Toward Optimal Semi-streaming Algorithm for $(1+\epsilon)$-approximate Maximum Matching

Abstract

We design a deterministic algorithm for the $(1+\epsilon)$-approximate maximum matching problem. As our main result, we show that this problem can be solved in $O(\epsilon^{-6})$ semi-streaming passes, improving the $O(\epsilon^{-19})$ pass-complexity obtained by the algorithm of [Fischer, Mitrović, and Uitto, STOC'22]. This significantly progresses toward resolving Open question 2 from [Assadi, SOSA'24]. By utilizing the framework proposed in [FMU'22], our algorithm yields the same round complexity speed-up for computing an $(1+\epsilon)$-approximate maximum matching in Massively Parallel Computation and CONGEST models.

The data structures our algorithm maintains are phrased in the language of blossoms and represented by alternating trees. This approach essentially allows us to simplify the correctness analysis by effectively treating certain aspects as if they were operating on bipartite graphs, thus circumventing certain technical intricacies appearing in prior work.

Contents

1 Introduction

One of the most intriguing problems in graph theory is related to bridging the complexity gaps between bipartite and non-bipartite matchings. With this respect, in recent years, we have witnessed several breakthroughs, e.g., in [San18, AV20] NC algorithms for planar perfect non-bipartite matchings have been shown after such algorithms have been known for bipartite case for 20 years already [MN95], or in [CGS15] after 10 years of research non-bipartite algorithms for weighted graphs based on fast matrix multiplication have been shown [San06]. Right now, a few intriguing open problems remain in this respect. Here, we tackle one of them, i.e., bridging the gap in the pass-complexity of non-bipartite and bipartite matchings in the semi-streaming model.

More specifically, given an undirected, unweighted graph $G = (V, E)$, 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 models such as semi-streaming [McG05, AG11, AG13, Tir18, AG18, GKMS19, FMU22, AJJ+22, Ass24, HS23], Massively Parallel Computation (MPC) [CŁM+18, ABB+19, GGK+18, GGM22, ALT21], CONGEST and LOCAL [LPSP15, BYCHGS17, FGK17, GHK18, GKMU18, Har19, FFK21]. One of the primary aims for these computational models is to deliver methods whose dependence on $1/\epsilon$ in the pass/round 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. For instance, approaches with $O(1/\epsilon^2)$ dependence have been known in CONGEST [AKO18], semi-streaming and MPC [AG13, ALT21]. (These bounds are stated with respect to the best-known dependence on n.)

For general graphs, the situation is very different. For instance, until very recently, the best pass complexity in the semi-streaming setting has either depended exponentially on $1/\epsilon$ [McG05, Tir18, GKMS19], or polynomially on $1/\epsilon$ but with a dependence on $\log n$ [AG13, AG11, AG18, AJJ+22, Ass24]; we provide an overview of existing results in Tables 1 and 2. Significant progress has been made by Fischer, Mitrović, and Uitto [FMU22], 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 [FMU22] makes important progress in understanding semi-streaming algorithms for approximating maximum matchings in general graphs, the analysis presented in that work is quite intricate, especially when compared to the analysis of known semi-streaming algorithms for bipartite graphs. 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 1.1. *Given any $\epsilon > 0$ and a graph G on n vertices, there exists a deterministic semi-streaming algorithm that 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 *structure*. The main conceptual contribution of our work is representing these structures by *alternating trees* and *blossoms*, introduced by Edmonds in his celebrated work [Edm65]. Alternating trees are formally defined in Definition 2.3, but for the sake of this section, it suffices to know that each path in this tree is alternating, and a vertex can also correspond to a contracted blossom. The most related work to ours, i.e., [FMU22], develops an ad-hoc structure. Although another related work [HS23] builds on blossom structure, it does it for reasons other than finding short augmentations; in fact, for finding short augmentation [HS23] uses [FMU22] 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 can re-use some of the well-established properties of blossoms; (2) our proof of correctness and our algorithm are simpler compared to that of [FMU22]; and (3) we are able to exhibit relations between different alternating trees during the short-augmentation search that eventually lead to our improved pass complexity, from $1/\epsilon^{19}$ to $1/\epsilon^{6}$. We hope that this perspective and simplification in the analysis will lead to further improvements in designing approximate maximum matching algorithms.

Reference	Passes	Deterministic	Weighted
[EKS09]	$O(1/\epsilon^8)$	Yes	No
[EKMS12]	$O(1/\epsilon^5)$	Yes	No
[AG13]	$O(1/\epsilon^5 \cdot \log 1/\epsilon)$	Yes	Yes
[Kap13]	$O(1/\epsilon^2)$ (vertex arrival)	Yes	No
[ALT21]	$O(1/\epsilon^2)$	Yes	No
[AJJ+22]	$O(\log(n)/\epsilon \cdot \log 1/\epsilon)$	Yes	No

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.

Implication to other models. [FMU22] 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. 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 [FMU22] and its predecessor [EKMS12], while several structural properties are borrowed from [Edm65]. We detail similarities and differences between our and [FMU22]'s techniques in Section 3.5.

Approximate maximum matchings in (semi-)streaming have been extensively studied from numerous perspectives. The closest to our work is [FMU22], who design a deterministic semi-streaming algorithm for ϵMM using $O(1/\epsilon^{19})$ passes. Prior to that work, [McG05, Tir18] 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

Reference	Passes	Deterministic	Weighted
[McG05]	$\exp(1/\epsilon)$	No	No
[AG11]	$O(\log(n)/\epsilon^7 \cdot \log 1/\epsilon)$	Yes	Yes
[AG13]	$O(\log(n)/\epsilon^4)$	Yes	Yes
[AG18]	$O(\log(n)/\epsilon)$	No	Yes
[Tir18]	$\exp(1/\epsilon)$	Yes	No
[GKMS19]	$\exp(1/\epsilon^2)$	No	Yes
[FMU22]	$O(1/\epsilon^{19})$	Yes	No
[HS23]	more than $O(1/\epsilon^{19})$	Yes	Yes
[Ass24]	$O(\log(n)/\epsilon)$	No	Yes
this work	$O(1/\epsilon^6)$	Yes	No

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.

have studied the question of estimating the size of the maximum matching [KKS14, BS15, AKL17, EHL+18, KMNT20]. The ϵMM problem has been considered in weighted graphs as well [AG11, AG13, BS15, AG18, GKMS19, HS23, Ass24]. Considering variants of the streaming setting, there have been works in dynamic streaming where one can both insert and remove edges [Kon15, BS15, CCE+16, AKLY16], in the vertex arrival model [KVV90, GKK12, Kap13, ELSW13, CTV15, BST19, GKM+19], and in random streaming where vertices or edges arrive in a random order [MY11, KMM12, GKMS19, FHM+20, Ber20, AB21].

Several works have studied lower bound questions in the streaming setting, both for exact [GO16, AR20, CKP+21] and approximate maximum matching [GKK12, Kap13, AKLY16, AKL17, AKSY20, Kap21, Ass22, AS23].

The ϵMM problem is well-studied in the classical centralized setting as well [DH03, DP14]. Furthermore, in recent years, there has been a growing interest in ϵMM in the area of dynamic algorithms [GLS+19, BGS20, ABD22, ABKL23, BK23, ZH23, BKS23] and also in sublinear time algorithms for approximate maximum matching [Beh21, BRR23, BRRS23].

1.3 Organization

After some preliminaries in Section 2, we give an overview of our approach in Section 3. Next, in Section 4 we describe our algorithm in detail. We argue about the correctness of our algorithm in Section 5 followed by the proof of the claimed complexity bounds of our algorithm in Section 6.

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 2.1 (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 2.2 (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 2.3 (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 2.4 (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 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* $e_i \in A_i \times A_{i+1}$*(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* $E_B = \bigcup_i E_{A_i} \cup \{e_0, e_1, \ldots, e_k\}$*.*

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 2.5 ([DP14])**.** *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 2.6 (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 [Edm65, Theorem 4.13].

Lemma 2.7 ([Edm65])**.** *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 [DP14].

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

(C1) *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 $A_0, \ldots, A_k \in \Omega$.*

(C2) *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 $1 \leq i < k$. 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 2.9 (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 [FKM+05], 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 [Ber57, Edm65, HK71]. 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., [McG05, EKMS12, Tir18, FMU22].

Remark: Throughout the paper, we attempt to use terminology as closely as possible to work prior, particularly the terminology used in [FMU22]. 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 reached. Figure 1 depicts an alternating tree.

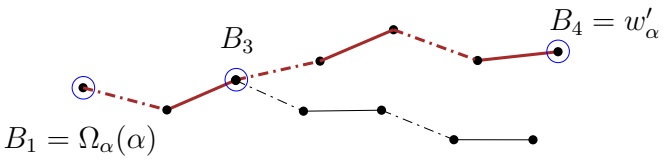


Figure 1: An example of alternating tree. Dashed and solid edges denote the unmatched and matched edges, respectively. The encircled vertices correspond to the non-trivial blossoms, i.e., B_1, B_3, and B_4 are non-trivial blossoms. $B_1 = \Omega_\alpha(\alpha)$ is the blossom containing a free vertex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

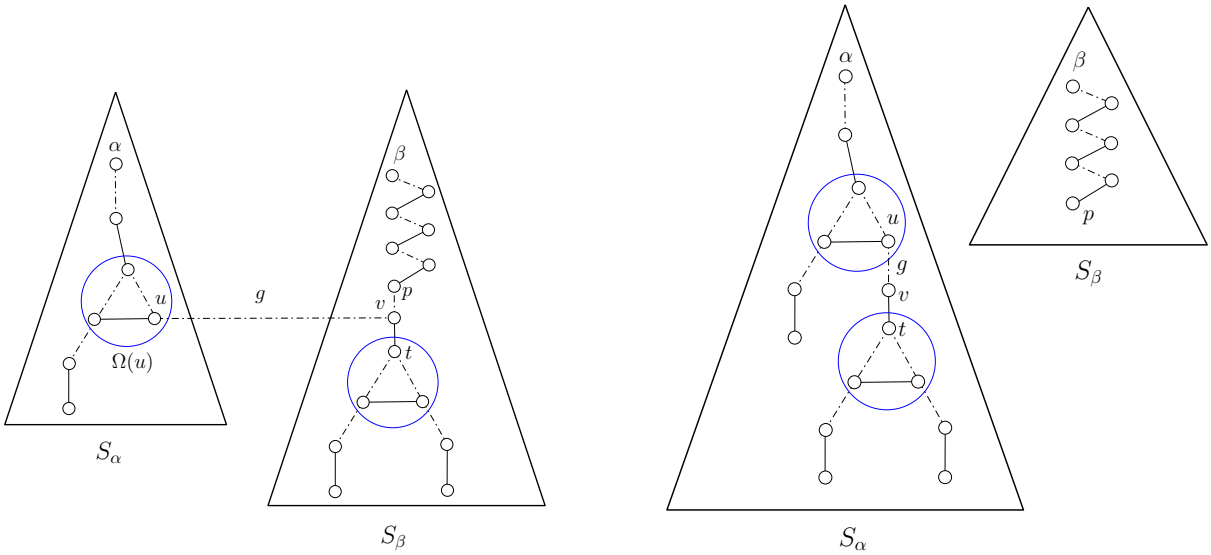
In a single pass, the working vertex v of S attempts to extend S by a length-2 path $\{u, v, t\}$, where $g = \{u, v\}$ is unmatched and $e = \{v, t\}$ is a matched edge; if it is impossible, just like in a typical DFS, this working vertex backtracks. Then, one of the following happens:

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 removed from the graph for some number of passes. This is done by procedure Augment, Section 4.5.1.

