Semi-Streaming Algorithms for Weighted k-Disjoint Matchings

S M Ferdous
Pacific Northwest National Laboratory, Richland, WA, USA

Bhargav Samineni
The University of Texas at Austin, TX, USA

Alex Pothen
Purdue University, West Lafayette, IN, USA

Mahantesh Halappanavar
Pacific Northwest National Laboratory, Richland, WA, USA

Bala Krishnamoorthy
Washington State University Vancouver, WA, USA

Abstract

We design and implement two single-pass semi-streaming algorithms for the maximum weight k-disjoint matching (k-DM) problem. Given an integer k, the k-DM problem is to find k pairwise edge-disjoint matchings such that the sum of the weights of the matchings is maximized. For $k \geq 2$, this problem is NP-hard. Our first algorithm is based on the primal-dual framework of a linear programming relaxation of the problem and is $\frac{1}{3+\varepsilon}$-approximate. We also develop an approximation preserving reduction from k-DM to the maximum weight b-matching problem. Leveraging this reduction and an existing semi-streaming b-matching algorithm, we design a $(\frac{1}{2+\varepsilon})(1 - \frac{1}{k+1})$-approximate semi-streaming algorithm for k-DM. For any constant $\varepsilon > 0$, both of these algorithms require $O(nk \log^2_{1+\varepsilon} n)$ bits of space. To the best of our knowledge, this is the first study of semi-streaming algorithms for the k-DM problem.

We compare our two algorithms to state-of-the-art offline algorithms on 95 real-world and synthetic test problems, including thirteen graphs generated from data center network traces. On these instances, our streaming algorithms used significantly less memory (ranging from 6× to 512× less) and were faster in runtime than the offline algorithms. Our solutions were often within 5% of the best weights from the offline algorithms. We highlight that the existing offline algorithms run out of 1 TB memory for most of the large instances (> 1 billion edges), whereas our streaming algorithms can solve these problems using only 100 GB memory for $k = 8$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Graph algorithms analysis; Mathematics of computing → Matchings and factors; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Matchings, Semi-Streaming Algorithms, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.53

Related Version *Full Version*: https://arxiv.org/abs/2311.02073

Supplementary Material *Software (Source Code)*: https://github.com/smferdous1/GraST

Funding *S M Ferdous*: Laboratory Directed Research and Development Program at PNNL.
Bhargav Samineni: U.S. DOE Science Undergraduate Laboratory Internships (SULI) program.
Alex Pothen: U.S. Department of Energy SC-0022260.
Mahantesh Halappanavar: U.S. Department of Energy 17-SC-20-SC (ECP ExaGraph) at PNNL.

32nd Annual European Symposium on Algorithms (ESA 2024).
Editors: Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman; Article No. 53; pp. 53:1–53:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given an undirected graph $G = (V, E, w)$ with weights $w\colon E \to \mathbb{R}_{>0}$ and an integer $k \geq 1$, the k-Disjoint Matching (k-DM) problem asks for a collection of k pairwise edge-disjoint matchings that maximize the sum of the weights of matched edges. The k-DM problem is a generalization of the classical Maximum Weight Matching (MWM) problem and is closely related to the Maximum Weight b-Matching (MWbM) problem. However, in contrast to these problems, k-DM is NP-hard and APX-hard already for $k \geq 2$ [14, 21]. Prior work has primarily studied k-DM in computational models where space complexity is not a limiting factor in designing algorithms. In this work, we study k-DM in the single-pass *semi-streaming* model [15, 33], which is used to solve massive graph problems with limited memory. In particular, we extend existing state-of-the-art semi-streaming matching [34, 18] and b-matching [23] algorithms to the k-DM problem. To the best of our knowledge, these are the *first* semi-streaming algorithms for the k-DM problem.

In the offline unweighted setting, k-DM in general graphs was originally studied by Feige et al. [14], who motivated the problem by applications in scheduling traffic in satellite-based communication networks. Cockayne et al. [7] modeled the problem of finding a maximal assignment of jobs to people such that no person performs the same job on two consecutive days using unweighted k-DM in bipartite graphs with $k = 2$.

In the weighted setting, k-DM was recently studied by Hanauer et al. [21, 19] in the offline and dynamic computation models. This was motivated by applications in designing reconfigurable optical topologies for data center networks [3, 4, 5, 31]. In contrast to static networks, reconfigurable networks use optical switches to quickly provide direct connectivity between racks, where each switch essentially acts as a reconfigurable optical matching. Given a traffic matrix and k optical switches, the underlying optimization problem becomes how to compute heavy disjoint matchings that carry a large amount of traffic for each switch, which is exactly the k-DM problem.

Algorithmic Contributions. We provide a primal-dual linear programming (LP) formulation of k-DM and use it to derive a $\frac{1}{3+\varepsilon}$-approximate single-pass semi-streaming algorithm that requires $O(nk \log^2 n)$ bits of space for any constant $\varepsilon > 0$. Our algorithm extends the seminal MWM semi-streaming algorithm by Paz and Schwartzman [34] by maintaining k stacks and employing approximate dual variables to decide which edges should be stored in those stacks. The post-processing phase that computes k edge-disjoint matchings from the stacks is more involved here since edges in a stack that are not included in a matching need to be considered for inclusion in higher-numbered stacks. The primal-dual analysis of the approximation ratio involves two sets of dual variables here, unlike the former algorithm.

We also reduce the k-DM problem to the MWbM problem. In particular, we show that a modified edge coloring algorithm on any α-approximate b-matching subgraph (with $b(v) = k$ for all $v \in V$) computes an $\alpha(1 - \frac{1}{k+1})$-approximate solution for k-DM. Using the $\frac{1}{2+\varepsilon}$-approximate semi-streaming MWbM algorithm of Huang and Sellier [23], we obtain a $(\frac{1}{2+\varepsilon})(1 - \frac{1}{k+1})$-approximate k-DM that requires $O(nk \log^2 n)$ bits of space for any constant $\varepsilon > 0$. This reduction, which was previously known for unweighted k-DM [14], is not specific to the semi-streaming setting, and thus could be used to develop algorithms for k-DM in other computational models where b-matching results are known.

Experimental Validation. We implement both algorithms and compare the memory used, running time required, and the weight computed with static offline approximation algorithms for this problem on several real-world and synthetic graphs, and several graphs generated from data center network traces. Our results show that the streaming algorithms reduce the memory needed to compute the matchings often by two orders of magnitude and are also faster than offline static algorithms. Indeed, the latter algorithms do not terminate on all but one of the larger graphs in our test set. The median weights computed by the streaming algorithms are only about 5% lower than the ones obtained by the static algorithms. Among the streaming algorithms, the primal-dual algorithm outperforms the b-matching-based algorithm in memory needed and weight, and also time (except for the data center problems).

2 Preliminaries

Notation. Consider a graph $G = (V, E, w)$ with weights $w\colon E \to \mathbb{R}_{>0}$. We denote $n \equiv |V|$ and $m \equiv |E|$ throughout the paper. For an edge $e = (u, v)$, we say that vertices u and v are *incident* on the edge e. Given a vertex $v \in V$, we denote by $\delta(v)$ the set of edges incident on v, and by $\deg(v) \coloneqq |\delta(v)|$ its degree. The maximum degree of G is $\Delta \coloneqq \max_{v\in V} \deg(v)$. We say that two edges e_1 and e_2 are *adjacent* if they share a common vertex. For an edge subset $H \subseteq E$, we let $V(H)$ denote the set of vertices incident on edges in H, and let $G[H]$ denote the subgraph *induced* by H (i.e., the subgraph whose edge set is H and vertex set is $V(H)$). Likewise, we denote by $\deg_H(v) \coloneqq |\delta(v) \cap H|$ the number of edges in H that a vertex $v \in V$ is incident on and let $\Delta_H \coloneqq \max_{v\in V} \deg_H(v)$. For a positive integer t, we use $[t]$ to represent the set of integers from 1 to t, inclusive. For an integer $s \le t$, we let $[s..t]$ denote the set of integers from s to t, inclusive.

