On the Assadi–Liu–Tarjan Auction Algorithm for Bipartite Matching: Simplification, Alternative Analysis, and Hard Instance

Christian Konrad
School of Computer Science, University of Bristol, UK

Kheeran K. Naidu
Qworky Research, London, UK

Archie Walton
School of Computer Science, University of Bristol, UK

Eric Wang
University of California, San Diego, CA, USA

Abstract

Assadi, Liu, and Tarjan [SOSA'21] gave an auction algorithm that outputs a $(1-\epsilon)$-approximation to Maximum Matching in bipartite graphs. Their algorithm computes a sequence of $O(\frac{1}{\epsilon^2})$ maximal matchings in subgraphs of the input graph and can be implemented in the multi-pass streaming setting with $O(\frac{1}{\epsilon^2})$ passes in a straightforward manner, which constitutes the state-of-the-art pass/approximation trade-off result in the multi-pass streaming setting. Their analysis uses tools from combinatorial auctions and, at its heart, relies on a clever potential function argument. Their proof, however, provides only limited insight into the inner workings of the algorithm.

In this paper, we revisit the ALT-algorithm and present the following contributions:

1. **Simplification.** The ALT-algorithm is built upon a *freezing mechanism* where vertices on one side of the bipartition that have already been rematched $\Theta(\frac{1}{\epsilon})$ times over the course of the algorithm remain matched to their current partner forever. We show that this mechanism is in fact unnecessary, i.e., no special treatment of such vertices is needed. With the freezing mechanism removed, the parameter ϵ now solely determines the total number of iterations/maximal matching computations, which provides the option of adaptively refining ϵ as the algorithm runs.
2. **Alternative Analysis.** We give an alternative analysis of the algorithm that is based on augmenting paths. Beyond the auction-perspective of the algorithm as established by Assadi et al., our analysis allows for a reinterpretation as one that follows the traditional approach of searching for and eliminating augmenting paths. Our analysis also copes with the removal of the freezing mechanism in a natural way, whereas the analysis of Assadi et al. strictly depends on its use.
3. **Hard Instance.** We provide the first hard instance on which the algorithm requires $\Omega(\frac{1}{\epsilon^2})$ iterations/maximal matching computations. The instance is a simple path graph, where we exhibit a cyclic behaviour that prevents fast progress. Hard instances for this algorithm therefore do not necessarily have to be dense.

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

Keywords and phrases Maximum Bipartite Matching, Augmenting Paths, Auction Algorithm, Approximation

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.120

Funding *Archie Walton*: Supported by an EPSRC DTP studentship.

1 Introduction

A *matching* in a graph is a subset of vertex-disjoint edges. A matching is *maximal* if it cannot be enlarged by adding an edge outside of the matching to it, and a matching is *maximum* if it is of largest possible size. Then, the Maximum Matching problem (MM) consists of computing a maximum matching, and, for any $0 < \alpha \leq 1$, an α-approximation algorithm to MM is an algorithm that outputs a matching of size at least α times the size of a maximum matching.

Motivated by the fact that, in various restricted computational models, computing a maximal matching is an easy task, there has been growing interest in algorithms for MM approximation that are solely based on computing a sequence of maximal matchings in subgraphs of the input graph so that the union of these maximal matchings contains a non-trivial approximation to MM [9, 16]. For example, in the *semi-streaming model* of computation [11], where an algorithm performs multiple passes over the edges of an n-vertex input graph while maintaining a memory of size $O(n \operatorname{poly} \log n)$, a single pass is sufficient for computing a maximal matching. In the $O(n)$-memory *massively parallel computation model* (MPC model) [12], where the input graph is partitioned between multiple machines with memory $O(n)$ each and the machines exchange messages in synchronous rounds, a maximal matching can be computed in $O(\log \log n)$ rounds [7]. Hence, any algorithm that finds an approximation to MM solely by computing p maximal matchings in subgraphs of the input graph can therefore give algorithms in restricted computational models with only a factor p overhead over the complexity of computing a maximal matching[1].

Assadi, Liu, and Tarjan [6] gave such an algorithm for bipartite graphs (henceforth the ALT-algorithm). For any small $\epsilon > 0$, the ALT-algorithm produces a $(1-\epsilon)$-approximation to MM and is solely based on the sequential computation of $O(\frac{1}{\epsilon^2})$ maximal matchings. When implemented in the semi-streaming model, the algorithm uses $O(\frac{1}{\epsilon^2})$ passes and constitutes the state-of-the-art bipartite matching algorithm in this model. We, however, note that streaming algorithms with pass complexity $O(\frac{1}{\epsilon} \cdot \log n)$ are also known [2, 5, 3], which run faster when $\epsilon = o(\frac{1}{\log n})$.

The ALT-algorithm as presented in [6] is phrased using terminology from combinatorial auctions, and their analysis also uses ideas from this area. We observe that this is very different to previous matching algorithms in the streaming model, which are mostly based on finding and eliminating augmenting paths [16, 4, 15, 13, 10, 14, 17], on linear programming [3, 1], and, more recently, on the edge-degree constrained subgraph technique [8, 4]. Regarding $(1-\epsilon)$-approximation multi-pass algorithms for bipartite graphs, Eggert et al. [10] previously gave an augmenting paths-based algorithm that runs in $O(\frac{1}{\epsilon^5})$ passes, and Ahn and Guha [1] gave a linear programming based algorithm that runs in $O(\frac{1}{\epsilon^2} \log \log(\frac{1}{\epsilon}))$ passes.

