Bipartite Matching in the Semi-Streaming Model

Sebastian Eggert, Lasse Kliemann,
Peter Munstermann, Anand Srivastav

Bericht Nr. 1101
Januar 2011

Bipartite Matching in the Semi-Streaming Model

Sebastian Eggert	see@informatik.uni-kiel.de
Lasse Kliemann	lki@informatik.uni-kiel.de
Peter Munstermann	pmu@informatik.uni-kiel.de
Anand Srivastav	asr@informatik.uni-kiel.de

Permanent ID of this document: 519a88bb-5f5a-409d-8293-13cd80a66b36

This version is dated 2011.01.08.

A preliminary version (extended abstract) appeared in the Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009), Copenhagen, Denmark, September 2009, pages 492–503 [2].

Bipartite Matching in the Semi-Streaming Model

Sebastian Eggert, Lasse Kliemann,
Peter Munstermann, Anand Srivastav

Abstract

We present the first *deterministic* $1+\epsilon$ approximation algorithm for finding a large matching in a bipartite graph in the semi-streaming model which requires only $O\left(({}^{1}/_{\epsilon})^{5}\right)$ passes over the input stream. In this model, the input graph $G=(V,E)$ is given as a stream of its edges in some arbitrary order, and storage of the algorithm is bounded by $O(n \operatorname{polylog} n)$ bits, where $n=|V|$. The only previously known arbitrarily good approximation for general graphs is achieved by the *randomized* algorithm of McGregor (2005), which uses $\Omega\left(({}^{1}/_{\epsilon})^{1/\epsilon}\right)$ passes. We show that even for bipartite graphs, McGregor's algorithm needs $\Omega\left({}^{1}/_{\epsilon}\right)^{\Omega\left({}^{1}/_{\epsilon}\right)}$ passes, thus it is necessarily exponential in the approximation parameter. The design as well as the analysis of our algorithm require the introduction of some new techniques. A novelty of our algorithm is a new deterministic assignment of matching edges to augmenting paths which is responsible for the complexity reduction, and gets rid of randomization.

We repeatedly grow an initial matching using augmenting paths up to a length of $2k+1$ for $k=\lceil {}^{2}/_{\epsilon} \rceil$. We terminate when the number of augmenting paths found in one iteration falls below a certain threshold also depending on k, that guarantees a $1+\epsilon$ approximation. The main challenge is to find those augmenting paths without requiring an excessive number of passes. In each iteration, using multiple passes, we grow a set of alternating paths in parallel, considering each edge as a possible extension as it comes along in the stream. Backtracking is used on paths that fail to grow any further. Crucial are the so-called *position limits*: when a matching edge is the ith matching edge in a path and it is then removed by backtracking, it will only be inserted into a path again at a position strictly lesser than i. This rule strikes a balance between terminating quickly on the one hand and giving the procedure enough freedom on the other hand.

Keywords: bipartite graph matching, streaming algorithms, approximation schemes, approximation algorithms.

1 Introduction

Given a bipartite graph $G = (A, B, E)$ with $n := |A \dot\cup B|$ vertices, the maximum cardinality matching problem (or *maximum matching problem*, as we will refer to it) is to find a cardinality-maximal set of edges such that no two intersecting edges are selected. An exact solution for this problem can be found in $O(n^{2.5})$ time with the algorithm of Hopcroft and Karp [4]. There is also a faster randomized algorithm by Mucha and Sankowski [6], which runs in $O(n^{\omega})$, where ω depends on the running time of the best known matrix multiplication algorithm; it is $\omega < 2.38$. For massive graphs, not only the time complexity, but also the space complexity plays an important role. In the streaming model, we assume that the entire graph cannot be stored in random access memory (RAM). Therefore the algorithm has no random access to the whole input, which is required for most of the algorithms. The set of edges has to be stored on an external device, like a disk or a tape, and is only presented 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). It sometimes is assumed that the order in which edges appear is arbitrary, but the same for each pass. However, our result works without such an assumption.

The number of passes used by the algorithm should be independent of the size of the input graph. One is interested in a good approximation of the optimum, and there the number of passes can and does depend on the approximation parameter, e.g., [3, 5].

Standard streaming models, for example a (poly)log-space streaming model, are too restrictive for graph problems. For example, it is impossible to decide if there is a path of length 2 between two given vertices x and y [3]. This shows that even easy graph problems are not solvable in this model. For solving graph problems, Muthukrishnan [7] proposed the semi-streaming model: memory of the algorithm is restricted to $O(n \operatorname{polylog} n)$, which is not enough to store the entire graph if the graph is sufficiently dense, but enough to store $\Theta(n)$ edges.

Feigenbaum et al. [3] showed that finding an exact solution of the maximum matching problem in one pass requires $\Omega(m)$ bits of space. Therefore, with the given memory restriction, we aim for an *approximation* of a maximum matching. We denote approximation factors as numbers from $[1, \infty)$. An approximation algorithm for the maximum matching problem has an *approximation factor* (or *approximation guarantee*) of $\rho \geq 1$ if it always delivers a matching M with $\rho|M| \geq \mathsf{OPT}$, where OPT denotes the cardinality of a maximum matching.

We also call such a matching a ρ approximation. It is desirable to achieve an arbitrarily good, say a $1+\epsilon$ approximation, where $\epsilon > 0$, called approximation parameter, can be arbitrarily small.[1] An algorithm for finding a $(\frac{2}{3}-\epsilon)^{-1}$ approximation of a maximum matching in bipartite graphs was given by Feigenbaum et al. [3]. This algorithm requires $O\left(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\right)$ passes over the input stream. For general graphs, McGregor in his pioneering paper [5] gave the first randomized algorithm in the semi-streaming model for finding a $1+\epsilon$ approximation of a maximum matching with a constant number of passes over the input stream, where ϵ is considered a constant. The dependence on $1/\epsilon$ is rather strong, namely $\Omega\left((1/\epsilon)^{1/\epsilon}\right)$ passes are needed. In Section 3 we show that this requirement essentially also holds for the special case of bipartite graphs. A more efficient approximation would insist on a *polynomial* dependence on relevant parameters. In the last years this has been a key issue in the qualification of efficiency, for example in research on parameterized complexity. For the maximum (bipartite) matching problem it was not known whether there is an approximation algorithm in the semi-streaming model requiring only a polynomial number of passes in $1/\epsilon$.

Our Contribution. We present in the semi-streaming model a deterministic $1+\epsilon$ approximation algorithm for the maximum matching problem in bipartite graphs with only $O\left((1/\epsilon)^5\right)$ passes over the input stream, and thus break the exponential barrier on the number of passes.

Both, McGregor's and our algorithm, build augmenting paths in a layer-wise fashion. McGregor repeatedly uses randomization to decide which matching edge may occur in which layer. As we show in Theorem 3.1, unfavorable random decisions can be responsible for the requirement of a large number of passes. The novelty of our approach is to introduce a new *dynamic and deterministic* assignment of matching edges to layers, which is responsible for the complexity reduction. We give a detailed description of our algorithm that can be easily implemented.