Matchings and b-Matchings. Given a function $b : V \to \mathbb{Z}_+$, a *b-matching* in a graph G is an edge subset $F \subseteq E$ such that $|F \cap \delta(v)| \le b(v)$ for each $v \in V$. The weight of a b-matching F is $w(F) \coloneqq \sum_{e\in F} w(e)$, and in the Maximum Weight b-Matching (MWbM) problem, we aim to maximize $w(F)$. When $b(v) = 1$ for all $v \in V$, we obtain a matching and the MWbM problem reduces to the Maximum Weight Matching (MWM) problem.

k-Disjoint Matchings. Given an integer $k \ge 1$, a k-disjoint matching in G is a collection of k matchings $\mathcal{M} = \{M_1, \dots, M_k\}$ that are pairwise edge-disjoint (i.e., $M_i \cap M_j = \emptyset$ for all $i, j \in [k], i \ne j$). Its weight is given by $w(\mathcal{M}) \coloneqq \sum_{i=1}^{k} w(M_i)$ and in the k-Disjoint Matching (k-DM) problem, we aim to maximize $w(\mathcal{M})$. A k-disjoint matching can also be described through an edge coloring viewpoint [21]. Consider a function $\mathcal{C}\colon E \to [k] \cup \{\perp\}$ that assigns edges a color from the palette $[k]$, or leaves them uncolored (color $\perp$). If $\mathcal{C}$ describes a proper k coloring (i.e., any two adjacent edges e_1, e_2 colored from $[k]$ satisfy $\mathcal{C}(e_1) \ne \mathcal{C}(e_2)$) then it also describes a k-disjoint matching. Prior work has shown that k-DM is NP-hard and APX-hard for $k \ge 2$ [14, 21].

An LP relaxation of the k-DM problem and its dual is shown in (P) and (D), respectively. For each edge $e = (u, v) \in E$ and color $c \in [k]$, we associate each primal variable $x(c, e)$ with the inclusion of edge e in the c^{th} matching, i.e., $x(c, e) = 1$ iff $e \in M_c$. The first constraint in (P) enforces that M_c is a valid matching for each $c \in [k]$, while the second constraint ensures each edge $e \in E$ belongs to at most one matching. For the dual (D), we define variables $y(c, v)$ for each color $c \in [k]$ and vertex $v \in V$ (corresponding to the first constraint in (P)), and $z(e)$ for each edge $e \in E$ (corresponding to the second constraint in (P)).

Semi-Streaming Model. For semi-streaming k-DM, in each pass, the edges of E are presented one at a time in an *arbitrary order*. We aim to compute a k-disjoint matching in G at the end of the algorithm, using limited memory and only a single pass. The semi-streaming

$$\text{(P) maximize} \quad \sum_{c\in[k]}\sum_{e\in E} w(e)x(c,e)$$
subject to
$$\sum_{e\in\delta(v)} x(c,e) \le 1 \quad \forall v\in V, c\in[k]$$
$$\sum_{c\in[k]} x(c,e) \le 1 \quad \forall e\in E$$
$$x(c,e) \ge 0 \quad \forall e\in E, c\in[k]$$

$$\text{(D) minimize} \quad \sum_{c\in[k]}\sum_{v\in V} y(c,v) + \sum_{e\in E} z(e)$$
subject to
$$y(c,u)+y(c,v)+z(e) \ge w(e) \quad \forall e=(u,v)\in E, c\in[k]$$
$$y(c,v) \ge 0 \quad \forall v\in V, c\in[k]$$
$$z(e) \ge 0 \quad \forall e\in E$$

Figure 1 LP Relaxation (P) of k-DM and its dual (D).

model allows memory size for processing proportional (up to polylog factors) to the size of the memory needed to store the output. For k-DM, the final solution size is $O(nk)$, and hence the memory limit is $O(nk \cdot \text{polylog}(n)) =: \tilde{O}(nk)$ bits of space. We assume that the ratio $W = w_{\max}/w_{\min}$ is poly(n), where $w_{\max} = \max_{e\in E}\{w(e)\}$ and $w_{\min} = \min_{e\in E}\{w(e)\}$. This allows for storing edge weights and their sums in $O(\log n)$ bits.

3 Related Work

Offline Approximation Algorithms. In the offline setting, Hanauer et al. [21] designed six approximation algorithms for k-DM. Three of these algorithms are based on an iterative matching framework where k matchings are successively computed by running a matching algorithm and removing the matched edges from the graph. This framework was used with the Blossom [10] algorithm, which computes an exact MWM solutions, and the Greedy and Global Path [29] algorithms, which compute $\frac{1}{2}$-approximate MWM solutions. They also designed a b-matching-based algorithm, where a Greedy $(k-1)$-matching is first found and then converted in a k-disjoint matching using the Misra-Gries edge coloring algorithm [32]. Additionally, two direct algorithms, NodeCentered and k-Edge Coloring, which do not use matching algorithms as a subroutine were also proposed. The NodeCentered algorithm assigns ratings to vertices, which are then processed in rating-decreasing order, and up to k edges a vertex is incident on are colored with any available color in weight-decreasing order. A threshold $\theta \in [0,1]$ is also introduced, which avoids an overly Greedy approach by deferring the coloring of edges with weight less than $\theta w_{\max}$. The k-Edge Coloring algorithm is an adaption of the Misra-Gries $(\Delta+1)$ edge coloring algorithm [32] that is restricted to using k colors and accounts for edge weights. The iterative GPA, b-matching based, NodeCentered, and k-Edge Coloring algorithms are shown to be *at most* $\frac{1}{2}$-approximate, while the Blossom variant is shown to be at most $\frac{7}{9}$-approximate and the Greedy variant is $\frac{1}{2}$-approximate.

Matchings in the Semi-Streaming Model. Matching is an active area of research in the semi-streaming model. For MWM in the single pass, arbitrary order stream setting, Feigenbaum et al. [15] first gave a $\frac{1}{6}$-approximation algorithm. This was improved on by a series of papers [8, 12, 30, 38], until the current state-of-the-art result by Paz and Schwartzman [34] who showed that a simple local-ratio algorithm achieves a $\frac{1}{2+\varepsilon}$-approximation. Ghaffari and Wajc [18] further simplified the analysis of this algorithm by giving both a primal-dual and charging-based analysis. This algorithm was implemented recently by Ferdous et al. [16] and it was shown to reduce memory requirements by one to two orders of magnitude over offline $\frac{1}{2}$-approximate algorithms, while being close to the best of them in run time and matching weight. On the hardness front, Kapralov [25] showed that no single-pass semi-streaming algorithm can have an approximation ratio better than $\frac{1}{1+\ln 2} \approx 0.59$ in arbitrary order streams. In random order streams, Gamlath et al. [17] designed a $(\frac{1}{2} + c)$-approximate algorithm, where $c > 0$ is some absolute constant.

For MWbM in the single pass, arbitrary order stream setting, Levin and Wajc [27] designed a $\frac{1}{3+\varepsilon}$-approximate algorithm using a primal-dual framework, which was recently improved to $\frac{1}{2+\varepsilon}$ by Huang and Sellier [23]. A variant of the latter algorithm requires $\tilde{O}(|F_{\max}| \log_{1+\varepsilon}(W/\varepsilon))$ bits, where $|F_{\max}|$ is the size of a max cardinality b-matching in G.

Edge Colorings and Unweighted k-DM. The k-DM problem is equivalent to a weighted variant of the Edge Coloring problem; in the latter, the goal is to find the *chromatic index* of a graph, i.e., the minimum number of colors needed such that adjacent edges receive distinct colors. Vizing [37] showed that the chromatic index of any simple graph G is in $\{\Delta, \Delta + 1\}$, but it is NP-hard to decide between them [22]. Hence, most edge coloring algorithms, like the $O(nm)$ time Misra-Gries algorithm [32], construct $(\Delta + 1)$-edge colorings. The k-DM problem can be seen as a "maximization" variant of Edge Coloring, where given the number of colors k as input, the goal is to find a maximum weight subgraph with chromatic index k.

Using this coloring viewpoint, Feige et al. [14] provided several hardness results and approximation algorithms for unweighted k-DM in the offline setting, which was later improved by Kamiński and Kowalik [24] for small k. Favrholdt and Nielson [13] additionally gave algorithms for this problem in the online setting. Recently, El-Hayek et al. [11] developed fully dynamic unweighted k-DM algorithms by reducing it to dynamic b-matching followed by edge coloring.

4 A Primal-Dual Approach

In this section we extend the streaming algorithm of Paz and Schwartzman (henceforth, PS) [34], and more specifically the primal-dual interpretation of it by Ghaffari and Wajc [18], for the MWM problem to the k-DM problem. We begin with an intuitive description of the PS algorithm; in the full version of the paper (see Page 1), we include a formal description.

Consider the non-streaming setting first. The algorithm chooses an edge with positive weight, includes it in a stack for candidate matching edges, and subtracts its weight from neighboring edges. It repeats this process as long as edges with positive weights remain. At the end, we unwind the stack and greedily add edges in the stack to the matching. This means that once an edge is added to the matching, any neighboring edges in the stack cannot be added to the matching.

