A 0.51-Approximation of Maximum Matching in Sublinear $n^{1.5}$ Time

Sepideh Mahabadi[*]   Mohammad Roghani[†]   Jakub Tarnawski[‡]

Abstract

We study the problem of estimating the size of a maximum matching in sublinear time. The problem has been studied extensively in the literature and various algorithms and lower bounds are known for it. Our result is a 0.5109-approximation algorithm with a running time of $\tilde{O}(n\sqrt{n})$.

All previous algorithms either provide only a marginal improvement (e.g., 2^{-280}) over the 0.5-approximation that arises from estimating a *maximal* matching, or have a running time that is nearly n^2. Our approach is also arguably much simpler than other algorithms beating 0.5-approximation.

1 Introduction

Given a graph $G = (V, E)$, a matching is a set of edges with no common endpoints, and the maximum matching problem asks for finding a largest such subset. Matching is a fundamental combinatorial optimization problem, and a benchmark for new algorithmic techniques in all major computational models. It also has a wide range of applications such as ad allocation, social network recommendations, and information retrieval, among others. Given that many of these applications need to handle large volumes of data, the study of sublinear time algorithms for *estimating* the maximum matching size has received considerable attention over the past two decades.[1] A sublinear time algorithm is not allowed to read the entire graph, which would take $\Omega(n^2)$ time where $n = |V|$; instead it is provided *oracle access* to the input graph and must run in $o(n^2)$ time. There are two main oracles for graph problems considered in the literature, which we also consider in this work.

- **Adjacency list oracle.** Here, the algorithm can query (v, i), where $v \in V$ and $i \le n$, and the oracle reports the i-th neighbor of the vertex v in its adjacency list, or NULL if i is larger than the number of v's neighbors.

- **Adjacency matrix oracle.** Here, the algorithm can query (u, v), where $u, v \in V$, and the oracle reports whether there exists an edge between u and v.

Earlier results on estimating the size of the maximum matching in sublinear time mostly focused on graphs with bounded degree Δ, starting with the pioneering work of Parnas and Ron [PR07],

---

[*]Microsoft Research. E-mail: smahabadi@microsoft.com.

[†]Stanford University. E-mail: roghani@stanford.edu. Work done while the author was an intern at Microsoft Research.

[‡]Microsoft Research. E-mail: jakub.tarnawski@microsoft.com.

[1]It is impossible to *find the edges* of any constant-approximate matching in time sublinear with respect to the size of the input [PR07].

and later works of [NO08, YYI09, RTVX11, ARVX12, LRY17, Gha22]. However, for $\Delta = \Omega(n)$ these do not lead to sublinear time algorithms. Thus, later works focused on general graphs with arbitrary maximum degree and managed to obtain sublinear time algorithms for them [CKK20, KMNFT20, Beh21, BRRS23, BKS23b, BKS23a]. In particular, the state of the art results can be categorized into two regimes:

- Algorithms that run in slightly sublinear time, i.e., $n^{2-\Omega_\varepsilon(1)}$. For example, the works of [BRR23b, BKS23b] gave a $(2/3 - \varepsilon)$-approximation algorithm that runs in time $n^{2-\Omega_\varepsilon(1)}$. Later, [BKS23a] improved the approximation factor to $1 - \varepsilon$.

- Algorithms whose approximation factor is $0.5+\varepsilon$. The state of the art result in this category is the work of [BRRS23], whose running time is $n^{1+\varepsilon}$ for an approximation factor of $0.5+\Omega_\varepsilon(1)$. However, the best approximation factor one can get using their trade-off is only $0.5 + 2^{-280}$.[2] Indeed, they mention: *We do not expect our techniques to lead to a better than, say, 0.51-approximation in $n^{2-\Omega(1)}$ time.*

Thus, all known algorithms either give only a marginal improvement over the 0.5-approximation arising from estimating the size of a *maximal* matching (which can be done even in $\widetilde{O}(n)$ time [Beh21]), or have a running time that is nearly n^2. Making a significant improvement on both fronts simultaneously has remained an elusive open question.

Our results. In this work, we show how to beat the factor 0.5 in time that is strongly sublinear. Specifically, we present an algorithm that runs in time $O(n\sqrt{n})$ and achieves an approximation factor of 0.5109 for estimating the size of the maximum matching.

It is worth noting that our algorithm is much simpler, both in terms of implementation and analysis, compared to the prior works that beat 0.5-approximation [BRRS23].

