

Bipartite Graph Matchings in the Semi-Streaming Model

(Extended Abstract)

Sebastian Eggert, Lasse Kliemann*, and Anand Srivastav
{see,lki,asr}@informatik.uni-kiel.de

Institut für Informatik
Christian-Albrechts-Universität Kiel
Christian-Albrechts-Platz 4
24118 Kiel

Bibliographic reference:

A. Fiat and P. Sanders (Eds.): ESA 2009, LNCS 5757, pp. 492–503, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The original publication is available at www.springerlink.com.
http://dx.doi.org/10.1007/978-3-642-04128-0_44

Abstract. We present an algorithm for finding a large matching in a bipartite graph in the semi-streaming model. In this model, the input graph $G = (V, E)$ is represented as a stream of its edges in some arbitrary order, and storage of the algorithm is bounded by $O(n \text{ polylog } n)$ bits, where $n = |V|$. For $\epsilon > 0$, our algorithm finds a $\frac{1}{1+\epsilon}$ -approximation of a maximum-cardinality matching and uses $O\left(\left(\frac{1}{\epsilon}\right)^8\right)$ passes over the input stream. The only previously known algorithm with such arbitrarily good approximation – though for general graphs – required exponentially many $\Omega\left(\left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}}\right)$ passes (McGregor 2005).

Keywords: bipartite graph matching, streaming algorithms, approx. algorithms.

1 Introduction

Given a bipartite graph $G = (A, B, E)$, $n := |A \cup B|$, 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 [3]. There is also a faster randomized algorithm by Mucha and Sankowski [5], 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

* Supported by Deutsche Forschungsgemeinschaft, Priority Program 1307 *Algorithm Engineering*, Grant Sr7/12-1.

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 n 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., [2] or [4].

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 [2]. This shows that even easy graph problems are not solvable in this model. For solving graph problems, Muthukrishnan [6] proposed the semi-streaming model: memory of the algorithm is restricted to $O(n \text{ polylog } n)$, which is not enough to store the entire graph if the graph is sufficiently dense, but enough to store $O(n)$ edges.

Feigenbaum et al. [2] 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 speak of an α -approximation, or α -factor approximation, for $0 < \alpha \leq 1$, if the algorithm delivers a matching of cardinality at least αOPT , where OPT is the cardinality of a maximum matching. It is desirable to achieve an arbitrarily good, say a $\frac{1}{1+\epsilon}$ -approximation, where $\epsilon > 0$, called approximation parameter, can be arbitrarily small. An algorithm for finding a $(\frac{2}{3} - \epsilon)$ -approximation of a maximum matching in bipartite graphs was given by Feigenbaum et al. [2]. 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 [4] gave the first randomized algorithm in the semi-streaming model for finding a $\frac{1}{1+\epsilon}$ -approximation of maximum matching with a constant number of passes over the input stream, where ϵ is considered a constant. The dependence on $\frac{1}{\epsilon}$ is rather strong, namely $\Omega\left(\left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}}\right)$ passes are needed. In Sec. 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 input 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 $\frac{1}{\epsilon}$.

Our Contribution. We present in the semi-streaming model a deterministic $\frac{1}{1+\epsilon}$ -approximation algorithm for the maximum matching problem in bipartite

graphs with only $O\left(\left(\frac{1}{\epsilon}\right)^8\right)$ 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 Thm. 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 will give a detailed description of our algorithm that can be easily implemented.

2 Preliminaries

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 a *maximal matching* if $M \cup \{e\}$ is not a matching for any $e \in M^c (= E \setminus M)$. A matching is called a *maximum matching* if it has maximum cardinality among all matchings. Obviously, a maximum matching is also a 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 $\text{free}(X)$. We often denote free vertices with small Greek letters, e.g., $\alpha \in \text{free}(A)$ 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. 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^c, M) -alternating and starts and ends with a free vertex, formally: $(v_0, e_1, v_1, m_1, v_2, \dots, v_{2k-1}, m_k, v_{2k}, e_{k+1}, v_{2k+1})$, where v_0 and v_{2k+1} are free vertices, $e_j = \{v_{2j-2}, v_{2j-1}\} \in M^c$ 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 $M \triangle E(P) = (M \cup E(P)) \setminus (M \cap E(P))$ 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 a 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 a maximal matching is already a $\frac{1}{2}$ -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, Lem. 3 and 4, which guarantee a good approximation if there are only few augmenting paths left. We will elaborate on this fact in Sec. 5.

