high probability. The additional space used in the even-numbered passes is trivially bounded by $O(n\log n)$. The approximation factor follows from the analysis of Ailon *et al.* [6]. □

5.2 min−disagree$_k$ with Unit Weights

Our result in this section is based the following algorithm of Giotis and Guruswami [30] that returns a $(1+\varepsilon)$-approximation for min−disagree$_k$ on unit-weight graphs. Their algorithm is as follows:

1. Sample $r = \text{poly}\,(1/\varepsilon, k) \cdot \log n$ nodes S and for every possible k-partition $\{S_i\}_{i\in[k]}$ of S:
   (a) Compute the cost of the clustering where $v \in V \setminus S$ is assigned to the ith cluster where

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

2. 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 (with the appropriate number of centers still to be determined) on the set of nodes in clusters that are smaller than $n/(2k)$.

We first observe the above algorithm can be emulated in $\min(k-1, \log n)$ passes in the data stream model. 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 Sect. 2.1 to find the best possible partitions during post-processing. It is not hard to argue that this algorithm terminates in $O(\log n)$ rounds, independent of k: Call clusters with fewer than $n/2k$ nodes "small", and those with at least $n/2k$ nodes "large". Observe that the number of nodes in small clusters halves

in each round since there are at most $k-1$ small clusters and each has at most $n/(2k)$ nodes. This would suggest a $\min(k-1, \log n)$ pass data stream algorithm, one pass to emulate each round of the offline algorithm. However, the next theorem shows that the algorithm can actually be emulated in $\min(k-1, \log\log n)$ passes.

Theorem 13 *There exists a* $\min(k-1, \log\log n)$*-pass* $O(\text{poly}(k, \log n, 1/\varepsilon)n)$*-space algorithm that* $(1+\varepsilon)$ *approximates* $\mathsf{min{-}disagree}_k(G)$.

Proof To design an $O(\log\log n)$ pass algorithm, we proceed as follows. At the start of the i-th pass, suppose we have k' clusters still to determine and that V_i is the set of remaining nodes that have not yet been included in large clusters. We will pick k' random sets of samples $S_1, \ldots, S_{k'}$ in parallel from V_i each of size

$$N_i = 2rn^{2^{i-1}/\log n}.$$

For each sampled node, we extract all edges to unclustered nodes. We will use this information to emulate one or more rounds of the algorithm. Note that since $N_i \geq n$ for $i \geq 1 + \log\log n$, the algorithm must terminate in $O(\log\log n)$ passes since in pass $1 + \log\log n$ we are storing all edges in the unclustered graph. What remains is to establish a bound on the space required in each of the passes. To do this we will first argue that in each pass, the number of unclustered nodes drops significantly, perhaps to zero.

Since there are only k' clusters still to determine, and every round of the algorithm fixes at least one cluster, it is conceivable that the sets $S_1, \ldots, S_{k'}$ could each be used to emulate one of the remaining $\leq k'$ rounds of the algorithm; this would suggest it is possible to completely emulate the algorithm in a single pass. However, this will not be possible if at some point there are fewer than r unclustered nodes remaining in all the sets $S_1, \ldots, S_{k'}$. At this point, we terminate the current set of samples, and take a new pass. Observe that in this case we have likely made progress, as the number of unclustered nodes over which we are working has likely dropped significantly. Specifically, suppose the number of unclustered nodes is greater than $|V_i|n^{2^{i-1}/\log n}$ before we attempt to use $S_{k'}$. By the principle of deferred decisions, the expected number of unclustered nodes in $S_{k'}$ is at least

$$\frac{|V_i|n^{2^{i-1}/\log n}}{|V_i|} N_i = 2r.$$

Therefore, by an application of the Chernoff bound, we can deduce that the number of unclustered nodes when we terminate the current pass is less than $|V_i|n^{2^{i-1}/\log n}$, i.e., the number of unclustered nodes has decreased by a factor of at least $n^{2^{i-1}/\log n}$ since the start of the pass.

