Faster Semi-Streaming Matchings via Alternating Trees

Slobodan Mitrović
Department of Computer Science, UC Davis, CA, USA
Department of Mathematics and Informatics, University of Novi Sad, Serbia

Anish Mukherjee
Department of Computer Science, University of Liverpool, UK

Piotr Sankowski
Institute of Informatics, University of Warsaw, Poland

Wen-Horng Sheu
Department of Computer Science, UC Davis, CA, USA

Abstract

We design a deterministic algorithm for the $(1+\epsilon)$-approximate maximum matching problem. Our primary result demonstrates that this problem can be solved in $O(\epsilon^{-6})$ semi-streaming passes, improving upon the $O(\epsilon^{-19})$ pass-complexity algorithm by [Fischer, Mitrović, and Uitto, STOC'22]. This contributes substantially toward resolving Open question 2 from [Assadi, SOSA'24]. Leveraging the framework introduced in [FMU'22], our algorithm achieves an analogous round complexity speed-up for computing a $(1+\epsilon)$-approximate maximum matching in both the Massively Parallel Computation (MPC) and CONGEST models.

The data structures maintained by our algorithm are formulated using blossom notation and represented through alternating trees. This approach enables a simplified correctness analysis by treating specific components as if operating on bipartite graphs, effectively circumventing certain technical intricacies present in prior work.

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

Keywords and phrases streaming algorithms, approximation algorithms, maximum matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.119

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2412.19057` [72]

Funding *Slobodan Mitrović*: Supported by the Google Research Scholar and NSF Faculty Early Career Development Program No. 2340048.
Piotr Sankowski: Supported by National Science Center grant no. 2020/37/B/ST6/04179.
Wen-Horng Sheu: Supported by NSF Faculty Early Career Development Program No. 2340048.

1 Introduction

Given an undirected, unweighted graph $G = (V, E)$ of n vertices and m edges, the task of maximum matching is to find the largest set of edges $M \subseteq E$ such that no two edges in M share an endpoint. With the prominence of large volumes of data and huge graphs, there has been a significant interest in finding simple and very fast algorithms, even at the expense of allowing approximation in the output. In particular, given a constant $\epsilon > 0$, the problem of finding an $(1+\epsilon)$-approximate maximum matching (that we denote by ϵMM) has been studied in the *semi-streaming model* [71, 2, 3, 73, 4, 49, 48, 11, 6, 61].

In the semi-streaming setting, the algorithm is assumed to have access to $O(n \operatorname{poly}\log n)$ space, which is generally insufficient to store the entire input graph. The graph G is provided as a stream of m edges arriving in an arbitrary order. The objective is to solve the problem with the minimum possible number of passes over the stream. The semi-streaming model has been the model of choice for processing massive graphs, as the traditional log-space streaming model proves too restrictive for many fundamental graph problems. In particular, it has been shown that testing graph connectivity requires $\Omega(n)$ space in the streaming setting, even with a constant number of passes allowed [59].

In the study of ϵMM within the semi-streaming setting, one of the primary aims is to deliver methods whose dependence on $1/\epsilon$ in the pass complexity is as small as possible while retaining the smallest known dependence on n. There has been ample success in this regard when the input graph is bipartite, where $O(1/\epsilon^2)$-pass algorithm is known [16]. We note that the pass complexity of this algorithm does not depend on n, but only a polynomial of $1/\epsilon$.

For general graphs, the situation is very different. For instance, until very recently, the best pass complexity in the semi-streaming setting either depends exponentially on $1/\epsilon$ [71, 73, 49], or polynomially on $1/\epsilon$ but with a dependence on $\log n$ [3, 2, 4, 11, 6]; we provide an overview of existing results in Tables 1 and 2. Significant progress has been made by Fischer, Mitrović, and Uitto [48], who introduced a semi-streaming algorithm for general graphs that outputs a $(1+\epsilon)$-approximate maximum matching in $\operatorname{poly}(1/\epsilon)$ passes, i.e., $O(1/\epsilon^{19})$ many passes with no dependence on n. Also, in that work, improvements of the same quality were obtained for the MPC and CONGEST models, albeit with a higher polynomial dependence on $1/\epsilon$.

Although the result [48] makes important progress in understanding semi-streaming algorithms for approximating maximum matchings in general graphs, the analysis presented in that work is quite intricate. In addition, the gap between known complexities for methods tackling bipartite and general graphs in semi-streaming remains relatively large, i.e., $1/\epsilon^2$ vs. $1/\epsilon^{19}$. This inspires the main question of our work:

Can we design simpler and more efficient semi-streaming algorithms for $(1+\epsilon)$-approximate maximum matching in general graphs?

1.1 Our contribution

The main technical claim of our work can be summarized as follows.

▶ **Theorem 1.** *There exists a deterministic semi-streaming algorithm that, given any $\epsilon > 0$ and a graph G on n vertices, outputs a $(1+\epsilon)$-approximate maximum matching in G in $O(1/\epsilon^6)$ passes. The algorithm uses $O(n/\epsilon^6)$ words of space.*

The very high-level idea behind our approach is finding "short" augmentations until only a few of them remain. It is folklore that such an algorithm yields the desired approximation. To find these "short" augmentations, each free vertex maintains some set of alternating paths originating at it; we refer to such a set of alternating paths by a *structure*. The main conceptual contribution of our work is representing these structures by *alternating trees* and *blossoms*, introduced by Edmonds in his celebrated work [39]. An alternating tree is a tree rooted at a free vertex such that every root-to-leaf path is an alternating path. A formal, recursive definition of blossoms is given in Definition 5. Informally, a blossom can be either an odd alternating cycle (that is, an odd cycle in which each vertex except one is matched to one of its neighbors in the cycle), or a set of smaller vertex-disjoint blossoms connected by an odd alternating cycle. Our algorithm represents each structure by an alternating tree in which each node can be either a vertex (of the input graph) or a blossom contracted into a single vertex. The most related work to ours, i.e., [48], develops an ad-hoc structure. Although another related work [61] builds on blossom structure, it does it for reasons other than finding short augmentations; in fact, for finding short augmentations [61] uses [48] essentially in a black-box manner. A more detailed overview of prior work is given in Tables 1 and 2.

Our approach provides several advantages: (1) we re-use some of the well-established properties of blossoms; (2) our proof of correctness and our algorithm are simpler compared to that of [48]; (3) we exhibit relations between different alternating trees during the short-augmentation search that eventually lead to our improved pass complexity, from $1/\epsilon^{19}$ to $1/\epsilon^{6}$; and (4) the size of structures in our algorithm is considerably smaller than the structure size in [48]'s algorithm, which leads to the improvement of space complexity from $O(n/\epsilon^{22})$ to $O(n/\epsilon^{6})$. We hope that this perspective and simplification in the analysis will lead to further improvements in designing approximate maximum matching algorithms.