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, Section 4.5.2. These contractions ensure that the exploration subgraph from each free vertex looks like a tree. (We point out that in the full algorithm, more edges are kept in addition to a DFS tree. In particular, our algorithm stores the in-blossom edges. Nevertheless, for the sake of overview, we skip those details.)

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, Section 4.5.3. Note that, by the construction, the

[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.



(a) Before Overtake (b) After Overtake

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 $\{v\} = \Omega(v)$.)

last edge appended to DFS exploration is matched. An example of Overtake is depicted in Figure 2.

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 (Section 5)

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:

(Informal version of Theorem 5.2) 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:

(Informal version of Invariant 5.4) 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 (Section 6)

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 Limit of our approach

We believe that our approach cannot go beyond $O(1/\epsilon^3)$ passes. Intuitively, the reason is that our analysis ties together the number of label reductions, which is at most $O(|M|/\epsilon)$, and the number of working vertices at any moment. We have to execute our algorithm until there is $\Omega(\epsilon^2|M|)$ working vertices to ensure a $(1+\epsilon)$-approximate maximum matching. However, in the worst case, each working vertex reduces only one edge label per pass. This yields $\Omega((|M|/\epsilon)/(\epsilon^2|M|)) = \Omega(1/\epsilon^3)$ passes if this worst-case behavior is possible.

It remains an intriguing open question on how to significantly improve the pass complexity our algorithm provides.

3.5 Comparison with [FMU22]

A fundamental difference between our approach and that of [FMU22] 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 [FMU22], 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 [FMU22], 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 (more details are provided in Section 6).

Finally, we believe the complexity of [FMU22] can be reduced to $O(1/\epsilon^{16})$ by a slightly more careful analysis and tweaking parameters. Moreover, 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", e.g., using our Lemma 6.2, in the complexity analysis in [FMU22] would result in yet another improvement by 2 in the exponent of pass-complexity. Nevertheless, it is unclear that, unless fundamental changes are made in the approach, [FMU22] can result in a pass-complexity better than $O(1/\epsilon^{12})$.

4 Algorithms

This section presents our algorithm approach in detail. Its analysis is deferred to Sections 5 and 6. 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: 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 [FMU22]. 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 4.1 (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.*

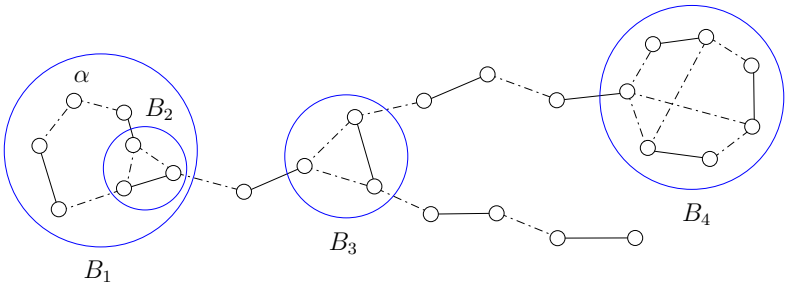
Each structure S_α satisfies the following properties.

1. ***Disjointness:** S_α is vertex-disjoint from all other structures.*
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 $\Omega_\alpha(v) = v'$.*

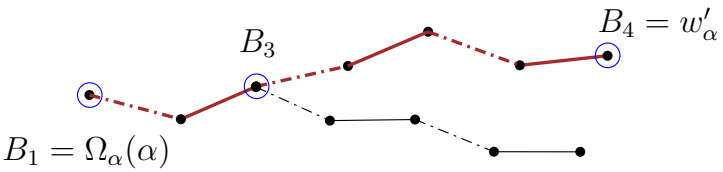
We denote the alternating tree G_α/Ω_α by T'_α. Since Ω_α is a regular set of blossom, each inner vertex of T'_α is a trivial blossom, whereas each outer vertex may be a non-trivial blossom. Figure 3b shows T'_α corresponding to the structure in Figure 3a. We remark that G_α may not be a vertex-induced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

Definition 4.2 (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 4.3 (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 4.4 (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 4.5 (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, ϵ, h) ▷ Nothing stored from the previous phase.
5: restore all vertices removed in the execution of Alg-Phase
6: augment the current matching M using the vertex-disjoint augmenting paths in $\mathcal{P}$
7: **return** M

Algorithm 1 provides an outline of our approach. Line 1 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 Line 2 corresponds to a scale h. In one scale h, each iteration of the for-loop in Line 3 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, Line 5 restores all removed vertices to G. After this step, G is identical to the input graph. Then, Line 6 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.

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. Lines 1 to 3 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, $\Omega_\alpha(\alpha)$. Line 4 computes two parameters limit_h and $\tau_{\max}(h)$. The purpose of these two parameters is detailed later. The for-loop in Line 5 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 five parts:

(1) Lines 6 to 9 initialize the status of each structure in this pass-bundle. A structure is marked

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 α, initialize its structure S_α
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 α **do**
7: if S_α has at least limit_h vertices, mark S_α as "on hold"
8: if S_α has less than limit_h vertices, mark S_α as "not on hold"
9: mark S_α as "not modified"
10: EXTEND-ACTIVE-PATH (Algorithm 3)
11: CONTRACT-AND-AUGMENT
12: BACKTRACK-STUCK-STRUCTURES
13: INCLUDE-UNMATCHED-EDGES
14: **return**

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 Section 4.6.

(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 Section 4.7.

(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 Section 4.8.

(5) After CONTRACT-AND-AUGMENT, new blossoms may be added to Ω. Next, procedure INCLUDE-UNMATCHED-EDGES is invoked to store new blossom arcs. More precisely, it makes a pass over a stream. For each blossom B, the procedure identifies all arcs in $E(B)$, and adds them to G_α, where α is a free vertex whose structure contains B. This procedure ensures that each structure stores full information about each blossom it contains. More details can be found in Section 4.9.

In Appendix A, 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 4.6. *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 4.1; in addition, Invariant 4.5 holds.*

4.4 Marking a structure on hold or modified

In the for-loop of Line 6, we mark a structure S_α *on hold* if and only if it contains at least limit_h vertices. See Lines 7 and 8 of Algorithm 2. This operation plays a crucial role in our analysis of

the pass-complexity of our algorithm, e.g., Lemmas 6.3 and 6.6.

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.

4.5 Basic operations on structures

In the following, we present three basic operations for modifying the structures. These operations are used in the execution of Extend-Active-Path and Contract-and-Augment. Whenever one of these operations is applied, the structures involved are marked as modified.

4.5.1 Procedure Augment$(g, \mathcal{P})$

Invocation reason: When our algorithm discovers an augmenting path in G.
Input:
- The set $\mathcal{P}$.
- An unmatched arc $g = (u, v)$, where $g \in E(G)$. The arc g must satisfy the following property: $\Omega(u)$ and $\Omega(v)$ are outer vertices of two different structures.

Since $\Omega(u)$ is an outer vertex, T'_α contains an even-length alternating path from the root $\Omega(\alpha)$ to $\Omega(u)$. Similarly, T'_β contains an even-length alternating path from $\Omega(\beta)$ to $\Omega(v)$. Since there is an unmatched arc $(\Omega(u), \Omega(v))$ in G', the two paths can be concatenated to form an augmenting path P' on G'.

By using Lemma 2.5, we obtain an augmenting path P on G by replacing each blossom on P' with an even-length alternating path. Augment adds P to $\mathcal{P}$ and removes S_α and S_β. That is, all vertices from $V(G_\alpha) \cup V(G_\beta)$ are removed from G, and Ω is updated as $\Omega - (\Omega_\alpha \cup \Omega_\beta)$. The vertices remain removed until the end of Alg-Phase. This guarantees that the paths in $\mathcal{P}$ remain disjoint. Recall that our algorithm adds these vertices back before the end of this phase, when Line 5 of Algorithm 1 is executed.

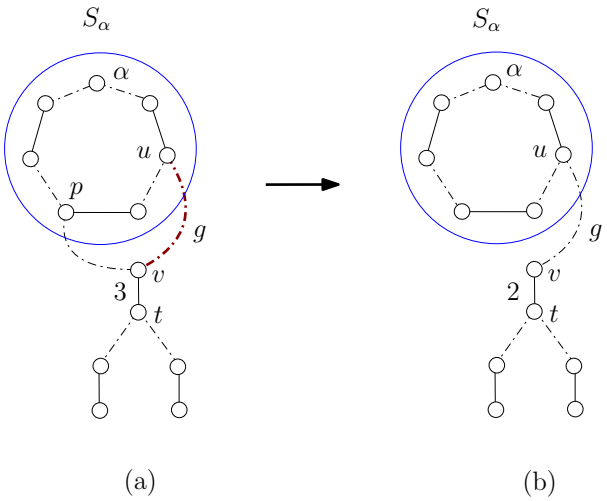
4.5.2 Procedure Contract(g)

Invocation reason: When a blossom in a structure is discovered.
Input:
- An unmatched arc $g = (u, v)$, where $g \in E(G)$, such that $\Omega(u)$ and $\Omega(v)$ are distinct outer vertices in the same structure, denoted by S_α. In addition, $\Omega(u)$ is the working vertex of S_α

Let g' denote the arc $(\Omega(u), \Omega(v))$. By Lemma 2.7, $T'_\alpha \cup \{g'\}$ contains a unique blossom B. The procedure contracts B by adding B to Ω_α; hence, T'_α is updated as T'_α/B after this operation. The arc g is added to G_α.