To adapt the algorithm to the streaming setting, an approximate dual variable $\phi(v)$ is kept for each vertex v that accumulates the weights of the edges incident on v that are added to the stack. When an edge arrives, we subtract the sum of the $\phi(\cdot)$ variables of the endpoints of the edge from its weight. If this reduced weight is positive, it is added to the stack; otherwise, it is discarded. The rest of the algorithm proceeds as in the non-streaming setting. To bound the size of the stack to $O(n \log n)$, we need one more idea, which is to add an edge $e = (u, v)$ to the stack only if its weight is greater than $(1 + \varepsilon)(\phi(u) + \phi(v))$, for a small constant $\varepsilon > 0$. This ensures that neighboring edges added to the stack have weights that increase exponentially in $(1 + \varepsilon)$. It can be shown that if the edge weights are polynomial in n, then the size of the stack is bounded as desired and that the approximation ratio becomes $\frac{1}{2+\varepsilon}$.

Algorithm 1 Semi-Streaming k-DM.

Input: A stream of edges E, an integer k, and a constant $\varepsilon > 0$
Output: A $\frac{1}{3+\varepsilon}$-approximate k-disjoint matching $\mathcal{M}$ using $O(nk \log^2 n)$ bits of space

```
▷ Initialization
∀v ∈ V, ∀c ∈ [k] : φ(c, v) ← 0
S ← {S_1, ..., S_k}, where S(c) denotes stack S_c

▷ Streaming Phase
for e = (u, v) ∈ E do
    for c ∈ [k] do
        φ_c = φ(c, u) + φ(c, v)
        if w(e) ≥ (1 + ε)φ_c then
            w'(c, e) ← w(e) − φ_c
            φ(c, u) ← φ(c, u) + w'(c, e)
            φ(c, v) ← φ(c, v) + w'(c, e)
            S(c).push(e); break

▷ Post-Processing
∀c ∈ [k] : M_c ← ∅
for c ∈ [k] do
    while S(c) is not empty do
        e = (u, v) ← S(c).pop()
        if V(M_c) ∩ {u, v} = ∅ then
            M_c ← M_c ∪ {e}
        else
            for j ∈ [c + 1..k] do
                φ_j = φ(j, u) + φ(j, v)
                if w(e) ≥ (1 + ε)φ_j then
                    w'(j, e) ← w(e) − φ_j
                    φ(j, u) ← φ(j, u) + w'(j, e)
                    φ(j, v) ← φ(j, v) + w'(j, e)
                    S(j).push(e); break
return M = {M_1, ..., M_k}
```

We adapt this general idea to develop our algorithm for k-DM in Algorithm 1. For each color $c \in [k]$, we maintain a stack $\mathcal{S}(c)$ that stores the eligible edges for the c^{th} matching. A matching M_c is then greedily computed from each stack $\mathcal{S}(c)$ in the post-processing phase. The algorithm maintains *approximate* dual variables $\phi(c, v)$ for each color $c \in [k]$ and $v \in V$, and uses $\varepsilon > 0$ to process only sufficiently heavy edges. For an edge $e = (u, v)$ in the stream, we iterate over the colors $c \in [k]$ to verify whether $w(e) \geq (1 + \varepsilon)\left(\phi(c, u) + \phi(c, v)\right)$. If the condition is not satisfied for any color, then the edge is discarded. Otherwise, let ℓ be the first color that satisfies it. The algorithm computes a reduced weight $w'(\ell, e)$ for e by subtracting the sum $\phi(\ell, u) + \phi(\ell, v)$ from its weight $w(e)$, pushes e into $\mathcal{S}(\ell)$, and increases $\phi(\ell, u)$ and $\phi(\ell, v)$ by the reduced weight $w'(\ell, e)$.

In the post-processing phase, each stack $\mathcal{S}(c)$ is processed in increasing order of the color c, and the edges in each stack are processed in reverse order in which they were added (i.e., by popping from the stack). For an edge $e = (u, v)$ popped from $\mathcal{S}(c)$, if no earlier popped edge from $\mathcal{S}(c)$ is incident on either u or v in M_c, then e is added to M_c. Otherwise, the algorithm checks to see if e can be added to a later stack $\mathcal{S}(j)$ where $j > c$, again based on the condition that $w(e) \geq (1 + \varepsilon)\left(\phi(j, u) + \phi(j, v)\right)$. At termination, the algorithm returns a k-disjoint matching $\mathcal{M} = \{M_1, \ldots, M_k\}$.

4.1 Analysis of the Algorithm

We prove the approximation ratio of Algorithm 1 using the standard primal-dual framework and adapting the analysis in [18]. We first show how to derive a feasible dual solution for LP (D) from the $\phi(\cdot, \cdot)$ values. By weak duality, the resulting dual objective immediately provides an upper bound on the weight of an optimal k-DM solution. Lemmas 2 and 3 then show lower bounds between the value of the k-disjoint matching $\mathcal{M}$ constructed by Algorithm 1 and the dual variables, which are then used to prove that $\mathcal{M}$ is $\frac{1}{3+2\varepsilon}$-approximate in Theorem 4. We also prove the space complexity of the algorithm in Lemma 6.

Dual Feasibility. At termination, we set $y(c,v) = (1+\varepsilon)\,\phi(c,v)$ for all $c \in [k]$ and $v \in V$. Recall that $y(c,v)$ is a dual variable from (D), and $\phi(c,v)$ is an approximate dual variable used in Algorithm 1. Unlike in classical MWM, for k-DM, we have to satisfy the dual constraints of each edge for all $c \in [k]$. Although the dual variables $z(\cdot)$ are unused in the algorithm, they help ensure dual feasibility; see below. If an edge $e = (u,v)$ is not in any matching (i.e, e is discarded either in the streaming or post-processing phase) then $y(c,u) + y(c,v) \geq w(e)$ for all $c \in [k]$, which satisfies the constraint. However, if $e \in M_\ell$ for some $\ell \in [k]$, the dual constraints for $c \in [\ell+1..k]$ may be violated. Thus, we set

$$z(e) = \max\left\{0,\ \max_{c\in[k]}\{w(e) - (1+\varepsilon)\,(\phi(c,u) + \phi(c,v))\}\right\}. \tag{1}$$

The following claim is immediate.

▷ Claim 1. For all vertices $v \in V$, edges $e \in E$, and $c \in [k]$, the dual variables $y(c,v)$ and $z(e)$ defined above constitute a feasible solution to (D).

Approximation Ratio. To prove the approximation ratio, we first separately relate the weight of the solution returned by Algorithm 1 to the summations of the $\phi(\cdot,\cdot)$ and $z(\cdot)$ variables.

▶ **Lemma 2.** *The solution $\mathcal{M}$ output by Algorithm 1 satisfies $w(\mathcal{M}) \geq \frac{1}{2}\sum_{c\in[k]}\sum_{v\in V}\phi(c,v)$.*

Proof. It suffices to show that $w(M_c) \geq \frac{1}{2}\sum_{v\in V}\phi(c,v)$, for any matching $M_c \in \mathcal{M}$. Let E_c be the set of edges that were pushed to the stack $\mathcal{S}(c)$ at some point in either the streaming (line 8) or the post-processing (line 23) phases. Note that only edges in E_c could have caused the $\phi(c,\cdot)$ values to increase. For ease of analysis, for an edge $e' = (s,t) \in E_c$ let $\phi^{\text{old}}_{e'}(c,\cdot)$ and $\phi^{\text{new}}_{e'}(c,\cdot)$ denote the $\phi(c,\cdot)$ values before and after e' is pushed to $\mathcal{S}(c)$, respectively. By definition of how we update the $\phi(c,\cdot)$ values, we have $\phi^{\text{new}}_{e'}(c,s) = \phi^{\text{old}}_{e'}(c,s) + w'(c,e')$, $\phi^{\text{new}}_{e'}(c,t) = \phi^{\text{old}}_{e'}(c,t) + w'(c,e')$, and $\phi^{\text{new}}_{e'}(c,r) = \phi^{\text{old}}_{e'}(c,r)$ for all $r \in V \setminus \{s,t\}$. This implies

$$w'(c,e') = \frac{1}{2}\sum_{x\in e'}\phi^{\text{new}}_{e'}(c,x) - \phi^{\text{old}}_{e'}(c,x). \tag{2}$$

Upon termination of Algorithm 1, since initially $\phi(c,v) = 0$ for all $v \in V$, we also have that

$$\phi(c,v) = \sum_{e'\in E_c}\phi^{\text{new}}_{e'}(c,v) - \phi^{\text{old}}_{e'}(c,v). \tag{3}$$