Table 1 A summary of the pass complexities of computing $(1+\epsilon)$-approximate maximum matching in **bipartite graphs**. Each algorithm uses $O(n \cdot \text{poly}(\log n, 1/\epsilon))$ space, although some have tighter guarantees. Using the framework in [27] and its follow-up [26], all results in this table also hold for the weighted case, except that the $\log n$ factors are increased to $\log(n/\epsilon)$.

Reference	Passes	Deterministic
[41]	$O(1/\epsilon^{8})$	Yes
[40]	$O(1/\epsilon^{5})$	Yes
[3]	$O(1/\epsilon^{2} \cdot \log(1/\epsilon))$	Yes
[62]	$O(1/\epsilon^{2})$ (vertex arrival)	Yes
[4]	$O(\log(n)/\epsilon)$	No
[16]	$O(1/\epsilon^{2})$	Yes
[11]	$O(\log(n)/\epsilon \cdot \log(1/\epsilon))$	Yes
[6]	$O(\log(n)/\epsilon)$	Yes

Implication to other models. The ϵMM problem has been comprehensively studied in other models of computation as well, such as Massively Parallel Computation (MPC) [36, 7, 51, 52, 16], CONGEST and LOCAL [69, 19, 47, 1, 53, 54, 58, 44]. [48] provides a framework which, through their semi-streaming algorithm, reduces the computation of $(1+\epsilon)$-approximate maximum matching to $\text{poly}(1/\epsilon)$ invocations of a $\Theta(1)$-approximate maximum matching algorithm. A straightforward adaptation of their framework to our result yields $1/\epsilon^{13}$ round-complexity improvement for MPC and CONGEST.

1.2 Related work

Our algorithmic setup is inspired by [48] and its predecessor [40], while several structural properties are borrowed from [39]. We detail similarities and differences between our and [48]'s techniques in Section 3.4.

Approximate maximum matchings in (semi-)streaming have been extensively studied from numerous perspectives. The closest to our work is [48], who design a deterministic semi-streaming algorithm for ϵMM using $O(1/\epsilon^{19})$ passes. Prior to that work, [71, 73]

Table 2 A summary of the pass complexities of computing $(1+\epsilon)$-approximate maximum matching in **general graphs**. Each algorithm uses $O(n \cdot \text{poly}(\log n, 1/\epsilon))$ space, although some have tighter guarantees.

Reference	Passes	Deterministic	Weighted
[71]	$\exp(1/\epsilon)$	No	No
[2]	$O(\log(n)/\epsilon^7 \cdot \log(1/\epsilon))$	Yes	Yes
[3]	$O(\log(n)/\epsilon^4)$	Yes	Yes
[4]	$O(\log(n)/\epsilon)$	No	Yes
[73]	$\exp(1/\epsilon)$	Yes	No
[49]	$\exp(1/\epsilon^2)$	No	Yes
[48]	$O(1/\epsilon^{19})$	Yes	No
[61]	more than $O(1/\epsilon^{19})$	Yes	Yes
[6]	$O(\log(n)/\epsilon)$	Yes	Yes
this work	$O(1/\epsilon^6)$	Yes	No

developed semi-streaming algorithms that use $\exp(1/\epsilon)$ many passes. Tables 1 and 2 list many other results for computing ϵMM in semi-streaming; in bipartite and general graphs, respectively.

Next, we briefly describe some related works on variants of the ϵMM problem or the underlying model of computation, that have been considered in the literature. A sequence of papers has studied the question of estimating the size of the maximum matching [64, 32, 13, 43, 65]. The ϵMM problem has been considered in weighted graphs as well [2, 3, 32, 4, 49, 61, 6]. On bipartite graphs and in the semi-streaming model, any algorithm for (unweighted) ϵMM can be converted to an algorithm for weighted ϵMM with slightly increased pass and space complexities [27, 26]. Using this conversion, all pass-complexity results in Table 1 also apply to the weighted case, except that the $\log n$ factors are increased to $\log(n/\epsilon)$. Considering variants of the streaming setting, there have been works in dynamic streaming where one can both insert and remove edges [67, 32, 35, 14], in the vertex arrival model [66, 55, 62, 42, 34, 31, 50], and in random streaming where vertices or edges arrive in a random order [70, 68, 49, 45, 25, 8].

Several works have studied lower bound questions in the streaming setting, both for exact [57, 17, 33] and approximate maximum matching [55, 62, 14, 13, 15, 63, 5, 18].

The ϵMM problem is well-studied in the classical centralized setting as well [37, 38]. Furthermore, in recent years, there has been a growing interest in ϵMM in the area of dynamic algorithms [56, 28, 10, 9, 30, 74, 29, 21, 12] and also in sublinear time algorithms for approximate maximum matching [20, 22, 23].

1.3 Organization

After some preliminaries in Section 2, we give an overview of our approach in Section 3. Next, in Section 4, we present our algorithm from a high level. The complete description and analysis of our algorithm are deferred to the full version [72].

2 Preliminaries

In the following, we first introduce all the terminology, definitions, and notations. We also recall some well-known facts about blossoms.

Let G be an undirected simple graph and $\epsilon \in (0, 1]$ be the approximation parameter. Without loss of generality, we assume that ϵ^{-1} is a power of 2. Denote by $V(G)$ and $E(G)$, respectively, the vertex and edge sets of G. Let n be the number of vertices in G and m be the number of edges in G. An undirected edge between two vertices u and v is denoted by $\{u, v\}$. Throughout the paper, if not stated otherwise, all the notations implicitly refer to a currently given matching M, which we aim to improve.

2.1 Alternating paths

▶ **Definition 2** (An unmatched edge and a free vertex)**.** *We say that an edge $\{u, v\}$ is* matched *iff $\{u, v\} \in M$, and* unmatched *otherwise. We call a vertex v* free *if it has no incident matched edge, i.e., if $\{u, v\}$ are unmatched for all edges $\{u, v\}$. Unless stated otherwise, α, β, γ are used to denote free vertices.*

▶ **Definition 3** (Alternating and augmenting paths)**.** *An* alternating path *is a simple path that consists of a sequence of alternately matched and unmatched edges. The* length *of an alternating path is the number of edges in the path. An* augmenting path *is an alternating path whose two endpoints are both free vertices.*

2.2 Alternating trees and blossoms

▶ **Definition 4** (Alternating trees, inner vertices, and outer vertices)**.** *A subgraph of G is an* alternating tree *if it is a rooted tree in which the root is a free vertex and every root-to-leaf path is an even-length alternating path. An* inner vertex *of an alternating tree is a non-root vertex v such that the path from the root to v is of odd length. All other vertices are* outer vertices*. In particular, the root vertex is an outer vertex.*

Note that in an alternating tree, every leaf is an outer vertex; every inner vertex v has exactly one child, which is matched to v. Hence, every non-root vertex in the tree is matched.