By Lemma 2.7, T'_α remains an alternating tree after the contraction, and B becomes an outer vertex of T'_α. Next, the procedure sets the label of each matched arc in $E(B)$ to 0. (After this step, for each matched arc $a \in E(B)$, both $\ell(a)$ and $\ell(\overleftarrow{a})$ are 0.)

Note that the working vertex of S_α, that is, $\Omega(u)$, is contracted into the blossom B. The procedure then sets B as the new working vertex of S_α. Then, S_α is marked as modified.



(a) (b)

Figure 4: Example of Case 2.1 of procedure OVERTAKE. In this case, $\Omega(v)$ is a child of $\Omega(u)$ before the overtaking operation. (a) The structure S_α before, where the arc g is highlighted. (b) The structure S_α after we invoke OVERTAKE via $g = (u, v)$.

4.5.3 Procedure OVERTAKE(g, a, k)

Invocation reason: When the active path of a structure S_α can be extended through g to *overtake* the matched arc a and reduce $\ell(a)$ to k.
Input:
- An unmatched arc $g = (u, v) \in G$.
- A non-blossom matched arc $a = (v, t) \in G$, which shares the endpoint v with g.
- A positive integer k.
- The input must satisfy the following.

(P1) $\Omega(u)$ is the working vertex of a structure, denoted by S_α.

(P2) $\Omega(v) \neq \Omega(u)$, and $\Omega(v)$ is either an unvisited vertex or an inner vertex of a structure S_β, where S_β can be S_α. In the case where $\Omega(v) \in S_\alpha$, $\Omega(v)$ is not an ancestor of $\Omega(u)$.

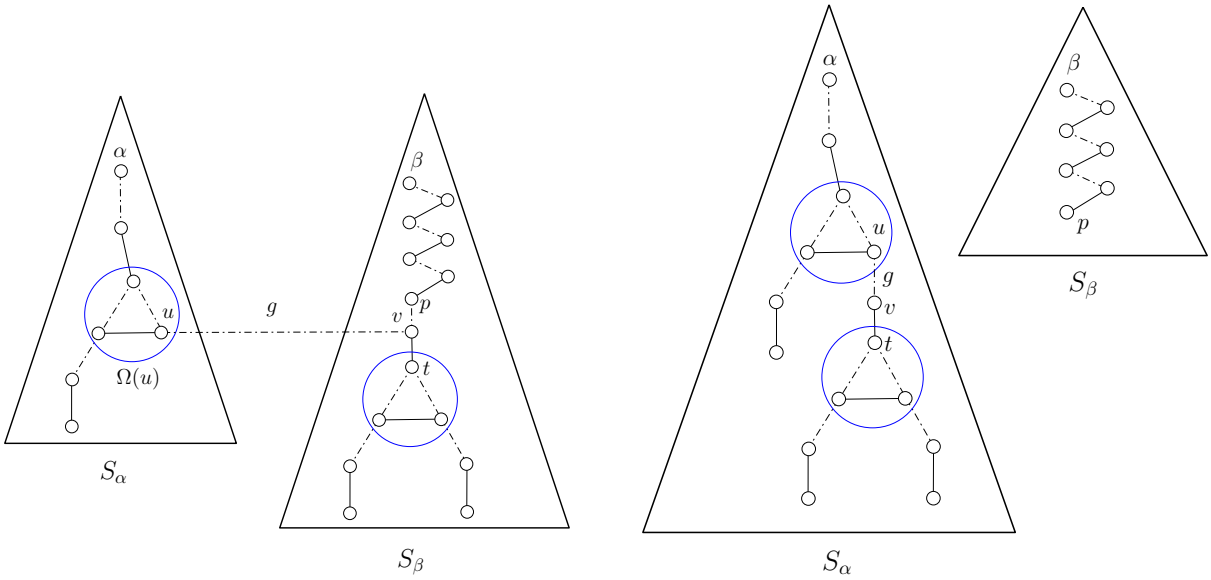
(P3) $k < \ell(a)$.

For ease of notation, we denote $\Omega(u), \Omega(v)$, and $\Omega(t)$ by u', v', and t', respectively. Since v' is not an outer vertex, it is the trivial blossom $\{v\}$. The procedure OVERTAKE performs a series of operations, detailed as follows. Consider three cases, where in all of them we reduce the label of a to k.

Case 1. a is not in any structure. We include the arcs g and a to G_α. The trivial blossoms v' and t' are added to Ω_α. The working vertex of S_α is updated as t', which is an outer vertex of T'_α. Then, S_α is marked as modified.

Case 2. a is in a structure S_β. By the definition of g, v' is an inner vertex. Thus, v' is not the root of T'_β. By Definition 4.1-3, there is a unique unmatched arc $(p, v) \in S_\beta$ such that $\Omega(p)$ is the parent of $\Omega(v)$. Two subcases are considered.

Case 2.1. $\alpha = \beta$. See Figure 4 for an example. By (P2), v' is not an ancestor of u'. The overtaking operation is done by updating $E(G_\alpha)$ as $E(G_\alpha) - \{(p, v)\} \cup \{g\}$. On the tree T'_α, this operation corresponds to re-assigning the parent of v' as u'. Then, we update the working vertex of S_α as t' and mark S_α as modified. We remark that it is possible that v'



(a) Before Overtake (b) After Overtake

Figure 5: Example of Case 2.2 of the procedure Overtake where $g = (u, v)$ connects the two structures S_α and S_β. While, in this example $\Omega(p) = \{p\}$ is a trivial blossom, it can be a non-trivial blossom in general.

is the child of u' before the overtaking operation, as shown in Figure 4.

Case 2.2. $\alpha \neq \beta$. See Figure 5 for an example. Similar to Case 2.1, the objective of the overtaking operation is to re-assign the parent of v' as u' on G'. However, we need to handle several additional technical details in this case. The overtaking operation consists of the following steps.

Step 1: Remove the arc (p, v) from G_β and add the arc (u, v) to G_α.

Step 2: Move, from G_β to G_α, all vertices x such that $\Omega(x)$ is in the subtree of v'

Step 3: Move, from G_β to G_α, all arcs (x, y) where x and y are both moved in Step 1.

Step 4: Move, from Ω_β to Ω_α, all blossoms that contain a subset of vertices moved in Step 1.

Step 5: If the working vertex of S_β was under the subtree of t' before Step 1, we set w'_α as w'_β and then update w'_β as $\Omega(p)$. Otherwise, set w'_α as t'.

After the overtaking operation, both S_α and S_β are marked as modified.

4.6 Procedure Extend-Active-Path

The goal of Extend-Active-Path is to *extend* each structure S_α, where S_α is not on hold, by performing at most one of the Augment, Contract, or Overtake operations.

The procedure works as follows. (See Algorithm 3 for a pseudocode.) The algorithm makes a pass over the stream to read each arc $g = (u, v)$ of G. When an arc g is read, it is mapped to an arc $g' = (\Omega(u), \Omega(v))$ of G'. In Extend-Active-Path, we only consider non-blossom unmatched arcs whose tail is a working vertex. Hence, if $\Omega(u) = \Omega(v)$, $\Omega(u)$ is not the working vertex of a structure, or g is a matched arc, then we simply ignore g. If one of u or v is removed, we also ignore g.

Algorithm 3 The execution of EXTEND-ACTIVE-PATH.

Input: a graph G, the parameter ϵ, the current matching M, the structure S_α of each free vertex α, the set of paths $\mathcal{P}$

```
1: for each arc g = (u, v) ∈ E(G) on the stream do
2:     if u or v was removed in this phase then
3:         continue with the next arc
4:     if Ω(u) = Ω(v), or Ω(u) is not the working vertex of any structure, or g is matched then
5:         continue with the next arc
6:     if u belongs to a structure that is marked as modified or on hold then
7:         continue with the next arc
8:     if Ω(u) is an outer vertex then
9:         if Ω(u) and Ω(v) are in the same structure then
10:            CONTRACT(g)
11:        else
12:            AUGMENT(g)
13:    else                                   ▷ Ω(v) is either unvisited or an inner vertex.
14:        compute distance(u)
15:        a ← the matched arc in G whose tail is v
16:        if distance(u) + 1 < ℓ(a) then
17:            OVERTAKE(g, a, distance(u) + 1)
```

Let S_α denote the structure whose working vertex is $\Omega(u)$, and let S_β denote the structure containing $\Omega(v)$. We ignore g if S_α is marked as on hold. Before this procedure, no structure is marked as modified. If α is marked as modified, we know it has extended during the execution of EXTEND-ACTIVE-PATH. In this case, we also ignore g. This ensures that each structure only extends once in the execution of EXTEND-ACTIVE-PATH. The above property is crucial to our analysis, e.g., in Theorem 5.2.

We examine whether g can be used for extending S_α as follows.

Case 1: $\Omega(v)$ is an outer vertex and $S_\alpha = S_\beta$. In this case, g' induces a blossom on T'_α. We invoke CONTRACT on g to contract this blossom.

Case 2: $\Omega(v)$ is an outer vertex and $S_\alpha \neq S_\beta$. In this case, the two structures can be connected to form an augmenting path. We invoke AUGMENT to compute this augmenting path and remove the two structures.

Case 3: $\Omega(v)$ is either an inner vertex or an unvisited vertex. Note that v cannot be a free vertex because, for each γ, it holds that $\Omega(\gamma)$ is an outer vertex. Therefore, v is the tail of a matched arc a. We determine whether S_α can overtake a by computing a number $\mathsf{distance}(u)$ and compare it with $\ell(a)$. $\mathsf{distance}(u)$ is computed as follows. Let Q^* be the shortest even-length alternating path in $E(\Omega(u)) \cap E(G_\alpha)$ from the base of $\Omega(u)$ to u. Recall that $E(\Omega(u))$ is the set of all arcs in G connecting a pair of vertices in the blossom $\Omega(u)$, which may include arcs not in G_α. Let d be the number of matched arcs in Q^*. If $\Omega(u)$ is the root $\Omega(\alpha)$ of T'_α, $\mathsf{distance}(u)$ is computed as d. Otherwise, there is a matched arc $a' \in T'_\alpha$ connecting $\Omega(u)$ with its parent, and we compute $\mathsf{distance}(u)$ as $\ell(a') + d$. If $\mathsf{distance}(u) + 1 < \ell(a)$, OVERTAKE is invoked to update the label of a as $\mathsf{distance}(u) + 1$. By Invariant 4.5, OVERTAKE is invoked when $\Omega(v) \in T'_\alpha$, then $\Omega(v)$ is not an ancestor of $\Omega(u)$.