Applying this analysis to all passes and using the fact that $|V_1| = n$, we conclude that

$$|V_{i+1}| \leq \frac{|V_i|}{n^{2^{i-1}/\log n}} \leq \frac{|V_1|}{n^{2^{1-1}/\log n} \cdot n^{2^{2-1}/\log n} \cdot \ldots \cdot n^{2^{i-1}/\log n}} = \frac{n}{n^{(2^i-1)/\log n}}.$$

The space needed by our algorithm for round i is therefore $O(|V_i|N_i k') = O(krn^{1+1/\log n}) = \tilde{O}(krn)$. □

6 Lower Bounds

Finally, we consider the extent to which our results can (not) be improved, by showing lower bounds for variants of problems that we can solve. All our proofs will use the standard technique of reducing from two-party communication complexity problems, i.e., Alice has input x and Bob has input y and they wish to compute some function $f(x, y)$ such that the number of bits communicated between Alice and Bob is small. A lower bound on the number of bits communicated can be used to lower bound the space complexity of a data stream algorithm as follows. Suppose Alice can transform x into the first part S_1 of a data stream and Bob can transform y into the second part S_2 such that the result of the data stream computation on $S_1 \circ S_2$ implies the value of $f(x, y)$. Then if the data stream algorithm takes p passes and uses s space, this algorithm can be emulated by Alice and Bob using $2p - 1$ messages each of size s bits; Alice starts running the data stream algorithm on S_1 and each time a player no longer has the necessary information to emulate the data stream algorithm they send the current memory state of the algorithm to the other player. Hence, a lower bound for the communication complexity problem yields a lower bound for the data stream problem.

Theorem 14 *A one-pass stream algorithm that tests whether* $\mathsf{min{-}disagree}(G) = 0$, *with probability at least* 9/10, *requires* $\Omega(n^2)$ *bits if permitted weights are* $\{-1, 0, 1\}$.

Proof The theorem follows from a reduction from the communication problem Index. Alice has a string $x \in \{0,1\}^{\binom{n}{2}}$, indexed as $[n] \times [n]$ and unknown to Bob, and Bob wants to learn $x_{i,j}$ for some $i, j \in [n]$ that is unknown to Alice. Any one-way protocol from Alice to Bob that allows Bob to learn $x_{i,j}$ requires $\Omega(n^2)$ bits of communication [1].

Consider the protocol for Index where Alice creates a graph G over nodes $V = \{v_1, \ldots, v_n\}$ and adds edges $\{\{v_i, v_j\} : x_{i,j} = 1\}$ each with weight -1. Suppose there were a data stream algorithm with properties as claimed in the statement of the Theorem. Alice could run such a data stream algorithm on G and send the state of the algorithm to Bob who would add positive edges $\{u, v_i\}$ and $\{u, v_j\}$ where u is a new node. All edges without a specified weight are treated as not present, or equivalently as having weight zero. Hence the set of weights used in this graph is $\{-1, 0, +1\}$. Now, if $x_{ij} = 0$, then $\mathsf{disagree}(G) = 0$: consider the partition containing $\{u, v_i, v_j\}$, with each other item comprising a singleton cluster. Alternatively, $x_{ij} = 1$ implies $\mathsf{disagree}(G) \geq 1$ since a clustering must disagree with one of the three edges on $\{u, v_i, v_j\}$. It follows that any data stream algorithm returning a multiplicative estimate of $\mathsf{min{-}disagree}(G)$ requires $\Omega(n^2)$ space. □

When permitted weights are restricted to $\{-1, 1\}$, the following multi-pass lower bounds holds:

Theorem 15 *A p-pass stream algorithm that tests whether* $\mathsf{min{-}disagree}(G) = 0$, *with probability at least* 9/10, *requires* $\Omega(n/p)$ *bits when permitted weights are* $\{-1, 1\}$.

Proof The proof uses a reduction from the communication problem of Disj where Alice and Bob have strings $x, y \in \{0, 1\}^n$ and wish to determine where there exists an i such that $x_i = y_i = 1$. Any p round protocol between Alice and Bob requires $\Omega(n)$ bits of communication [39] and hence there must be a message of $\Omega(n/p)$ bits.

Consider the protocol for Disj on a graph G with nodes $V = \{a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n\}$. For each $i \in [n]$, Alice adds an edge $\{a_i, b_i\}$ with weight $(-1)^{x_i+1}$. She runs a data stream algorithm with properties as stated in the theorem statement on G and sends the state of the algorithm to Bob. For each $i \in [n]$, Bob adds an edge $\{b_i, c_i\}$ of weight $(-1)^{y_i+1}$ along with negative edges

$$\{\{a_i, c_i\} : i \in [n]\} \cup \{\{u, v\} : u \in \{a_i, b_i, c_i\}, v \in \{a_j, b_j, c_j\}, i \neq j\} .$$

Note that $\mathsf{min{-}disagree}(G) > 0$ iff there exists i with $x_i = y_i = 1$. Were there no such i, the positive edges would all be isolated, whereas if $x_i = y_i = 1$ then every partition violates one of the edges on $\{a_i, b_i, c_i\}$. It follows that every p-pass data stream algorithm returning a multiplicative estimate of $\mathsf{min{-}disagree}(G)$ requires $\Omega(n/p)$ space. □

Next we show a lower bound that applies when the number of negative weight edges in bounded. This shows that our upper bound in Theorem 9 is essentially tight.