Our Results. The purpose of this paper is to provide a deeper understanding of the ALT-algorithm. We present the following contributions:

1. **Simplification.** The ALT-algorithm operates on a bipartite graph $G = (A, B, E)$ and maintains an initially empty matching M that, at the end of the algorithm, constitutes a $(1-\epsilon)$-approximation. The algorithm is round-based, and, in each round of the algorithm, a matching M' between the currently unmatched A-vertices and their neighbors is computed in a carefully chosen *demand subgraph* (see Section 2 for precise definitions).

[1] As long as the subgraphs on which the maximal matching algorithm is to be executed can be specified/-computed in the respective computational model.

The matching M is then updated by setting

$$M \leftarrow M' \cup \{e \in M \ : \ e \text{ not incident to an edge in } M' \} .$$

One property of proceeding in this way is that, over the course of the algorithm, B-vertices change their partners many times. The ALT-algorithm is built upon a *freezing mechanism* that effectively freezes a B-vertex to its current match once it has been rematched $\Theta(\frac{1}{\epsilon})$ times. This somewhat artificial rule is crucial for Assadi et al.'s analysis to work.
We show that the freezing mechanism is in fact unnecessary and no special treatment of vertices that have changed their partners often is required. Removing the freezing mechanism simplifies the algorithm and has the added benefit that the parameter ϵ now solely determines the number of iterations/maximal matching computations of the algorithm. This implies that adaptive refinements of ϵ while the algorithm runs are now possible.

2. **Alternative Analysis.** While our analysis reuses some of the facts established by Assadi et al., we employ an entirely different argument at its core that deals with the removal of the freezing mechanism in a natural way. We establish a connection between the number of times $c(b)$ that a B-vertex b has changed partners and the length of any augmenting path that runs through b. More precisely, if there exists an augmenting path starting at an unmatched vertex $a \in A$ that goes through the vertex b then this path is of length at least $2c(b) + 1$. Hence, the lengths of augmenting paths increase over the course of the algorithm whenever a B-vertex is rematched. Since longer augmenting paths imply larger matchings, this connection immediately explains how the algorithm makes progress.
3. **Hard Instance.** We give the first hard instance on which the ALT-algorithm requires $\Omega(\frac{1}{\epsilon^2})$ iterations. Our instance is the path graph, where we demonstrate a cyclic behaviour that prevents fast progress. In this instance, the algorithm repeatedly cycles through very similar matchings many times. Interestingly, most edges of the path are inserted and removed from the current matching $\Theta(\frac{1}{\epsilon})$ times, a key property that demonstrates that hard instances for this algorithm do not necessarily have to be dense.

Outline. In Section 2, we discuss the ALT-algorithm, and we give an augmenting-paths based analysis of the algorithm in Section 3. Then, in Section 4, we show that the algorithm may require $\Omega(\frac{1}{\epsilon^2})$ passes on a path graph. Finally, we conclude in Section 5.

2 Algorithm

The ALT-algorithm is depicted in Algorithm 1.

The listing differs in two aspects from its original version in [6]. First, in our version, the quantity $c(b)$, for every $b \in B$, when increased is increased by one, while this quantity is increased by ϵ in [6]. The reason for scaling these values is that our interpretation of $c(b)$ is different as in [6] – see further below for a discussion.

Second, and more importantly, the original version in [6] *freezes* B-vertices once they have been rematched $\Theta(\frac{1}{\epsilon})$ times. This translates into adding the constraint "and $c(b) \leq \frac{3}{\epsilon}$" to the end of Line 4. As previously discussed, one of our key contributions is the insight that the freezing mechanism is unnecessary and can be removed.

We note that removing the freezing mechanism has a significant benefit. In our version of the ALT-algorithm, the parameter ϵ solely determines the number of iterations, while, in its original version by Assadi et al., the parameter ϵ also determines the moment at which vertices are frozen. Hence, in our version, the execution of the algorithm for a parameter

Algorithm 1 Assadi-Liu-Tarjan Algorithm: $(1-\epsilon)$-approximation for MM on bipartite graphs.

Input: Bipartite input graph $G = (A, B, E)$, parameter $\epsilon > 0$

1: Let $M = \varnothing$ and, for all $b \in B$, let $c(b) = 0$
2: **for each** round $r = 1, 2, \ldots, 49/\epsilon^2$ **do**
3: For every $a \in A \setminus A(M)$, let $c_{\min}(a) := \min\{c(b) : b \in \Gamma(a)\}$
4: Let $H_r \subseteq G$ be the subgraph spanned by all edges $(a, b) \in (A \setminus A(M)) \times B$ such that $c(b) = c_{\min}(a)$
5: Compute a maximal matching N_r in H_r
6: $M \leftarrow \{ab \in M \ : \ b \notin B(N_r)\} \cup N_r$
7: Increment the count $c(b)$ of each $b \in B(N_r)$ by 1
8: **end for**
9: **return** M

$\epsilon' > \epsilon$ is a prefix of the execution of the algorithm with parameter ϵ, which means that the precision parameter ϵ can be updated while the algorithm is running. This is not possible in its original version.

The algorithm works as follows. Given the bipartite input graph $G = (A, B, E)$, the algorithm maintains a matching M. The matching M evolves over the course of the algorithm but maintains the invariant that if a vertex $b \in B$ is matched in M at some moment then the vertex b remains matched throughout the algorithm. We note that b may change mates along the way, however, it will never become unmatched again. The matching M thus cannot decrease in size.