We remark on a technical detail as follows. In the description above, the path Q^* is computed using the set of arcs $E(\Omega(u)) \cap E(G_\alpha)$. The purpose of this step is to find a

shortest even-length alternating path in $E(\Omega(u))$. However, since this step must be done without reading any arc from the stream, the computation is performed using only the arcs already stored in S_α, that is, $E(\Omega(u)) \cap E(G_\alpha)$. In the analysis of our algorithm (see Theorem 5.2), we show that at the moment Q^* is computed, it holds that $E(\Omega(u)) \cap E(G_\alpha) = E(\Omega(u))$. Therefore, Q^* is the shortest even-length alternating path in $E(\Omega(u))$ from the base of $\Omega(u)$ to u.

4.7 Procedure Contract-and-Augment

The procedure Contract-and-Augment performs two steps to identify augmenting paths and blossoms:

> Step 1: Repeatedly invoke Contract on an arc connecting two outer vertices of the same structure, where one of the outer vertices is the working vertex.
>
> Step 2: For each arc g connecting outer vertices of different structures, invoke Augment with g.

Step 1 is implemented as follows. First, for each free vertex α, compute A_α as the set of arcs connecting two vertices in G_α. The computation of A_α for all free vertices α can be done in one pass over the stream. Next, we repeatedly perform the following operation on each structure S_α: While there exists an arc $(u, v) \in A_\alpha$ such that $\Omega(u)$ is the working vertex of S_α and $\Omega(v) \neq \Omega(u)$ is an outer vertex, invoke Contract on (u, v) to contract the blossom.

Step 2 is implemented by scanning each arc (u, v) over the stream, and if $\Omega(u)$ and $\Omega(v)$ are outer vertices of different structures, we invoke Augment on (u, v).

The purpose of Contract-and-Augment is to ensure an invariant (see Invariant 5.4) that we leverage in our correctness analysis.

4.8 Procedure Backtrack-Stuck-Structures

The purpose of Backtrack-Stuck-Structures is to backtrack the structures that do not make progress in a pass-bundle. More formally, each structure that is not on hold and not modified is backtracked. Note that if a structure is marked as not modified when the procedure Backtrack-Stuck-Structures is invoked, then it did not extend in Extend-Active-Path, and Contract-and-Augment could not contract any blossom inside the structure.

Consider a structure S_α that is not on hold and not modified. The backtracking is performed as follows. If w'_α is a non-root outer vertex of T'_α, we update the working vertex as the parent of the parent of w'_α, which is an outer vertex. Otherwise, w'_α is the root $\Omega(\alpha)$. In this case, we set the working vertex of S_α as $\emptyset$, which makes S_α inactive.

Backtrack-Stuck-Structures may change w'_α of some structures S_α. We do not mark these changed structures as modified because the mark is not used in the rest of the pass-bundle.

4.9 Procedure Include-Unmatched-Edges

The procedure Include-Unmatched-Edges makes a pass over the stream and examines each arc $(u, v) \in E(G)$. If (u, v) is a blossom arc, that is, $\Omega(u) = \Omega(v)$, then it is added to the structure containing $\Omega(u)$.

It is not hard to see that this procedure ensures the following invariant.

Invariant 4.7 (Representation of blossoms)**.** *At the beginning of each pass-bundle, the following holds. Consider a structure S_α. For each blossom $B \in \Omega(\alpha)$, G_α contains all arcs $(u, v) \in E(G)$ such that $\Omega(u) = \Omega(v) = B$.*

We remark that Invariant 4.7 only holds at the beginning of each pass-bundle. During the execution of a pass-bundle, new blossoms may be added to Ω by Contract. At the moment a blossom B is added, the structure containing B may not store all blossom arcs in $E(B)$.

Include-Unmatched-Edges may change G_α of some structures S_α. We do not mark these structures as modified because the mark is not used in the rest of the pass-bundle.

5 Correctness

The goal of this section is to show that our algorithm does not miss any short augmentation. That is, if our algorithm is left to run indefinitely and no structure is on hold, then at some point, the remaining graph will have no short augmentation left. Formally, we show the following.

Definition 5.1 (Critical arc and vertex)**.** *Recall that the active path of a structure is a path in G'. We say a non-blossom arc $(u, v) \in G$ is* critical *if the arc $(\Omega(u), \Omega(v)) \in G'$ is active. In particular, all blossom arcs in a structure are not critical, even if they are in an active blossom. We say a free vertex $\alpha \in G$ is* critical *if S_α is active.*

Theorem 5.2 (No short augmenting paths is missed)**.** *At the beginning of each pass-bundle, the following holds. Let $P = (\alpha, a_1, a_2, \ldots, a_k, \beta)$ be an augmenting path in G such that no vertex in P is removed in this phase and $k \leq \ell_{\max}$. At least one of the following holds: α is critical, or P contains a critical arc.*

Properties of a phase. Our analysis of Theorem 5.2 relies on the following two simple properties.

Lemma 5.3 (Outer vertex has been a working one)**.** *Consider a pass-bundle τ. Suppose that G' contains an outer vertex v' at the beginning of τ. Then, there exists a pass-bundle $\tau' \leq \tau$ such that v' is the working vertex at the beginning of τ'.*

Invariant 5.4. *At the beginning of each pass-bundle, no arc in G' connects two outer vertices.*

Lemma 5.5. *Invariant 5.4 holds.*

Proofs of these two claims are deferred to Sections 5.2 and 5.3.

We remark that Invariant 5.4 only holds at the beginning of each pass-bundle. During the execution of Extend-Active-Path, some structures may include new unvisited vertices or contract a blossom in the structure. These operations create new outer vertices that may be adjacent to existing outer vertices.

5.1 No short augmentation is missed (Proof of Theorem 5.2)

For a pass-bundle τ, we use Ω^τ and ℓ^τ to denote, respectively, the set of blossoms and labels at the beginning of τ.

Consider a fixed phase and a pass-bundle τ in the phase. Suppose, toward a contradiction, that at the beginning of τ, there exists an augmenting path $P = (\alpha, a_1, a_2, \ldots, a_k, \beta)$ in G, where $k \leq \ell_{\max}$, such that:

(i) none of the vertices in P is removed in the phase, and

(ii) α and all arcs in P are not critical.

Recall that for an arc to be critical, by Definition 5.1, it has to be non-blossom. Hence, some blossom arcs of P may be inside of an active blossom at this moment. For $i = 1, 2, \ldots, k$, let $a_i = (u_i, v_i)$. Let $v_0 = \alpha$. Two cases are considered:

Case 1: There exists an index q such that $\ell^\tau(a_q) > q$. Let q be the smallest index such that $\ell^\tau(a_q) > q$. Since $\ell^\tau(a_q) > q > 0$, a_q must be a non-blossom arc. Let p be the smallest index such that $p < q$ and $a_{p+1}, \ldots, a_{q-1}$ are blossom arcs. See Figure 6 for a sketch example.

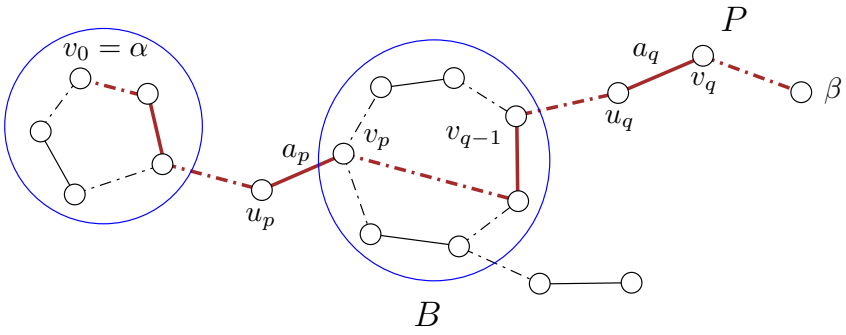


Figure 6: This sketch is used to illustrate the proof of Theorem 5.2, Case 1. The highlighted path P is an augmenting path between α and β. The blossom B contains all vertices and arcs in the path from v_p to a_{q-1}. Note that the shortest even-length alternating path from v_p to v_{q-1} contains an arc not in E_B.

If $p = 0$, then $v_p = \alpha$ and thus $\Omega(v_p)$ is an outer vertex. Otherwise, since $\ell^\tau(a_p) \leq p \leq \ell_{\max}$, a_p is visited. Hence, a_p is a non-blossom arc contained in a structure. We also have that $\Omega(v_p)$ is an outer vertex. For $p < i < q$, since a_i is a blossom arc, $\Omega(u_i)$ and $\Omega(v_i)$ are both outer vertices. Hence, by Invariant 5.4, all vertices in the path $(v_p, a_{p+1}, a_{p+2}, \ldots, a_{q-1})$ must be in the same blossom at the beginning of τ. Denote this blossom by B. Clearly, if $p = 0$, then $v_p = \alpha$ is the base of B. Otherwise, since a_p is a non-blossom matched arc adjacent to B, we also have v_p as the base of B. Since a_p is non-critical at the beginning of τ, we have that B **is inactive at the beginning of** τ.

Let $\tau' \leq \tau$ be the last pass-bundle such that B is the working vertex of some structure S_γ at the beginning of τ'. By Lemma 5.3, τ' exists. In τ', S_γ backtracks from B, and B remains inactive until at least the beginning of τ. Hence, if $p > 0$, $\ell(a_p)$ is not updated between the end of τ' and the beginning of τ. Therefore, $\ell^{\tau'}(a_p) = \ell^\tau(a_p) \leq p$.