▶ **Definition 5** (Blossoms and trivial blossoms)**.** *A blossom is identified with a vertex set B and an edge set E_B on B. If $v \in V(G)$, then $B = \{v\}$ is a* trivial blossom *with $E_B = \emptyset$. Suppose there is an odd-length sequence of vertex-disjoint blossoms $A_0, A_1, \ldots, A_k$ with associated edge sets $E_{A_0}, E_{A_1}, \ldots, E_{A_k}$. If $\{A_i\}$ are connected in a cycle by edges $e_0, e_1, \ldots, e_k$, where $e_i \in A_i \times A_{i+1}(\text{modulo } k+1)$ and $e_1, e_3, \ldots, e_{k-1}$ are matched, then $B = \bigcup_i A_i$ is also a blossom associated with edge set $E_B = \bigcup_i E_{A_i} \cup \{e_0, e_1, \ldots, e_k\}$.*

▶ Remark 6. In the literature, a blossom is often defined as an odd-length cycle in G consisting of $2k + 1$ edges, where exactly k of these edges belong to the matching M. Here, we use the definition in [38], in which a blossom is defined recursively as an odd-length cycle alternating between matched and unmatched edges, whose components are either single vertices or blossoms in their own right. This recursive definition characterizes the subgraphs contracted in Edmond's algorithm.

Consider a blossom B. A short proof by induction shows that $|B|$ is odd. In addition, $M \cap E_B$ matches all vertices except one. This vertex, which is left unmatched in $M \cap E_B$, is called the *base* of B. Note that $E(B) = E(G) \cap (B \times B)$ may contain many edges outside of E_B. Blossoms exhibit the following property.

▶ **Lemma 7** ([38]). *Let B be a blossom. There is an even-length alternating path in E_B from the base of B to any other vertex in B.*

▶ **Definition 8** (Blossom contraction). *Let B be a blossom. We define the contracted graph G/B as the undirected simple graph obtained from G by contracting all vertices in B into a vertex, denoted by B.*

The following lemma is proven in [39, Theorem 4.13].

▶ **Lemma 9** ([39]). *Let T be an alternating tree of a graph G and $e \in E(G)$ be an edge connecting two outer vertices of T. Then, $T \cup \{e\}$ contains a unique blossom B. The graph T/B is an alternating tree of G/B. It contains B as an outer vertex. Its other inner and outer vertices are those of T which are not in B.*

Consider a set Ω of blossoms. We say Ω is *laminar* if the blossoms in Ω form a laminar set family. Assume that Ω is laminar. A blossom in Ω is called a *root blossom* if it is not contained in any other blossom in Ω. Denote by G/Ω the undirected simple graph obtained from G by contracting each root blossom of Ω. For each vertex in $\bigcup_{B\in\Omega} B$, we denote by $\Omega(v)$ the unique root blossom containing v. If Ω contains all vertices of G, we denote by M/Ω the set of edges $\{\{\Omega(u), \Omega(v)\} \mid \{u, v\} \in M \text{ and } \Omega(u) \neq \Omega(v)\}$ on the graph G/Ω. It is known that M/Ω is a matching of G/Ω [38].

In our algorithm, we maintain several vertex-disjoint subgraphs. Each subgraph is associated with a *regular* set of blossoms, which is a set of blossoms whose contraction would transform the subgraph into an alternating tree satisfying certain properties. A regular set of blossoms is formally defined as follows.

▶ **Definition 10** (Regular set of blossoms). *A regular set of blossoms of G is a set Ω of blossoms satisfying the following:*

1. *Ω is a laminar set of blossoms of G. It contains the set of all trivial blossoms in G. If a blossom $B \in \Omega$ is defined to be the cycle formed by $A_0, \ldots, A_k$, then $A_0, \ldots, A_k \in \Omega$.*
2. *G/Ω is an alternating tree with respect to the matching M/Ω. Its root is $\Omega(\alpha)$ and each of its inner vertex is a trivial blossom (whereas each outer vertex may be a non-trivial blossom).*

2.3 Representation of undirected graphs

In our algorithm, each undirected edge $\{u, v\}$ is represented by two directed *arcs* (u, v) and (v, u). Let (u, v) be an arc. We say (u, v) is *matched* if $\{u, v\}$ is a matched edge; otherwise, (u, v) is *unmatched*. The vertex u and v are called, respectively, *tail* and *head* of (u, v). We denote by $\overleftarrow{(u, v)} = (v, u)$ the reverse of (u, v).

Let $P = (u_1, v_1, \ldots, u_k, v_k)$ be an alternating path, where u_i and v_i are vertices, (u_i, v_i) are matched arcs, and (v_i, u_{i+1}) are unmatched ones. Let $a_i = (u_i, v_i)$. We often use $(a_1, a_2, \ldots, a_k)$ to refer to P, i.e., we omit specifying unmatched arcs. Nevertheless, it is guaranteed that the input graph contains the unmatched arcs (v_i, u_{i+1}), for each $1 \le i < k$. If P is an alternating path that starts and/or ends with unmatched arcs, e.g., $P = (x, u_1, v_1, \ldots, u_k, v_k, y)$ where (x, u_1) and (v_k, y) are unmatched while $a_i = (u_i, v_i)$ for $i = 1 \ldots k$ are matched arcs, we use $(x, a_1, ..., a_k, y)$ to refer to P. In our case, very frequently, x and y will be free vertices, usually $x = \alpha$ and $y = \beta$.

▶ **Definition 11** (Concatenation of alternating paths). *Let $P_1 = (a_1, a_2, \ldots, a_k)$ and $P_2 = (b_1, b_2, \ldots, b_s)$ be alternating paths. We use $P_1 \circ P_2 = (a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_s)$ to denote their concatenation. Note that, the alternating path $P_1 \circ P_2$ also contains the unmatched edge between a_k and b_1.*

2.4 Semi-streaming model

In the semi-streaming model [46], we assume that the algorithm has no random access to the input graph. The set of edges is represented as a stream. In this stream, each edge is presented exactly once, and each time the stream is read, edges may appear in an arbitrary order. The stream can only be read as a whole and reading the whole stream once is called a *pass* (over the input). The main computational restriction of the model is that the algorithm can only use $O(n \operatorname{poly} \log n)$ words of space, which is not enough to store the entire graph if the graph is sufficiently dense.

3 Overview of Our Approach

The starting point of our approach is the classical idea of finding augmenting paths to improve the current matching [24, 39, 60]. It is well-known that it suffices to search for $O(1/\epsilon)$ long augmenting paths, i.e., it suffices to search for relatively short paths, to obtain a $(1+\epsilon)$-approximate maximum matching (ϵMM). *However, how can this short-augmentations search be performed in a small number of passes?*

To make this search efficient in the semi-streaming setting, the general idea is to search for *many* augmenting paths during the same pass. This is achieved by a depth-first-search (DFS) exploration truncated at depth $O(1/\epsilon)$ from each free vertex; that kind of approach was employed in prior work, e.g., [71, 40, 73, 48].