Now for an edge $e = (u,v) \in M_c$, let

$$\mathcal{P}_<(c,e) \coloneqq \{e' \in E_c \colon e \cap e' \neq \emptyset, e' \text{ added to } \mathcal{S}(c) \text{ before } e\},$$

i.e., the set of edges adjacent to e that were pushed to $\mathcal{S}(c)$ before e was, and let $\mathcal{P}(c,e) \coloneqq \mathcal{P}_<(c,e) \cup \{e\}$. Note that since we construct M_c greedily, no edge $e' \in \mathcal{P}_<(c,e)$ is included in M_c and $E_c = \bigcup_{e\in M_c}\mathcal{P}(c,e)$. By definition of how we update the $\phi(c,\cdot)$ values, we have that $\phi^{\text{old}}_e(c,u) + \phi^{\text{old}}_e(c,v) = \sum_{e'\in\mathcal{P}_<(c,e)} w'(c,e')$. Additionally, by the definition of $w'(c,e)$,

$$\begin{aligned} w(e) &= w'(c,e) + \phi^{\text{old}}_e(c,u) + \phi^{\text{old}}_e(c,v) \\ &= \sum_{e'\in\mathcal{P}(c,e)} w'(c,e') = \frac{1}{2}\sum_{e'\in\mathcal{P}(c,e)}\sum_{x\in e'}\phi^{\text{new}}_{e'}(c,x) - \phi^{\text{old}}_{e'}(c,x), \end{aligned}$$

where the last equality follows by Eq. (2). Hence,

$$w(M_c) = \sum_{e \in M_c} w(e) = \frac{1}{2} \sum_{e \in M_c} \sum_{e' \in \mathcal{P}(c,e)} \sum_{x \in e'} \phi_{e'}^{\text{new}}(c,x) - \phi_{e'}^{\text{old}}(c,x)$$
$$\geq \frac{1}{2} \sum_{e \in E_c} \sum_{v \in e} \phi_e^{\text{new}}(c,v) - \phi_e^{\text{old}}(c,v)$$
$$= \frac{1}{2} \sum_{v \in V} \sum_{e \in E_c} \phi_e^{\text{new}}(c,v) - \phi_e^{\text{old}}(c,v) = \frac{1}{2} \sum_{v \in V} \phi(c,v).$$

The inequality follows since each edge $e' = (u,v) \notin M_c$ appears in at least one and at most two $\mathcal{P}(c,\cdot)$ sets (say, if there exists $e_1, e_2 \in M_c$ that u and v are incident on, respectively) and the last equality follows by Eq. (3). ◀

▶ **Lemma 3.** *The solution $\mathcal{M}$ output by Algorithm 1 satisfies $w(\mathcal{M}) \geq \sum_{e \in E} z(e)$.*

Proof. From the definition of $z(\cdot)$ in Eq. (1), we have $z(e) \leq w(e)$ for all $e \in E$. Moreover, we can show that $z(e) = 0$ for each edge $e = (u,v) \notin \mathcal{M}$. This holds since either e was discarded during the streaming phase, or during the post-processing phase. In either case, $w(e) < (1+\varepsilon)(\phi(c,u) + \phi(c,v))$ for all $c \in [k]$, which gives $z(e) = 0$. Hence, $\sum_{e \in E} z(e) = \sum_{e \in \mathcal{M}} z(e) + \sum_{e \in E \setminus \mathcal{M}} z(e) \leq \sum_{e \in \mathcal{M}} w(e) = w(\mathcal{M})$. ◀

Using Lemmas 2 and 3 and weak duality, we can now show the approximation ratio.

▶ **Theorem 4.** *For any constant $\varepsilon > 0$, the k-disjoint matching $\mathcal{M}$ returned by Algorithm 1 is a $\frac{1}{3+2\varepsilon}$-approximate solution to k-DM.*

Proof. Let $\mathcal{M}^*$ be an optimal solution to k-DM. By weak duality and the fact that (P) is an LP-relaxation of k-DM, we have that $w(\mathcal{M}^*) \leq \sum_{c \in [k]} \sum_{v \in V} y(c,v) + \sum_{e \in E} z(e)$ for the dual variables $y(\cdot,\cdot)$ and $z(\cdot)$ defined in Claim 1. Recalling that we set $y(c,v) = (1+\varepsilon)\phi(c,v)$, Lemmas 2 and 3 imply that $(2(1+\varepsilon))w(\mathcal{M}) \geq \sum_{c \in [k]} \sum_{v \in V} y(c,v)$ and $w(\mathcal{M}) \geq \sum_{e \in E} z(e)$, respectively. Combining these, we obtain

$$(3+2\varepsilon)w(\mathcal{M}) \geq \sum_{c \in [k]} \sum_{v \in V} y(c,v) + \sum_{e \in E} z(e) \geq w(\mathcal{M}^*),$$

which when rearranged gives $w(\mathcal{M}) \geq \frac{1}{3+2\varepsilon} w(\mathcal{M}^*)$. ◀

Time and Space Complexity. The total runtime of Algorithm 1 is $O(km)$, which follows as the processing time for each edge is $O(k)$ as it may be considered for insertion into each of the k stacks. Additionally, the size of each stack is trivially bounded by m, so the post-processing step of unwinding the stacks takes $O(km)$ time. The space complexity of Algorithm 1 can also easily be bound. We first make the following useful observation.

▶ **Observation 5.** *When an edge $e = (u,v)$ gets pushed to a stack $\mathcal{S}(c)$, both $\phi(c,v)$ and $\phi(c,u)$ increase by at least a factor of $1+\varepsilon$.*

Proof. Let $\phi_e^{\text{old}}(c,\cdot)$ and $\phi_e^{\text{new}}(c,\cdot)$ be the values of $\phi(c,\cdot)$ before and after e is pushed to $\mathcal{S}(c)$, respectively. Note that since e is pushed to $\mathcal{S}(c)$, it must be that $w(e) \geq (1+\varepsilon)\Phi_e^{\text{old}}$, where $\Phi_e^{\text{old}} := \phi_e^{\text{old}}(c,u) + \phi_e^{\text{old}}(c,v)$. Additionally, by how we update the $\phi(c,\cdot)$ values, we have $\phi_e^{\text{new}}(c,v) - \phi_e^{\text{old}}(c,v) = w'(c,e) = w(e) - \Phi_e^{\text{old}}$. Thus,

$$\phi_e^{\text{new}}(c,v) - \phi_e^{\text{old}}(c,v) = w(e) - \Phi_e^{\text{old}} \geq (1+\varepsilon)\Phi_e^{\text{old}} - \Phi_e^{\text{old}} \geq \varepsilon\phi_e^{\text{old}}(c,v).$$

Rearranging, we get $\phi_e^{\text{new}}(c,v) \geq (1+\varepsilon)\phi_e^{\text{old}}(c,v)$. The same argument holds for vertex u. ◀

▶ **Lemma 6.** *For any constant $\varepsilon > 0$, Algorithm 1 uses $O(nk\log^2 n)$ bits of space.*

Proof. Consider a vertex $v \in V$ and color $c \in [k]$. Let $e = (u, v)$ be an edge that is pushed to $\mathcal{S}(c)$, and let $\phi_e^{\text{old}}(c, \cdot)$ and $\phi_e^{\text{new}}(c, \cdot)$ denote the values of $\phi(c, \cdot)$ before and after e is pushed to $\mathcal{S}(c)$, respectively. Suppose that after e is pushed, we have that v is incident on d edges in $\mathcal{S}(c)$. For the special case of $d = 1$, corresponding to the first edge incident on v included in $\mathcal{S}(c)$, we can derive a lower bound on $\phi_e^{\text{new}}(c, v)$. We use $\phi_e^{\text{old}}(c, v) = 0$ and $w(e) \geq (1 + \varepsilon)\phi_e^{\text{old}}(c, u)$ to obtain

$$\phi_e^{\text{new}}(c, v) = w'(c, e) = w(e) - \phi_e^{\text{old}}(c, u) \geq w(e) - \frac{w(e)}{1+\varepsilon} \geq \frac{\varepsilon w_{\min}}{1+\varepsilon}.$$