Theorem 1.1. *There exists an algorithm that, given access to the adjacency list of a graph, estimates the size of the maximum matching with a multiplicative approximation factor of* 0.5109 *and runs in* $\widetilde{O}(n\sqrt{n})$ *time with high probability.*

Theorem 1.2. *There exists an algorithm that, given access to the adjacency matrix of a graph, estimates the size of the maximum matching with a multiplicative-additive approximation factor of* $(0.5109, o(n))$ *and runs in* $\widetilde{O}(n\sqrt{n})$ *time with high probability.*

Moreover, our algorithm can be employed as a subroutine for Theorem 2 of [Beh23] to obtain an improved approximation ratio in the dynamic setting. More precisely, that result requires a subroutine for estimating the maximum matching size in $\widetilde{O}(n\sqrt{n})$ time, for which it uses the 0.5-approximation of [Beh21]. Our algorithm can be used instead, resulting in a very slight improvement to the overall approximation guarantee for [Beh23].

We note that the framework of [BKS23a] can also be used to obtain a similar result. Their algorithm, which performs a single iteration to find a constant fraction of augmenting paths of length three on top of a maximal matching, likewise yields a better-than-2 approximate matching with $n^{2-\Omega(1)}$ running time. However, the trade-off in this approach is worse in terms of both the approximation ratio and the running time.

---

[2]More specifically, for $\varepsilon \in (0, 1/4)$, they get an algorithm with approximation factor of $0.5 + 2^{-70/\varepsilon}$ that runs in time $O(n^{1+\varepsilon})$.

Related work. On the lower bound front, Parnas and Ron [PR07] demonstrated that any algorithm getting a constant approximation of the maximum matching size needs at least $\Omega(n)$ time. More recently, the work of [BRR23b] established that any algorithm providing a $(2/3 + \Omega(1), \varepsilon n)$-multiplicative-additive approximation requires at least $n^{6/5-o(1)}$ time. For sparse graphs, a lower bound of $\Delta^{\Omega(1/\varepsilon)}$ was shown for any εn additive approximation [BRR23a]. For dense graphs, [BRR24] showed a lower bound of $n^{2-O_\varepsilon(1)}$ for the runtime of algorithms achieving such additive approximations.

Paper organization. In Section 2, we provide an overview of the challenges encountered while designing our algorithm and the techniques used to address them. We first develop an algorithm for bipartite graphs with a multiplicative-additive error in Section 4, avoiding additional challenges that arise from general graphs, trying to obtain multiplicative error (in the adjacency list model), or working with the adjacency matrix. In Section 5, we extend our algorithm to handle general graphs. In Section 6, we demonstrate how to achieve a multiplicative approximation guarantee. Finally, in Section 7, we present a simple reduction showing that our algorithm also works in the adjacency matrix model with a multiplicative-additive error.

2 Technical Overview

In this section, we provide an overview of the techniques used in this paper to design our algorithm. We begin with the two-pass semi-streaming algorithm of Konrad and Naidu [KN21] for bipartite graphs. In the first pass, the algorithm constructs a maximal matching M. In the second pass, it constructs a maximal b-matching between vertices matched in M and those unmatched in M. More specifically, each vertex in $V(M)$ has a capacity of k, while each vertex in $V \setminus V(M)$ has a capacity of kb, where $b = 1 + \sqrt{2}$ and k is a large constant. The idea is that if M is far from maximum, the b-matching will contain many length-3 augmenting paths that can be used to augment M. This algorithm obtains a $(2 - \sqrt{2}) \approx 0.585$-approximation.

Our goal is to develop a sublinear-time algorithm by translating this semi-streaming two-pass algorithm to the sublinear time model. When trying to do so, several challenges arise. In this section we describe them step by step, and show how to overcome them.

Challenge (1): constructing a maximal matching in sublinear time is not possible. In fact, finding all edges of any constant-factor approximation of the maximum matching is impossible in sublinear time due to [PR07]. Dynamic algorithms for maximum matching [Beh23, BKSW23] use the following approach: they maintain a maximal matching M and then apply the sublinear-time **random greedy maximal matching (RGMM)** algorithm of Behnezhad [Beh21] to estimate the size of the maximal b-matching. In our setting, we cannot afford to explicitly construct M. However, we can obtain oracle access to M using the sublinear-time RGMM algorithm of [Beh21]. More specifically, we can query whether a vertex v is matched in M or not in $\widetilde{O}(n)$ time. Therefore, a possible solution to address the first challenge is to design two nested oracles: the outer oracle attempts to build a maximal b-matching, whereas the inner oracle checks the status of vertices (matched or not in M) to correctly filter edges and assign capacities to each vertex.