3 McGregor's Algorithm

We briefly describe McGregor's Algorithm [4]. It is a randomized algorithm and finds a $\frac{1}{1+\epsilon}$ -approximation of a maximum matching in general graphs in the semi-streaming model. We then show that even on a *bipartite* input, it requires $\Omega\left(\left(\frac{1}{\epsilon} - 1\right)^{\frac{1}{\epsilon}-1}\right)$ passes over the input stream in order to achieve the claimed success probability of $1 - e^{-1}$, cf. [4, Th. 2]. It is hence – on bipartite graphs – outperformed by our algorithm, which only needs a number of passes *polynomial* in $\frac{1}{\epsilon}$, and moreover is deterministic.

McGregor's algorithm takes a general graph G and a parameter ϵ as input. It first computes a 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, $i := 1, \dots, k$, of a sub-routine 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{P} is chosen which yields the largest augmented matching $M \triangle E(\mathcal{P})$, 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 $\Omega\left(\left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}}\right)$ 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 $\frac{1}{2}$. Then matching edges M are randomly partitioned into M_1, \dots, M_i , each one independently assigned to one of these sets with probability $\frac{1}{i}$. Each matching edge $\{u, v\}$ is given an orientation randomly: either we have (u, v) or we have (v, u) , each chosen with probability $\frac{1}{2}$ and independently for each edge.¹ Denote the sets of oriented edges by $\vec{M}_1, \dots, \vec{M}_i$. The random choices reflected by sets $F_{\text{left}}, F_{\text{right}}$ and $\vec{M}_1, \dots, \vec{M}_i$ fully specify the bipartite graph $G' = (V', E')$ in the following way. We interpret the free vertices and the directed matching edges as vertices of G' , i.e., $V' = F_{\text{left}} \cup F_{\text{right}} \cup \bigcup_{j=1}^i \vec{M}_j$. Edges run as follows:

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

We can think of G' being in layers: on the left is F_{left} , then we have² $\vec{M}_1, \dots, \vec{M}_i$, 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.

¹ In [4] symbols “ a ” and “ b ” are used to mark the orientation.

² In [4] layers are enumerated with *decreasing* indices from left to right.

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.

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

Proof. Let $l \in \mathbb{N}$ and G be the path on $2l - 1$ edges, and $\epsilon := \frac{1}{l}$. Suppose the algorithm chooses an initial 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 $(1 + \epsilon)|M| = (1 + \frac{1}{l})(l - 1) = l - 1 + \frac{l-1}{l} < l = \text{OPT}$. Hence M is not a $\frac{1}{1+\epsilon}$ -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 $-1 < x$ we have $\ln(1 + x) \leq x$,

$$\begin{aligned} d &\geq \frac{-1}{\ln(1 - p)} = \frac{1}{\ln\left(\frac{1}{1-p}\right)} = \frac{1}{\ln\left(1 + \left(\frac{1}{1-p} - 1\right)\right)} \\ &\geq \frac{1}{\frac{1}{1-p} - 1} = \frac{1}{p} - 1 \geq (l - 1)^{l-1} - 1 = \left(\frac{1}{\epsilon} - 1\right)^{\frac{1}{\epsilon}-1} - 1. \quad \square \end{aligned}$$

This lower bound still stands if we do a straightforward modification of 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 Matching Algorithm

The main algorithm is `approx-maximum-matching`, shown on page 8. It implements the usual scheme of starting with an arbitrary 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 augmenting paths found falls below the threshold of $2\delta|M|$. We only consider M -augmenting paths of length at most $2k + 1$. These are computed by sub-routine `disjoint-paths`. That sub-routine is the most complex part; it is displayed on page 8. We now

give an explanatory description of it; an illustration is given in Fig. 1 on page 9. Input is the bipartite graph G , given as a stream, a threshold parameter δ , a length parameter k , and a maximal matching M . The task is to find many pairwise vertex-disjoint M -augmenting paths of length at most $2k + 1$. The state of its main loop, starting in line 4, is (essentially) given by:

- a position number $i \in \{1, \dots, k + 1\}$;
- for each matching edge $m \in M$ a position limit $\ell(m) \in \{1, \dots, k + 1\}$;
- the remaining vertices V' ;
- for each $\alpha \in \text{free}(A)$ a path $P(\alpha) = (\alpha, \dots)$ of length at most $2k + 1$, being called a *constructed path*.