The algorithm maintains, for each $b \in B$, a counter $c(b)$, which counts the number of times the vertex has changes its mate in M. In its original version in [6], once $c(b)$ reaches the count $\Theta(\frac{1}{\epsilon})$ then b ceases to swap partners and remains matched to the current partner until the end of the algorithm. In our version, this step is unnecessary and b continues to swap partners until the algorithm terminates.

In each iteration of the algorithm, the currently unmatched A-vertices compete for a new mate, which often requires unmatching a currently matched A-vertex. To this end, each unmatched A-vertex $a \in A \setminus A(M)$ first computes the quantity $c_{\min}(a) = \min\{c(b) \ : \ b \in \Gamma(a)\}$, which is the smallest number of times that any neighbour of a has changed mates so far. Then, a *demand subgraph* H_r is established, where, for each unmatched $a \in A \setminus A(M)$, every edge ab is included such that $c(b) = c_{\min}(a)$. Next, a maximal matching N_r in H_r is computed and the current matching M is updated so that every edge of M that is incident to a matched B-vertex in N_r is replaced with the corresponding edge in N_r. Finally, the count $c(b)$ of every vertex b that has received a new mate is increased by 1 to reflect the previous step.

Phrased in the language of auctions as in [6], the A-vertices constitute the set of bidders and the B-vertices constitute the set of items. Then, for an item $b \in B$, the quantity $c(b)$ is the current price of an item, which increases by 1 every time the item is rematched to a different bidder. For a bidder $a \in A$, the quantity $c_{\min}(a)$ corresponds to the price of the cheapest item in the neighbourhood of a, i.e., items that the bidder a could acquire. The demand subgraph H_r thus only consists of edges between unmatched bidders and the cheapest items available to them.

Basic Properties. We highlight some basic facts about the algorithm that are easy to verify and that were also used by Assadi et al. [6]:

P1: Once a B-vertex is matched, it will never become unmatched again. The matching M therefore never decreases in size.

P2: For any vertices $a \in A$ and $b \in B$, the quantities $c_{\min}(a)$ and $c(b)$, respectively, never decrease.

P3: Consider an unmatched A-vertex $a \in A \setminus A(M)$ at the start of any iteration, and suppose that it remains unmatched by the end of the iteration. Then, the value $c_{\min}(a)$ increases by one in this iteration.

Furthermore, we say that an unmatched A-vertex a is *saturated* if $c_{\min}(a) \geq 6/\epsilon$. An A-vertex that is not saturated is *unsaturated*.

3 Analysis

Let OPT be an arbitrary but fixed maximum matching in the input graph $G = (A, B, E)$. We write $A_{\text{OPT}} := A(OPT)$ to denote the A-vertices incident to OPT.

Assadi et al.'s analysis has two main components. First, they argue that if a specific configuration of the variables is achieved then the matching M must be of size at least $(1-\epsilon)|OPT|$ (*approximation part*). Second, they argue that such a configuration must necessarily be reached at some moment during the execution (*progress part*).

In our analysis below, we introduce augmenting paths-based arguments. These arguments are used to both give an entirely different proof of the approximation part, and to augment Assadi et al.'s arguments for the progress part in order to show that the algorithm equally works without the freezing mechanism.

Assadi et al.'s key lemma regarding the approximation factor states the following. They define the notion of ϵ-*happiness* of a bidder, which translates to saying that an A-vertex is ϵ-happy either if it is matched or if its $c_{\min}(a)$ value has reached $\Theta(1/\epsilon)$ – the latter of which corresponds to our notion of an unmatched A-vertex being saturated. The interpretation of this notion is that if a bidder is matched then it is clearly happy, while if it is not matched but all items in their neighbourhood have become quite expensive then the unmatched A-vertex is still *relatively* happy, or ϵ-happy, as they would need to spend a large amount of money to purchase the item. They prove that if in some iteration of the algorithm there are $(1-\frac{1}{2}\epsilon)|OPT|$ ϵ-happy bidders then the resulting matching is a $(1-\epsilon)$-approximation. Their proof uses a potential function argument, summing the "utilities" of the ϵ-happy bidders and giving both upper and lower bounds on this quantity.

Our key insight into an augmenting paths-based analysis of the approximation factor is as follows. Consider an unmatched vertex $a \in A_{\text{OPT}} \setminus A(M)$ with its min-cost $c_{\min}(a)$. We observe that the quantity $c_{\min}(a)$ directly translates into a lower bound on the length of an augmenting path in $M \oplus OPT = (M \setminus OPT) \cup (OPT \setminus M)$ that starts at vertex a, i.e., we prove that the length of the augmenting path starting at a is of length at least $2c_{\min}(a)+1$. Hence, if an unmatched vertex has a large $c_{\min}$ value then the damage incurred by the fact that a is not matched is small since the augmenting path is long. Recall that, in an augmenting path of length $2\ell+1$, we have only one unmatched A-vertex and ℓ matched A-vertices. Thus, locally, the approximation factor on this path is $\ell/(\ell+1)$. Hence, if we can ensure that $c_{\min}(a) = \Omega(\frac{1}{\epsilon})$ holds for most unmatched A_{OPT}-vertices then the current matching M is a $(1-\epsilon)$-approximation. We will see that this follows from the progress part of the analysis.