▶ Remark 12. Throughout the paper, we attempt to use terminology as closely as possible to work prior, particularly the terminology used in [48]. We hope that in this way, we aid readers in comparing our contributions to priors.

3.1 Augmentation search via alternating trees

In our algorithm, each free vertex maintains an alternating tree. These trees are created via a DFS exploration in an alternating-path manner. Consider an alternating tree S. S is associated with a so-called *working vertex*, which represents the last vertex the DFS exploration has currently reached. Figure 1 depicts an alternating tree.

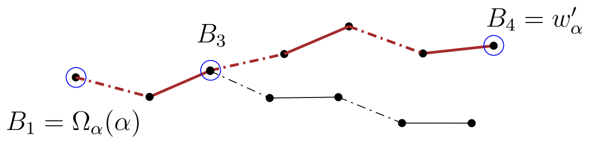


Figure 1 An example of alternating tree. Dashed and solid edges denote the unmatched and matched edges, respectively. The encircled vertices correspond to the non-trivial blossoms, i.e., B_1, B_3, and B_4 are non-trivial blossoms. $B_1 = \Omega_\alpha(\alpha)$ is the blossom containing a free vertex α. w'_α is the working vertex and the highlighted path, from B_1 to B_4, is the active path.

Recall that the goal is to look for *short* augmentations. Hence, these DFS explorations are carried out by attempting to visit an edge by as short an alternating path as possible. In particular, each matched edge e maintains a *label* $\ell(e)$ representing the so-far shortest discovered alternating path to a free vertex.[1] Observe that only matched edges maintain labels. That enables storing those labels in $O(n)$ words.

[1] Our algorithm maintains arc labels; an edge $\{u, v\}$ is represented by arcs (u, v) and (v, u), and the algorithm maintains $\ell((u, v))$ and $\ell((v, u))$. For the sake of simplicity, in this overview, we only talk about edge labels.

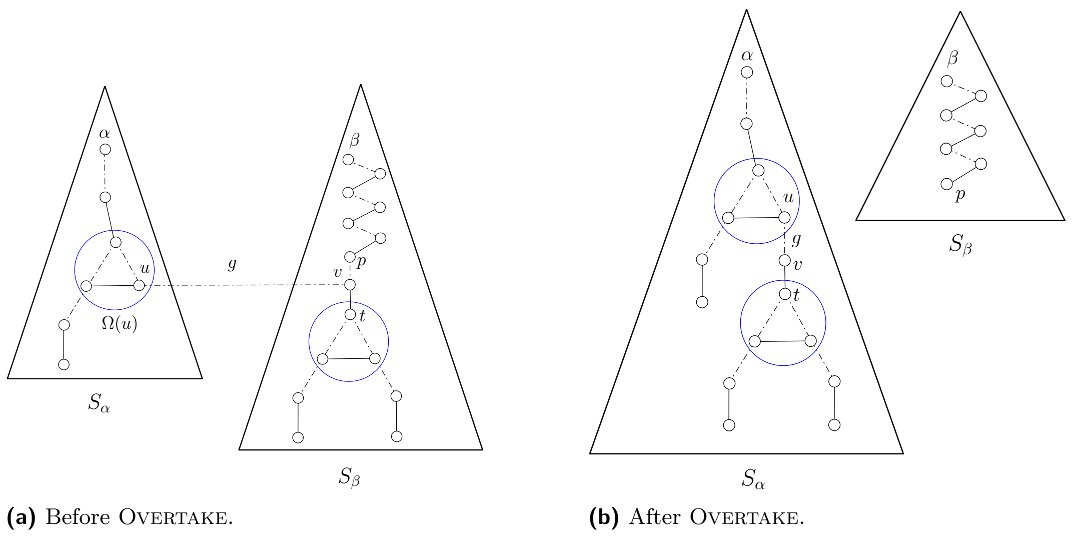


Figure 2 An example of Overtake. In this example, $g = (u, v)$ connects two alternating trees S_α and S_β, with $\Omega(u)$ being the working vertex of S_α before Overtake. The circles represent blossoms, which are not contracted in this sketch so as to illustrate possible situations better. The alternating path from α to matched edge $(\Omega(v), \Omega(t))$ along g improves the label of $(\Omega(v), \Omega(t))$, and hence Overtake is invoked. (Observe that $\{v\} = \Omega(v)$.)

In a single pass, the working vertex u of S attempts to extend S by a length-2 path $\{u, v, t\}$, where $g = \{u, v\}$ is unmatched and $e = \{v, t\}$ is a matched edge; if it is impossible, just like in a typical DFS, this working vertex backtracks. Then, one of the following happens (see the full version for details):

1. The edge g connects S with an alternating tree S', different than S, such that there is an augmenting path between the roots of S and S' involving g. In that case, this augmenting path is recorded, and S and S' are temporarily removed from the graph. This is done by procedure Augment. (After the DFS exploration is completed, the algorithm restores all temporarily removed vertices and augments the matching using the recorded augmenting paths.)
2. If g connects two vertices in S such that it creates a blossom, then this blossom is contracted. This is done by procedure Contract. These contractions ensure that the exploration subgraph from each free vertex looks like a tree.
3. The alternating path along S to e is shorter than $\ell(e)$. Then, g and e are added to S, and $\ell(e)$ is updated accordingly. If e belongs to another alternating *subtree*, which might belong to S or another alternating tree, then the entire subtree together with e is appended to u. This is done by procedure Overtake. Note that, by the construction, the last edge appended to DFS exploration is matched. An example of Overtake is depicted in Figure 2.

   ▶ Remark 13. The intuition behind the overtake operation is as follows. In our algorithm, an edge label represents the shortest alternating distance from a free vertex to the edge. Thus, when S finds a shorter path to e, Overtake allows S to take over the search on e and reduce its label. This operation helps the algorithm find shorter alternating paths to each matched edge.

The described operations are very natural when we take the perspective of maintaining edge labels and alternating trees. *However, why do they yield an approximate maximum matching? Moreover, how many passes does the entire process take?*

3.2 Correctness argument

Recall that our algorithm executes many DFS explorations in parallel, each originating at a free vertex. When a DFS exploration from a free vertex has no new edges to visit, we say that the free vertex becomes inactive; otherwise, it is active. Our algorithm terminates when the number of active free vertices becomes a small fraction of the current matching size. The main goal of our correctness proof is to show that terminating the search for augmentations is justified. Speaking informally, the intuition behind our correctness argument is that if our algorithm runs indefinitely, each short augmentation will eventually intersect an augmentation our algorithm has already found.

Our proof of correctness boils down to showing the following:

Consider a short augmenting path P in the input graph G. Then, at any point in time, it holds:
- our algorithm has already found an augmentation *intersecting* P, or
- there is an ongoing DFS exploration *intersecting* P.