The set of constructed paths is partitioned into *incomplete paths* \mathcal{I} and *augmenting paths* \mathcal{A} . Set \mathcal{I} consists of $(M^{\mathbb{C}}, M)$ -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 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 $(M^{\mathbb{C}}, M)$ -alternating. Their vertices are alternately from A and B . Incomplete paths end with a matching edge and with a vertex from A .

We think of constructed paths starting at the left and proceeding to the right. 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 $2\ell(m)$.

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 $\alpha \in \text{free}(A)$. Then the algorithm does several passes over the input. It cycles the position i after each pass, realized by the `for` loop in line 5. We call one execution of that `for` loop a *sweep*. Hence, a sweep consists of $k + 1$ passes, with positions $i = 1, \dots, k + 1$.

We 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. Two conditions have to be met for a $P(\alpha)$ to be eligible for extension:

- (i) $P(\alpha)$ must have length $2(i - 1)$;
- (ii) $P(\alpha)$ must have a as its endpoint, i.e., $P(\alpha) = (\alpha, \dots, a)$.

Since all paths are pairwise vertex-disjoint, there can be at most one path that fulfills condition (ii). If there is none, we discard e and continue the pass with the next edge. Otherwise, let $P(\alpha^*) = (\alpha^*, \dots, a)$ be that path.

If b is a free vertex, we have found an augmenting path, namely $A := (\alpha^*, \dots, a, e, b)$. We set $\mathcal{A} := \mathcal{A} \cup \{A\}$, $\mathcal{I} := \mathcal{I} \setminus \{P(\alpha^*)\}$, and update the set of remaining vertices V' .

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, i.e., $i < \ell(m)$. If so, 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 set³ $P(\alpha^*) := (\alpha^*, \dots, a, e, b, m, \Gamma_M(b))$ 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 $P(\tilde{\alpha}) = (\tilde{\alpha}, \dots, \tilde{a}, \tilde{e}, b, m, \Gamma_M(b), \dots)$. The order is in fact always $(\dots, b, m, \Gamma_M(b), \dots)$, since $b \in B$ and $\Gamma_M(b) \in A$; moreover it is 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 $P(\alpha^*) := (\alpha^*, \dots, a, e, b, m, \Gamma_M(b), \dots)$. We set $\ell(m) := i$ and also update the position limits of any matching edges to the right of m in the sub-path which we just moved, i.e., the next matching edge m' will get $\ell(m') := i + 1$, and so on.

When the pass is over and $i = 1$, a test is done whether we shall carry on or finish and return \mathcal{A} to the main algorithm.⁴ We finish when $|\mathcal{I}_{>0}| \leq \delta|M|$. We will later show that disjoint-paths cannot find more than $2|\mathcal{I}_{>0}|$ additional augmenting paths. When the pass is over and $i = k + 1$, i.e., we just completed a pass at maximal position, we do backtracking. This means to remove the last two edges from each incomplete path in $\mathcal{I}_{>0}$, i.e., if $P(\alpha) = (\alpha, \dots, a, e, b, m, a')$ we set $P(\alpha) := (\alpha, \dots, a)$ for each $\alpha \in \text{free}(A)$ such that $P(\alpha) \in \mathcal{I}_{>0}$. The justification for this is that an incomplete path $P(\alpha) \in \mathcal{I}_{>0}$ that could not be completed in the sweep which just finished will also not be completed in any following sweep – any admissible way of extending $P(\alpha)$ 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.

Then the next pass begins, with the next position, which is either $i + 1$ if $i < k + 1$, or 1 if $i = k + 1$.