Regarding the progress part of the analysis, we use similar argument as Assadi et al. However, we include the insight that the number of unmatched A_{OPT}-vertices with large min-cost value is bounded, which is a property that follows from our augmenting paths-based

analysis. In more detail, we show that there are at most $|OPT|/k$ unmatched A_{OPT}-vertices with min-cost value $\Omega(k)$, for integers $k \geq 1$. This property is crucial to show that the freezing mechanism is not needed.

We will now give the proof of our key lemma:

▶ **Lemma 1.** *Consider the state of the algorithm at the beginning of any iteration. For any $k \geq 0$, let $\mathcal{P}$ be an augmenting path of length $2k+1$ in $M \oplus OPT$, and let $a_0 \in A_{OPT}$ be the unmatched A-vertex in $\mathcal{P}$. Then:*

$$k \geq c_{min}(a_0) \ .$$

Furthermore, the set $M \oplus OPT$ contains at most $\frac{1}{k+1} \cdot |OPT|$ augmenting paths that start at A_{OPT}-vertices a with $c_{min}(a) \geq k$.

Proof. Let $\mathcal{P} = a_0 b_1 a_1 b_2 a_2 \dots b_{k-1} a_{k-1} b_k \in M \oplus OPT$ be an augmenting path of length $2k+1$, i.e., we have $a_i b_{i+1} \in OPT$ and $a_i b_i \in M$, for every $i \in [k]$. We first observe that, by definition of $c_{\min}(a_i)$ and the fact that b_{i+1} is in the neighbourhood of a_i, it holds that $c(b_{i+1}) \geq c_{\min}(a_i)$.

Next, we will argue that, for every $0 \leq i \leq k-1$, we have $c_{\min}(a_i) - 1 \leq c_{\min}(a_{i+1})$. Indeed, as argued above, we have $c(b_{i+1}) \geq c_{\min}(a_i)$. Next, observe that a_{i+1} is matched in M to b_{i+1}. Consider the iteration r when a_{i+1} was most recently matched to b_{i+1}. Then, denote by $c^r(b_{i+1})$ the value of $c(b_{i+1})$ in this iteration, and, similarly, denote by $c^r_{\min}(a_i)$ the value of $c_{\min}(a_i)$ in this iteration. In iteration r, the value of $c^r(b_{i+1})$ was increased to its current value $c(b_{i+1})$, i.e., we have $c^r(b_{i+1}) = c(b_{i+1}) - 1$. Since in iteration r, the edge (a_{i+1}, b_{i+1}) was contained in H_r, we have that $c^r_{\min}(a_{i+1}) = c^r(b_{i+1}) (= c(b_{i+1}) - 1)$. Last, since $c_{\min}(a_{i+1}) \geq c^r_{\min}(a_{i+1})$ (Property **P2**), we obtain the claimed bound.

The previous bound implies that $c_{\min}(a_k) \geq c_{\min}(a_0) - k$, and this in turn implies that $c(b_k) \geq c_{\min}(a_0) - k$. Since b_k is unmatched, which, by Property **P3**, means that b_k has not previously been matched, we have that $c(b_k) = 0$. We thus obtain $0 \geq c_{\min}(a_0) - k$ as desired.

Next, consider the set of augmenting paths $\mathcal{P}_{\geq k} \subseteq M \oplus OPT$ starting at a vertex $a \in A_{\text{OPT}}$ with $c_{\min}(a) \geq k$. Observe that these paths are vertex-disjoint, and, as discussed above, each of these paths visits at least $k+1$ A_{OPT} vertices. Hence, there can be at most $|A_{\text{OPT}}|/(k+1) = |OPT|/(k+1)$ such paths, which completes the proof. ◀

Using the insight from the previous lemma, we show that it is enough to find an upper bound on the number of unmatched A_{OPT} vertices with low neighbourhood cost in order to establish the approximation factor of the algorithm.

▶ **Corollary 2.** *Consider an iteration r where the number of unmatched A_{OPT}-vertices with minimum neighbourhood cost less than $2/\epsilon$ is at most $\epsilon/2 \cdot |OPT|$. Then, the current matching M is a $(1-\epsilon)$-approximation.*

Proof. Consider the set of augmenting paths $\mathcal{A}$ in $M \oplus OPT$. Let $A' \subseteq A_{OPT} \setminus A(M)$ be the set of starting vertices of these augmenting paths in A. Partition A' between two sets, $A^+ \subseteq A'$ being the vertices in A' with minimum neighbourhood cost at least $2/\epsilon$, and $A^- = A' \setminus A^+$. By Lemma 1, we have $|A^+| \leq \frac{1}{\frac{2}{\epsilon}+1}|OPT| \leq \frac{\epsilon}{2}|OPT|$. Then:

$$|\mathcal{A}| = |A'| = |A^+| + |A^-| \leq \epsilon/2 \cdot |OPT| + \epsilon/2 \cdot |OPT| = \epsilon \cdot |OPT| \ .$$

Since $|M| + |\mathcal{A}| = |OPT|$, the result follows. ◀

▶ **Theorem 3.** *Our version of the ALT-algorithm listed in Algorithm 1 is a* $(1-\epsilon)$*-approximation algorithm for MM on bipartite graphs.*

Proof. Let M_{out} be the final matching output by the algorithm, and let $B_{\text{out}} = B(M_{\text{out}})$. In this proof, we denote by $c^r(b)$ the quantity $c(b)$ in the beginning of iteration r.