[1] In the framework of approximation algorithms such an algorithm is called a *PTAS* (polynomial-time approximation scheme) if it has polynomial running time for fixed ϵ, and an *efficient PTAS* if the running time is an ϵ-independent polynomial in the input size times a quantity dependent on ϵ, and an *FPTAS* (fully polynomial-time approximation scheme) if the running time is polynomial in the input size and $1/\epsilon$.

Organization. In Section 2 we introduce all relevant notions for graphs and matchings. In Section 3 we briefly describe McGregor's algorithm and show that even on bipartite input it may require a number of passes exponential in $1/\epsilon$.

Section 4 explains the theoretical basis of our approximation technique. In Section 5 we state our algorithm and give a detailed informal description. Sections 6 and 7 provide analysis for approximation guarantee, number of passes, time complexity, and space requirements. We conclude in Section 8 with some open problems. Most interesting is the challenge to design a randomized or deterministic algorithm for maximum matching in general graphs in the semi-streaming model using only polynomially many passes over the input stream.

2 Preliminaries

We denote $\mathbb{N} := \{1, 2, 3, \ldots\}$ and $[n] := \{1, \ldots, n\}$ for each $n \in \mathbb{N}$. We denote by $G = (A, B, E)$ a bipartite graph with vertex set $V = A \cup B$, where $A \cap B = \emptyset$, and edge set E. A set $M \subseteq E$ is called a *matching* if no two edges in M intersect. A matching M is an *inclusion-maximal matching* if $M \cup \{e\}$ is not a matching for any $e \in M^{\complement} = E \setminus M$. A matching is called a *maximum matching* if it has maximal cardinality among all matchings. Obviously, a maximum matching is also an inclusion-maximal matching. A vertex is called *free* if it does not appear in any edge from M, and *matched* otherwise. The free vertices of a subset $X \subseteq V$ are denoted $\mathrm{free}(X)$. We often denote free vertices with small Greek letters, e.g., $\alpha \in \mathrm{free}(A)$ for a free vertex of partition A. For a matched vertex v denote $\Gamma_M(v) := u$ where $\{v, u\} \in M$. We denote paths as sequences of vertices and edges; this notation has redundancy, but will be helpful in some places. Denote $E(P)$ the edges in a path P and $V(P)$ the vertices; we extend these notions to sets of paths, i.e., if $\mathcal{P}$ is a set of paths then $E(\mathcal{P})$ is the set of all edges that occur in at least one of those paths. The *length* of a path P, denoted $|P|$, is its number of edges. An M-*augmenting* path of length $2k+1$ is a path with $2k+1$ edges and $2k+2$ vertices which is $(M^{\complement}, M)$-alternating and starts and ends with a free vertex, formally: $(v_0, e_1, v_1, m_1, v_2, \ldots, v_{2k-1}, m_k, v_{2k}, e_{k+1}, v_{2k+1})$, where v_0 and v_{2k+1} are free vertices, $e_j = \{v_{2j-2}, v_{2j-1}\} \in M^{\complement}$ for $j \in [k+1]$ and $m_j = \{v_{2j-1}, v_{2j}\} \in M$ for $j \in [k]$. For a matching M and an M-augmenting path P, the symmetric difference $M \bigtriangleup E(P) = (M \cup E(P)) \setminus (M \cap E(P))$ is a matching of size $|M| + 1$.

Motivated by Berge's famous theorem [1], stating that a matching is a maximum matching if and only if there is no M-augmenting path, a majority of matching algorithms uses augmenting paths for finding a maximum matching. As well, our algorithm uses augmenting paths for increasing the size of the matching constructed so far. It starts with an inclusion-maximal matching, which can be found easily in one pass over the input by selecting an edge iff this edge is not incident with any already selected edge. It is a well-known fact that an inclusion-maximal matching is already a 2 approximation of a maximum matching. But for our result, this approximation factor is not good enough. On the other hand, we will not aim for eliminating *all* augmenting paths, but only up to a certain limit. We will prove two lemmas, Lemma 4.1 and 4.2, which guarantee a good approximation if there are only few augmenting paths left. We will elaborated on this in Section 4.

3 McGregor's Algorithm

We briefly describe McGregor's Algorithm [5]. It is a randomized algorithm and finds a $1 + \epsilon$ approximation of a maximum matching in general graphs in the semi-streaming model. We show that even on a *bipartite* input, it requires $\Omega\left({}^{1}/_{\epsilon}\right)^{\Omega({}^{1}/_{\epsilon})}$ passes over the input stream in order to achieve the claimed success probability of $1 - e^{-1}$, cf. [5, Theorem 2]. It is hence – on bipartite graphs – outperformed by our algorithm, which only needs a number of passes *polynomial* in ${}^{1}/_{\epsilon}$, and moreover is deterministic.

McGregor's algorithm takes a general graph G and a parameter ϵ as input. It first computes an inclusion-maximal matching M, which is done in one pass over the input stream. Define $k := \lceil \frac{1}{\epsilon} + 1 \rceil$ and $r := \Omega(k^k)$. The algorithm performs r iterations, each of them called a *phase*. A phase consists of k iterations, $i := 1, \ldots, k$, of a subroutine that tries to find a large set of pairwise vertex-disjoint M-augmenting paths, each of length $2i + 1$. Of those k sets of M-augmenting paths, one set $\mathcal{A}$ is chosen which yields the largest augmented matching $M \vartriangle E(\mathcal{A})$, and then M is replaced by that augmented matching, and the next phase starts. Each phase needs at least one pass over the input, and so we have $\Omega(k^k)$ passes, which means $\Omega\left(({}^{1}/_{\epsilon})^{{}^{1}/_{\epsilon}}\right)$ passes. For each i in each phase, a bipartite graph G' is (implicitly) constructed, that captures certain aspects of the relation of M to G and helps finding M-augmenting paths. We describe the construction of the bipartite graph. The set of free vertices is partitioned randomly into F_{left} and F_{right}, with each vertex being independently assigned to one of the sets with probability ${}^{1}/_{2}$. Then matching edges M are randomly partitioned into $M_1, \ldots, M_i$, each one independently assigned to one of these sets with probability ${}^{1}/_{i}$. Each matching edge $\{u, v\}$

is given an orientation randomly: either we have (u,v) or we have (v,u), each chosen with probability $1/2$ and independently for each edge.[2] Denote the sets of oriented edges by $\overrightarrow{M}_1, \dots, \overrightarrow{M}_i$. The random choices reflected by sets $F_{\text{left}}, F_{\text{right}}$ and $\overrightarrow{M}_1, \dots, \overrightarrow{M}_i$ fully specify the bipartite graph $G' = (V', E')$ in the following way. We interpret free vertices and directed matching edges as vertices of G', i.e., $V' = F_{\text{left}} \cup F_{\text{right}} \cup \bigcup_{j=1}^{i} \overrightarrow{M}_j$. Edges run as follows:

$$\begin{aligned}
E' = &\left\{\{x,(u,v)\};\ x \in F_{\text{left}},\ (u,v) \in \overrightarrow{M}_1,\ \{x,u\} \in E\right\} \\
&\cup \bigcup_{j=1}^{i-1} \left\{\{(s,t),(u,v)\};\ (s,t) \in \overrightarrow{M}_j,\ (u,v) \in \overrightarrow{M}_{j+1},\ \{t,u\} \in E\right\} \\
&\cup \left\{\{(u,v),y\};\ (u,v) \in \overrightarrow{M}_i,\ y \in F_{\text{right}},\ \{v,y\} \in E\right\} \ .
\end{aligned}$$