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. 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

³ Strictly, we should write $\mathcal{I} := (\mathcal{I} \setminus (\alpha^*, \dots, a)) \cup \{(\alpha^*, \dots, a, e, b, m, \Gamma_M(b))\}$. However, in order to simplify notation, we treat \mathcal{I} as a set of ‘mutable’ objects, to which we refer by $P(\cdot)$.

⁴ We could as well do this test for all i , but it suffices for $i = 1$.

line 23. During a pass with position i , incomplete paths of length at most $2(i-2)$ or length exactly $2i$ are not changed in any way. Incomplete paths with length at least $2(i+1)$ may be reduced, namely when they contain a matching edge (at some position right of i) that can be used to extend some incomplete path of length $2(i-1)$.

Algorithm 1: approx-maximum-matching

Input: bipartite graph $G = (A, B, E)$, parameter $\epsilon > 0$
Output: matching M

- 1 set $k := \lceil \frac{1}{\epsilon} \rceil$ and set $\delta := \frac{1}{8k(k+1)(k+2)}$
- 2 set $M :=$ maximal matching
- 3 **repeat**
- 4 set $c := |M|$
- 5 set $\mathcal{A} :=$ disjoint-paths(G, δ, k, M)
- 6 set $M := M \triangle E(\mathcal{A})$
- 7 **until** $|\mathcal{A}| \leq 2\delta c$
- 8 **return** M

Algorithm 2: disjoint-paths

Input: bipartite $G = (A, B, E)$, $\delta > 0, k \in \mathbb{N}$, maximal matching M
Output: M -augmenting paths \mathcal{A} of length at most $2k+1$

- 1 **forall** $m \in M$ **do** set $\ell(m) := k+1$
- 2 **forall** $\alpha \in \text{free}(A)$ **do** set $P(\alpha) := (\alpha)$; set $\mathcal{I} := \mathcal{I} \cup \{P(\alpha)\}$
- 3 set $\mathcal{A} := \emptyset$ and set $V' := A \cup B$
- 4 **repeat**
- 5 **for** $i := 1$ **to** $k+1$ **do**
- 6 **forall** $e = \{a, b\} \in E$ **do** /* assume $a \in A, b \in B$ */
- 7 **if** $a, b \in V'$ and $\exists P(\alpha^*) = (\alpha^*, \dots, a) \in \mathcal{I} : |P(\alpha^*)| = 2(i-1)$ **then**
- 8 **if** b is a free vertex **then**
- 9 set $P(\alpha^*) := (\alpha^*, \dots, a, e, b)$
- 10 set $\mathcal{A} := \mathcal{A} \cup \{P(\alpha^*)\}$ /* store */
- 11 set $\mathcal{I} := \mathcal{I} \setminus \{P(\alpha^*)\}$
- 12 set $V' := V' \setminus V(P(\alpha^*))$
- 13 **else if** $i < \ell(m)$ with $m = \{b, \Gamma_M(b)\}$ **then**
- 14 **if** b is in no incomplete path **then**
- 15 set $P(\alpha^*) := (\alpha^*, \dots, a, e, b, m, \Gamma_M(b))$
- 16 set $\ell(m) := i$
- 17 **else**
- 18 let $P(\tilde{\alpha}) = (\tilde{\alpha}, \dots, \tilde{a}, \tilde{e}, b, m, \Gamma_M(b), \dots) \in \mathcal{I}$
- 19 /* move right wing of b from $P(\tilde{\alpha})$ to $P(\alpha)$ */
- 20 set $P(\tilde{\alpha}) := (\tilde{\alpha}, \dots, \tilde{a})$
- 21 set $P(\alpha^*) := (\alpha^*, \dots, a, e, b, m, \Gamma_M(b), \dots)$
- 22 adjust ℓ -values on new edges of $P(\alpha^*)$
- 23 **if** $i = 1$ and $|\mathcal{I}_{>0}| \leq \delta|M|$ **then break and return** \mathcal{A}
- 24 **forall** $P \in \mathcal{I}_{>0}$ **do** remove last two edges from P
- 25 **until forever**