Recall that our algorithm maintains augmenting trees from free vertices. On a very high level, this enables us to think about augmentation search as, informally speaking, it is done on bipartite graphs. In particular, we show the following invariant:

At the beginning of every pass, if a non-tree edge induces an odd cycle, that edge cannot be reached by any so far discovered alternating path starting at a free vertex.

One can also view this invariant as a way of saying that no relevant odd cycle is visible to the algorithm. Of course, our algorithmic primitives and analysis must ensure that this view is indeed tree-like. Once we established this, we could bypass the technical intricacies of prior work.

3.3 Pass and space complexity, and approximation guarantee

Our algorithm progressively finds a better approximation of the current approximate matching. We implement that by dividing our augmentation search into different *scales*. A fixed scale guides the granularity of the search of the rest of the algorithm, and the scale values range from $1/2$ to $O(1/\epsilon^2)$ in powers of 2. Each scale is further divided into many *phases*.

Our pass complexity balances and ties several parameters guiding the algorithm. These parameters are the number of edge-label reductions, the sizes of alternating trees, the scale values, and the number of phases in a scale. The most important of these parameters are scale and the upper bound on an alternating tree size.

When phases are executed for a given scale h, the attempt is to arrive at a $(1 + O(h/\epsilon))$-approximate maximum matching. Hence, in the beginning, when there are many augmentations, large values of h imply that the algorithm will soon arrive at the desired approximation. Importantly, this also means that fewer augmenting paths must be found for the next scale, i.e., scale $h/2$, because scale $h/2$ starts with a better approximation than scale h. Therefore, scales enable us to balance the quality of approximation we want to achieve with the number of augmentations that must be found: the tighter the approximation requirement is, the slower the algorithm is; the fewer the augmentations must be found, the faster the algorithm is.

3.4 Comparison with [48]

A fundamental difference between our approach and that of [48] is that the search structure from a free vertex can be seen as a tree that we also refer to by structure. The same as in [48], in our work, DFS structures S_α and S_β originating at different free vertices α and β might affect each other – either by moving a part of S_α to S_β via Overtake, or by finding an augmentation between α and β. In [48], these structures and blossoms are represented as a union of alternating paths, with some edges being marked as belonging to odd cycles. There is no special treatment of blossoms, nor do those structures have any particular shape. On the other hand, we represent these structures as alternating trees; some vertices in those trees might correspond to blossoms. Crucially, it enables us to simplify structure-related procedures, provide a simpler proof of correctness, and prove new properties about structure sizes, allowing us to significantly reduce the pass complexity.

Finally, we believe the complexity of [48] can be reduced to $O(1/\epsilon^{16})$ by a slightly more careful analysis and tweaking parameters. Adding the scales would improve the exponent in the pass-complexity by an additional 2. In addition, replacing "a maximal set of augmenting paths" with "the maximum set of augmenting paths" in the complexity analysis in [48] would result in yet another improvement by 2 in the exponent of pass-complexity (e.g., using Lemma 6.2 in the full version). Nevertheless, it is unclear that, unless fundamental changes are made in the approach, [48] can result in a pass-complexity better than $O(1/\epsilon^{12})$.

▶ Remark 14. The ideas of the truncated DFS exploration, maintaining edge labels, and the idea of the overtaking operation were first proposed in [41] and later used in [48]. The actual formulation of overtaking in our work is inspired by but different from [41] or [48].

4 Algorithms Overview

In this section, we present only the basic components of our main algorithm. Details and analysis are deferred to the full version [72]. We start by presenting two data structures that our algorithm maintains: the edge-exploration each free vertex maintains, which we call *structure* (Section 4.1.1), and a label that each matched arc maintains (Section 4.1.2). In Section 4.2 we provide the base of our approach, which consists of many phases. The algorithms handling those phases are described in the subsequent sections, with Section 4.3 providing an overview of a single phase.

▶ Remark 15. As already noted, throughout the paper, we attempt to use the algorithmic design as closely as possible to work prior, particularly the one used in [48]. We hope that in this way, we aid readers in comparing our contributions to priors.

4.1 Algorithms' preliminaries

4.1.1 Free-vertex structures

In our algorithm, each free vertex α maintains a *structure* (see Figure 3 for an example), defined as follows.

▶ **Definition 16** (The structure of a free vertex). *The structure of a free vertex α, denoted by S_α, is a tuple $(G_\alpha, \Omega_\alpha, w'_\alpha)$, where*

- *G_α is a subgraph of G,*
- *Ω_α is a regular set of blossoms of G_α, and*
- *w'_α is either $\emptyset$ or an outer vertex of the alternating tree G_α/Ω_α.*

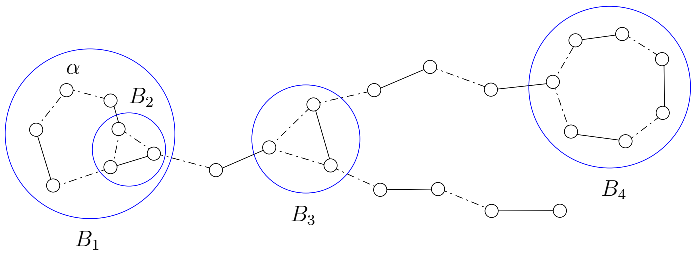
Each structure S_α satisfies the following properties.

1. ***Disjointness:*** *For any free vertex $\beta \neq \alpha$, G_α is vertex-disjoint from G_β.*
2. ***Tree representation:*** *The subgraph G_α contains a set of arcs satisfying the following: If G_α contains an arc (u, v) with $\Omega_\alpha(u) \neq \Omega_\alpha(v)$, then $\Omega_\alpha(u)$ is the parent of $\Omega_\alpha(v)$ in the alternating tree G_α/Ω_α.*
3. ***Unique arc property:*** *For each arc $(u', v') \in E(G_\alpha/\Omega_\alpha)$, there is a unique arc $(u, v) \in G_\alpha$ such that $\Omega_\alpha(u) = u'$ and $\Omega_\alpha(v) = v'$.*

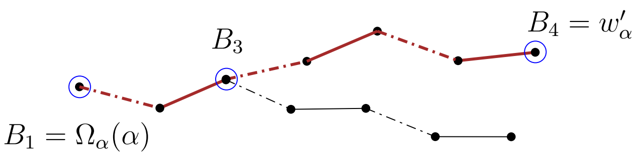
We denote the alternating tree G_α/Ω_α by T'_α. Since Ω_α is a regular set of blossom, each inner vertex of T'_α is a trivial blossom, whereas each outer vertex may be a non-trivial blossom. Figure 3b shows T'_α corresponding to the structure in Figure 3a. We remark that G_α may not be a vertex-induced subgraph. That is, G may contain arcs that are not in G_α but connect two vertices in G_α.