We can think of G' being in layers: on the left is F_{left}, then we have[3] $\overrightarrow{M}_1, \dots, \overrightarrow{M}_i$, and on the right we have F_{right}. All directed edges (which are vertices of G') point to the right. Edges in G' only run between neighboring layers.

Storing E' in memory is impossible in general, and so we only store V' and determine parts of E' as needed by passes over the input stream. A path in G' between the 'end' layers F_{left} and F_{right} yields an M-augmenting path of length $2i+1$ in G. McGregor's algorithm tries to find a large collection of such paths once G' is constructed; we skip those details here. In [5, Theorem 2] it is proved that with probability at least $1 - e^{-1}$, the algorithm finds a $1+\epsilon$ approximation within the r phases. We show that this number of phases is actually required even on bipartite input.

3.1 Theorem. *There are bipartite instances on which McGregor's algorithm requires $\Omega\left(1/\epsilon\right)^{\Omega(1/\epsilon)}$ passes over the input stream to succeed with probability at least $1 - e^{-1}$.*

Proof. Let $0 < \epsilon \leq 1/2$ and $t \in \{1, \dots, \lfloor \frac{1}{2\epsilon} \rfloor\}$ and $l := \lfloor 1/t\epsilon \rfloor$. Then $l \geq 2$ and $t\epsilon l \leq 1$. Let G be the path on $2l-1$ edges. Suppose the algorithm chooses an initial inclusion-maximal matching M such that there exists exactly one M-augmenting path of length $2l-1$ in G, namely G itself. Then $|M| = l-1$, and the size of a maximum matching is l. This gives

$$(1+t\epsilon)\ |M| = (1+t\epsilon)\ (l-1) = l + t\epsilon\,(l-1) - 1 < l + t\epsilon l - 1 \leq l = \mathsf{OPT} \ .$$

[2]In [5] symbols "a" and "b" are used to mark the orientation.
[3]In [5] layers are enumerated with *decreasing* indices from left to right.

Hence M is not a $1+\epsilon$ approximation, in fact it is not even a $1+t\epsilon$ approximation. The only way to find the desired approximation is to find that one M-augmenting path, which is G itself. To this end, the two free vertices must be assigned to different layers, and all matching edges must be assigned to the layers and oriented in a way that G occurs as a path in G'. This can only happen in an iteration where $i = l-1$, and then the probability that it happens is $p := \frac{1}{2} \frac{1}{(l-1)^{(l-1)}} \frac{1}{2^{(l-1)}}$. The probability that it does *not* happen within d phases is $(1-p)^d$. To push this probability below e^{-1}, we need $d \ln(1-p) \leq -1$, hence, using that for all $0 \leq x$ we have $\ln(1+x) \leq x$,

$$d \geq \frac{-1}{\ln(1-p)} = \frac{1}{\ln\left(\frac{1}{1-p}\right)} = \frac{1}{\ln\left(1+\left(\frac{1}{1-p}-1\right)\right)} \geq \frac{1}{\frac{1}{1-p}-1} \geq (l-1)^{l-1} - 1 \ .$$

This is $\Omega\left({}^{1}/_{\epsilon}\right)^{\Omega({}^{1}/_{\epsilon})}$ for $t = O(1)$. □

This lower bound still stands if we do a straightforward modification to the algorithm, allowing it to use the knowledge that it is working on a bipartite graph $G = (A, B, E)$. Consider the following modification. We put the free vertices of A into F_{left} and the free vertices of B into F_{right}. Each matching edge $\{a, b\}$ with $a \in A$ and $b \in B$ is oriented (b, a). Probability p in the above example is now $p := \frac{1}{(l-1)^{(l-1)}}$, yielding the same lower bound for d.

4 Our Approximation Technique

Like many previous approaches, we use augmenting paths to improve a given matching. However, we do not aim to eliminate all augmenting paths, which would guarantee optimality by Berge's theorem. Instead, we do two restrictions. For each $k \in \mathbb{N}$, we call a path of length at most $2k+1$ a *k-path*. Then we can express our restrictions as follows:

– We only consider augmenting k-paths, for a k that depends on the approximation parameter ϵ, namely $k = \lceil {}^{2}/_{\epsilon} \rceil$. Since a 2 approximation is already achieved by an inclusion-maximal matching, we are only interested in $\epsilon < 1$ and so $k \geq 3$.

– We do not aim at finding *all* augmenting k-paths, but we are satisfied when there exists a small enough inclusion-maximal set of k-paths.

For each $k \in \mathbb{N}$ let $\delta(k) := \frac{1}{2k(k+2)}$. The following two lemmas say that our approach indeed works to gain an approximation of $1 + {}^{2}/_{k} \leq 1 + \epsilon$, provided that we can approximate inclusion-maximal sets of augmenting k-paths. The following lemma is similar to [5, Lemma 1].

4.1 Lemma. *Let M be an inclusion-maximal matching and M^* a maximum matching. Let $D := M \bigtriangleup M^*$ and for each $i \in \mathbb{N}_0$ let $\mathcal{D}_i \subseteq 2^D$ be the set of all connected components of D that have i edges from M and $i+1$ edges from M^*. (Thus $\mathcal{D}_i$ is a set of sets of edges.) For each $i \in \mathbb{N}$ define $\alpha_i := |\mathcal{D}_i|/|M| \in [0,1]$. Assume that there exists $k \in \mathbb{N}$ such that $\sum_{i=1}^{k} \alpha_i \leq 1/k$. Then $(1 + 2/k)\,|M| \geq |M^*|$, i.e., M is a $1 + 2/k$ approximation.*