By definition of Backtrack-Stuck-Structures (Section 4.8), S_γ is not marked as on hold or modified in τ'. Thus, when Extend-Active-Path read the unmatched arc $g = (v_{q-1}, u_q) \in G$ from the stream, $\Omega(v_{q-1}) = B$ is the working vertex of S_γ. We claim that $\overleftarrow{a_q}$ cannot be in any structure at this moment. If $\overleftarrow{a_q}$ is in any structure, then $\Omega(u_q)$ is an outer vertex, and Extend-Active-Path should invoke either Augment or Contract on g. This leads to a contradiction because either S_γ is removed or it is marked as modified in this pass-bundle. Hence, $\overleftarrow{a_q}$ is not in any structure, and thus $\Omega(u_q)$ is either unvisited or an inner vertex. That is, **at the moment g is read, Line 14 of Extend-Active-Path is executed to compute** $\mathsf{distance}(v_{q-1})$.

Recall that v_p is the base of B. Thus, $\mathsf{distance}(u)$ is computed by finding the shortest even-length alternating path Q^* in $E(B) \cap E(G_\gamma)$ from v_p to v_{q-1}. By Invariant 4.7, G_γ contains all blossom arcs in $E(B)$ at the beginning of τ'. Since S_γ is not marked as modified in the pass-bundle, G_γ still contains these arcs when Q^* is computed. Hence, all arcs in the path $(v_p, a_{p+1}, \ldots, a_{q-1})$ are stored in G_γ at this moment. As a result, the length of Q^* is at most the length of $(v_p, a_{p+1}, \ldots, a_{q-1})$. If $p = 0$, $\mathsf{distance}(u)$ is computed as the number of matched arcs in Q^*, which is at most $q-p-1 = q-1$; otherwise, $\mathsf{distance}(u)$ is computed as $\ell^{\tau'}(a_p)+(q-p-1) \leq q-1$, where we use our conclusion from above that $\ell^{\tau'}(a_p) = \ell^\tau(a_p) \leq p$. This implies that Extend-Active-Path should have updated the label of a_q to q in τ', which contradicts the definition of q.

Case 2: For each $i = 1, 2, \ldots, k$ it holds $\ell^\tau(a_i) \leq i$. Let $p \leq k$ be the smallest index such that all vertices in $a_{p+1}, a_{p+2}, \ldots, a_k$ are blossom arcs at the beginning of τ. Similar to Case 1, if $p = 0$, then $v_p = \alpha$; otherwise, a_p is a non-blossom arc with $\ell^\tau(a_p) \leq p$. Hence, $\Omega(v_p)$ is an outer vertex.

For each $i > p$, since a_i is a blossom arc, $\Omega(u_i)$ and $\Omega(v_i)$ are both outer vertices. Hence, by Invariant 5.4, all vertices in the path $(v_p, a_{p+1}, a_{p+2}, \ldots, a_k)$ must be in the same blossom at the beginning of τ. Denote this blossom by B. Since $\Omega^\tau(\beta)$ is the root of T'_β, it is also an outer vertex. That is, $\Omega^\tau(v_k)$ and $\Omega^\tau(\beta)$ are both outer vertices.

Since P contains an unmatched arc (v_k, β), by Invariant 5.4, $\Omega^\tau(\beta) = \Omega^\tau(v_k) = B$. If $p = 0$, this leads to a contradiction because B contains two free vertices $v_p = \alpha$ and β. If $p > 0$, then $B = \Omega(\beta)$ is not a free vertex because it is adjacent to a non-blossom matched arc a_p, which is also a contradiction.

5.2 Outer vertex has been a working one (Proof of Lemma 5.3)

Suppose that v' is a trivial blossom. Then, v' is a working vertex when it is added to a structure. Denote by $\tau'' < \tau$ the pass-bundle in which v' is first added to a structure. We claim that v' is the working vertex at the beginning of $\tau'' + 1$. After v' is added to a structure in τ', v' may be overtaken by other structures. By the definition of Overtake (Section 4.5.3), when a structure overtakes v', it also sets v' to be its working vertex. Hence, v' is a working vertex after the invocation of Extend-Active-Path (Section 4.6). Since v' remains in G' at least until τ, we know that v' is not removed or contracted by Augment (Section 4.5.1) or Contract (Section 4.5.2). Therefore, v' is the working vertex of a structure at the beginning of $\tau'' + 1$.

Next, consider the case when v' is a non-trivial blossom. Then, v' is a working vertex at the moment it is added to Ω by Contract. Let $\tau'' < \tau$ denote the pass-bundle such that v' is added to Ω. By a similar argument, we know that v' is the working vertex of a structure at the beginning of $\tau'' + 1$.

5.3 No arc between outer vertices (Proof of Lemma 5.5)

Suppose, by contradiction, that at the beginning of some pass-bundle τ, there is an arc $g' \in E(G')$ connecting two outer vertices u' and v'. Let $\tau_{u'} < \tau$ (resp. $\tau_{v'} < \tau$) be the first pass-bundle in which u' (resp. v') is added to a structure by either Overtake or Contract. Without loss of generality, assume that $\tau_{u'} \geq \tau_{v'}$. There are two cases:

1. u' was an unvisited vertex before $\tau_{u'}$ and is added to a structure by Overtake in $\tau_{u'}$.
2. u' is a non-trivial blossom added to Ω in $\tau_{u'}$.

In the first case, by the proof of Lemma 5.3, we know that u' is a working vertex after the invocation of Extend-Active-Path in $\tau_{u'}$. Therefore, in Contract-and-Augment (Section 4.7), either Augment or Contract will be invoked on g', which contradicts the definition of g'.

In the second case, u' remains a working vertex after it is added to Ω (by Contract, in either Extend-Active-Path or Contract-and-Augment) and before the end of $\tau_{u'}$. Hence, in Contract-and-Augment, either Augment or Contract will be invoked on g', leading to a contradiction.

6 Pass and Space Complexity, and Approximation Guarantee

In this section, we show that our algorithm finds a $(1+\epsilon)$-approximate matching using $O(n/\epsilon^6)$ words of space and $O(1/\epsilon^6)$ passes over the stream.

6.1 Overview of parameters

As a reminder, our algorithm is parameterized by h (see Algorithm 1), which we refer to by scale. Intuitively, h dictates the fraction of remaining augmentations our algorithm seeks. The idea is that initially, our algorithm is aggressive and looks for many augmentations at once. Only after, once the algorithm knows there are not many augmentations left, does the algorithm fine-tune by searching for a relatively few remaining augmentations. Since not many augmentations are left, this fine-tuning can be implemented more efficiently than without varying scales.

Our analysis ties together several parameters used in our algorithm, summarized in Table 3. Those parameters are:

- $h_{\min}$, which is the minimum scale,
- $t_{\max}(h)$, which is the number of phases for a given scale,
- Δ_h, which is an upper bound on the number of vertices that a structure can have.

Parameter	Definition	Asymptotic value	Exact value
$\ell_{\max}$	maximum label for a visited matched arc	$\Theta(\epsilon^{-1})$	$3\epsilon^{-1}$
$h_{\min}$	the minimum scale	$\Theta(\epsilon^2)$	$\epsilon^2/64$
$t_{\max}(h)$	number of phases in a scale	$\Theta((h\epsilon)^{-1})$	$144(h\epsilon)^{-1}$
$\tau_{\max}(h)$	number of pass-bundles in a phase	$\Theta((h\epsilon)^{-1})$	$72(h\epsilon)^{-1}$
limit_h	lower bound on the size of an on-hold structure	$\Theta(h^{-1})$	$6h^{-1}+1$
Δ_h	upper bound on the size of any structure	$\Theta((h\epsilon)^{-1})$	$36(h\epsilon)^{-1}$

Table 3: Descriptions of the parameters used in our algorithm.

Recall that on Line 7 of Alg-Phase (Algorithm 2), the algorithm stops extending structure whose size reaches limit_h. However, Δ_h, which denotes the maximum size of a structure in a scale h, is not trivially bounded by $O(\mathsf{limit}_h)$. To see this, notice that although we stop extending a structure S_α once its size exceeds limit_h, another structure S_β can overtake S_α. This can potentially make the size of S_β almost $2 \cdot \mathsf{limit}_h$. Then, another structure S_γ can overtake S_β, making S_γ of size almost $3 \cdot \mathsf{limit}_h$, and so on. In Lemma 6.6, we show that this type of growth cannot proceed for "too long". By leveraging the fact that each free vertex essentially maintains a tree, we obtain an upper bound of $\mathsf{limit}_h \cdot \ell_{\max}$ on the structure size.

6.2 Approximation analysis

This section proves that Algorithm 1 outputs a $(1+\epsilon)$-approximate maximum matching. Our analysis starts with the following well-known fact.

Definition 6.1 (M-augmenting k-path)**.** *For a matching M and an integer $k \geq 1$, define an M-augmenting k-path as an augmenting path with respect to M consisting of at most k matched edges.*

Lemma 6.2. *Consider a graph G, a matching M of G, and a fixed real number $p > 0$. Let $\mathcal{P}^*$ be the maximum set of disjoint M-augmenting $\ell_{\max}$-paths in G. If $|\mathcal{P}^*| \leq p|M|$, then $|M|$ is a $(1+p) \cdot (1+1/\ell_{\max})$-approximate maximum matching.*

The claims akin to Lemma 6.2 are well-established. For the sake of completeness, we provide its proof in Section 6.4.

Recall that $\ell_{\max} = 3/\epsilon$. By letting $p = \frac{1}{\ell_{\max}}$ in Lemma 6.2, if at any moment the graph contains less than $p|M|$ disjoint M-augmenting $\ell_{\max}$-paths, M is smaller than the optimal matching by at most factor of

$$\left(1 + \frac{1}{\ell_{\max}}\right) \cdot \left(1 + \frac{1}{\ell_{\max}}\right) \leq \left(1 + \frac{3}{\ell_{\max}}\right) = 1 + \epsilon.$$

However, our algorithm does not immediately let $p = 1/\ell_{\max}$, but it rather arrives at that fraction gradually by considering different scales. The idea behind using varying scales builds on two observations: (1) the larger the value of p, the sooner the algorithm stops; (2) the fewer augmenting paths there are, to begin with, the sooner the algorithm stops for a fixed p. Hence, the algorithm finds a certain fraction of augmentations for a scale h, and then a smaller fraction of augmentations remains to be found for $h/2$. We formalize this by the following claim, whose proof is deferred to Section 6.5.