That is, the minimum non-zero value of $\phi(c, v)$ is at least $\frac{\varepsilon w_{\min}}{1+\varepsilon}$. Using this together with Observation 5 implies that for arbitrary values of d, $\phi_e^{\text{new}}(c, v) \geq \frac{\varepsilon w_{\min}}{1+\varepsilon}(1 + \varepsilon)^{d-1}$. Moreover, by definition of how we compute reduced weights and update the $\phi(c, \cdot)$ values, we have that $\phi_e^{\text{new}}(c, v) \leq w_{\max}$. Recalling that $W = \frac{w_{\max}}{w_{\min}}$ and using these two bounds, we find that $(1 + \varepsilon)^{d-2} \leq W\varepsilon^{-1}$. Taking the logarithm of both sides, we get

$$d \leq 2 + \log_{1+\varepsilon}(W\varepsilon^{-1}) = O(\log n),$$

since we assume ε is constant and W is poly(n). That is, v can be incident on at most $O(\log n)$ edges in $\mathcal{S}(c)$. Hence, $|\mathcal{S}(c)| = O(n \log n)$ and the total number of edges stored in all the stacks is $O(nk \log n)$. Each edge weight requires $O(\log n)$ bits; similarly, each $\phi(\cdot, \cdot)$ variable requires $O(\log n)$ bits as it is the sum of at most $\Delta < n$ edge weights, giving the space complexity of $O(nk \log^2 n)$ bits. ◀

5 A b-Matching Based Approach

Recall that a b-matching generalizes a matching by allowing each vertex to be incident to *at most* $b(v)$ matched edges for some function $b \colon V \to \mathbb{Z}_+$. When $b(v) = k$ for all $v \in V$, where k is some positive integer, we refer to the matching as a k-matching and consider the Max Weight k-Matching (MWkM) problem. Note that k-disjoint matchings always induce valid k-matchings, but the reverse need not hold (e.g., the triangle graph with $k = 2$). In this sense, MWkM provides a relaxation of k-DM (i.e., if F^* and $\mathcal{M}^*$ are optimal solutions to MWkM and k-DM on the same graph, respectively, then $w(F^*) \geq w(\mathcal{M}^*)$). This leads to the following approach to construct a feasible k-disjoint matching:

1. Solve MWkM on the graph G, which gives a k-matching F. Note that Δ_F, the maximum degree of a vertex in the induced graph $G[F]$, may be less than k.
2. Properly $(\Delta_F + 1)$-edge color the subgraph $G[F]$, which may use up to $k + 1$ colors.
3. Return $\mathcal{M}$, the collection of edges colored by the k heaviest color classes.

This approach was originally used for unweighted k-DM by Feige et al. [14], where they showed it provided a $(1 - \frac{1}{k+1})$-approximation guarantee. Here we extend this to weighted k-DM and show that the reduction is approximation preserving.

▶ **Lemma 7.** *Let F be an α-approximate solution to MWkM on a graph G. If the induced subgraph $G[F]$ is properly $(\Delta_F + 1)$ colored, the set of edges colored by the k heaviest color classes is an $\alpha(1 - \frac{1}{k+1})$-approximate solution to k-DM on G.*

Proof. Let $\mathcal{M}$ represent a solution to k-DM on G. Additionally, let F^* and $\mathcal{M}^*$ be the optimal solutions to MWkM and k-DM on G, respectively.

Algorithm 2 Semi-Streaming k-DM

Input: A stream of edges E, an integer k, and a constant $\varepsilon > 0$
Output: A $(\frac{1}{2+\varepsilon})(1 - \frac{1}{k+1})$-approximate k-disjoint matching $\mathcal{M}$ using $\tilde{O}(nk)$ bits

▷ *Initialization*
$\forall v \in V : b(v) \leftarrow k$

▷ *Streaming Phase*
$F \leftarrow$ SS-bM$(E,\, b,\, \frac{\varepsilon}{2})$ ▷ *MWkM*

▷ *Post-Processing*
$\Delta_F \leftarrow \max_{v \in V} \deg_F(v)$ ▷ $\Delta_F \leq k$
$\mathcal{C} \leftarrow$ Color$(G[F])$ ▷ *Uses* $[\Delta_F + 1]$ *colors*
if $\Delta_F + 1 = k + 1$ **then**
  Let $k+1$ be the color class with min weight
$\forall i \in [k] \colon M_i \leftarrow \{e \in F \colon \mathcal{C}(e) = i\}$
return $\mathcal{M} = \{M_1, \ldots, M_k\}$

By definition of a k-matching, we have that $\Delta_F \leq k$. If $\Delta_F < k$, then the edge coloring used at most k colors, and we can return $\mathcal{M} = \{M_1, \ldots, M_k\}$, where M_i is the set of edges colored with i for $i \in [k]$. In this case, we have $w(\mathcal{M}) = w(F)$. Otherwise, if $\Delta_F = k$, then the edge coloring may have used $k+1$ colors. Without loss of generality, let $k+1$ denote the color class with the minimum weight. Again let $\mathcal{M} = \{M_1, \ldots, M_k\}$. By discarding the edges with color $k+1$, at most a $\frac{1}{k+1}$ fraction of the weight of F is lost. Thus, in either case

$$w(\mathcal{M}) \geq \left(1 - \frac{1}{k+1}\right) w(F) \geq \alpha \left(1 - \frac{1}{k+1}\right) w(F^*) \geq \alpha \left(1 - \frac{1}{k+1}\right) w(\mathcal{M}^*),$$

where the penultimate inequality follows from the definition of F, and the last inequality follows from MWkM being a relaxation of k-DM. ◀

Note that properly $(\Delta + 1)$-edge coloring a graph G can be done in $O(m)$ space using the $O(nm)$ time Misra-Gries algorithm [32]. If we use a semi-streaming algorithm for MWkM to handle the streaming process and find some k-matching F, the remaining steps of the algorithm only require memory linear in $|F|$, resulting in a semi-streaming algorithm for k-DM. Using the semi-streaming $\frac{1}{2+\varepsilon}$-approximation algorithm of Huang and Sellier [23] for MWbM with $b(v) = k$ for all $v \in V$, Lemma 7 implies a semi-streaming $(\frac{1}{2+\varepsilon})(1 - \frac{1}{k+1})$-approximation algorithm for k-DM. The space requirement is $O(nk \log^2 n)$ bits, and it is determined by the Huang and Sellier algorithm. We describe the algorithm formally in Algorithm 2, where SS-bM and Color refer to the algorithms of Huang and Sellier [23] and Misra and Gries [32], respectively. In the full version of this paper, we give a detailed summary of these algorithms.

▶ **Theorem 8.** *For any constant $\varepsilon > 0$, Algorithm 2 is a $(\frac{1}{2+\varepsilon})(1 - \frac{1}{k+1})$-approximate semi-streaming algorithm for k-DM that uses $O(nk \log^2 n)$ bits of space.*

The streaming phase requires $O(k)$ processing time per edge, while constructing the k-matching F takes $O(m)$ time. By definition of a k-matching, $|F| = O(kn)$, so the post-processing coloring step requires $O(kn^2)$ time. Thus the time complexity of Algorithm 2 is $O(km + kn^2)$.

6 Heuristic Improvements

In this section, we describe some heuristics we employ to speed up and improve the weight of both streaming algorithms we have presented.

Dynamic Programming (DP) Based Weight Improvement. Manne and Halappanavar [28] have proposed a general scheme to enhance the weight of a matching by computing *two* edge-disjoint matchings M_1 and M_2. The induced subgraph $G[M_1 \cup M_2]$ contains only cycles of even length or paths. Utilizing a linear-time dynamic programming approach, an optimal matching M' can be derived from the induced graph $G[M_1 \cup M_2]$. The weight of M' is guaranteed only to be at least as large as $\max\{w(M_1), w(M_2)\}$, but in practice this heuristic results in substantially improved weight.

We adapted this method for Algorithm 1 as follows: instead of computing a k-disjoint matching, we first compute a $2k$-disjoint matching. These $2k$ matchings are then merged into k matchings. While various strategies can be used for this merging process, we have merged the i^{th} matching with the $(2k - i + 1)^{\text{th}}$ matching, for $i \in [k]$. This approach does not change the asymptotic memory or time complexities for streaming algorithms since each merge requires only $O(n)$ time and space.

Common Color and Merge. For the b-matching based Algorithm 2, we used two heuristics. The first is the *common color* heuristic described by Hanauer et al. [21], which attempts to color an edge by first determining if there is a common free color on both of its endpoints before going through the Misra-Gries routine. The second is the *merge* heuristic, which is used when the number of color classes is $k + 1$; it tries to improve the solution weight by merging the lowest- and second-lowest-weight color classes instead of completely discarding the lowest-weight one, again through the dynamic programming approach described above.

7 Experiments and Results

This section reports experimental results for 95 real-world and synthetic graphs. All the codes were executed on a node of a community cluster computer with 128 cores in the node, where the node is an AMD EPYC 7662 with 1 TB of total memory over all the cores. The machine has three levels of cache memory. The L1 data and instruction caches, the L2 cache, and the L3 cache have 4 MB, 32 MB, and 256 MB of memory, respectively. The page size of the node is 4 KB.