Proof. We start by upper-bounding $\sum_{i \geq k+1} \alpha_i$, i.e., the sum of those α_i not covered by the sum $\sum_{i=1}^{k} \alpha_i$. At most $|M| - \sum_{i=1}^{k} |\mathcal{D}_i|$ edges from M are available to participate in components in $\mathcal{D}_i$ for $i \geq k+1$. Each component in such a $\mathcal{D}_i$ needs at least $k+1$ edges from M. Hence

$$\begin{aligned}\sum_{i \geq k+1} \alpha_i\,|M| = \sum_{i \geq k+1} |\mathcal{D}_i| &\leq \left(|M| - \sum_{i=1}^{k} |\mathcal{D}_i|\right) \frac{1}{k+1} \\ &= |M| \left(\frac{1}{k+1} - \frac{1}{k+1} \sum_{i=1}^{k} \alpha_i\right) \quad ,\end{aligned}$$

and so

$$\sum_{i \geq k+1} \alpha_i \leq \frac{1}{k+1} - \frac{1}{k+1} \sum_{i=1}^{k} \alpha_i \quad . \tag{1}$$

Next we relate $|M^*|$ to $|M|$. We do so by counting all edges in M^* in a particular scheme. Clearly, $M^* = (M \setminus D) \mathbin{\dot\cup} (M^* \cap D)$. Consider the second part, i.e., the edges of M^* in D. The components of D can be classified as follows: those which have one edge less from M than from M^* (those are collected in $\mathcal{D}_1, \mathcal{D}_2, \ldots$; note that $\mathcal{D}_0 = \emptyset$ since M is inclusion-maximal); those which have an equal number of edges from M and M^*; and those which have one edge more from M than from M^*. The latter do not occur, since they would be M^*-augmenting paths. So if we count edges in $M^* \cap D$, we can as well first count $M \cap D$ and then one more edge for each component in one of the $\mathcal{D}_1, \mathcal{D}_2, \ldots$. It follows

$$|M^*| = |M \setminus D| + |M \cap D| + \sum_{i \in \mathbb{N}} |\mathcal{D}_i| = |M| + \sum_{i \in \mathbb{N}} |\mathcal{D}_i| = |M| + \sum_{i \in \mathbb{N}} \alpha_i |M|$$

$$
\begin{aligned}
&= |M| \left(1 + \sum_{i=1}^{k} \alpha_i + \sum_{i \geq k+1} \alpha_i \right) \\
&\leq |M| \left(1 + \sum_{i=1}^{k} \alpha_i + \frac{1}{k+1} - \frac{1}{k+1} \sum_{i=1}^{k} \alpha_i \right) && \text{by (1)} \\
&= |M| \left(1 + \frac{1}{k+1} + \frac{k}{k+1} \sum_{i=1}^{k} \alpha_i \right) \\
&\leq |M| \left(1 + \frac{1}{k+1} + \frac{k}{k+1} \cdot \frac{1}{k} \right) && \text{by assumption} \\
&= |M| \left(1 + \frac{2}{k+1} \right) \leq |M| \left(1 + \frac{2}{k} \right) \quad .
\end{aligned}
$$

□

4.2 Lemma. *Let M be an inclusion-maximal matching. Let $\mathcal{Y}$ be an inclusion-maximal set of pairwise vertex-disjoint M-augmenting k-paths such that $|\mathcal{Y}| \leq 2\,\delta(k)\,|M|$. Then M is a $1 + {}^{2}/_{k}$ approximation of a maximum matching.*

Proof. Let M^* be a maximum matching. Let $\mathcal{X}$ be a cardinality-maximal set of pairwise vertex-disjoint M-augmenting k-paths. Then $|\mathcal{Y}| \geq \frac{1}{k+2}|\mathcal{X}|$, since each path $P \in \mathcal{Y}$ can be incident with at most $k+2$ paths from $\mathcal{X}$, because P has two free vertices and at most k edges from M. It follows $|\mathcal{X}| \leq (k+2)\,|\mathcal{Y}| \leq 2\,(k+2)\,\delta(k)\,|M|$.

Let $D := M \triangle M^*$ and $\mathcal{D}_i$, α_i for each $i \in \mathbb{N}$ defined as in Lemma 4.1, and let $\mathcal{Z} := \bigcup_{i=1}^{k} \mathcal{D}_i$. Then $\mathcal{Z}$ is a set of pairwise vertex-disjoint M-augmenting k-paths, so $|\mathcal{Z}| \leq |\mathcal{X}|$. Hence

$$
\sum_{i=1}^{k} \alpha_i |M| = |\mathcal{Z}| \leq |\mathcal{X}| \leq 2\,(k+2)\,\delta(k)\,|M| = \frac{1}{k}\,|M| \quad .
$$

Lemma 4.1 now guarantees that M is a $1 + {}^{2}/_{k}$ approximation. □

5 Our Matching Algorithm

The main algorithm is shown on page 13. It implements the usual scheme of starting with an arbitrary inclusion-maximal matching M, then repeatedly computing a set of vertex-disjoint M-augmenting paths and replacing M by an augmented version. We stop when the number of M-augmenting paths found falls below the threshold of $\delta(k)\,|M|$. (Together with the stopping criterion of the subroutine described below, this will later, in the proof of Theorem 6.5, establish the "$2\delta(k)\,|M|$" bound found in the previous lemma.) We only consider M-augmenting paths of length at most $2k+1$, i.e., k-paths. These are computed by a subroutine Disjoint Paths. That subroutine is the most complex part; it is displayed on page 14. We now give an explanatory description of it; an illustration is given in Figure 1 on page 13. Input is the bipartite graph G, given as a stream, a threshold parameter δ (usually chosen as $\delta(k)$), a length parameter k, and an inclusion-maximal matching M. The task is to find many pairwise vertex-disjoint M-augmenting k-paths. The state of its main loop, starting in line 6, is (essentially) given by:

– for each matching edge $m \in M$ a *position limit* $\ell(m) \in \{1, \ldots, k+1\}$;

– the remaining vertices V';

– for each $\alpha \in \mathrm{free}(A)$ a k-path $P(\alpha) = (\alpha, \ldots)$, called a *constructed* path;

– a boolean variable $stuck_\alpha$, which is used as an indicator to decide whether $P(\alpha)$ got stuck, so it is time for backtracking.

The set of constructed paths is partitioned into *incomplete paths* $\mathcal{I}$ and *augmenting paths* $\mathcal{A}$. Set $\mathcal{I}$ consists of $(M^{\complement}, M)$-alternating paths which could not (yet) be completed to M-augmenting paths. Set $\mathcal{A}$ consist of M-augmenting paths; it is the result of Disjoint Paths when it terminates. Once an incomplete path is completed to an M-augmenting path, it is moved from $\mathcal{I}$ to $\mathcal{A}$ and never touched again until the end of this run of Disjoint Paths; its vertices are removed from V'. Denote $\mathcal{I}_{>0}$ the set of all incomplete paths that consist of at least one edge (they in fact consist of at least two edges then). Several invariants hold for the constructed paths during execution:

– Any two constructed paths are vertex-disjoint.