Lemma 6.3 (Upper bound on the number of active structures)**.** *Consider a fixed phase for a scale h. Let M be the matching at the beginning of the phase. Then, at the end of that phase, there are at most $h|M|$ active structures.*

We turn Lemma 6.3 into a claim about the matching-size improvement, proved in Section 6.6 and formalized as follows:

Lemma 6.4. *Consider a fixed phase in a scale h. Let M be the matching at the beginning of the phase. Let $\mathcal{P}^*$ be the maximum set of disjoint M-augmenting $\ell_{\max}$-paths in G. If $|\mathcal{P}^*| \geq 4h\ell_{\max}|M|$, then at the end of the phase the size of M is increased by a factor of at least $1 + \frac{h\ell_{\max}}{\Delta_h}$.*

It is not hard to build on Lemma 6.4 and obtain a guarantee on the maximum matching approximation after all the phases of a fixed scale. In particular, for a fixed scale h, our algorithm transforms M into a $(1 + 4h\ell_{\max}) \cdot (1 + 1/\ell_{\max})$-approximate maximum matching, implying that M becomes a $(1 + \epsilon)$-approximation for scale $h = \Theta\left(1/\ell_{\max}^2\right)$.

Lemma 6.5 (Matching size at the end of a scale)**.** *At the end of each scale h, the matching M is a $(1 + 4h\ell_{\max}) \cdot \left(1 + \frac{1}{\ell_{\max}}\right)$-approximate maximum matching.*

We defer the proof of Lemma 6.5 to Section 6.7.

The remaining piece we need for our analysis is an upper bound on the structure size, i.e., Δ_h, which figures in the guarantee of Lemma 6.4. As discussed in Section 6.1, Δ_h is not trivially upper bounded by limit_h. Nevertheless, in Section 6.8, we show that it can only be by a factor of $\ell_{\max}$ larger than limit_h.

Lemma 6.6 (Upper bound on structure size)**.** *Consider a fixed phase of a scale h. At any moment of the phase, the size of each structure is at most $\mathsf{limit}_h \cdot \ell_{\max}$.*

We can now conclude the approximation analysis.

Theorem 6.7. *Algorithm 1 outputs a $(1 + \epsilon)$-approximate maximum matching.*

Proof. Recall that $\Delta_h = 36(h\epsilon^{-1})$, which is greater that $\mathsf{limit}_h \cdot \ell_{\max}$ (see Table 3). By Lemma 6.6, at any moment of scale h, the size of each structure is upper bounded by Δ_h.

By Lemma 6.5, at the end of scale $h = \frac{\epsilon^2}{64} \leq \frac{1}{4\ell_{\max}^2}$, M is a $\left(1 + \frac{1}{\ell_{\max}}\right) \cdot \left(1 + \frac{1}{\ell_{\max}}\right)$-approximate maximum matching. Since

$$\left(1 + \frac{1}{\ell_{\max}}\right) \cdot \left(1 + \frac{1}{\ell_{\max}}\right) \leq \left(1 + \frac{3}{\ell_{\max}}\right) = (1 + \epsilon),$$

the output of our algorithm is a $(1 + \epsilon)$-approximate maximum matching. □

6.3 Pass and space complexity

In this section we prove the following claim.

Lemma 6.8. *Algorithm 1 uses $O\left(1/\epsilon^6\right)$ passes and $O\left(n/\epsilon^6\right)$ space.*

The pass complexity. See Table 3 for the parameters used in this algorithm. Consider a fixed scale h. The number of phases is $t_{\max}(h) = O(\frac{1}{h\epsilon})$, each phase executes $\tau_{\max}(h) = O(\frac{1}{h\epsilon})$ pass-bundles, and each pass-bundle requires 4 passes over the stream. Hence, the number of passes in scale h is

$$4 \cdot t_{\max}(h) \cdot \tau_{\max}(h) = 4 \cdot \frac{144}{h\epsilon} \cdot \frac{72}{h\epsilon} = O\left(\frac{1}{\epsilon^2} \cdot \frac{1}{h^2}\right).$$

Since we iterate through $h = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots, h_{\min} = \frac{\epsilon^2}{64}$, the total number of passes is

$$O\left(\frac{1}{\epsilon^2} \cdot (2^2 + 4^2 + 8^2 + \cdots + (64/\epsilon^2)^2)\right) = O\left(\frac{1}{\epsilon^6}\right).$$

This completes the proof of the pass complexity.

The space complexity. Consider a fixed scale h. At any moment in the scale, we need to store the labels of each matched arc and the structure of each free vertex. Since there are $2|M|$ matched arcs, the labels can be stored using $O(|M|) = O(n)$ space.

Since the structures are vertex-disjoint, the total number of vertices of all structures is bounded by $O(n)$. Each structure S_α may contain up to $O(|V(G_\alpha)|^2)$ arcs. (For example, if G_α is a single blossom, our algorithm stores all arcs connecting two vertices in the blossom.) By Lemma 6.6, the number of vertices in a structure is upper-bounded by $\Delta_h = O(1/(h\epsilon))$. Hence, the number of arcs in any structure is at most $O(1/(h\epsilon)^2)$. Since there are at most n structures, the arcs of all structures can be stored in $O(n/(h\epsilon)^2)$ space.

In the execution of a phase, we maintain the set of vertices that are removed so that they can be added back at the end of the phase. This requires $O(n)$ space.

In Contract-and-Augment, we compute the set of arcs A_α for each free vertex α. Recall that A_α is the set of all arcs connecting two vertices in G_α. Hence, the size of each A_α is at most $O(1/(h\epsilon^2))$. As a result, Contract-and-Augment can be implemented using $O(n/(h\epsilon)^2)$ space.

In Include-Unmatched-Edges, we store all blossom arcs in $E(G)$. Since each blossom arc connects two vertices of the same structure, the total number of blossom arcs is at most the total number of arcs in all structures, which is $O(n/(h\epsilon)^2)$. Consequently, a scale h can be implemented using $O(n) + O(n/(h\epsilon)^2)$ space. Since the minimum scale is $h_{\min} = \Theta(\epsilon^2)$, our algorithm requires $O(n/(h_{\min}\epsilon)^2) = O(n/\epsilon^6)$ space.

6.4 Proof of Lemma 6.2

For two sets of edges E_1 and E_2, we use $E_1 \oplus E_2$ to denote the *symmetric difference* of E_1 and E_2, defined as $(E_1 \cup E_2) - (E_1 \cap E_2)$. Let M^* be an optimal matching of G. It is known that each component in $M \oplus M^*$ either is an M-augmenting path or has the same number of edges from M and M^*. Let $\mathcal{P}_i$ denote the set of M-augmenting i-paths in $M \oplus M^*$.

Let M' denote the matching obtained by augmenting M using all paths in $\bigcup_{i \le \ell_{\max}} \mathcal{P}_i$. That is, $M' = M \oplus \bigcup_{i \le \ell_{\max}} E(\mathcal{P}_i)$. Since $\bigcup_{i \le \ell_{\max}} \mathcal{P}_i$ is a set of disjoint M-augmenting $\ell_{\max}$-paths, its size is at most $|\mathcal{P}^*| \le p|M|$. Hence, $|M'| \le (1+p)|M|$. By the definition of M', no component

in $M' \oplus M^*$ is an M'-augmenting $\ell_{\max}$-path. Hence, if a component in $M' \oplus M^*$ has i edges from M', where $i \geq \ell_{\max} + 1$, it has at most

$$i + 1 = i \cdot \frac{i+1}{i} \leq i \cdot \frac{\ell_{\max} + 2}{\ell_{\max} + 1} = i \cdot \left(1 + \frac{1}{\ell_{\max} + 1}\right)$$

edges from M^*. This implies that $|M^*| \leq \left(1 + \frac{1}{\ell_{\max}+1}\right) \cdot |M'|$. Hence, we have

$$|M^*| \leq \left(1 + \frac{1}{\ell_{\max} + 1}\right) \cdot |M'| \leq \left(1 + \frac{1}{\ell_{\max} + 1}\right) \cdot (1 + p) \cdot |M|.$$

This completes the proof.

6.5 Upper bound on the number of active structures (Proof of Lemma 6.3)

Suppose, by contradiction, that there are more than $h|M|$ active structures at the end of a phase. Then, there are more than $h|M|$ active structures in each pass-bundle of this phase. Consider any pass-bundle τ in the phase. At the end of τ, the status of each active structure S_α falls into one of the following:

(1) S_α is on hold,

(2) α reduces the label of an arc in Overtake,

(3) α waits for the next pass-bundle because part of its structure is overtaken, or

(4) α contracts a blossom in S_α.

(5) α backtracks from the last matched arc in its active path.

Since each on-hold structure contains at least $\mathsf{limit}_h - 1 = \frac{6}{h}$ matched vertices, there are at most $2|M|/\mathsf{limit}_h = h|M|/3$ on-hold structures.[2] Therefore, at least $2h|M|/3$ free vertices satisfy one of (2)-(5).

For each α satisfying (2) or (4), a label of a matched arc is reduced by at least one. For each α satisfying (5), a free vertex backtracks from a matched arc whose label was reduced previously. Each free vertex in (2) causes at most one free vertex to satisfy (3). Therefore, in τ, there are at least $h|M|/3$ free vertices satisfying (2), (4), or (5).

In a phase, the label of each matched arc is reduced at most $\ell_{\max} + 1$ times. Since there are $2|M|$ matched arcs, the total number of label changes in the phase is at most $2(\ell_{\max} + 1)|M| \leq 4\ell_{\max}|M|$, and the total number of backtracking operations is thus at most $4\ell_{\max}|M|$. This shows that the total number of pass-bundles is less than

$$\frac{8\ell_{\max}|M|}{h|M|/3} = \frac{24\ell_{\max}}{h},$$

which contradicts the fact that the algorithm executes $\tau_{\max}(h) = \frac{24\ell_{\max}}{h}$ pass-bundles in this phase. Hence, the lemma holds.

6.6 Proof of Lemma 6.4

Consider an augmenting path P in $\mathcal{P}^*$. By Theorem 5.2, at the end of the phase, either some vertices in P are removed, or P contains a critical arc or a critical vertex. By Lemma 6.3, at the end of the phase, there are at most $h|M|$ active structure. Clearly, each active structure S_α contains exactly one critical vertex, that is, the vertex α. Therefore, each active structure

[2] As M, in the worst-case, is a 2-approximate maximum matching, there are at most $2|M|$ free vertices.

contains one critical vertex and, at most, $\ell_{\max}$ critical arcs. That is, at most $h|M| \cdot (\ell_{\max}+1)$ paths in $\mathcal{P}^*$ contain a critical arc or a critical vertex; recall that all the paths in $\mathcal{P}^*$ are vertex-disjoint.