Our implementation uses C++17 and is compiled with g++9.3.0 with the -O3 optimization flag. The streaming algorithms are simulated by sequentially reading and processing edges from a file using the C++ `fstream` class. We compare them against several offline algorithms in the DJ-Match software suite [20]. All the streaming and offline algorithms are sequential, and the reported runtimes *do not* include file reading times and (for the offline algorithms) graph construction times. For memory, we use the `getrusage` system call to report the maximum resident set size (RSS) during the program's execution.

7.1 Datasets and Benchmark Algorithms

Real-World and Synthetic Graphs. Following [21, 26], we include ten weighted graphs from the SuiteSparse Matrix Collection [9] labeled as SMALL. Similar to [21], we also generated 66 synthetic instances, labeled as RMAT, using the R-MAT model [6] with 2^x vertices, where $x \in [10, 11, \ldots, 20]$. We used three initiator matrices, $\mathsf{rmat}_{\mathrm{b}} = (0.55, 0.15, 0.15, 0.15)$, $\mathsf{rmat}_{\mathrm{g}} = (0.45, 0.15, 0.15, 0.25)$, and $\mathsf{rmat}_{\mathrm{er}} = (0.25, 0.25, 0.25, 0.25)$. For all these graphs, we assign real-valued random weights in the range $[1, 2^{19}]$ drawn from uniform or exponential distributions. Our LARGE dataset consists of six of the *largest* undirected graphs in the SuiteSparse Matrix Collection [9], each having more than 1 billion edges. For the unweighted graphs, we assign uniform random real weights in the range $[1, 10^6]$. In the full version of this paper, we include a Table listing the sizes and degree measures of these graphs.

Table 1 Benchmark approximation algorithms. LS: Local swaps, CC: Common color, RL: Rotate long, M: Merge, Agg: Aggregation, srt(x): Time complexity of sorting x elements.

Algorithm	Heuristics	Approx.	Time Complexity
GRDY-IT	LS	$1/2$	$O(\text{srt}(m) + km)$
GPA-IT	LS	$\leq 1/2$	$O(\text{srt}(m) + km)$
NC	$\theta = 0.2$, Agg=sum	$\leq 1/2$	$O(\text{srt}(n) + n \cdot \text{srt}(\Delta) + km)$
K-EC	CC-RL	$\leq 1/2$	$O(\text{srt}(m) + kn^2)$
STK	DP	$\frac{1}{3+\epsilon}$	$O(km)$
STKB	CC-M	$\frac{k}{(2+\varepsilon)(k+1)}$	$O(km + kn^2)$

Network Trace Data. Similar to [21], our network trace (TRACE) dataset consists of i) Facebook Data Traces [36]: Six production-level traces of three clusters from Facebook's Altoona Data Center, ii) HPC Data [2]: MPI traces for four different applications run in parallel, and iii) pFabric Data [1, 2]: Three synthetic pFabric traces generated from Poisson processes with flow rates in $\{0.1, 0.5, 0.8\}$. From these trace data, we pre-compute graphs by assigning the total demand of a pair of nodes (i.e., the number of times they appear in the trace) as the edge weight. In the full version of this paper, we include a Table listing detailed statistics of these generated graphs.

Benchmark Algorithms and Heuristics. We summarize the algorithms we compare in Table 1. For our streaming algorithms, we use STK to denote the primal-dual based Algorithm 1, and STKB to denote the b-matching based Algorithm 2. We compare these against four of the offline algorithms that were determined to be the most practical (in terms of runtime and solution quality) by Hanauer et al. [21]. These include the iterative Greedy (GRDY-IT) and iterative Global Paths algorithms (GPA-IT), the NodeCentered algorithm (NC), and the k-Edge Coloring algorithm (K-EC) that we have described in Section 3. For these four offline algorithms, we use the heuristics and post-processing steps recommended in [21], which we list in Table 1. We refer to [21] for a detailed description of these heuristics. For our semi-streaming algorithms, we implement the three heuristics described in Section 6. We use DP to denote the dynamic programming heuristic for the STK algorithm, and CC and M for the common color and merge heuristics, respectively, for the STKB algorithm.

7.2 Comparison of Streaming Algorithms

We first compare six variants of our streaming algorithms amongst themselves. For the primal-dual approach, we include the standard STK algorithm and the STK-DP heuristic. For the b-matching-based approach, we have the CC (common color) and M (merge) heuristics in addition to the standard STKB algorithm, for a total of four combinations.

In Figure 2, we show the relative quality results on the SMALL graphs for the streaming algorithms. We set $\varepsilon = 0.001$ and tested with $k \in \{2, 4, 8, 16, 32, 64, 96\}$, but observed that beyond $k = 32$, all the algorithms computed similar weights, as at this point, the solutions likely contained nearly the entire graph. Hence, we only report results up to $k = 32$. For each graph, algorithm, and k value combination, we conduct *five* runs and record the mean runtime, memory usage, and solution weight. We calculate *relative time* by taking the ratio of the mean runtime for each algorithm to the mean runtime of a baseline algorithm. *Relative memory* and *relative weight* are similarly computed. We choose STK as the baseline algorithm for runtime and memory comparisons and STK-DP as the baseline for weight comparisons.

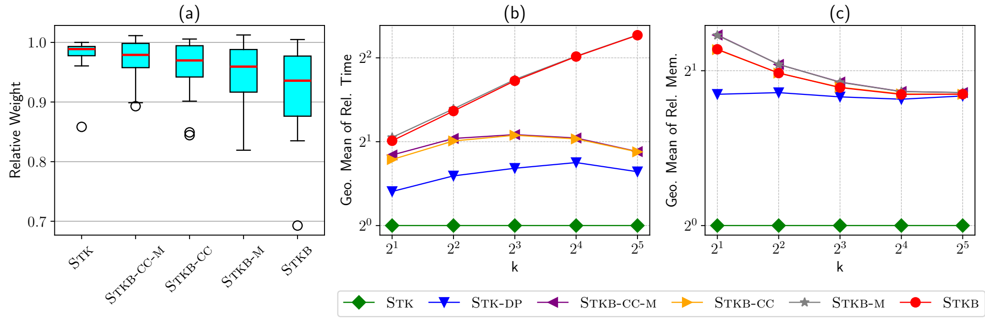

Figure 2 Summary plots for SMALL instances on different streaming algorithms with $\varepsilon = 0.001$. Plot (a) is a boxplot of relative weights across all instances and k values for each algorithm. Plots (b) and (c) give the geometric mean of the relative time and memory, respectively, across all instances with increasing k values. STK is the baseline algorithm for relative time and memory, while STK-DP is the baseline for relative weight.

We show geometric means of the relative weights computed by each algorithm across all graphs and k value combinations as box plots in Figure 2 (a). The relative time and relative memory metrics across increasing k values are plotted in Figure 2 (b) and (c), respectively.

The relative weight of STK-DP is always one, so we do not show it in the plot. In terms of median relative weight (the red line), STK is the second best, and STKB-CC-M is the third best. Surprisingly, while the worst-case approximation guarantee of the primal-dual-based approach is weaker than the b-matching-based approaches, it provides weights that are better than the latter in nearly all instances. For runtimes, we see that the fastest algorithm is STK, while the slowest are STKB and STKB-M. STKB-CC and STKB-CC-M both have similar runtimes and are faster than STKB and STKB-M. The runtime of STK-DP is between STK and STKB-CC-M. In terms of memory usage, STK requires the least, while STK-DP requires roughly twice as much memory as STK($1.76 - 1.86\times$ across k). The other four b-matching-based algorithms behave similarly to each other and are worse than both STK and STK-DP.

From this experiment, *we conclude that among these six streaming algorithm variants, the best three are STK, STK-DP, and STKB-CC-M.* Hence, all the remaining experiments will report results only for these three variants of the streaming algorithms.

7.3 Comparison with Offline Algorithms

Next, we compare the three streaming algorithms with the four offline algorithms listed in Table 1. We show the relative runtime, memory, and weight plots for the algorithms on the SMALL dataset in Figure 3. Due to space constraints, we show the results for the RMAT dataset in the full version of the paper. We follow the experimental settings and computations as in Section 7.2 with STK as the baseline for relative time and memory results, and GPA-IT with local swaps as the baseline for weight results, as these generally performed the best on their respective metrics.