Theorem 16 *A one-pass stream algorithm that tests whether* $\mathsf{min{-}disagree}(G) = 0$, *with probability at least* 9/10, *requires* $\Omega(n + |E^-|)$ *bits if permitted weights are* $\{-1, 0, 1\}$.

Proof A lower bound of $\Omega(|E^-|)$ follows by considering the construction in Theorem 14 on $\sqrt{|E^-|}$ nodes. A lower bound of $\Omega(n)$ when $n \geq |E^-|$ follows by considering the construction in Theorem 15 without adding the negative edges $\{uv : u \in \{a_i, b_i, c_i\}, v \in \{a_j, b_j, c_j\}, i \neq j\}$. □

Finally, we show that the data structure for evaluating 2-clusterings of arbitrarily weighted graphs (Sect. 2.3) cannot be extended to clusterings with more clusters.

Theorem 17 *When* $|\mathcal{C}| = 3$, *a data structure that returns a multiplicative estimate of* $\mathsf{disagree}(G, \mathcal{C})$ (i.e., *answers* $\mathsf{disagree}_3$ *queries) with probability at least* 9/10, *requires* $\Omega(n^2)$ *space.*

Proof We show a reduction from the communication problem of Index where Alice has a string $x \in \{0, 1\}^{n^2}$ indexed as $[n] \times [n]$ and Bob wants to learn $x_{i,j}$ for some

$i, j \in [n]$ that is unknown to Alice. A one-way protocol from Alice to Bob that allows Bob to learn $x_{i,j}$ requires $\Omega(n^2)$ bits of communication [1]. Consider the protocol for INDEX where Alice creates a graph G over nodes $V = \{a_1, \ldots, a_n, b_1, \ldots, b_n\}$ and adds edges $\{a_u b_v : x_{u,v} = 1\}$ each with weight -1. She encodes the graph G into a data structure with properties as described in the theorem statement, and sends the state of the structure to Bob who then queries the partition $\mathcal{C} = \{a_i b_j, \{a_\ell : \ell \neq i\}, \{b_\ell : \ell \neq j\}\}$. Since $\mathsf{disagree}(G, \mathcal{C}) = x_{ij}$ it follows that every data structure allowing a multiplicative estimate of $\mathsf{disagree}(G, \mathcal{C})$ requires $\Omega(n^2)$ space. □

Appendix: Extension to Bounded Weights

In this section, we detail the simple changes that are required in the paper by Gionis and Guruswami [30] such that their result extends to the case where there are no zero weights and the magnitude of all non-zero weights is bounded between 1 and w_* where we will treat w_* as constant.

Max-Agreement. See Sect. 2.2 for a description of the max-agreement algorithm. The proof in the unweighted case first shows a lower bound for $\mathsf{max{-}agree}_k(G)$ of

$$\max(|E^+|, |E^-|(1 - 1/k)) \geq n^2/16\,.$$

In the bounded-weights case, the magnitude of every edge only increases and so the same bound holds. Hence, for the purpose of returning a $(1 + O(\varepsilon))$ multiplicative approximation, it still suffices to find an εn^2 additive approximation. Indeed, the argument of Giotis and Guruswami [30] still applies, with small changes by decreasing ε by a factor w_* and increasing r by a factor of w_*^2. Rather than retread the full analysis of Giotis and Guruswami [30], we just identify the places where their argument is altered.

The central result needed is that estimating the cost associated with placing each node in a given cluster can be done accurately from a sample of the clustered nodes. This is proved via a standard additive Chernoff bound [30, Lemma 3.3]. It is natural to define the weighted generalization of this estimate based on the weights of edges in the sample and to rescale accordingly. One can then apply the additive Chernoff bound over random variables which are constrained to have magnitude in the range $\{1, 2, \ldots, w_*\}$, rather than $\{0, 1\}$ as in the unit-weights case. The number of nodes whose estimated relative contribution deviates by more than $(\varepsilon/32w_*)$ from its (actual) contribution to the optimal clustering is then bounded by applying the Markov inequality. Provided we increase the sample size r by a factor of w_*^2, these bounds all hold with the necessary probability.