We proceed to bound the number of paths in $\mathcal{P}^*$ containing a removed arc or vertex. Let $\mathcal{P}$ be the set of disjoint M-augmenting paths found by Alg-Phase in this phase. When an augmenting path between two free vertices α and β is found, our algorithm also removes S_α and S_β. Each removed structure contains at most Δ_h vertices. Hence, at most $|\mathcal{P}| \cdot 2\Delta_h$ paths in $\mathcal{P}^*$ contain a removed matched arc or removed free vertex. Consequently, we obtain $|\mathcal{P}^*| \leq h|M|(\ell_{\max}+1) + |\mathcal{P}| \cdot 2\Delta_h$. Assume that $|\mathcal{P}^*|$ contains more than $4h|M|\ell_{\max}$ paths, then we have

$$|\mathcal{P}| \geq \frac{|\mathcal{P}^*| - h|M|(\ell_{\max}+1)}{2\Delta_h} \geq \frac{2h\ell_{\max}}{2\Delta_h}|M| = \frac{h\ell_{\max}}{\Delta_h}|M|.$$

Our algorithm augments M by using the augmenting paths in $\mathcal{P}$ at the end of the phase. Hence, the size of M is increased by a factor of $(1+\frac{h\ell_{\max}}{\Delta_h})$ at the end of this phase. This completes the proof of the claim.

6.7 Matching size at the end of a scale (Proof of Lemma 6.5)

We prove this lemma by induction.

Base case: At the beginning of our algorithm, M is initialized as a 2-approximate maximum matching. The size of M never decreases throughout the execution. In the first scale, $h = \frac{1}{2}$. Since $(1+4h\ell_{\max}) \cdot \left(1+\frac{1}{\ell_{\max}}\right) \geq 2$, M is indeed a $(1+4h\ell_{\max}) \cdot \left(1+\frac{1}{\ell_{\max}}\right)$-approximate maximum matching at the end of h.

Inductive case: Let $h < \frac{1}{2}$ be a scale. Assume that the lemma holds for the scale $2h$. Then, M is a $(1+8h\ell_{\max}) \cdot \left(1+\frac{1}{\ell_{\max}}\right)$-approximate maximum matching at the beginning of scale h by the inductive hypothesis.

Suppose, toward a contradiction, that M is not a $(1+4h\ell_{\max}) \cdot \left(1+\frac{1}{\ell_{\max}}\right)$-approximate maximum matching at the end of scale h. Since the size of M never decreases, M is not a $(1+4h\ell_{\max}) \cdot \left(1+\frac{1}{\ell_{\max}}\right)$-approximate maximum matching at the beginning of each phase in h.

By setting $p = 4h\ell_{\max}$ in Lemma 6.2, at the beginning of each phase in h, G contains at least $4h\ell_{\max}$ M-augmenting $\ell_{\max}$-paths. By Lemma 6.4, after each phase, $|M|$ is increased by a factor of $\frac{h\ell_{\max}}{\Delta_h}$. Let $T = \frac{144}{h\epsilon}$ be the number of phases in the scale. Since $T > 4\Delta_h$, $|M|$ is increased by a factor of

$$\left(1+\frac{h\ell_{\max}}{\Delta_h}\right)^T \geq 1 + T\frac{h\ell_{\max}}{\Delta_h} > 1 + 4h\ell_{\max},$$

where the first inequality comes from the fact that $(1+x)^k \geq 1+kx$ for all positive integer k and positive real number x.[3]

Let M^* be the maximum matching. Recall that M is a $(1+8h\ell_{\max}) \cdot \left(1+\frac{1}{\ell_{\max}}\right)$-approximate maximum matching at the beginning of h. At the end of h, $|M^*|/|M|$ is at most

$$(1+8h\ell_{\max}) \cdot \left(1+\frac{1}{\ell_{\max}}\right) / (1+4h\ell_{\max}) \leq (1+4h\ell_{\max}) \cdot \left(1+\frac{1}{\ell_{\max}}\right),$$

which contradicts the assumption that M is not a $(1+4h\ell_{\max}) \cdot \left(1+\frac{1}{\ell_{\max}}\right)$-approximate maximum matching at the end of h. This completes the proof.

[3] This inequality can be proven by an induction on k. In the inductive argument, we use the fact that $(1+(k-1)x)(1+x) \geq 1+kx$.

6.8 Upper bound on structure size (Proof of Lemma 6.6)

Consider a matched arc $a' = (u', v')$ of G'. We define $T'(a')$ as follows. If a' is in some structure S_α, $T'(a')$ is the subtree of T'_α rooted at u'; otherwise, $T'(a')$ is defined as the tree consisting of a single arc a'. Define $size(a')$ as the number of vertices $v \in G_\alpha$ such that $\Omega(v) \in T'(a')$. That is, $size(a')$ is the number of vertices of G in $T'(a')$, including those vertices of G representing contracted blossoms of $T'(a')$.

To prove this lemma, we first show the following invariant:

Invariant 6.9. *At any moment of the phase, $size(a') \leq (\ell_{\max} - \ell(a'))\mathsf{limit}_h + 1$ holds for each matched arc a' in G'.*

Proving that the invariant holds. For each unvisited matched arc a', it holds that $size(a') = 1$. Hence, the invariant holds at the beginning of the first pass-bundle. Suppose, by contradiction, that the invariant is violated at some moment in the phase. For a matched arc $a' \in E(G')$ that is contained in a structure S_α, there are only four types of events that may change $size(a')$:

1. S_α overtakes a matched arc and potentially a part of another structure,
2. part of S_α is overtaken,
3. Augment is invoked to remove S_α,
4. Contract is invoked to contract a blossom in S_α.

Note that Include-Unmatched-Edges does not change $size(a')$ because it only includes blossom arcs to S_α.

Consider the first event after which a matched arc $a' \in E(G')$ has $size(a') > (\ell_{\max} - \ell(a'))\mathsf{limit}_h + 1$. Right before the event, a' must be in a structure S_α, for otherwise $size(a') = 1$ after the event. We claim that the increase of $size(a')$ must be because S_α overtakes a part of another structure. Suppose this is not true. If part of S_α is overtaken, $size(a')$ cannot increase. If Augment removes S_α, a' is also removed and thus no longer a matched arc after the event, which contradicts the definition of a'. If Contract contracts a blossom in S_α, then either a' is contracted (which is a contradiction) or $size(a')$ is unchanged. Hence, the increase of $size(a')$ is due to an overtaking operation performed by S_α.

Let $a^* \in E(G')$ be the matched arc overtaken by S_α in the event. Since $size(a')$ increases in the event, the overtaking operation must move the subtree $T'(a^*)$ under the subtree of $T'(a')$. This implies that $\ell(a^*) > \ell(a')$ before the event. Consider the moment right before the event. Since the invariant holds, we have $size(a^*) \leq (\ell_{\max} - \ell(a^*))\mathsf{limit}_h + 1$. Since S_α is not on hold and not modified, $size(a') \leq \mathsf{limit}_h - 1$. Hence, after the overtaking operation, $size(a')$ becomes at most

$$(\ell_{\max} - \ell(a^*))\mathsf{limit}_h + 1 + (\mathsf{limit}_h - 1) < (\ell_{\max} - \ell(a') - 1)\mathsf{limit}_h + \mathsf{limit}_h \leq (\ell_{\max} - \ell(a'))\mathsf{limit}_h,$$

which contradicts the definition of a'. Therefore, the invariant holds.

From the invariant to the structure size. We proceed to show that the size of each structure is at most $\ell_{\max} \cdot \mathsf{limit}_h$. Towards a contradiction, assume that S_α has a structure size greater than $\ell_{\max} \cdot \mathsf{limit}_h$, and consider the first moment when it happens.

Using a similar argument to the one for showing the invariant above, it follows that S_α overtakes a matched arc a^*. Right before the overtaking, the size of S_α is at most $\mathsf{limit}_h - 1$ because it is not on hold and not modified; furthermore, by the invariant,

$$size(a^*) \leq (\ell_{\max} - \ell(a^*))\mathsf{limit}_h + 1 \leq (\ell_{\max} - 1)\mathsf{limit}_h + 1.$$

Thus, after the overtaking operation, the size of S_α is at most

$$(\ell_{\max} - 1)\mathsf{limit}_h + 1 + (\mathsf{limit}_h - 1) \leq \ell_{\max} \cdot \mathsf{limit}_h,$$

which contradicts the definition of α. Consequently, the lemma holds.

A Proof of Lemma 4.6

In Alg-Phase, there are five procedures that can modify the structures: Augment, Contract, Overtake, Backtrack-Stuck-Structures, and Include-Unmatched-Edges. We prove that these procedures preserve all properties of a structure, which includes the disjointness, tree representation, and unique arc properties in Definition 4.1. We also show that Invariant 4.5 is preserved by the procedures.

Augment removes both structures that it modifies; Backtrack-Stuck-Structures only modifies the working vertex of each structure; Backtrack-Stuck-Structures only includes blossom arcs to structure. Therefore, they clearly preserve the invariant and all properties of a structure. In the following, we argue that Contract and Overtake also preserve these properties.

Lemma A.1. *Let (g, a, k) be the input to an invocation of Overtake. After the invocation, the updated S_α is a structure per Definition 4.1. If $a \in S_\beta$ for $\alpha \neq \beta$, after the same invocation S_β is also a structure per Definition 4.1. Moreover, if Invariant 4.5 holds before this invocation, it holds after the invocation as well.*

Proof. Let $g = (u, v)$ and $a = (v, t)$. Without loss of generality, we assume that a is in a structure $S_\beta \neq S_\alpha$. The overtaking operation moves the subtree of $\Omega(v)$ from T'_β to T'_α. We denote by T'' the subtree moved in the operation. Define p as in Section 4.5.3; that is, $p \in G$ is the vertex such that $(\Omega(p), \Omega(v))$ is the unique unmatched arc connecting $\Omega(v)$ with its parent before the invocation of Overtake.