We first discuss the SMALL graph results. *All of the streaming algorithms are significantly faster than the offline ones.* The fastest among these is the STK algorithm, while the slowest is the b-matching based STKB-CC-M. Among the offline algorithms, GPA-IT is the slowest, more than $20\times$ slower in geometric mean than STK, while GRDY-IT is more than $15\times$ slower. The other two algorithms are relatively faster with similar runtimes but still slower than

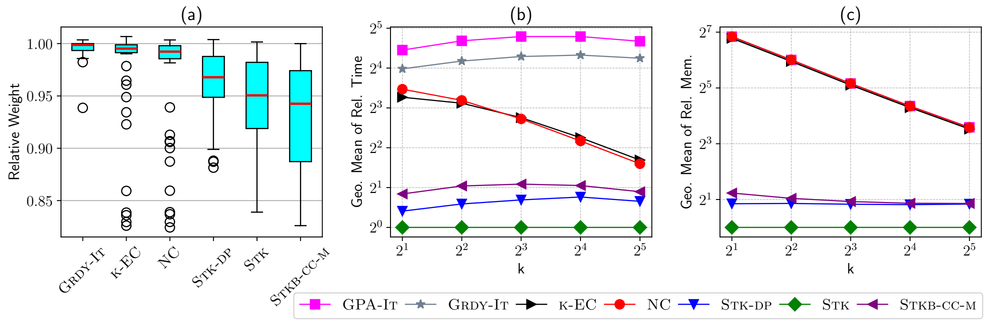

Figure 3 Summary plots the streaming and offline algorithms on SMALL dataset with $\varepsilon = 0.001$ for the streaming algorithms. Plot (a) is a boxplot of relative weights across all instances and k values for each algorithm. Plots (b) and (c) give the geometric mean of the relative time and memory, respectively, across all instances with increasing k values. STK is the baseline algorithm for relative time and memory, while GPA-IT is the baseline for relative weight.

all streaming algorithms. The speedup for STK w.r.t to NC and K-EC ranges from 3 to 11 across k. As an example, for $k = 8$, both NC and K-EC are more than 6× slower than STK. We also observe that both NC and K-EC get relatively more efficient as k increases, which was also reported in [21]. For the memory results, we see that STK requires the least, while the other two streaming algorithms take almost twice the memory, on average. All the offline algorithms behave similarly in terms of memory consumption since they all need to store the entire graph, which dominates the total memory consumption. *We see a substantial memory reduction when using the streaming algorithms, with improvement ranging from 114× to 11× in geometric mean across k.* For smaller values of k this reduction is more pronounced.

We now focus on the case $k = 8$. All the streaming algorithms consume at least 16× less memory than the offline algorithms, while for STK it is 32×. For the largest graph (kron_g500-logn21) in this set, we see all the offline algorithms require at least 45 GB of memory while the streaming algorithms consume less than 1GB of memory. *We emphasize that the higher memory requirement of the offline algorithms prohibits them from being run on larger datasets*, as we will see later. While the streaming algorithms are very efficient in terms of memory and time, we also see they obtain reasonably high solution weights. For the weight results, we set GPA-IT as the baseline algorithm; hence, we do not include it in the box plot. All the offline algorithms find heavier weights than the streaming algorithms; for the NC and K-EC algorithms, we see many outliers compared to the other algorithms. *Among the streaming algorithms, STK-DP obtains the heaviest weight, with only less than 4% median deviation from the best weight.* For STK-DP, the geometric mean of relative weights is 0.96 at $k = 2$ and improves to 0.97 at $k = 32$. The corresponding geometric mean of relative weights for faster offline algorithms, NC and K-EC are as follows: for $k = 2$, the means are 0.96 and 0.97, respectively, and for $k = 32$, they are 0.97 and 0.98, respectively. This highlights STK-DP's comparable quality to the closest practical offline alternatives. The STK and STKB-CC-M algorithms compute weights where the median deviation from the best weight is less than 5% and 6%, respectively.

In the full version of this paper, we report results from similar experiments on the RMAT dataset. *Overall, a similar conclusion can be drawn as the* SMALL *instances.* The random graphs generated are much smaller than the SMALL instances, and hence the memory improvements obtained by the streaming algorithms are smaller (6× to 38× in geometric mean). *For the* RMAT *instances, the streaming algorithms obtain better quality results than*

Table 2 Comparison of streaming algorithms for $k = 8$ and $\varepsilon = 0.001$ on LARGE graphs.

	STK			STK-DP			STKB-CC-M		
Graph	Time (s)	Weight	Mem. (GB)	Rel. Time	% Wt. Imprv.	Rel. Mem.	Rel. Time	% Wt. Imprv.	Rel. Mem.
AGATHA_2015	1377.54	1.60e+14	49.41	1.64	0.67	1.90	0.99	-1.51	1.69
MOLIERE_2016	736.75	8.26e+6	23.28	1.64	2.03	1.78	1.48	0.26	1.22
GAP-kron	629.37	1.20e+10	29.66	1.85	3.05	1.90	1.06	-0.46	1.62
GAP-urand	679.73	9.83e+10	53.25	1.67	3.71	1.57	2.11	-1.30	1.75
com-Friendster	475.13	1.02e+14	22.66	1.62	2.84	1.81	1.49	-4.58	1.54
mycielskian20	86.14	1.99e+12	0.65	2.34	5.30	2.17	0.98	-10.10	1.03

the SMALL *instances*. The difference between the streaming and the NC and K-EC algorithms is *smaller* than seen in the SMALL instances. Both NC and STK-DP achieve similar relative weights, while K-EC is marginally (within 1%) better.

7.4 LARGE Graph Results

We now discuss our LARGE graph experiments. Since these graphs require longer runtimes, and our experiments on the smaller graphs reveal little deviation in runtime and memory across runs (the weight remains constant as our algorithms are deterministic), we report in Table 2 the results of a *single run* of our streaming algorithms. We chose $k = 8$ and set $\varepsilon = 0.001$ for this experiment. The first three columns represent the time in seconds, weight, and memory in GB for the baseline STK algorithm, while the next six columns represent the relative metrics for the STK-DP and STKB-CC-M algorithms. For all the instances, using STK-DP yields an increase in solution quality over STK, with the average increase being 2.93%. Consistent with the results on smaller graphs, STKB-CC-M obtains the lowest weight among the streaming algorithms with weight decreasing in almost all the instances compared to STK and the average decrease is 2.95%. In terms of memory and runtime, STK-DP and STKB-CC-M require at most twice as much memory and time as the STK algorithm. The geometric mean of relative memory and runtime of STK-DP is 1.85 and 1.78, respectively, and for STKB-CC-M they are 1.45 and 1.30, respectively.

For the offline algorithms, we chose NC and K-EC, since the previous experiments show they have much lower runtimes than the other two iterative matching algorithms. *These algorithms could only be run on the smallest graph in this dataset (mycielskian20) while respecting the* 1 *TB memory limit.* For this graph, K-EC and NC obtained weights of 1.70e+12 and 1.68e+12, respectively, which are around 18% less than STK-DP. The K-EC algorithm required more than two hours to compute a solution, while NC required about twenty minutes. This is much worse than any of the streaming algorithms, as even the slowest one (STK-DP) required less than four minutes. Both the NC and K-EC algorithms used around 640 GB of memory, while the memory usage of the streaming algorithms ranges from 660 MB for STK to 1.4 GB for STK-DP, which provides at least a 450-fold reduction.

Effect of varying ε. In the full version of the paper, we show experiments highlighting the effects of varying ε on the LARGE graphs for the STK-DP algorithm. The ε parameter influences both the memory consumption and weight of the solution returned by the algorithm, and we find that as expected, increasing ε decreases both of these values. However, in almost all cases, the decrease in weight is relatively much smaller than the decrease in memory, which suggests that using larger values of ε in practice can substantially decrease the memory usage of the algorithm without significantly decreasing the weight of the solution returned.

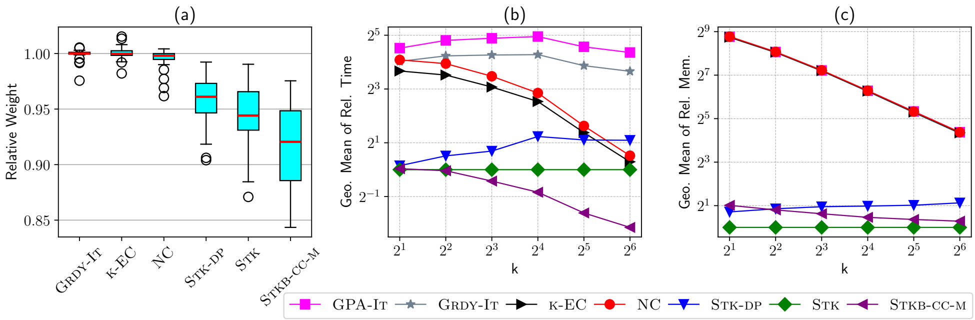

Figure 4 Summary plots the streaming and offline algorithms on Facebook TRACE dataset with $\varepsilon = 0.001$ for the streaming algorithms. Plot (a) is a boxplot of relative weights across all instances and k values for each algorithm. Plots (b) and (c) give the geometric mean of the relative time and memory, respectively, across all instances with increasing k values. STK is the baseline algorithm for relative time and memory, while GPA-IT is the baseline for relative weight.