The other steps in the argument are modified in a similar way: we analyze the total weight of edges in agreement, rather than their number. Specifically, applying this modification to [30, Lemma 3.4], we bound the impact of misplacing one node in the constructed clustering compared to the optimal clustering. With the inequality from the above Chernoff bound argument, the impact of this can, as in the orignal argument, be bounded in the weighted case by $(\varepsilon/8)n$. The number of nodes for

which this does not hold is at most a fraction $(\varepsilon/8w_*)$ of each partition, and so contribute to a loss of at most $(\varepsilon^2/8)n^2$ (weighted) agreements in each step of the argument, as in the original analysis.

Min-Agreement. See Sect. 5.2 for a description of the min-agreement algorithm. Again, the central step is the use of a Chernoff bound on edges incident on sampled nodes. Modifying this to allow for bounded-weight edges again incurs a factor of w_*^2, but is otherwise straightforward. It then remains to follow through the steps of the original argument, switching from cardinalities of edgesets to their weights.

Acknowledgements K. J. Ahn: The author is currently at Google, kookjin@google.com. G. Cormode: Supported in part by European Research Council grant ERC-2014-CoG 647557, a Royal Society Wolfson Research Merit Award and the Yahoo Faculty Research Engagement Program. S. Guha: supported by NSF Award CCF-1546141. A. McGregor: Supported by NSF Award CCF-1637536, CCF-1908849, and CCF-1934846. A. Wirth: Supported in part by ARC Future Fellowship FT120100307.

References

1. Ablayev, F.M.: Lower bounds for one-way probabilistic communication complexity and their application to space complexity. Theor. Comput. Sci. **157**(2), 139–159 (1996). https://doi.org/10.1016/0304-3975(95)00157-3
2. Ahn, K.J., Guha, S.: Linear programming in the semi-streaming model with application to the maximum matching problem. Inf. Comput. (ICALP 2011 Special Issue) **222**, 59–79 (2013). https://doi.org/10.1016/j.ic.2012.10.006
3. Ahn, K.J., Guha, S.: Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In: Transactions in Parallel Computing (TOPC), special issue for Symposium on Parallelism in Algorithms and Architectures (SPAA), 2015, vol. 4 (2018). https://doi.org/10.1145/2755573.3154855
4. Ahn, K.J., Guha, S., McGregor, A.: Analyzing graph structure via linear measurements. In: Symposium on Discrete Algorithms: SODA, pp. 459–467 (2012). https://doi.org/10.1137/1.9781611973099.40
5. Ahn, K.J., Guha, S., McGregor, A.: Graph sketches: sparsification, spanners, and subgraphs. In: Principles of Database Systems: PODS, pp. 5–14 (2012). https://doi.org/10.1145/2213556.2213560
6. Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: ranking and clustering. J. ACM **55**(5) (2008). https://doi.org/10.1145/1411509.1411513
7. Ailon, N., Jaiswal, R., Monteleoni, C.: Streaming *k*-means approximation. In: Conference on Neural Information Processing Systems: NIPS, pp. 10–18 (2009). http://books.nips.cc/papers/files/nips22/NIPS2009_1085.pdf
8. Ailon, N., Karnin, Z.S.: A note on: No need to choose: How to get both a PTAS and sublinear query complexity. CoRR **abs/1204.6588** (2012)
9. Arora, S., Hazan, E., Kale, S.: The multiplicative weights update method: a meta algorithm and applications. Theory Comput. **8**(6), 121–164 (2012)