▶ **Definition 17** (The working vertex and active path of a structure). *Consider a structure S_α. The* working vertex *of S_α is defined as the vertex w'_α, which can be $\emptyset$. If $w'_\alpha \neq \emptyset$, we define the* active path *of S_α as the unique path on T'_α from the root $\Omega_\alpha(\alpha)$ to w'_α. Otherwise, the active path is defined as $\emptyset$.*

▶ **Definition 18** (Active vertices, arcs, and structures). *A vertex or arc of T'_α is said to be* active *if and only if it is on the active path. We say S_α is active if $w'_\alpha \neq \emptyset$.*



(a) The graph G_α.



(b) The contracted graph G_α/Ω_α.

Figure 3 Example of a structure S_α. Dashed and solid edges denote the unmatched and matched edges, respectively. α is a free vertex. Ω_α contains all trivial blossoms in G_α and the non-trivial blossoms $\{B_1, B_2, B_3, B_4\}$, where $B_1 = \Omega_\alpha(\alpha)$ and B_4 is the working vertex w'_α in G_α/Ω_α. (a) The graph G_α. (b) The corresponding contracted graph G_α/Ω_α. The encircled vertices correspond to the non-trivial blossoms. w'_α is the working vertex and the highlighted path, from B_1 to B_4, is the active path.

Let F be the set of free vertices. Throughout the execution, we maintain a set Ω of blossoms, which consists of all blossoms in $\bigcup_{\alpha \in F} \Omega_\alpha$ and all trivial blossoms. Note that Ω is a laminar set of blossoms. We denote by G' the contracted graph G/Ω. The vertices of G' are classified into three sets: (1) the set of inner vertices, which contains all inner vertices in $\bigcup_{\alpha \in F} V(T'_\alpha)$; (2) the set of outer vertices, which contains all outer vertices in $\bigcup_{\alpha \in F} V(T'_\alpha)$; (3) the set of *unvisited vertices*, which are the vertices not in any structure.

Similarly, we say a vertex in G is *unvisited* if it is not in any structure. An arc $(u, v) \in G$ is a *blossom arc* if $\Omega(u) = \Omega(v)$; otherwise, (u, v) is a *non-blossom arc*. An *unvisited arc* is an arc $(u, v) \in E(G)$ such that u and v are unvisited vertices.

4.1.2 Labels of matched arcs

Our algorithm stores the set of all matched arcs throughout its execution. Each matched arc is associated with a *label*, defined as follows.

▶ **Definition 19** (The label of a matched arc). *Each matched arc $a^* \in G$ is assigned a label $\ell(a^*)$ such that $1 \leq \ell(a^*) \leq \ell_{\max} + 1$, where $\ell_{\max}$ is defined as $3/\epsilon$.*

Each matched arc $a' \in G'$ corresponds to a unique non-blossom matched arc $a \in G$; for ease of presentation, we denote by $\ell(a')$ the label of a.

Our algorithm maintains the following invariant.

▶ **Invariant 20** (Increasing labeling). *For any alternating path $(\Omega(\alpha), a'_1, a'_2, \ldots, a'_k)$ on T'_α starting from the root, it holds that $\ell(a'_1) < \ell(a'_2) < \cdots < \ell(a'_k)$.*

4.2 Algorithm overview

In the following, we will sketch our algorithm, incrementally providing more details. Algorithm 1 gives a high-level description of the algorithm. Recall that $\frac{1}{\epsilon}$ is assumed to be a power of 2.

Algorithm 1 A high-level algorithm description.

Input: a graph G and the approximation parameter ϵ
Output: a $(1 + \epsilon)$-approximate maximum matching

```
1: compute, in a single pass, a 2-approximate maximum matching $M$
2: for scale $h = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots, \frac{\epsilon^2}{64}$ do
3:     for phases $t = 1, 2, \ldots, \frac{144}{h\epsilon}$ do
4:         $\mathcal{P} \leftarrow$ ALG-PHASE$(G, M, \epsilon, h)$        ▷ Nothing stored from the previous phase.
5:         restore all vertices removed in the execution of ALG-PHASE
6:         augment $M$ using the vertex-disjoint augmenting paths in $\mathcal{P}$
7: return $M$
```

Algorithm 1 provides an outline of our approach. Section 4.2 applies a simple greedy algorithm to find a maximal matching, which is a 2-approximation for the problem. Starting from this maximal matching, our algorithm repeatedly finds augmenting paths to improve the current matching. This is done by executing several *phases* with respect to different *scales*, detailed as follows.

Each iteration of the for-loop in Section 4.2 corresponds to a scale h. In one scale h, each iteration of the for-loop in Section 4.2 is called a phase with respect to the scale h. In each phase, the procedure ALG-PHASE is invoked to find a set $\mathcal{P}$ of vertex-disjoint augmenting paths. In the execution of ALG-PHASE, we may *conceptually remove* some vertices from G. After the execution of ALG-PHASE, Section 4.2 restores all removed vertices to G. After this step, G is identical to the input graph. Then, Section 4.2 augments the current matching using the set $\mathcal{P}$ of vertex-disjoint augmenting paths, which increase the size of M by $|\mathcal{P}|$.

The scale h is a parameter that determines the number of phases executed and the number of passes spent on each phase. By passing a smaller scale to Alg-Phase, Alg-Phase would spend more passes attempting to find more augmenting paths in the graph. Our algorithm decreases the scale gradually so that more and more augmenting paths in the graph can be discovered.

Algorithm 2 Alg-Phase: the execution of a single phase.

Input: a graph G, the current matching M, the parameter ϵ, and the current scale h
Output: a set $\mathcal{P}$ of *disjoint* M-augmenting paths

```
1: $\mathcal{P} \leftarrow \emptyset$
2: $\ell(a) \leftarrow \ell_{\max} + 1$ for each arc $a \in M$
3: for each free vertex $\alpha$, initialize its structure $S_\alpha$
4: compute parameters $\mathsf{limit}_h = \frac{6}{h} + 1$ and $\tau_{\max}(h) = \frac{72}{h\epsilon}$
5: **for** pass-bundles $\tau = 1, 2, \ldots, \tau_{\max}(h)$ **do**
6:     **for** each free vertex $\alpha$ **do**
7:         if $S_\alpha$ has at least $\mathsf{limit}_h$ vertices, mark $S_\alpha$ as "on hold"
8:         if $S_\alpha$ has less than $\mathsf{limit}_h$ vertices, mark $S_\alpha$ as "not on hold"
9:         mark $S_\alpha$ as "not modified"
10:    Extend-Active-Path
11:    Contract-and-Augment
12:    Backtrack-Stuck-Structures
13: **return**
```

4.3 A phase overview (Alg-Phase)

We now proceed to outline what the algorithm does in a single phase, whose pseudocode is given as Algorithm 2. In each phase, our algorithm executes DFS explorations from all free vertices in parallel. Details of the parallel DFS are described as follows. Section 4.2 initialize the set of paths $\mathcal{P}$, the label of each arc, and the structure of each free vertex. The structure of a free vertex α is initialized to be an alternating tree of a single vertex α. That is, G_α and Ω_α are set to be a graph with a single vertex α and a set containing a single trivial blossom $\{\alpha\}$, respectively; the working vertex w'_α is initialized as the root of T'_α, that is, $\Omega_\alpha(\alpha)$. Section 4.2 computes two parameters limit_h and $\tau_{\max}(h)$. The purpose of these two parameters is detailed later. The for-loop in Section 4.2 executes $\tau_{\max}(h)$ iterations, where each iteration is referred to as a *pass-bundle*. The execution of a pass-bundle corresponds to one step in the parallel DFS. Each pass-bundle consists of four parts:

(1) Section 4.2 initialize the status of each structure in this pass-bundle. A structure is marked as *on hold* if and only if it contains at least limit_h vertices. Each structure S_α is marked as *not modified*. The purpose of this part is described in Section 4.4.