7.5 TRACE Graph Results

Figure 4 shows experimental results for the TRACE dataset. Due to space constraints, we include only the results for the Facebook graphs since they are the largest. In the full version, we also show the results for the HPC and pFabric data. We use the same baseline algorithms and similar setup as the SMALL dataset experiments. Overall the conclusion is similar to the earlier experiments, except that for these graphs, STKB-CC-M is the fastest. This is because the edge coloring step in the post-processing for the Facebook graphs is much faster than for the other graphs. For STK-DP, STK and STKB-CC-M the median values of the geometric means of the relative weights are 0.96, 0.94, and 0.92, respectively. There are also substantial runtime and memory (10×-512×) improvements compared to the offline algorithms.

8 Conclusions and Future Work

Earlier work on offline maximum weight matching algorithms showed that exact algorithms do not terminate on graphs with hundreds of millions of edges. Hence, offline approximation algorithms with near-linear time complexities based on short augmentations were designed [35]. However, our results show that on graphs with billions of edges, even these algorithms require over 1 TB of memory for the k-DM problem, and do not terminate on such graphs.

Streaming algorithms are designed to reduce memory usage, and our streaming k-DM algorithms effectively reduce it by one to two orders of magnitude on our test set. Our results also show that the streaming algorithms are theoretically and empirically faster. In particular, *we conclude that the STK-DP algorithm is the best performer since it only requires modestly more memory and runtime than the STK algorithm while still computing solutions comparable (within 5%) to the best offline algorithm.* Despite its weaker worst-case approximation ratio, we also find that STK consistently outperforms STKB-CC-M in solution weight. This raises the question of whether the approximation ratio of STK could be improved to $\frac{1}{2+\varepsilon}$.

References

1 Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review*, 43(4):435–446, 2013.

2 Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. On the complexity of traffic traces and implications. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(1):1–29, 2020.

3 Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, and Hugh Williams. Sirius: A flat datacenter network with nanosecond optical switching. In *Proc. of the 2020 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM 2020)*, pages 782–797, 2020. doi:10.1145/3387514.3406221.

4 Marcin Bienkowski, David Fuchssteiner, Jan Marcinkowski, and Stefan Schmid. Online dynamic b-matching: With applications to reconfigurable datacenter networks. *SIGMETRICS Performance Evaluation Review*, 48(3):99–108, 2021. doi:10.1145/3453953.3453976.

5 Marcin Bienkowski, David Fuchssteiner, and Stefan Schmid. Online b-matchings for reconfigurable datacenters: The power of randomization. *arXiv preprint arXiv:2209.01863*, 2022. doi:10.48550/arXiv.2209.0186.

6 Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In *Proc. of the 2004 SIAM International Conference on Data Mining (SDM 2004)*, pages 442–446, 2004. doi:10.1137/1.9781611972740.43.

7 Ernest J. Cockayne, Bert L. Hartnell, and Stephen T. Hedetniemi. A linear algorithm for disjoint matchings in trees. *Discrete Mathematics*, 21(2):129–136, 1978. doi:10.1016/0012-365X(78)90085-7.

8 Michael S. Crouch and Daniel M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Proc. of the 17th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2014)*, pages 96–104, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

9 Timothy A. Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1), 2011. doi:10.1145/2049662.2049663.

10 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

11 Antoine El-Hayek, Kathrin Hanauer, and Monika Henzinger. On b-matching and fully-dynamic maximum k-edge coloring, 2023. doi:10.48550/arXiv.2310.01149.

12 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011. doi:10.1137/100801901.

13 Lene M. Favrholdt and Morten N. Nielsen. On-line edge-coloring with a fixed number of colors. *Algorithmica*, 35:176–191, 2003. doi:10.1007/s00453-002-0992-3.

14 Uriel Feige, Eran Ofek, and Udi Wieder. Approximating maximum edge coloring in multigraphs. In *Proc. of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2002)*, pages 108–121, 2002. doi:10.1007/3-540-45753-4_11.

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

16 SM Ferdous, Alex Pothen, and Mahantesh Halappanavar. Streaming matching and edge cover in practice. In Leo Liberti, editor, *Proceedings of the 22nd International Symposium on Experimental Algorithms (SEA 2024), LIPIcs*, volume 301, 2024.

17 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proc. of the 2019 ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 491–500, 2019. doi:10.1145/3293611.3331603.

18 Mohsen Ghaffari and David Wajc. Simplified and space-optimal semi-streaming $(2+\epsilon)$-approximate matching. In *Proc. of the 2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, pages 13:1–13:8, 2019. doi:10.4230/OASIcs.SOSA.2019.13.

19 Kathrin Hanauer, Monika Henzinger, Lara Ost, and Stefan Schmid. Dynamic demand-aware link scheduling for reconfigurable datacenters. In *Proc. of the 42nd IEEE Conference on Computer Communications (INFOCOM 2023)*, 2023. doi:10.48550/arXiv.2301.05751.

20 Kathrin Hanauer, Monika Henzinger, Stefan Schmid, and Jonathan Trummer. DJ-Match/DJ-Match: Version 1.0.0, January 2022. doi:10.5281/zenodo.5851268.

21 Kathrin Hanauer, Monika Henzinger, Stefan Schmid, and Jonathan Trummer. Fast and heavy disjoint weighted matchings for demand-aware datacenter topologies. In *Proc. of the 41st IEEE Conference on Computer Communications (INFOCOM 2022)*, pages 1649–1658, 2022. doi:10.1109/INFOCOM48880.2022.9796921.

22 Ian Holyer. The NP-Completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981. doi:10.1137/0210055.

23 Chien-Chung Huang and François Sellier. Semi-streaming algorithms for submodular function maximization under b-matching constraint. In *Proc. of the 24th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2021)*, pages 14:1–14:18, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.14.

24 Marcin Kamiński and Łukasz Kowalik. Beyond the Vizing's bound for at most seven colors. *SIAM Journal on Discrete Mathematics*, 28(3):1334–1362, 2014. doi:10.1137/120899765.

25 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In *Proc. of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 1874–1893, 2021. doi:10.1137/1.9781611976465.112.

26 Arif Khan, Krzysztof Choromanski, Alex Pothen, S. M. Ferdous, Mahantesh Halappanavar, and Antonino Tumeo. Adaptive anonymization of data using b-edge cover. In *Proc. of the 2018 International Conference for High Performance Computing, Networking, Storage, and Analysis (SC 2018)*, pages 59:1–59:11, 2018. doi:10.1109/SC.2018.00062.

27 Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. In *Proc. of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 1914–1933, 2021. doi:10.1137/1.9781611976465.114.

28 Fredrik Manne and Mahantesh Halappanavar. New effective multithreaded matching algorithms. In *Proc. of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS 2014)*, pages 519–528, 2014. doi:10.1109/IPDPS.2014.61.

29 Jens Maue and Peter Sanders. Engineering algorithms for approximate weighted matching. In *Proc. of the 6th International Conference on Experimental Algorithms (WEA 2007)*, pages 242–255, 2007. doi:10.1007/978-3-540-72845-0_19.

30 Andrew McGregor. Finding graph matchings in data streams. In *Proc. of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2005)*, pages 170–181, 2005. doi:10.1007/11538462_15.

31 William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proc. of the 2017 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM 2017)*, pages 267–280, 2017. doi:10.1145/3098822.3098838.

32 Jayadev Misra and David Gries. A constructive proof of Vizing's Theorem. *Information Processing Letters*, 41(3):131–133, 1992. doi:10.1016/0020-0190(92)90041-S.

33 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005. doi:10.1561/0400000002.

34 Ami Paz and Gregory Schwartzman. A $(2+\epsilon)$-approximation for maximum weight matching in the semi-streaming model. *ACM Transactions on Algorithms*, 15(2):18:1–18:15, 2019. doi:10.1145/3274668.

35 Alex Pothen, SM Ferdous, and Fredrik Manne. Approximation algorithms in combinatorial scientific computing. *Acta Numerica*, 28:541–633, 2019. doi:10.1017/S0962492919000035.

36 Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137, 2015.

37 Vadim G. Vizing. The chromatic class of a multigraph. *Cybernetics*, 1(3):32–41, 1965. doi:10.1007/BF01885700.

38 Mariano Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20, 2012. doi:10.1007/s00453-010-9438-5.