We will first argue that the number of iterations X where the matching N_r computed is of size at least $|N_r| \geq \frac{\epsilon}{3}|OPT|$ is at most $36/\epsilon^2$. To this end, consider such an iteration r and partition the $B(N_r)$-vertices into $B_<(N_r)$ and $B_\geq(N_r)$ such that each $b \in B_<(N_r)$ is such that $c^r(b) < 6/\epsilon$ and each $b \in B_\geq(N_r)$ is such that $c^r(b) \geq 6/\epsilon$. Observe that, by Lemma 1, there are at most $(\epsilon/6) \cdot |OPT|$ unmatched and saturated A-vertices. Hence, at most $(\epsilon/6) \cdot |OPT|$ vertices in $B_\geq(N_r)$ can be matched in iteration r and increase their costs since only unmatched saturated A-vertices can match to these vertices. This in turn implies that at least $(\epsilon/6) \cdot |OPT|$ vertices of $B_<(N_r)$ are matched in the current iteration.

Observe further that $B(N_r) \subseteq B_{\text{out}}$, which follows from Property **P1**. Over the course of the X iterations, we have thus seen at least

$$X \cdot \frac{\epsilon}{6} \cdot |OPT| \tag{1}$$

increases of the costs of B_{out}-vertices b when they had a current value $c(b) < \frac{6}{\epsilon}$. This number, however, is also trivially bounded from above by $\frac{6}{\epsilon} \cdot |B_{\text{out}}| \leq \frac{6}{\epsilon} \cdot |OPT|$, which together with the lower bound in 1 implies that $X \leq \frac{36}{\epsilon^2}$ as desired.

The previous claim implies that the number of iterations where $|N_r| \leq \frac{\epsilon}{3}|OPT|$ is at least $\frac{49}{\epsilon^2} - \frac{36}{\epsilon^2} = \frac{13}{\epsilon^2}$ (recall that Algorithm 1 overall executes $\frac{49}{\epsilon^2}$ iterations). Suppose for the sake of a contradiction that M_{out} is not a $(1-\epsilon)$-approximation. Then, by Corollary 2, in each iteration, there are at least $\frac{\epsilon}{2} \cdot |OPT|$ unmatched vertices $a \in A_{\text{OPT}}$ with $c_{\min}(a) < 2/\epsilon$. Since the matching found in any of these $13/\epsilon^2$ iterations is of size at most $\epsilon/3 \cdot |OPT|$, in each such iteration there are at least $\frac{\epsilon}{2} \cdot |OPT| - \frac{\epsilon}{3} \cdot |OPT| = \frac{\epsilon}{6} \cdot |OPT|$ vertices of A_{OPT} with min-cost smaller than $2/\epsilon$ that are not matched. Observe that, by Property **P3**, each such vertex increases its $c_{\min}$ value. Hence, over the course of the algorithm, we observe at least

$$13/\epsilon^2 \cdot \frac{\epsilon}{6} \cdot |OPT| = \frac{13}{6\epsilon}|OPT| \tag{2}$$

increases in the min-cost value of A_{OPT} vertices when their current min-cost was below $2/\epsilon$. This quantity is trivially bounded by $\frac{2}{\epsilon} \cdot |A_{\text{OPT}}| = \frac{2}{\epsilon} \cdot |OPT|$, which yields a contradiction to the lower bound established in Equation 2. Hence, M_{out} is a $(1-\epsilon)$-approximation. ◀

Why/how does ALT find augmenting paths? Given a matching M, traditional augmenting paths-based matching algorithms determine the edges of an augmenting path $\mathcal{P}$ and then swap the edges of M with the edges of OPT along $\mathcal{P}$, which increases the size of M by 1. The streaming algorithm by Eggert et al. [10] searches for many augmenting paths in parallel and computes in this fashion a set of disjoint augmenting paths that can be augmented simultaneously. Their search is highly structured and based on DFS explorations including backtracking up to a depth of $O(\frac{1}{\epsilon})$ starting at every unmatched vertex.

The ALT-algorithm can also be regarded as one that searches for augmenting paths. In each iteration, the currently unmatched A-vertices form the beginning of augmenting paths. Similar to Lemma 1, it can be seen that, for every unmatched vertex $a \in A$, the quantity $c_{\min}(a)$ gives a lower bound on the length of any augmenting path starting at a. The fact that the algorithm only considers neighbours b of a as potential partners with minimal $c(b)$ value indicates that the search for augmenting paths is directed towards the shortest possible

augmenting paths. Interestingly, the algorithm does not wait with swapping edges in and out of the matching until completed augmenting paths are found as the matching N_r is immediately incorporated into the matching M. This is beneficial from the perspective of augmenting paths since if a was matched to b in N_r and the vertex a', who was the previous mate of b, was unmatched in this process then a' is now in a *stronger* position to complete the augmenting path since its $c_{\min}$ value cannot be larger than the $c_{\min}$ value of a. One may now trace this sequence of swaps with increasingly smaller $c_{\min}$ values in order to identify completed augmenting paths.

4 Hard Instance

We now show that, even on path graphs, the ALT-algorithm may require $\Omega(n^2)$ iterations to obtain a maximum matching. This directly translates to a proof that the number of iterations required for a $(1-\epsilon)$-approximation to MM is $\Theta(\frac{1}{\epsilon^2})$.