(2) The procedure Extend-Active-Path makes a pass over the stream and attempts to extend each structure that is not on hold. Details of this procedure are given in the full version [72].

(3) After the execution of Extend-Active-Path, the subgraph G_α maintained in each structure S_α may change. The procedure Contract-and-Augment is then invoked to identify blossoms and augmenting paths. The procedure makes a pass over the stream, contracts some blossoms that contain the working vertex of a structure, and identifies pairs of structures that can be connected to form augmenting paths. Details of this procedure are given in the full version.

(4) The procedure Backtrack-Stuck-Structures examines each structure. If a structure is not on hold and fails to extend in this pass, Backtrack-Stuck-Structures backtracks the structure by removing one matched arc from its active path. Details of this procedure are given in the full version.

In the full version, we prove the following lemma, showing that all invariants presented in Sections 4.1.1 and 4.1.2 are preserved in the execution of Alg-Phase.

▶ **Lemma 21.** *The following holds throughout the execution of* Alg-Phase*. For each free vertex α that is not removed, S_α is a structure per Definition 16; in addition, Invariant 20 holds.*

4.4 Marking a structure on hold or modified

In the for-loop of Section 4.2, we mark a structure S_α *on hold* if and only if it contains at least limit_h vertices. See Section 4.2 of Algorithm 2. This operation plays a crucial role in our analysis of the pass-complexity of our algorithm. See the full version for details.

In the for-loop, we also mark each structure as *not modified.* Recall that each structure S_α is represented by a tuple $(G_\alpha, \Omega_\alpha, w'_\alpha)$. In the execution of the pass-bundle, we may modify some structures by, for example, adding new arcs to G_α. Whenever G_α, Ω_α, or w'_α is changed, we mark S_α as modified. In other words, if a structure S_α is marked as not modified, $(G_\alpha, \Omega_\alpha, w'_\alpha)$ is unchanged since the beginning of the current pass-bundle.

References

1 Mohamad Ahmadi, Fabian Kuhn, and Rotem Oshman. Distributed approximate maximum matching in the congest model. In *32nd International Symposium on Distributed Computing (DISC 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018.

2 Kook Jin Ahn and Sudipto Guha. Laminar families and metric embeddings: Non-bipartite maximum matching problem in the semi-streaming model. *arXiv preprint arXiv:1104.4058*, 2011. arXiv:1104.4058.

3 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation*, 222:59–79, 2013. doi:10.1016/J.IC.2012.10.006.

4 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *ACM Transactions on Parallel Computing (TOPC)*, 4(4):1–40, 2018. doi:10.1145/3154855.

5 Sepehr Assadi. A two-pass (conditional) lower bound for semi-streaming maximum matching. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 708–742. SIAM, 2022. doi:10.1137/1.9781611977073.32.

6 Sepehr Assadi. A simple $(1-\epsilon)$-approximation semi-streaming algorithm for maximum (weighted) matching. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 337–354. SIAM, 2024. doi:10.1137/1.9781611977936.31.

7 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. Coresets meet edcs: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1616–1635. SIAM, 2019. doi:10.1137/1.9781611975482.98.

8 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. In *48th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 198 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.19.

9 Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. On regularity lemma and barriers in streaming and dynamic matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC*, pages 131–144. ACM, 2023. doi:10.1145/3564246.3585110.

10 Sepehr Assadi, Aaron Bernstein, and Aditi Dudeja. Decremental matching in general graphs. In *49th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 229 of *LIPIcs*, pages 11:1–11:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.11.

11 Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 627–669. SIAM, 2022. doi:10.1137/1.9781611977073.29.

12 Sepehr Assadi, Sanjeev Khanna, and Peter Kiss. Improved bounds for fully dynamic matching via ordered Ruzsa-Szemerédi graphs. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2971–2990. SIAM, 2025. doi:10.1137/1.9781611978322.96.

13 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1723–1742. SIAM, 2017. doi:10.1137/1.9781611974782.113.

14 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1345–1364. SIAM, 2016. doi:10.1137/1.9781611974331.CH93.

15 Sepehr Assadi, Gillat Kol, Raghuvansh R. Saxena, and Huacheng Yu. Multi-pass graph streaming lower bounds for cycle counting, max-cut, matching size, and other problems. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 354–364. IEEE, 2020. doi:10.1109/FOCS46700.2020.00041.

16 Sepehr Assadi, S Cliff Liu, and Robert E Tarjan. An auction algorithm for bipartite matching in streaming and massively parallel computation models. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 165–171. SIAM, 2021. doi:10.1137/1.9781611976496.18.

17 Sepehr Assadi and Ran Raz. Near-quadratic lower bounds for two-pass graph streaming algorithms. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 342–353. IEEE, 2020. doi:10.1109/FOCS46700.2020.00040.

18 Sepehr Assadi and Janani Sundaresan. Hidden permutations to the rescue: Multi-pass streaming lower bounds for approximate matchings. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 909–932. IEEE, 2023. doi:10.1109/FOCS57990.2023.00058.

19 Reuven Bar-Yehuda, Keren Censor-Hillel, Mohsen Ghaffari, and Gregory Schwartzman. Distributed approximation of maximum independent set and maximum matching. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 165–174, 2017. doi:10.1145/3087801.3087806.

20 Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 873–884. IEEE, 2021. doi:10.1109/FOCS52979.2021.00089.

21 Soheil Behnezhad and Alma Ghafari. Fully Dynamic Matching and Ordered Ruzsa-Szemerédi Graphs . In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 314–327, Los Alamitos, CA, USA, October 2024. IEEE Computer Society. doi:10.1109/FOCS61266.2024.00027.

22 Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC*, pages 267–280. ACM, 2023. doi:10.1145/3564246.3585231.

23 Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. Beating greedy matching in sublinear time. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 3900–3945. SIAM, 2023. doi:10.1137/1.9781611977554.CH151.

24 Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957.

25 Aaron Bernstein. Improved bounds for matching in random-order streams. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 12:1–12:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.12.