– All constructed paths are $(M^{\complement}, M)$-alternating.

– Their vertices are alternately from A and B.

– Incomplete paths end with a vertex from A; if they have positive length, they end with a matching edge.

We think of constructed paths starting at the left and proceeding to the right.[4] So, in each incomplete path, each vertex from B has a matching edge to its right. If a matching edge m is contained in an incomplete path P, its position limit $\ell(m)$ is exactly so that m is matching edge number $\ell(m)$ counted from the left of P, that is edge number $2\,\ell(m)$.

[4]The basic idea of growing vertex-disjoint paths in such a way is inspired by McGregor's subroutine "Find-Layer-Paths" [5, Section 2.2].

Initialization is done by setting position limits $\ell(m) := k+1$ for each $m \in M$ (which is an impossible value for an edge inside a constructed path) and constructed paths $P(\alpha) := (\alpha)$ for each $\alpha \in \text{free}(A)$. By passing over the input, the algorithm tries to extend the constructed paths. One execution of the forall loop from line 8 to line 27 constitutes one pass over the input, and we will also refer to one execution of that loop as a "pass". Execution of one iteration of the repeat loop from line 6 to the last line is referred to as one *phase*. The number of passes is equal to the number of phases (when we count the last iteration of the repeat loop, which ends in line 28, as one phase).

We first explain what happens for each edge during a pass. Let the current edge be $e = \{a, b\}$, $a \in A$, $b \in B$, between two remaining vertices. It is tested whether we can – and wish to – use e to extend an incomplete path. A path $P(\alpha^*)$ can only be extended if it has a as its endpoint, i.e., $P(\alpha^*) = (\alpha^*, \ldots, a)$. Since all paths are pairwise vertex-disjoint, there can be at most one path that fulfills this condition. If there is none, we discard e and continue the pass with the next edge. Otherwise, let $P(\alpha^*) = (\alpha^*, \ldots, a)$ be that path. We then have to distinguish between two cases.

The first case is that b is a free vertex. Then we have found an M-augmenting path, namely $P := (\alpha^*, \ldots, a, e, b)$. We set $\mathcal{A} := \mathcal{A} \cup \{P\}$, $\mathcal{I} := \mathcal{I} \setminus \{P(\alpha^*)\}$, and update the set of remaining vertices V'. Since our approximation theory is based on *inclusion-maximal* sets of M-augmenting paths, we cannot make a 'mistake' by choosing the first occurring way to complete an incomplete path to an augmenting one.

The other case is that b is a matched vertex, let $m := \{b, \Gamma_M(b)\} \in M$. Then we check whether we are below m's position limit, that is, whether the position of m in $P(\alpha^*)$ upon insertion would be lesser than $\ell(m)$. This means we check for $i < \ell(m)$, where $i = \frac{|P(\alpha^*)|}{2} + 1$. If this is the case, we set $\mathit{stuck}_{\alpha^*} := \mathsf{ff}$ because $P(\alpha^*)$ will be extended. (We set $\mathit{stuck}_{\alpha} := \mathsf{tt}$ for all α before each pass; only if a path's end is modified, it is not considered stuck. We will later explain the meaning of the stuck flags in more detail.) Now there are two more cases to consider. The first is that m is in no incomplete path (it then is in no constructed path at all). Then we set[5] $P(\alpha^*) := (\alpha^*, \ldots, a, e, b, m, \Gamma_M(b))$ and $\ell(m) := i$. That is, we append e and its matching edge m to the incomplete path and update m's position limit. The second case is that m is included in another incomplete path $P(\tilde{\alpha}) = (\tilde{\alpha}, \ldots, \tilde{a}, \tilde{e}, b, m, \Gamma_M(b), \ldots)$. The order is in fact always $(\ldots, b, m, \Gamma_M(b), \ldots)$, since $b \in B$ and $\Gamma_M(b) \in A$; moreover it is

[5]Strictly, we should write $\mathcal{I} := (\mathcal{I} \setminus (\alpha^*, \ldots, a)) \cup \{(\alpha^*, \ldots, a, e, b, m, \Gamma_M(b))\}$. However, in order to simplify notation, we treat $\mathcal{I}$ as a set of mutable objects, to which we refer by $P(\cdot)$.

always $e \neq \tilde{e}$. We now move b and its right wing in $P(\tilde{\alpha})$ to $P(\alpha^*)$, i.e., we set $P(\tilde{\alpha}) := (\tilde{\alpha}, \dots, \tilde{a})$ and $P(\alpha^*) := (\alpha^*, \dots, a, e, b, m, \Gamma_M(b), \dots)$. Since $P(\tilde{\alpha})$ changed it is important to set $\mathit{stuck}_{\tilde{\alpha}} := \mathsf{ff}$ as well because there now might be a way to extend $P(\tilde{\alpha})$ with an edge $e \in E$ which has already gone by in this pass. We set $\ell(m) := i$ and also update the position limits of any matching edges to the right of m in the subpath which we just moved, i.e., the next matching edge m' will get $\ell(m') := i + 1$, and so on.

Whenever a pass is over a test is done whether we shall carry on or finish and return $\mathcal{A}$ to the main algorithm. We finish when $|\mathcal{I}_{>0}| \leq \delta|M|$. We will later show that Disjoint Paths cannot find more then $|\mathcal{I}_{>0}|$ additional M-augmenting paths.

Right after that test, it is time for backtracking if needed, performed in lines 29 and 30. Backtracking is used for every path $P(\alpha)$ which would – in its current form – certainly not grow during the next pass. This is the case when the end of $P(\alpha)$ remained unmodified during the previous pass, which is reflected by its stuck flag stuck_α. During backtracking we remove the last two edges from each incomplete path in $\mathcal{I}_{>0}$ which has its stuck flag set, i.e., if we have $P(\alpha) = (\alpha, \dots, a, e, b, m, a')$ then we set $P(\alpha) := (\alpha, \dots, a)$ for each $\alpha \in \mathrm{free}(A)$ such that $P(\alpha) \in \mathcal{I}_{>0}$ and $\mathit{stuck}_\alpha = \mathsf{tt}$. The justification for this is that at this point, any admissible way of extending such a path has already been tried. The removed edges will never be put at that position in $P(\alpha)$ again, due to their position limits. This gives a chance to other edges to be included there instead in future passes.

We can think of all matching edges being to the right, at position $k + 1$, in the beginning. This is an impossible position to obtain inside of a constructed path. Then, matching edges move to the left. Each time a matching edge m is inserted into a constructed path, it moves at least one position to the left, accompanied by a decrement of its position limit $\ell(m)$. When it is removed by backtracking, its position limit is not changed, hence the edge is still eligible for being inserted in any position left of its last one. In particular, each position limit $\ell(m)$ is non-increasing over the steps of Disjoint Paths. If a matching edge has reached the left end, that is, its position limit has been decreased to 1, and if it then is removed by backtracking, it will not be inserted into any constructed path again. If an edge is in some constructed path, the only way by which it is made a non-member of any constructed path is that it is captured by backtracking in line 30.