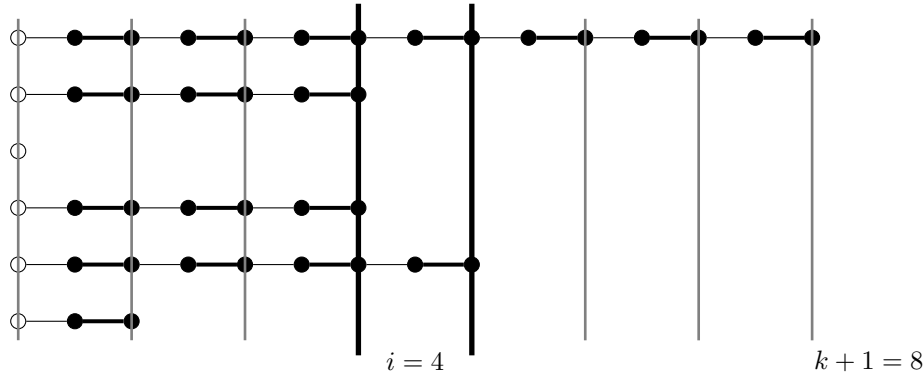


Fig. 1: A state of disjoint-paths with position $i = 4$. We call the region between the two thick vertical lines the *focus* of the sweep; the other vertical lines mark possible other positions for the sweep. There are 6 incomplete paths, 5 of them in $\mathcal{I}_{>0}$. The non-filled vertices on the left are free vertices of A . Thick edges are matching edges. Length parameter is $k = 7$, so we are looking for augmenting paths of length at most $2k + 1 = 15$, and the maximal position for the sweep is $k + 1 = 8$. Edges may only be inserted in the focus of the sweep, so for the current position $i = 4$, only paths number 2 and 4 can be extended. Edges may only be removed from paths provided that they are positioned outside of the focus and to its right. So, we can only remove edges from path number 1 here. In particular, any edges inserted in the focus will not be removed during the rest of this sweep (unless their path is completed).

5 Analysis – Approximation

The key idea is to consider a relaxation of the semi-streaming model restrictions in the following way: we allow more than a constant number of passes. This gives us the possibility to find a *maximal* set of vertex-disjoint M -augmenting paths of length at most $2k + 1$.

Lemma 1. *If condition “ $|\mathcal{I}_{>0}| \leq \delta|M$ ” in line 22 is substituted by “ $|\mathcal{I}_{>0}| = 0$,” then algorithm disjoint-paths finds a maximal set of pairwise vertex-disjoint M -augmenting paths of length at most $2k + 1$.*

Proof. Let the stop condition in line 22 be “ $i = 1$ and $|\mathcal{I}_{>0}| = 0$.” Consider the point of time when this condition is reached and the algorithm terminates. Let us assume that there exists a path $(\alpha, e_1, v_1, m_1, \dots, m_r, v_{2r}, e_{r+1}, \beta)$ in G with $1 \leq r \leq k$, $e_j \in M^c$ for all $j \in [r + 1]$, $m_j \in M$ for all $j \in [r]$, $\alpha \in \text{free}(A)$ and $\beta \in \text{free}(B)$, 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. 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 augmenting path in \mathcal{A} , its path remains incomplete until the end. In the last sweep, the stop

condition is fulfilled, meaning that there are no incomplete paths of positive length left at the end of the pass with $i = 1$. Due to the test in line 13, once a matching edge is added to some incomplete path at position 1 during a sweep S , or if it is there at the start of a sweep S , it remains there for the rest of S , unless its path is completed and stored in \mathcal{A} . In the given situation, this means that at the start of the last pass (with $i = 1$), the incomplete path containing α has length 0, or otherwise it would still be in $\mathcal{I}_{>0}$ when the test in line 22 is made. Moreover, e_1 , which has α as an endpoint, and m_1 are still in $G[V']$. So, α occurs as vertex a in some pass with b such that $\{b, \Gamma_M(b)\} = m_1$. Since, by induction, $\ell(m_1) > 1$, the test in line 13 is true in this pass, resulting in e_1 and m_1 being added behind α . This is a contradiction and the proof is finished.