We use the family of path graphs on $6x-2$ vertices for any $x \geq 2$, with $3x-1$ A and B vertices, respectively. We label the B-vertex that is an endpoint of the path as b_0. We label the next B-vertex along the path as b_1, and label the rest of the B-vertices by $b_2, ..., b_{3x-2}$ in the same manner. We then label the A-vertices such that a_i is adjacent to b_i and b_{i+1}. The vertex a_{3x-2} is the A-vertex that is only adjacent to b_{3x-2}. This is the other endpoint of the path. We split the A-vertices into three sets A_0, A_1 and A_2 such that $A_k := \{a_\ell : \ell \equiv k \mod 3\}$. We split the B-vertices into B_0, B_1, B_2 in the same way.

We note that the only maximum matching in this graph is the matching between a_k and b_k for each k from 0 to $3x-2$. We set $\epsilon = \frac{1}{3x}$ so that only the singular maximum matching is a $(1-\epsilon)$-approximate matching.

We now describe the four "setup" rounds. An example of a path on 16 vertices after each of these rounds is given in Figure 1.

Round 1: Let the matching returned be the set of edges between A_0 and B_1, as well as the edges between A_2 and B_2. We note that this matching is maximal. The vertices in A_1 and B_0 remain unmatched.

Round 2: Each vertex in A_1 has two neighbours of cost 1. When a vertex has multiple min-cost neighbours, we will take the B-vertex of larger index, represented by the diagonal edges in Figure 1. We add all of the edges between A_1 and B_2. Each vertex in B_2 has its cost increased to 2, and the vertices in A_2 become unmatched. Additionally, the edge between a_{3x-2} and b_{3x-2} is matched.

Round 3: Each of the vertices in A_2 is adjacent to an unmatched vertex in B_0, each of which are added to the matching. The only A-vertex that is not matched after this round is a_{3x-3}, as it is blocked from matching to b_{3x-3} by a_{3x-4}.

Round 4: a_{3x-3} must match to b_{3x-3}, a_{3x-4} becomes unmatched.

After the fourth round, the graph is in the following state. Each a_k for $k \leq 3x-5$ is matched to b_{k+1}, a_{3x-3} is matched to b_{3x-3} and a_{3x-2} is matched to b_{3x-2}. a_{3x-4} is unmatched. All vertices in B_0 have cost 1, except b_0 which has cost 0 and b_{3x-3} which has cost 2. All vertices in B_1 have cost 1, except b_{3x-2} which has cost 2. All vertices in B_2 have cost 2.

During the remainder of the execution of the algorithm, we will use the same tie-break property as described in the second round. That is, the algorithm will take the edge connecting the unmatched A-vertex to the B-vertex of higher index.

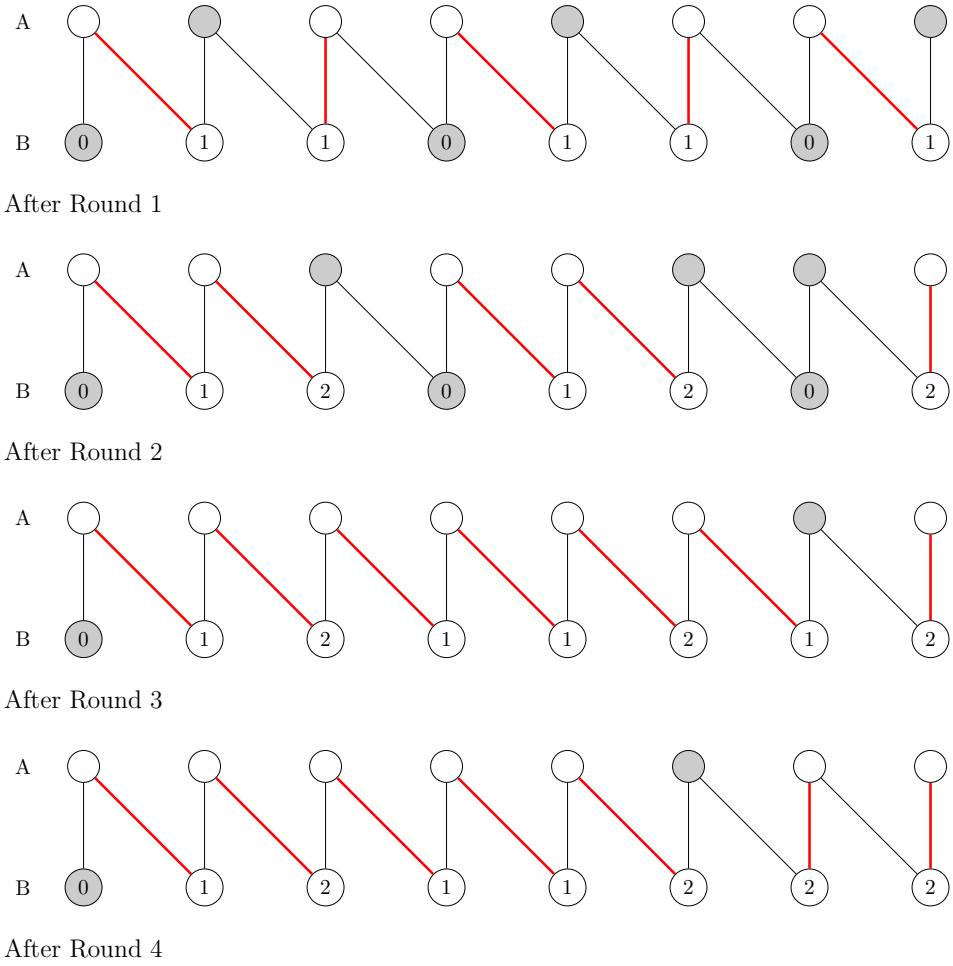

Figure 1 The state of a path on 16 vertices after each setup round. Edges in red are part of the matching. Unfilled vertices are matched, gray vertices are unmatched. The numbers in the B-vertices represent their cost.