10. Arora, S., Kale, S.: A combinatorial, primal-dual approach to semidefinite programs. In: ACM Symposium on Theory of Computing: STOC, pp. 227–236 (2007). https://doi.org/10.1145/1250790.1250823
11. Bagon, S., Galun, M.: Large scale correlation clustering optimization. arXiv:1112.2903v1 (2011)
12. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Mach. Learn. **56**(1–3), 89–113 (2004). https://doi.org/10.1023/B:MACH.0000033116.57574.95
13. Benczúr, A.A., Karger, D.R.: Approximating $s-t$ minimum cuts in $\tilde{O}(n^2)$ time. In: Symposium on Theory of Computing: STOC, pp. 47–55 (1996)
14. Bonchi, F., Garcia-Soriano, D., Liberty, E.: Correlation clustering: From theory to practice. In: International Conference on Knowledge Discovery and Data Mining: KDD, pp. 1972. ACM, New York, NY, USA (2014). https://doi.org/10.1145/2623330.2630808
15. Braverman, V., Chung, K., Liu, Z., Mitzenmacher, M., Ostrovsky, R.: AMS without 4-wise independence on product domains. In: International Symposium on Theoretical Aspects of Computer Science: STACS, pp. 119–130 (2010). https://doi.org/10.4230/LIPIcs.STACS.2010.2449
16. Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental clustering and dynamic information retrieval. SIAM J. Comput. **33**(6), 1417–1440 (2004). https://doi.org/10.1137/S0097539702418498
17. Charikar, M., Guruswami, V., Wirth, A.: Clustering with qualitative information. J. Comput. Syst. Sci. **71**(3), 360–383 (2005). https://doi.org/10.1016/j.jcss.2004.10.012
18. Charikar, M., O'Callaghan, L., Panigrahy, R.: Better streaming algorithms for clustering problems. In: Symposium on Theory of Computing: STOC, pp. 30–39 (2003). https://doi.org/10.1145/780542.780548
19. Chawla, S., Makarychev, K., Schramm, T., Yaroslavtsev, G.: Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In: Symposium on Theory of Computing: STOC (2015)
20. Chierichetti, F., Dalvi, N.N., Kumar, R.: Correlation clustering in mapreduce. In: International Conference on Knowledge Discovery and Data Mining: KDD, pp. 641–650 (2014). https://doi.org/10.1145/2623330.2623743
21. Coleman, T., Saunderson, J., Wirth, A.: A local-search 2-approximation for 2-correlation-clustering. In: European Symposium on Algorithms: ESA, pp. 308–319 (2008). https://doi.org/10.1007/978-3-540-87744-8_26
22. Cormode, G., Yi, K.: Small Summaries for Big Data. CUP (2020)
23. Demaine, E.D., Emanuel, D., Fiat, A., Immorlica, N.: Correlation clustering in general weighted graphs. Theor. Comput. Sci. **361**(2–3), 172–187 (2006). https://doi.org/10.1016/j.tcs.2006.05.008
24. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2006)
25. Elsner, M., Schudy, W.: Bounding and comparing methods for correlation clustering beyond ILP. In: Workshop on Integer Linear Programming for Natural Langauge Processing: ILP, pp. 19–27. Association for Computational Linguistics, Stroudsburg, PA, USA (2009). http://www.anthology.aclweb.org/W/W09/W09-1803.pdf
26. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: On graph problems in a semi-streaming model. Theor. Comput. Sci. **348**(2–3), 207–216 (2005). https://doi.org/10.1016/j.tcs.2005.09.013
27. Frieze, A., Jerrum, M.: Improved approximation algorithms for MAX k-cut and MAX BISECTION. Algorithmica **18**, 67 (1997)
28. Garg, N., Vazirani, V.V., Yannakakis, M.: Approximate max-flow min-(multi)cut theorems and their applications. In: ACM Symposium on Theory of Computing: STOC, pp. 698–707 (1993). https://doi.org/10.1145/167088.167266
29. Gionis, A., Mannila, H., Tsaparas, P.: Clustering aggregation. ACM Trans. Knowl. Disc. Data (TKDD) **1**(1), 4 (2007)
30. Giotis, I., Guruswami, V.: Correlation clustering with a fixed number of clusters. Theory Comput. **2**(1), 249–266 (2006). https://doi.org/10.4086/toc.2006.v002a013
31. Goel, A., Kapralov, M., Post, I.: Single pass sparsification in the streaming model with edge deletions. CoRR **abs/1203.4900** (2012). http://arxiv.org/abs/1203.4900
32. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: exact algorithms for clique generation. Theory Comput. Syst. **38**(4), 373–392 (2005). https://doi.org/10.1007/s00224-004-1178-y
33. Gruenheid, A., Dong, X.L., Srivastava, D.: Incremental record linkage. Proc. VLDB Endow. **7**(9), 697–708 (2014)