We start the induction. Consider case $j = r$, i.e., the induction base. If $\ell(m_j) = k + 1$, then we are done. Assume hence for contradiction that $\ell(m_j) \leq k$, which means that m_j was inserted into some constructed path – and removed again by backtracking, since in the end it is neither in any path of \mathcal{A} nor in any path of \mathcal{I} (since $\mathcal{I}_{>0}$ is empty). Let S be a sweep directly after which m_j is removed from its path the last time, i.e., made a non-member of any path. The removal can only happen by backtracking, line 23. This means that m_j has to be at the end of its path after the last pass of S . We consider all possible developments that can lead to this, and show that each induces a contradiction. The reader is encouraged to frequently look at Fig. 1 while reading the proof. We need three general claims. The first follows directly from the test in line 13:

Let $m \in M$ be in some incomplete path. Position limit $\ell(m)$ does not change in a pass with position i for which $\ell(m) \leq i$. This holds in particular for such a pass in which m is inserted into an incomplete path. (+)

The second claim is a direct consequence of where the algorithm inserts edges into incomplete paths:

For any matching edge m : if during a pass with position i , position limit $\ell(m)$ is reduced at all, it is reduced to no less than i . (++)

The third claim in particular implies that no edges may be added behind m_j in sweep S :

Once some edges are added behind m_j (which would happen in line 15 or line 20) in some sweep, edge m_j will never be at the end of some incomplete path again during the same sweep. (*)

Proof of ().* Let e and $m \in M$ be edges added behind m_j in a pass with position i_0 . Then $\ell(m) = i_0$ for the rest of this sweep by (+). These edges could, *during* this sweep, only be removed from this incomplete path in line 19. For this to happen in a pass with position $i \geq i_0$, it is required that $\ell(m) > i$. This is impossible since $i \geq i_0 = \ell(m)$. This proves (*).

Recall that we consider a sweep S directly after which m_j is removed. By (*), the following three cases are left to consider:

- (1) m_j is at the end of an incomplete path when S starts and it remains at the end of an incomplete path until the end of S .
- (2) $m_j = \{b_j, a_j\}$ is somewhere in the middle of an incomplete path $P(\tilde{\alpha})$ when S starts, say $(\tilde{\alpha}, \dots, b_j, m_j, a_j, e', b', m', a', \dots)$, and then (e', b', m', a', \dots) is removed from $P(\tilde{\alpha})$ in line 19, and henceforth m_j remains at the end of $P(\tilde{\alpha})$ until the end of S .
- (3) m_j is inserted into an incomplete path *during* S and remains at its end until the end of S .

We can bring all three cases to a contradiction by the following claim:

m_j cannot be at the end of an incomplete path at the start of a pass (#)
with position $\ell(m_j) + 1$ in sweep S .

Proof of (#). Assume m_j is at the end of an incomplete path $P(\alpha^*)$ at the start of a pass with position $i = \ell(m_j) + 1$. Then $|P(\alpha^*)| = 2\ell(m_j) = 2(i - 1)$, and so it is eligible for being extended. However, m_j remains at the end of this path during this pass because of (*) (so the path is in fact not extended), and also m_j 's position limit $\ell(m_j)$ does not change because of (+). This implies a contradiction immediately: when we reach e_{j+1} in the input stream, the algorithm *will* use it to complete m_j 's incomplete path, with β as endpoint. This is a contradiction to that m_j does not occur in any augmenting path in \mathcal{A} . We have proved (#).

This immediately rules out case (1). Case (2) is also ruled out: the removal of (e', b', m', a', \dots) could only happen in a pass with position $i_0 < \ell(m') = \ell(m_j) + 1$, and then m_j would be at the end of an incomplete path at the start of all later passes, i.e., passes with positions $i_0 + 1, i_0 + 2, \dots$. By (++) this is true in particular for that pass with position $\ell(m_j) + 1$, with $\ell(m_j)$ taken at the time when the pass occurs; a contradiction by (#).

The final possibility to consider is case (3). If m_j is inserted into its path during S in a pass with position i_0 , henceforward we have $\ell(m_j) = i_0$ for the rest of S by (+), and so in particular the next pass has position $i_0 + 1 = \ell(m_j) + 1$. This is a contradiction by (#).