To lower bound the number of rounds required for a maximum matching, we will show that using the tie-break condition above causes the algorithm to repeatedly re-match the same vertices multiple times. We will call each of these processes a *sweep*.

The conditions required for a sweep to begin are the following, where a_k is the unmatched vertex.

C1: b_k has cost at least that of b_{k+1}.
C2: For all $i > k$, the cost of b_{i+1} is either the same as the cost of b_i, or one more.

We note that both of these conditions hold for the state of the matching after the fourth round. This means that a sweep will begin after the setup rounds. We now describe what occurs during a sweep. An example of a sweep is given in Figure 2.

The algorithm matches a_k to b_{k+1}, the cost of b_{k+1} increases by 1, and a_{k+1} becomes unmatched. By Condition **C2**, the cost of b_{k+2} was at most the cost of b_{k+1} plus one before this round. Therefore after this round the cost of b_{k+2} is at most the cost of b_{k+1} afterwards. This is Condition **C1** for unmatched vertex a_{k+1}. All the other costs are unchanged, so

Condition **C2** remains true. This means that the sweep will continue by matching a_{k+1} to b_{k+2}. This must then continue all the way down the path until the endpoint a_{3x-2} becomes unmatched. We will call this process the *rightward* portion of the sweep.

Once a_{3x-2} becomes unmatched, the only option for the algorithm in the next round is to immediately rematch it to b_{3x-2}, and so a_{3x-3} becomes unmatched again. At the start of the sweep, b_{3x-2} had cost at most that of b_{3x-3} plus 1 by Condition **C2**. At this point the cost of b_{3x-3} has increased by 1 and the cost of b_{3x-2} has increased by 2. Thus, the cost of b_{3x-2} must now be strictly larger than the cost of b_{3x-3}, and so a_{3x-3} must match to b_{3x-3} and increase the cost of b_{3x-3} by 1. We can now apply the same argument again to the state of the matching after this round, so a_{3x-4} matches to b_{3x-4}. This process, matching a_k to b_k, must therefore continue down the path until a_k becomes unmatched again. We will call this the *leftward* portion of the sweep. The state of the matching is now the same as before the sweep, except with all the costs of b_i for $i > k$ increased by 2.

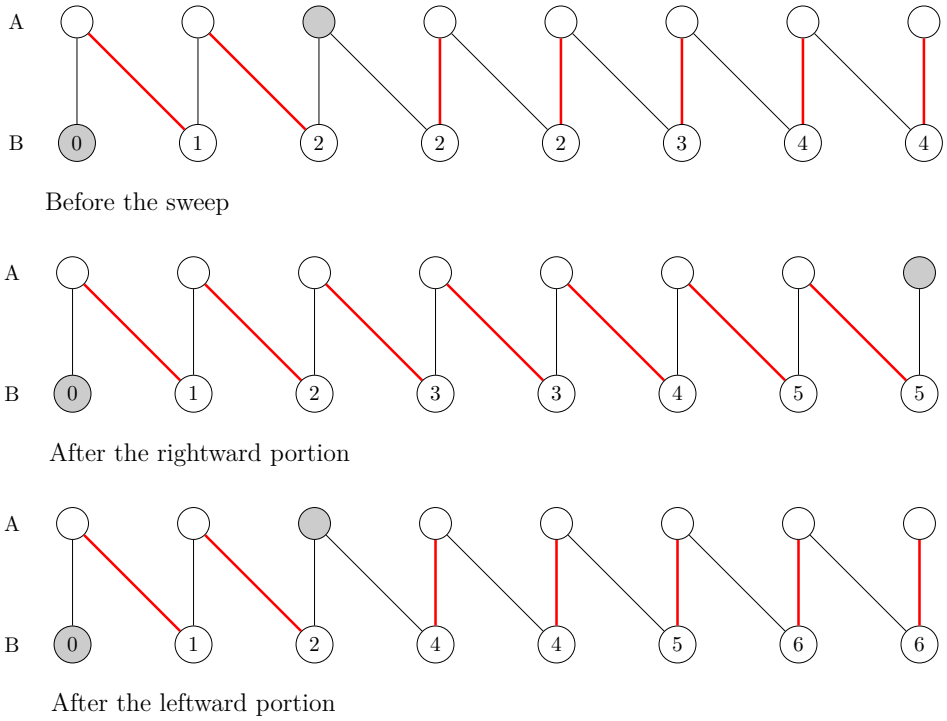

Figure 2 An example of a path during stages of a sweep. Edges in red are part of the matching. Unfilled vertices are matched, grey vertices are unmatched. The numbers in the B-vertices represent their cost.

We now show that sweeps are triggered regularly during the execution of the algorithm. After a sweep has finished, a_k matches to b_k as the cost of b_{k+1} has increased by 2 since the start of the last sweep. The algorithm only ever triggers sweeps at a_k after a_{k+1} has just been unmatched, so the vertices with indices less than k have not been altered since the end of the setup rounds. Therefore, in the next round a_{k-1} will be matched to b_{k-1} as it has cost 1 compared to cost 3 for b_k. In the subsequent round, a_{k-2} will be matched to b_{k-2} as it has cost 1 while b_{k-1} now has cost 2. However, in the next round a_{k-3} has both neighbours with cost 2. This satisfies Condition **C1** for a sweep to begin. Condition **C2** is also satisfied. This is because the B-vertices with indices greater than k have had each of

their costs increased by 2 since the start of the previous sweep, and this does not affect this condition holding. Furthermore, the items from b_{k-2} to b_{k+1} have costs $2, 2, 3, 4$ which also satisfy this condition. Therefore a new sweep must start at a_{k-3}.