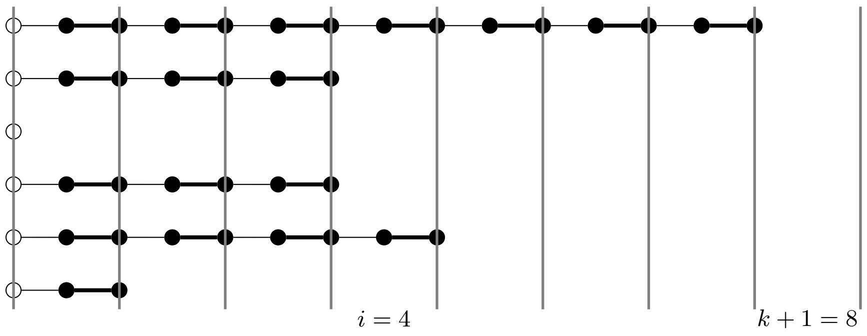

Figure 1. A state of Disjoint Paths. The non-filled vertices on the left are free vertices of A, thick edges are matching edges. There are 6 incomplete paths, 5 of them in $\mathcal{I}_{>0}$. Length parameter is $k = 7$, so we are looking for M-augmenting paths of length at most $2k + 1 = 15$. If the current edge in the stream is eligible for extending path number 2 or 4 (counted from the top), the hypothetical new position of the appended matching edge would be $i = 4$. Hence, any matching edge m to be used to extend these paths must have $4 = i < \ell(m)$. The matching edge may, however, already be part of another incomplete path, e.g., it may be one of the last three matching edges of the path at the top.

Algorithm 1: Approximate Maximum Matching

```
  Input: bipartite graph $G = (A, B, E)$, parameter $\epsilon > 0$
  Output: matching $M$
1 set $k := \lceil 2/\epsilon \rceil$ ;
2 set $M :=$ inclusion-maximal matching ;
3 repeat
4     set $c := |M|$ ;
5     set $\mathcal{A} :=$ Disjoint Paths$(G, \delta(k), k, M)$ ;
6     set $M := M \triangle E(\mathcal{A})$ ;
7 until $|\mathcal{A}| \leq \delta(k)c$;
8 return $M$ ;
```

Algorithm 2: Disjoint Paths

```
Input: bipartite $G = (A, B, E)$, $\delta > 0$, $k \in \mathbb{N}$, incl.-max. matching $M$
Output: $M$-augmenting $k$-paths $\mathcal{A}$
forall the $m \in M$ do set $\ell(m) := k + 1$;
set $\mathcal{A} := \emptyset$; set $\mathcal{I} := \emptyset$; set $V' := A \cup B$;
forall the $\alpha \in \text{free}(A)$ do
    set $P(\alpha) := (\alpha)$;
    set $\mathcal{I} := \mathcal{I} \cup \{P(\alpha)\}$;
repeat
    forall the $\alpha \in \text{free}(A)$ do set $stuck_\alpha :=$ tt;
    forall the $e = \{a, b\} \in E$ do /* assume $a \in A$, $b \in B$ */
        if $a, b \in V'$ and $\exists P(\alpha^*) = (\alpha^*, \dots, a) \in \mathcal{I}$ then
            /* position of hypothetical new matching edge: */
            let $i = \frac{|P(\alpha^*)|}{2} + 1$;
            if $b$ is a free vertex then
                set $P(\alpha^*) := (\alpha^*, \dots, a, e, b)$;
                set $\mathcal{A} := \mathcal{A} \cup \{P(\alpha^*)\}$ /* store */;
                set $\mathcal{I} := \mathcal{I} \setminus \{P(\alpha^*)\}$;
                set $V' := V' \setminus V(P(\alpha^*))$;
            else if $i < \ell(m)$ with $m = \{b, \Gamma_M(b)\}$ then
                if $b$ is in no incomplete path then
                    set $P(\alpha^*) := (\alpha^*, \dots, a, e, b, m, \Gamma_M(b))$;
                    set $stuck_{\alpha^*} :=$ ff;
                    set $\ell\ (m) := i$;
                else
                    let $P(\tilde{\alpha}) = (\tilde{\alpha}, \dots, \tilde{a}, \tilde{e}, b, m, \Gamma_M(b), \dots) \in \mathcal{I}$;
                    /* move right wing of $b$ from $P(\tilde{\alpha})$ to $P(\alpha)$:
                       */
                    set $P(\tilde{\alpha}) := (\tilde{\alpha}, \dots, \tilde{a})$;
                    set $P(\alpha^*) := (\alpha^*, \dots, a, e, b, m, \Gamma_M(b), \dots)$;
                    set $stuck_{\tilde{\alpha}} :=$ ff;
                    set $stuck_{\alpha^*} :=$ ff;
                    adjust $\ell$-values on new edges of $P(\alpha^*)$;
    if $|\mathcal{I}_{>0}| \leq \delta\,|M|$ then break and return $\mathcal{A}$ ;
    forall the $P(\alpha') \in \mathcal{I}_{>0}$ with $stuck_{\alpha'} =$ tt do
        remove last two edges from $P(\alpha')$ /* backtrack */ ;
until forever;
```

6 Analysis – Approximation

We show in two steps that Disjoint Paths computes a sufficiently good approximation of an inclusion-maximal set of M-augmenting k-paths. We first show that a tightening of the stopping condition will make it find such an inclusion-maximal set, i.e., an optimal solution to this subproblem. Then we show that with our original stopping condition, we only miss a small number of additional M-augmenting paths. Finally, we invoke Lemma 4.2 to prove the approximation guarantee. We start with a simple proposition, that will be used on two occasions.

6.1 Proposition. *If an incomplete path has zero length after a pass, then it had zero length before that pass already.*

Proof. We only have to observe that none of the operations conducted during a pass can reduce a path's length to zero. Paths are only reduced in line 23 when we go from $(\tilde{\alpha}, \ldots, \tilde{a}, \tilde{e}, b, m, \Gamma_M(b), \ldots)$ to $(\tilde{\alpha}, \ldots, \tilde{a})$. The test in line 16 ensures $2 \leq \ell(m)$, and so the new path cannot have zero length. $\square$

6.2 Lemma. *If condition "$|\mathcal{I}_{>0}| \leq \delta |M|$" in line 28 is substituted by "$|\mathcal{I}_{>0}| = 0$", then algorithm* Disjoint Paths *finds an inclusion-maximal set of pairwise vertex-disjoint M-augmenting k-paths.*

Proof. Let the stop condition in line 28 be "$|\mathcal{I}_{>0}| = 0$". Consider the point of time when this condition is reached and the algorithm terminates. Let us assume that there exists a path $(\alpha, e_1, v_1, m_1, \ldots, m_r, v_{2r}, e_{r+1}, \beta)$ in G with $1 \leq r \leq k$, $e_j \in M^{\complement}$ for all $j \in [r+1]$, $m_j \in M$ for all $j \in [r]$, $\alpha \in \mathrm{free}(A)$ and $\beta \in \mathrm{free}(B)$, which is vertex-disjoint to all paths in $\mathcal{A}$. In the following we show that this assumption leads to a contradiction.