We have completed the induction base. The induction step works similar: claim (#) holds with a different proof, which uses the induction hypothesis, and the treatment of the three cases works the same. □

We can show that with the relaxation of the semi-streaming model, the algorithm would find only a small number of additional M -augmenting paths.

Lemma 2. *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 $2r$ additional pairwise vertex-disjoint M -augmenting paths of length at most $2k + 1$.*

We apply this lemma with $r := \delta|M|$ and consider the set \mathcal{Y} of M -augmenting paths which would be constructed by *disjoint-paths* with stop condition “ $|\mathcal{I}_{>0}| = 0$ ”.

Corollary 1. *Let \mathcal{A} be the output of *disjoint-paths*(G, δ, k, M). Then there exists a maximal set \mathcal{Y} of pairwise vertex-disjoint M -augmenting paths of length at most $2k + 1$ such that $|\mathcal{Y}| \leq |\mathcal{A}| + 2\delta|M|$.*

The following lemma, similar to [4, Lem. 1], is used in the proof of Lem. 4, which allows us to deduce the main approximation result in Thm. 2 from Cor. 1.

Lemma 3. *Let M be a maximal and M^* a maximum matching. Let $k \in \mathbb{N}$. Choose $\alpha_i \in [0, 1]$ for each $i \geq 1$ such that $\alpha_i |M|$ is the number of connected components in $M \triangle M^*$ with i edges from M and $i + 1$ edges from M^* . If $\sum_{i=1}^k \alpha_i \leq \frac{1}{2^{k(k+1)}}$ then $(1 + \frac{1}{k})|M| \geq |M^*|$.*

Lemma 4. *Let M be a maximal and M^* a maximum matching. Let \mathcal{Y} be a maximal set of pairwise vertex-disjoint M -augmenting paths of length at most $2k + 1$ such that $|\mathcal{Y}| \leq \frac{1}{2^{k(k+1)(k+2)}}|M|$. Then $(1 + \frac{1}{k})|M| \geq |M^*|$.*

Theorem 2. *Let M^* be a maximum matching and $\epsilon > 0$. Algorithm *approx-maximum-matching* finds a matching M with $(1 + \epsilon)|M| \geq |M^*|$.*

6 Analysis – Running Time and Space Requirements

Lemma 5. *Algorithm *disjoint-paths* needs at most $(k + 1)^{2\frac{1}{\delta}}$ passes.*

Proof (Idea). It is sufficient to bound the number of sweeps. For each sweep (except the last one), we can guarantee a minimum number of edges being removed from incomplete paths. This implies a certain demand for edges being *inserted* into those paths. Each insertion, however, decreases a position limit, and this can only happen a limited number of times. \square

Using this lemma, we can show the claimed bound on the number of passes. Since we only have to store $O(n)$ edges at a time, we also get the necessary bound on the space requirement.

Theorem 3. *Algorithm *approx-maximum-matching* needs $O\left(\left(\frac{1}{\epsilon}\right)^8\right)$ passes and $O(n \log n)$ bits of space.*

7 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.

References

1. Berge, C.: Two theorems in graph theory. Proceedings of the National Academy of Sciences of the United States of America **43**(9) (1957) 842–844

2. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: On graph problems in a semi-streaming model. In Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D., eds.: Automata, Languages and Programming: 31st International Colloquium (ICALP 2004). Volume 3142 of Lecture Notes in Computer Science., Turku, Finland, Springer (July 2004) 531–543
3. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing* **2**(4) (1973) 225–231
4. McGregor, A.: Finding graph matchings in data streams. In Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L., eds.: 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2005) and 9th International Workshop on Randomization and Computation (RANDOM 2005). Volume 3624 of Lecture Notes in Computer Science., Berkeley, CA, USA, Springer (August 2005) 170–181
5. Mucha, M., Sankowski, P.: Maximum matchings via Gaussian elimination. In: 45th Annual Symposium on Foundations of Computer Science (FOCS 2004), Rome, Italy (October 2004) 248–255
6. Muthukrishnan, S.M.: Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science* **1**(2) (2005) 67 pages