26 Aaron Bernstein, Jiale Chen, Aditi Dudeja, Zachary Langley, Aaron Sidford, and Ta-Wei Tu. Matching composition and efficient weight reduction in dynamic matching. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2991–3028, 2025. doi:10.1137/1.9781611978322.97.

27 Aaron Bernstein, Aditi Dudeja, and Zachary Langley. A framework for dynamic matching in weighted graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 668–681, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451113.

28 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 1123–1134. IEEE, 2020. doi:10.1109/FOCS46700.2020.00108.

29 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\epsilon)$-approximate matching size in truly sublinear update time. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 1563–1588. IEEE, 2023. doi:10.1109/FOCS57990.2023.00095.

30 Joakim Blikstad and Peter Kiss. Incremental $(1-\epsilon)$-approximate dynamic matching in $o(poly(1/\epsilon))$ update time. In *31st Annual European Symposium on Algorithms, ESA*, volume 274 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.22.

31 Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. *Algorithmica*, 81(5):1781–1799, 2019. doi:10.1007/S00453-018-0505-7.

32 Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 263–274. Springer, 2015. doi:10.1007/978-3-662-48350-3_23.

33 Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh R. Saxena, Zhao Song, and Huacheng Yu. Almost optimal super-constant-pass streaming lower bounds for reachability. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 570–583. ACM, 2021. doi:10.1145/3406325.3451038.

34 Ashish Chiplunkar, Sumedh Tirodkar, and Sundar Vishwanathan. On randomized algorithms for matching in the online preemptive model. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2015. doi:10.1007/978-3-662-48350-3_28.

35 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1326–1344. SIAM, 2016. doi:10.1137/1.9781611974331.CH92.

36 Artur Czumaj, Jakub Łącki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 471–484, 2018.

37 Doratha E. Drake and Stefan Hougardy. Improved linear time approximation algorithms for weighted matchings. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, 6th International Workshop APPROX and 7th International Workshop RANDOM Proceedings*, volume 2764 of *Lecture Notes in Computer Science*, pages 14–23. Springer, 2003. doi:10.1007/978-3-540-45198-3_2.

38 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1), January 2014. doi:10.1145/2529989.

39 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

40 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1):490–508, 2012. doi:10.1007/S00453-011-9556-8.

41 Sebastian Eggert, Lasse Kliemann, and Anand Srivastav. Bipartite graph matchings in the semi-streaming model. In *European Symposium on Algorithms*, pages 492–503. Springer, 2009. doi:10.1007/978-3-642-04128-0_44.

42 Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved bounds for online preemptive matching. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS*, volume 20 of *LIPIcs*, pages 389–399. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICS.STACS.2013.389.

43 Hossein Esfandiari, MohammadTaghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. *ACM Trans. Algorithms*, 14(4):48:1–48:23, 2018. doi:10.1145/3230819.

44 Salwa Faour, Marc Fuchs, and Fabian Kuhn. Distributed congest approximation of weighted vertex covers and matchings. *arXiv preprint arXiv:2111.10577*, 2021. arXiv:2111.10577.

45 Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1773–1785. SIAM, 2020. doi:10.1137/1.9781611975994.108.

46 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/J.TCS.2005.09.013.

47 Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 180–191. IEEE, 2017. doi:10.1109/FOCS.2017.25.

48 Manuela Fischer, Slobodan Mitrović, and Jara Uitto. Deterministic $(1+\varepsilon)$-approximate maximum matching with $\text{poly}(1/\varepsilon)$ passes in the semi-streaming model and beyond. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 248–260. Association for Computing Machinery, 2022. doi:10.1145/3519935.3520039.

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

50 Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 26–37. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00011.

51 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 129–138, 2018. doi:10.1145/3212734.3212743.

52 Mohsen Ghaffari, Christoph Grunau, and Slobodan Mitrović. Massively parallel algorithms for b-matching. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 35–44, 2022. doi:10.1145/3490148.3538589.

53 Mohsen Ghaffari, David G Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 662–673. IEEE, 2018. doi:10.1109/FOCS.2018.00069.

54 Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. Deterministic distributed edge-coloring with fewer colors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 418–430, 2018. doi:10.1145/3188745.3188906.

55 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 468–485. SIAM, 2012. doi:10.1137/1.9781611973099.41.

56 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1 + \epsilon)$-approximate incremental matching in constant deterministic amortized time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1886–1898. SIAM, 2019. doi:10.1137/1.9781611975482.114.

57 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016. doi:10.1007/S00453-016-0138-7.

58 David G Harris. Distributed local approximation algorithms for maximum matching in graphs and hypergraphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 700–724. IEEE, 2019. doi:10.1109/FOCS.2019.00048.

59 Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. *Computing on data streams*, pages 107–118. American Mathematical Society, USA, 1999.

60 John E Hopcroft and Richard M Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 122–125. IEEE, 1971. doi:10.1109/SWAT.1971.1.

61 Shang-En Huang and Hsin-Hao Su. $(1 - \epsilon)$-approximate maximum weighted matching in $poly(1/\epsilon, \log n)$ time in the distributed and parallel settings. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, pages 44–54, 2023.

62 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1679–1697. SIAM, 2013. doi:10.1137/1.9781611973105.121.

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

64 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 734–751. SIAM, 2014. doi:10.1137/1.9781611973402.55.

65 Michael Kapralov, Slobodan Mitrovic, Ashkan Norouzi-Fard, and Jakab Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1753–1772. SIAM, 2020. doi:10.1137/1.9781611975994.107.

66 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 352–358. ACM, 1990. doi:10.1145/100216.100262.

67 Christian Konrad. Maximum matching in turnstile streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 840–852. Springer, 2015. doi:10.1007/978-3-662-48350-3_70.

68 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX, and 16th International Workshop, RANDOM Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0_20.

69 Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. *Journal of the ACM (JACM)*, 62(5):1–17, 2015. `doi:10.1145/2786753`.

70 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC*, pages 597–606. ACM, 2011. `doi:10.1145/1993636.1993716`.

71 Andrew McGregor. Finding graph matchings in data streams. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 170–181. Springer, 2005. `doi:10.1007/11538462_15`.

72 Slobodan Mitrović, Anish Mukherjee, Piotr Sankowski, and Wen-Horng Sheu. Faster semi-streaming matchings via alternating trees. *arXiv preprint arXiv:2412.19057*, 2025.

73 Sumedh Tirodkar. Deterministic algorithms for maximum matching on general graphs in the semi-streaming model. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018.

74 Da Wei Zheng and Monika Henzinger. Multiplicative auction algorithm for approximate maximum weight bipartite matching. In *Integer Programming and Combinatorial Optimization - 24th International Conference, IPCO Proceedings*, volume 13904 of *Lecture Notes in Computer Science*, pages 453–465. Springer, 2023. `doi:10.1007/978-3-031-32726-1_32`.