We show by induction over j, from $j = r$ downwards to 1, that for all $1 \leq j \leq r$ it holds that $\ell(m_j) > j$. In particular, $\ell(m_1)$ is thus greater than 1 all the time, since position limit values are non-increasing. We can then conclude as follows. In the beginning of the algorithm, free vertex α is made the starting vertex of an incomplete path. Since α is in no M-augmenting path in $\mathcal{A}$ in the end, its path remains incomplete all the time. In the last phase, when the stop condition is fulfilled, there are no incomplete paths of positive length left, i.e., all incomplete paths that remain incomplete consist of only one vertex. Path $P(\alpha)$ is one of these. By Proposition 6.1 $P(\alpha)$ must have had zero length already at the beginning of the pass. However, e_1, which has α as an endpoint, and m_1 are in $G[V']$ at all times. So, α occurs as vertex a in some pass with b

such that $a, b \in V'$ and $\{b, \Gamma_M(b)\} = m_1$. Since $\ell(m_1) > 1$, the test in line 16 is true, which results in e_1 and m_1 being added behind α. They stay there until the end of the pass. This is a contradiction to that after this pass there are no incomplete paths with positive length left. So the proof is finished.

We start the induction with $j = r$. Suppose that $\ell(m_r) \leq r < k + 1$, which means that m_r was inserted into some incomplete path – and was removed again by backtracking, since in the end it is in no path from $\mathcal{I}_{>0}$ (since this set is empty) nor in any path from $\mathcal{A}$. Being removed by backtracking means that m_r has to be at the end of an incomplete and stuck path directly after a pass, say in path $P(\alpha')$. Being stuck in particular means that no changes were done to $P(\alpha')$ in the preceding pass. So $P(\alpha')$ had the form $(\alpha', \ldots, m_r, v_{2r})$ during the whole pass. But that way $P(\alpha')$ would have been completed to $(\alpha', \ldots m_r, v_{2r}, e_{r+1}, \beta)$ in the pass, because of $e_{r+1} = \{v_{2r}, \beta\} \in E$; a contradiction.

For the induction step, let $1 \leq j < r$ and suppose that $\ell(m_{j'}) > j'$ for all $j < j'$. We argue similar to before, so assume that $\ell(m_j) \leq j < k + 1$, which means that m_j was inserted into some incomplete path and removed again by backtracking. During a pass directly before a removal, m_j has to constantly be at the end of an incomplete path. Since the other matching edge m_{j+1} is in no path in $\mathcal{A}$ in the end, it is in some incomplete path or in no constructed path at all during that pass. By induction, we have $\ell(m_{j+1}) > j+1 > j \geq \ell(m_j)$. Hence, e_{j+1} and m_{j+1} are inserted behind m_j during that pass; a contradiction. □

6.3 Lemma. *Let $r \geq 0$. If algorithm* Disjoint Paths *terminates when $|\mathcal{I}_{>0}| = 0$ instead of $|\mathcal{I}_{>0}| \leq r$, then it finds at most r additional pairwise vertex-disjoint M-augmenting k-paths.*

Proof. The lemma follows when we show that an incomplete path with zero length at some point in time in line 28 will remain at zero length throughout the rest of the algorithm.

Let $P(\alpha)$ have zero length in line 28 in some phase, i.e., $P(\alpha) = (\alpha)$. By Proposition 6.1, this path already had zero length before the preceding pass. So during that pass, the algorithm was unable to grow this path. Therefore there exists no edge $e = \{\alpha, b\} \in E$ with $\ell(\{b, \Gamma_M(b)\}) \geq 2$. Since the position limits do not increase during the algorithm, there will be no such an edge $e \in E$ in the following and the path will remain $P(\alpha) = (\alpha)$ until Disjoint Paths terminates. □

6.4 Corollary. *Let $\mathcal{A}$ be the output of* Disjoint Paths(G, δ, k, M). *Then there exists an inclusion-maximal set $\mathcal{Y}$ of pairwise vertex-disjoint M-augmenting k-paths such that $|\mathcal{Y}| \leq |\mathcal{A}| + \delta\,|M|$.*

Proof. Apply Lemma 6.3 with $r := \delta\,|M|$ and choose as $\mathcal{Y}$ the set of M-augmenting paths which would be constructed by Disjoint Paths with stop condition "$|\mathcal{I}_{>0}| = 0$". $\square$

The main approximation result now follows easily.

6.5 Theorem. *Let $\epsilon > 0$. Our matching algorithm, Algorithm 1, finds a $1+\epsilon$ approximation of a maximum matching.*

Proof. When the algorithm terminates, the last set of found M-augmenting paths $\mathcal{A}$ satisfies $|\mathcal{A}| \leq \delta(k)\,|M|$, with M referring to the non-augmented matching, that is the matching for which $\mathcal{A}$ was computed. We apply Corollary 6.4 with $\delta = \delta(k)$ and receive existence of an inclusion-maximal set $\mathcal{Y}$ of pairwise vertex-disjoint M-augmenting k-paths with $|\mathcal{Y}| \leq |\mathcal{A}| + \delta(k)\,|M| \leq \delta(k)\,|M| + \delta(k)\,|M| = 2\delta(k)\,|M|$. The approximation follows from Lemma 4.2 with $k = \lceil 2/\epsilon \rceil$. $\square$

7 Analysis – Pass, Time, and Space Requirements

7.1 Lemma. *Algorithm* Disjoint Paths *needs at most $2k\delta^{-1} + 1$ passes.*

Proof. The number of passes equals the number of phases, so we need to bound the number of phases. Let Π be the set of all phases, save the last one, in which the algorithm terminates. For each $\pi \in \Pi$ let $i(\pi)$ be the number of executions of line 20, $a(\pi)$ be the number of executions of line 27, and $b(\pi)$ be the number of executions of line 30 during π. In other words, we count the events that a matching edge is inserted into a path after it was in no path at all (and its position limit is adjusted), when position limits on the migrated right wing of a path are adjusted, and when a matching edge is removed from some path by backtracking.