34. Guha, S.: Tight results for clustering and summarizing data streams. In: International Conference on Database Theory: ICDT, pp. 268–275 (2009). https://doi.org/10.1145/1514894.1514926
35. Guha, S., McGregor, A., Tench, D.: Vertex and hyperedge connectivity in dynamic graph streams. In: Proceedings of the 34th ACM Symposium on Principles of Database Systems (PODS), pp. 241–247 (2015)
36. Guha, S., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams. In: IEEE Foundations of Computer Science: FOCS, pp. 359–366 (2000). http://doi.ieeecomputersociety.org/10.1109/SFCS.2000.892124
37. Hassanzadeh, O., Chiang, F., Lee, H.C., Miller, R.J.: Framework for evaluating clustering algorithms in duplicate detection. Proc. VLDB Endow. **2**(1), 1282–1293 (2009)
38. Indyk, P., McGregor, A.: Declaring independence via the sketching of sketches. In: Symposium on Discrete Algorithms: SODA, pp. 737–745 (2008). http://dl.acm.org/citation.cfm?id=1347082.1347163
39. Kalyanasundaram, B., Schnitger, G.: The probabilistic communication complexity of set intersection. SIAM J. Discrete Math. **5**(4), 545–557 (1992). https://doi.org/10.1137/0405044
40. Kane, D.M., Nelson, J., Woodruff, D.P.: On the exact space complexity of sketching and streaming small norms. In: ACM-SIAM Symposium on Discrete Algorithms: SODA, pp. 1161–1178 (2010). https://doi.org/10.1137/1.9781611973075.93
41. Kapralov, M., Lee, Y.T., Musco, C., Musco, C., Sidford, A.: Single pass spectral sparsification in dynamic streams. CoRR **abs/1407.1289** (2014). http://arxiv.org/abs/1407.1289
42. McCutchen, R.M., Khuller, S.: Streaming algorithms for *k*-center clustering with outliers and with anonymity. International Workshop on Approximation Algorithms for Combinatorial Optimization: APPROX pp. 165–178 (2008). https://doi.org/10.1007/978-3-540-85363-3_14
43. McGregor, A.: Graph stream algorithms: a survey. SIGMOD Record **43**(1), 9–20 (2014). https://doi.org/10.1145/2627692.2627694
44. Pan, X., Papailiopoulos, D.S., Oymak, S., Recht, B., Ramchandran, K., Jordan, M.I.: Parallel correlation clustering on big graphs. In: Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, pp. 82–90 (2015). http://papers.nips.cc/paper/5814-parallel-correlation-clustering-on-big-graphs
45. Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Appl. Math. **144**(1), 173–182 (2004)
46. Silva, J.A., Faria, E.R., Barros, R.C., Hruschka, E.R., Carvalho, A.C.P.L.F.d., Gama, J.a.: Data stream clustering: A survey. ACM Comput. Surv **46**(1), 13:1-13:31 (2013). https://doi.org/10.1145/2522968.2522981
47. Steurer, D.: Fast SDP algorithms for constraint satisfaction problems. In: Symposium on Discrete Algorithms: SODA, pp. 684–697 (2010)
48. Swamy, C.: Correlation clustering: maximizing agreements via semidefinite programming. In: Symposium on Discrete Algorithms: SODA, pp. 526–527 (2004). https://doi.org/10.1145/982792.982866
49. Verroios, V., Garcia-Molina, H.: Entity resolution with crowd errors. In: 2015 IEEE 31st International Conference on Data Engineering, pp. 219–230. IEEE (2015)
50. Vesdapunt, N., Bellare, K., Dalvi, N.: Crowdsourcing algorithms for entity resolution. Proc. VLDB Endow. **7**(12), 1071–1082 (2014)
51. Wirth, A.I.: Approximation algorithms for clustering. Ph.D. thesis, Princeton University (2004). ftp://ftp.cs.princeton.edu/reports/2004/716.pdf

Authors and Affiliations

Kook Jin Ahn[1] · Graham Cormode[2] · Sudipto Guha[1] · Andrew McGregor[3] · Anthony Wirth[4]

Kook Jin Ahn
kookjin@cis.upenn.edu; kookjin@google.com

Sudipto Guha
sudipto@cis.upenn.edu

Andrew McGregor
mcgregor@cs.umass.edu

Anthony Wirth
awirth@unimelb.edu.au

1 University of Pennsylvania, Philadelphia, USA

2 University of Warwick, Coventry, UK

3 University of Massachusetts Amherst, Amherst, USA

4 School of Computing and Information Systems, The University of Melbourne, Parkville, Victoria, Australia