Challenge (2): two nested oracles require $\Omega(n^2)$ time. The algorithm of [Beh21] runs in $\widetilde{O}(\bar{d}(G))$ time, where $\bar{d}(G)$ denotes the average degree of the graph G. Additionally, for the outer oracle, it requires $\widetilde{O}(\bar{d}(G[V(M), V \setminus V(M)]))$ time (i.e., queries to the inner oracle). Unfortunately, it is possible for both $\bar{d}(G)$ and $\bar{d}(G[V(M), V \setminus V(M)])$ to be as large as $\Omega(n)$. For example,
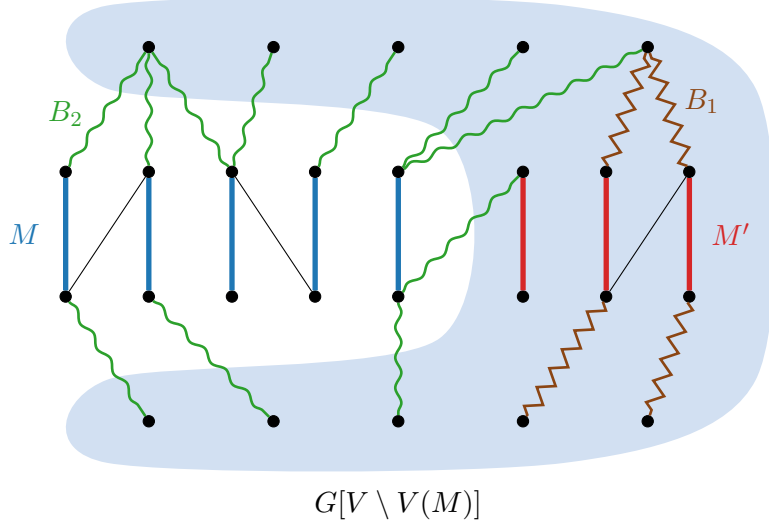
3

Figure 1: Our algorithm explicitly constructs a matching M (blue), which need not be maximal in G. We extend it with another matching M' (red), such that $M \cup M'$ is maximal. The highlighted (light blue) subgraph $G[V \setminus V(M)]$ has degree at most $\sqrt{n}$ with high probability. In case 1, our algorithm augments M' using a b-matching B_1 (zigzag edges, brown). In case 2, our algorithm augments M using a b-matching B_2 (swirly edges, green).

consider a graph with a vertex set $A \cup B$, where $|A| = |B| = n/2$. The edges within A form a complete bipartite graph, while there is an $\varepsilon n/2$-regular graph between A and B. After running the RGMM algorithm, most edges in the maximal matching belong to $G[A]$, and most vertices in B are unmatched. Consequently, we have $\bar{d}(G[V(M), V \setminus V(M)]) = \Omega(n)$.

To address this issue, we sparsify the original graph while manually constructing a matching M. In a preprocessing step, starting from an empty M, for each unmatched vertex in the graph, we sample $\widetilde{\Theta}(\sqrt{n})$ neighbors uniformly at random. If an unmatched neighbor exists, we match the two vertices and add this edge to M. Using this preprocessing step, we show that after spending $\widetilde{O}(n\sqrt{n})$ time, the induced subgraph of vertices that remain unmatched in M has a maximum degree of $\sqrt{n}$ with high probability. Moreover, since we explicitly materialize M, we are able to check if any vertex is matched in M in $O(1)$ time, eliminating the need for costly oracle calls. Note that M need not be maximal in G; therefore we next extend it to a maximal matching.

Let M' be a maximal matching in $G[V \setminus V(M)]$ obtained by running the sublinear time RGMM algorithm of [Beh21]. Now $M \cup M'$ is a maximal matching for G. Inspired by the two-pass semi-streaming algorithm of [KN21], we attempt to augment the maximal matching $M \cup M'$ in two possible ways (see also Figure 1):

1. Augment M' using a b-matching between $V(M')$ and $V \setminus V(M) \setminus V(M')$. The algorithm then outputs the size of the augmented matching, plus the size of the previously constructed matching M.

2. Augment M using a b-matching between $V(M)$ and $V \setminus V(M)$. The algorithm then outputs the size of the augmented matching.