Each position limit $\ell(m)$, $m \in M$ is initialized with $\ell(m) = k+1$. Hence $\ell(m)$ can be decreased at most k times during the run of the algorithm. Altogether there are at most $k\,|M|$ decrements of position limits. These decrements happen in line 20 or line 27. Hence

$$\sum_{\pi\in\Pi} (i(\pi) + a(\pi)) \leq k\,|M| \ . \tag{2}$$

Fix a phase $\pi \in \Pi$. Since the algorithm does not terminate in π, we have $|\mathcal{I}_{>0}| > \delta(k)\,|M|$ in line 28. The next backtracking applies to all paths in $\mathcal{I}_{>0}$

that are marked being stuck. Since all paths are marked being stuck at the beginning of a phase, the only paths to which backtracking does not apply are those for which line 19, line 25, or line 26 were executed during π. The number of paths being marked unstuck is coupled with the numbers $i(\pi)$ and $a(\pi)$. Each execution of line 20 can lead to at most one additional path not being marked stuck. Each execution of line 27 can lead to at most two additional paths not being marked stuck. With

$$U(\pi) := \{P(\alpha) \in \mathcal{I}_{>0};\ \mathit{stuck}_\alpha = \mathsf{ff} \text{ after the pass in } \pi\}$$

it follows $|U(\pi)| \leq i(\pi) + 2a(\pi)$ and hence

$$\delta\,|M| < |\mathcal{I}_{>0}| = b(\pi) + |U(\pi)| \leq b(\pi) + i(\pi) + 2a(\pi)\ . \tag{3}$$

Now we sum over all $\pi \in \Pi$. An edge must be inserted into some path before it can be removed by backtracking. Hence $\sum_{\pi\in\Pi} b(\pi) \leq \sum_{\pi\in\Pi} i(\pi)$. It follows:

$$|\Pi|\,\delta\,|M| \overset{(3)}{\leq} \sum_{\pi\in\Pi}(b(\pi) + i(\pi) + 2a(\pi)) \leq 2\sum_{\pi\in\Pi}(i(\pi) + a(\pi)) \overset{(2)}{\leq} 2k\,|M|\ .$$

Hence $|\Pi| \leq 2k\delta^{-1}$. Since Π does not include the last phase, we have to add 1 to get a bound on the total number of phases. □

Using this lemma, we can show the claimed bound on the number of passes.

7.2 Theorem. *Algorithm 1 needs $O\left(({}^1\!/\!_\epsilon)^5\right)$ passes. The per-edge processing time is $O(|V|)$. The total processing time is $O\left(({}^1\!/\!_\epsilon)^5\ |V|\,|E|\right)$, so we have an FPTAS.*

Proof. The iteration of the loop in Algorithm 1 starts with an inclusion-maximal matching $M =: M_0$. In each iteration, the matching grows by at least $\delta(k)\,|M_0|$. Since M_0 is inclusion-maximal, in total it cannot grow by more than $|M_0|$ edges. Hence there are at most $\delta(k)^{-1} = 2k\,(k+2)$ iterations. Each iteration by Lemma 7.1 needs at most $2k\delta(k)^{-1} + 1 = 4k^2\,(k+2) + 1$ passes. Altogether, using $k \geq 3$, the number of passes is at most

$$\begin{aligned} 2k\,(k+2)\ (4k^2\,(k+2)+1) &= 8k^3\,(k+2)^2 + 2k\,(k+2) \\ \leq 8k^3\,(k+2)^2 + k^2\,(k+2) &\leq 9k^3\,(k+2)^2 = O\left(k^5\right) = O\left(({}^1\!/\!_\epsilon)^5\right)\ . \end{aligned}$$

As for the time complexity, the per-edge processing time of $O(|V|)$ is clear from the algorithm Disjoint Paths. It follows that one pass is in $O(|E|\,|V|)$.

Other operations done in one phase are clearly in $O(|V|)$. The number of phases equals the number of passes. Hence the total time spent in phases is $O\left(({}^1\!/\!_\epsilon)^5\,|V|\,|E|\right)$. One iteration of the main loop without subroutines is in $O(|V|)$. The number of passes clearly upper-bounds the number of iterations of the main loop. This yields the stated time complexity. $\square$

7.3 Theorem. *Algorithm 1 needs $O(n \log n)$ bits of space.*

Proof. Storing one edge requires storing its endpoints, which are two numbers between 1 and n, that is $O(\log n)$ bits. All the intermediate matchings M have $O(n)$ edges each. All constructed paths which have to be stored at the same time are pairwise vertex-disjoint, hence have $O(n)$ edges in total. Hence, we require storage for $O(n)$ edges, which is $O(n \log n)$ bits. In addition, the position limits need to be stored. These are $O(|M|) = O(n)$ numbers, each between 1 and $k+1$, that is $O(n \log k)$ in total. We may assume $k = O(n)$, since no improvements can be made by larger k (there are no augmenting paths on more than n vertices). Clearly, all other state information, like the set of remaining vertices V' and the position number i, can be expressed by $O(n)$ numbers between 1 and n, and hence fits into $O(n \log n)$ bits. $\square$

8 Open Questions

The most intriguing question is whether our approach can be adapted to the maximum matching problem in general graphs, while maintaining a substantially lower number of passes than McGregor's algorithm needs. It is also worth asking whether the number of passes can be reduced further for the bipartite case.

Acknowledgements

Lasse Kliemann thanks the German Research Foundation (Deutsche Forschungsgemeinschaft (DFG)) for financial support through Priority Program 1307 "Algorithm Engineering" (Grant SR7/12-1).

References

[1] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842–844, 1957. URL `http://www.pnas.org/content/43/9/842.short`.

[2] Sebastian Eggert, Lasse Kliemann, and Anand Srivastav. Bipartite graph matchings in the semi-streaming model. In *Proceedings of the 17th Annual European Symposium on Algorithms, Copenhagen, Denmark, September 2009 (ESA 2009)*, pages 492–503, 2009. URL `http://dx.doi.org/10.1007/978-3-642-04128-0_44`. Available also at `http://www.informatik.uni-kiel.de/~discopt/person/asr/publication/streaming_esa_09.pdf`. Presented also at the MADALGO Workshop on Massive Data Algorithmics, June 2009, Århus, Denmark.

[3] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming, Turku, Finland, July 2004 (ICALP 2004)*, pages 531–543, 2004. URL `http://dx.doi.org/10.1007/b99859`.

[4] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[5] Andrew McGregor. Finding graph matchings in data streams. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and Randomization and Computation, Berkeley, CA, USA, August 2005 (APPROX RANDOM 2005)*, pages 170–181, 2005. URL `http://dx.doi.org/10.1007/11538462_15`.

[6] Marcin Mucha and Piotr Sankowski. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, Rome, Italy, October 2004 (FOCS 2004)*, pages 248–255, 2004. URL `http://doi.ieeecomputersociety.org/10.1109/FOCS.2004.40`. Available also at `http://www.mimuw.edu.pl/~mucha/pub/mucha_sankowski_focs04.pdf`.

[7] S. Muthu Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):67 pages, 2005. URL `http://algo.research.googlepages.com/eight.ps`.