G_α is a subgraph, Ω_α is a regular set of blossoms, and w'_α is a vertex of T'_α: Clearly, G_α is a subgraph after the invocation. Before the overtaking operation, the root of T'' was the inner vertex $\Omega(v)$. After the operation, it becomes a child of the outer vertex $\Omega(u)$. Hence, each inner vertex (resp. outer vertex) of T'' remains to be an inner vertex (resp. outer vertex) after the operation. By the observation above, each inner vertex of T'_α is a trivial blossom after the operation. Consequently, Ω_α is a regular blossom after the invocation. After the invocation of Overtake, w'_α is either set to be $\Omega(t)$ or w'_β (that is, the working vertex of S_β before the invocation). In either case, w'_α is an outer vertex of T'_α.

G_β is a subgraph, Ω_β is a regular set of blossoms, and w'_β is a vertex of T'_β: Clearly, G_β is a subgraph after the invocation. Since Overtake only removes the rooted subtree T'' from T'_β, Ω_β is a regular blossom after the invocation. In the overtaking operation, w'_β is either unchanged or updated as $\Omega(p)$. In either case, it is an outer vertex of T'_β after the invocation.

The disjointness property holds: Since Overtake only moves a set of vertices and arcs from S_α to S_β, the disjointness property holds after the invocation.

The tree representation and unique arc properties hold : The first step of the overtaking operation is to remove (p, v) from S_β and add (u, v) to S_α. By the unique arc property, after (p, v) is removed, the subtree T'' is disconnected from T'_β. By adding (u, v) to S_α, we ensures that T'_α contains an arc from the new parent of $\Omega(v)$ (that is, $\Omega(u)$) to $\Omega(v)$. In addition, since (u, v) is the only arc in G_α that is adjacent to T'', we know that no other arc $(x, y) \in G_\alpha$

corresponds to an arc between $\Omega(u)$ and $\Omega(v)$. Hence, both the tree representation and unique arc properties hold for T'_α after the invocation.

Since Overtake only moves a subtree from S_β to S_α, the two properties also hold for S_β after the invocation.

Invariant 4.5: For ease of presentation, we say an alternating tree T^* of G' is *properly labeled* if the following holds: For each root-to-leaf path $(a'_1, a'_2, \ldots, a'_k)$ on T^*, it holds that $\ell(a'_1) < \ell(a'_2) < \cdots < \ell(a'_k)$. To prove Invariant 4.5, it suffices to show that T'_α and T'_β are properly labeled after the invocation.

We first argue that T'_α is properly labeled. Since Invariant 4.5 holds before the invocation, T'' is properly labeled. Suppose that $\Omega(u)$ is the root of T'_α. Then, Overtake only attaches T'' as a child of the root. Since T'' is properly labeled, it is easy to see that T'_α is also properly labeled after the addition.

Suppose that $\Omega(u)$ is not the root of T'_α. In this case, Overtake add attaches T'' as a child of the non-root vertex $\Omega(u)$. Let a' and a'' denote, respectively, the matched arc adjacent to $\Omega(u)$ and $\Omega(v)$. Since T'' is properly labeled and its root $\Omega(v)$ only has one child (because $\Omega(v)$ is an inner vertex), it suffices to show that $\ell(a') < \ell(a'')$ after the invocation. Recall that Overtake is only invoked in the execution of Extend-Active-Path. As defined in Section 4.6, The input k to Overtake is computed as $\ell(a') + d + 1$, where $d \geq 0$ is the number of matched arcs in Q^*. Hence, it holds that $k > \ell(a')$. Since $\ell(a'')$ is updated as k, we know that $\ell(a') < \ell(a'')$ after the invocation. Consequently, T'_α is properly labeled after the invocation.

We proceed to argue that T'_β is properly labeled after the invocation. By Invariant 4.5, every subtree of T'_β is properly labeled before the invocation. In the overtaking operation, only the rooted subtree T'' is removed from S_β. Therefore, it is easy to see that T'_β remains properly labeled after the removal. Since both T'_α and T'_β are properly labeled after the invocation, Invariant 4.5 is preserved.

In the argument above, it has been assumed that a is in a structure S_β. We remark that if a is not in any structure, we may as well think of the operation as overtaking a subtree consisting of a single matched arc from a structure S_β. Hence, a similar argument can be applied for the case. □

Lemma A.2. *Let $g = (u, v)$ be the input to an invocation of Contract. After the invocation, the updated S_α is a structure per Definition 4.1. Moreover, if Invariant 4.5 holds before this invocation, it holds after the invocation as well.*

Proof. Let B denote the blossom contracted in the invocation. Let T'_1 and T'_2 denote, respectively, the tree T'_α before and after the invocation. Let Ω_1 and Ω_2 denote, respectively, the set Ω_α before and after the invocation. Let u' and v' denote, respectively $\Omega_1(u)$ and $\Omega_1(v)$.

G_α is a subgraph, Ω_α is a regular set of blossoms, and w'_α is a vertex of T'_α: Since Contract does not change the vertex set of G_α, G_α is a subgraph after the invocation. By Lemma 2.7, T'_2 is an alternating tree containing B as an outer vertex; the other inner and outer vertices of T'_2 are those of T'_1 which are not in B. Therefore, Ω_α is a regular set of blossoms after the invocation. Since B is set to be the working vertex, w'_α is an outer vertex of T'_α after the invocation.

The disjointness property holds: Since Contract does not change the vertex set of G_α, the disjointness property holds after the invocation.

The tree representation and unique arc properties hold: To prove the property, we first show that each arc (x', y') of T'_2 can be mapped to an non-blossom arc (x, y) in G_α.

Since the arcs of T_2' correspond to the arcs of T_1 that are not contracted, there is an injection f_1 from $E(T_2')$ to $E(T_1')$. The injection is formally defined as follows. Consider an arc (x', y') in T_2'. If $x', y' \neq B$, then T_1' also contains the arc (x', y'), and we map (x', y') to itself. If $x' = B$, then the parent p' of y' on T_1' was contracted in the invocation; in this case, we map (x', y') to (p', y'). Suppose that $y' = B$. Since B is an outer vertex of T_2', x' is an inner vertex in T_2'. By Lemma 2.7, x' is also an inner vertex in T_1'. Let c' be the only child of x' in T_1'. In this case, we map (x', y') to (x', c').

Since the two properties hold before the invocation, there is a bijection f_2 from the arcs in T_1' to the non-blossom arcs (with respect to Ω_1) in G_α. The composition, $f_2 \circ f_1$, of the two mappings is an injection from the arcs of T_2' to the non-blossom arcs (with respect to Ω_1) in G_α. It is not hard to verify that $f_2 \circ f_1$ maps an arc $(x', y') \in T_2'$ to the unique arc $(x, y) \in G_\alpha$ such that $\Omega_2(x) = x'$ and $\Omega_2(y) = y'$. Hence, the two properties hold after the invocation.

Invariant 4.5 holds: Consider an alternating tree. We say a matched arc (u', v') in the tree is *above* another matched arc (x', y') if v' is an ancestor of x'.

Observe that the contraction of B preserves the ancestor-descendant relationships; more precisely, if a matched arc a_1' is above a matched arc a_2' in T_2', then $f_1(a_1')$ is also above $f_1(a_2')$ in T_1'. Consider a root-to-leaf path $(a_1', a_2', \ldots, a_k')$ on T_2'. By the observation above, for each $i \leq k$, $f_1(a_i')$ is above $f_1(a_{i+1}')$ in T_1'. Since Contract does not change the label of each arc that is not contracted, we have $\ell(a_i') = \ell(f_1(a_i'))$ for all $1 \leq i \leq k$. Since Invariant 4.5 holds before the invocation, we know that $\ell(a_1') < \ell(a_2') < \cdots < \ell(a_k')$. Since the argument above holds for all root-to-leaf paths on T_2', Invariant 4.5 is preserved. □

Lemma A.1 and Lemma A.2 complete the proof of Lemma 4.6.

References

[AB21] 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.

[ABB+19] 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.

[ABD22] 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.

[ABKL23] 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.

[AG11] 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.

[AG13] 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.

[AG18] 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.

[AJJ+22] 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.

[AKL17] 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.

[AKLY16] 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.

[AKO18] 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.

[AKSY20] 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.

[ALT21] 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.

[AR20] 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.

[AS23] 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.

[Ass22] 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.

[Ass24] 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.

[AV20] Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in NC. *J. ACM*, 67(4), may 2020.

[Beh21] 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.

[Ber57] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957.

[Ber20] 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.

[BGS20] 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.

[BK23] 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.

[BKS23] 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.

[BRR23] 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.

[BRRS23] 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.

[BS15] 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.

[BST19] Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. *Algorithmica*, 81(5):1781–1799, 2019.

[BYCHGS17] 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.

[CCE+16] 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.

[CGS15] Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4), sep 2015.

[CKP+21] 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.

[CŁM+18] 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.

[CTV15] 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.

[DH03] 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.

[DP14] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1), jan 2014.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[EHL+18] 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.

[EKMS12] Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1):490–508, 2012.

[EKS09] 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.

[ELSW13] 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.

[FFK21] Salwa Faour, Marc Fuchs, and Fabian Kuhn. Distributed congest approximation of weighted vertex covers and matchings. *arXiv preprint arXiv:2111.10577*, 2021.

[FGK17] 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.

[FHM+20] 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.

[FKM+05] 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.

[FMU22] Manuela Fischer, Slobodan Mitrović, and Jara Uitto. Deterministic $(1+\varepsilon)$-approximate maximum matching with 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, page 248–260. Association for Computing Machinery, 2022.

[GGK+18] 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.

[GGM22] 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.

[GHK18] 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.

[GKK12] 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.

[GKM+19] 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.

[GKMS19] 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.

[GKMU18] 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.

[GLS+19] 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.

[GO16] Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016.

[Har19] 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.

[HK71] 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.

[HS23] 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.

[Kap13] 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.

[Kap21] 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.

[KKS14] 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.

[KMM12] 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.

[KMNT20] 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.

[Kon15] 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.

[KVV90] 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.

[LPSP15] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. *Journal of the ACM (JACM)*, 62(5):1–17, 2015.

[McG05] Andrew McGregor. Finding graph matchings in data streams. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 170–181. Springer, 2005.

[MN95] Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24:1002–1017, 1995.

[MY11] 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.

[San06] Piotr Sankowski. Weighted bipartite matching in matrix multiplication time. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 2006.

[San18] Piotr Sankowski. NC Algorithms for Weighted Planar Perfect Matching and Related Problems. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 97:1–97:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

[Tir18] 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.

[ZH23] 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.