This means that a sweep is triggered each time we unmatch a vertex in A_2. We can use the lengths of these sweeps to lower bound the number of rounds required for a maximum matching.

▶ **Theorem 4.** *The ALT-algorithm requires $\Omega(n^2)$ rounds in order to return a maximum matching.*

Proof. Recall that the algorithm performs a sweep starting at each vertex in A_2. If a sweep starts at a_k, it matches each b_i for $i > k$ exactly twice during the sweep. The number of passes required for each of the sweeps is therefore $2((3x-2)-k)$. Summing up the lengths of each of the sweeps, we find a lower bound for the number of passes required.

$$2\sum_{i=0}^{x-1}(3x-2)-(3i-1) = 2\sum_{i=1}^{x} 3i-1 \in \Omega(x^2) = \Omega(n^2). \quad \blacktriangleleft$$

We can now use this result to bound the number of passes required for a $(1-\epsilon)$-approximation.

▶ **Corollary 5.** *The ALT-algorithm requires $\Omega(\frac{1}{\epsilon^2})$ rounds in order to return a $(1-\epsilon)$ approximation to a maximum matching, with or without the freezing mechanism.*

Proof. Choose x to be the integer such that $3x-1 < \frac{1}{\epsilon} \leq 3x+2$. The only $(1-\epsilon)$-approximation to a maximum matching on this graph can only be the singular maximum matching, as we have that $(1-\frac{1}{\epsilon})(3x-1) > (1-\frac{1}{3x-1})(3x-1) = 3x-2$. By Theorem 4, the ALT-algorithm requires $\Omega(x^2) = \Omega(\frac{1}{\epsilon^2})$ passes in order to return a maximum matching.

Additionally, since the vertex costs are always non-decreasing at each step of the algorithm, each vertex cost is bounded above by the cost of b_{3x-2} at the end of the execution of the algorithm. This vertex is matched twice during the setup phase, and twice during each sweep. Therefore, the cost of each vertex at the end of the execution is at most $2x+2$. For $x \geq 3$ this is at most $3x-1 < \frac{1}{\epsilon}$. Hence the freezing mechanism is not triggered, and so the proof also applies when it is included. ◀

5 Conclusion

In this paper, we advance the understanding of the ALT-algorithm through three contributions. First, we simplify the algorithm by showing that the freezing mechanism as used in its original version by Assadi et al. is unnecessary. Second, we provide an alternative analysis based on augmenting paths, which explains the algorithm's behaviour more naturally and deals with the removal of the freezing mechanism in a natural way. Third, we construct the first hard instance on which the algorithm indeed requires $\Omega(\frac{1}{\epsilon^2})$ iterations.

We conclude with two open questions.

First, and most importantly, are there algorithms that solely rely on the computation of maximal matchings that requires fewer than $\Theta(\frac{1}{\epsilon^2})$ iterations/matching computations to find a $(1-\epsilon)$-approximation?

Second, can we prove lower bounds for such algorithms? For example, can we show that $\Omega(\frac{1}{\epsilon})$ matching computations are necessary to obtain a $(1-\epsilon)$-approximation?

References

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

2 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *ACM Trans. Parallel Comput.*, 4(4):17:1–17:40, 2018. doi:10.1145/3154855.

3 Sepehr Assadi. A simple (1 - ϵ)-approximation semi-streaming algorithm for maximum (weighted) matching. In Merav Parter and Seth Pettie, editors, *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 337–354. SIAM, 2024. doi:10.1137/1.9781611977936.31.

4 Sepehr Assadi and Soheil Behnezhad. On the Robust Communication Complexity of Bipartite Matching. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, volume 207 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.48.

5 Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 627–669. SIAM, 2022. doi:10.1137/1.9781611977073.29.

6 Sepehr Assadi, S. Cliff Liu, and Robert E. Tarjan. An auction algorithm for bipartite matching in streaming and massively parallel computation models. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 165–171. SIAM, 2021. doi:10.1137/1.9781611976496.18.

7 Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G. Harris. Exponentially faster massively parallel maximal matching. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1637–1649. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00096.

8 Aaron Bernstein. Improved bounds for matching in random-order streams. *Theory Comput. Syst.*, 68(4):758–772, 2024. doi:10.1007/S00224-023-10155-7.

9 Lidiya Khalidah binti Khalil and Christian Konrad. Constructing large matchings via query access to a maximal matching oracle. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.26.

10 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1-2):490–508, 2012. doi:10.1007/S00453-011-9556-8.

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

12 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010. doi:10.1137/1.9781611973075.76.

13 Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 74:1–74:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.74.

14 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0_20.

15 Christian Konrad and Kheeran K. Naidu. On two-pass streaming algorithms for maximum bipartite matching. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 19:1–19:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.19.

16 Christian Konrad, Kheeran K. Naidu, and Arun Steward. Maximum matching via maximal matching queries. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 41:1–41:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.41.

17 Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2005. doi:10.1007/11538462_15.