The key intuition here (think of the case when $M \cup M'$ yields only a 0.5-approximation) is that either M' is sufficiently large, making $|M| + (2 - \sqrt{2}) \cdot 2|M'|$ larger than the approximation

guarantee, or M itself is large enough so that $(2 - \sqrt{2}) \cdot 2|M|$ meets our approximation guarantee (note that $(2 - \sqrt{2})$ is the approximation guarantee of [KN21]). Augmenting M using a b-matching is easier since we have explicit access to M and only need to run a single RGMM oracle to estimate the size of the b-matching. Our first estimate, which requires finding a b-matching between $V(M')$ and $V \setminus V(M) \setminus V(M')$, is more challenging since we do not have explicit access to M'. To avoid the $\Omega(n^2)$ running time of the two nested oracles, we make crucial use of the aforementioned property that the subgraph of vertices unmatched in M has low induced degree (at most $\sqrt{n}$); this is the reason why we only try to augment M' rather than $M \cup M'$. We will discuss this in the next paragraphs.

Challenge (3): the algorithm does not have access to the adjacency list of $G[V \setminus V(M)]$. After the sparsification step, the average degree d of $G[V \setminus V(M)]$ is at most $\sqrt{n}$. Hence, if the algorithm had access to the adjacency list of $G[V \setminus V(M)]$, it could run the nested oracles in $\widetilde{O}(d^2) = \widetilde{O}(n)$ time by executing two RGMM algorithms: inner oracle for computing M' and outer oracle for the b-matching to augment M'. But, since the nested oracles may visit up to n vertices, and retrieving the full adjacency list of a vertex in $G[V \setminus V(M)]$ requires $\Omega(n)$ time, it seems that the overall running time of the algorithm could still be as high as $\Omega(n^2)$.

Here, we leverage two key properties of the RGMM algorithm to refine the runtime analysis. The first property is that at each step, the algorithm requires only a random neighbor of the currently visited vertex. Intuitively, if a vertex has degree $\Theta(n)$ in G, in expectation it takes $O(n/d)$ samples from the adjacency list of the original graph to encounter a vertex from $G[V \setminus V(M)]$. Thus, if all vertices in $G[V \setminus V(M)]$ had degree d, one could easily argue that the running time of the algorithm is $\widetilde{O}(d^2 \cdot n/d) = \widetilde{O}(n\sqrt{n})$. However, vertex degrees can vary, and for a vertex with a constant degree, we would need $\Omega(n)$ samples from the adjacency list of G to find a single neighbor in $G[V \setminus V(M)]$. To address this challenge, we utilize another property of the RGMM algorithm, recently proven by [MRTV25]. Informally, this result shows that during oracle calls for RGMM, a vertex is visited proportionally to its degree, implying that low-degree vertices are visited only a small number of times.

Challenge (4): outer oracle creates biased inner oracle queries. The final main challenge we discuss here is that the simple $\widetilde{O}(n\sqrt{n})$ bound, which we informally proved in the previous paragraph, relies on the tacit assumption that the inner oracle queries generated by the outer oracle correspond to $\widetilde{O}(\sqrt{n})$ uniformly random calls to the inner oracle. Indeed, the running time of the algorithm of [Beh21] is analyzed for a uniformly random query vertex; however, there may exist a vertex v in the graph for which calling the inner oracle takes significantly more than $\widetilde{O}(d)$ time. Consequently, if all outer oracle calls end up querying v, the running time could be significantly worse than $\widetilde{O}(n\sqrt{n})$. To overcome this issue, we use the result of [MRTV25] along with the fact that the maximum degree of $G[V \setminus V(M)]$ is $\widetilde{O}(\sqrt{n})$. We show that for any vertex v, the outer oracle queries the inner oracle for v at most $\widetilde{O}(\deg_{G[V \setminus V(M)]}(v)/\sqrt{n})$ times in expectation. This enables us to formally prove that the total running time of the algorithm remains at most $\widetilde{O}(n\sqrt{n})$.

General graphs and the adjacency matrix model. There are additional challenges when dealing with general graphs as opposed to bipartite graphs, such as the fact that the sizes of the maximal matching and the b-matching alone are insufficient to achieve a good approximation ratio. For general graphs, our algorithm estimates the maximum matching in the union of the maximal matching and the b-matching, which requires using the $(1 - \varepsilon)$-approximate local computation

5

algorithm (LCA) by [LRY17] on the subgraph formed by this union, to which we only have oracle access. We encourage readers to refer to Section 5 for more details about the techniques used there.

Additionally, for more information on the extension of the algorithm that operates in the adjacency matrix model, we recommend readers to check Section 7.

3   Preliminaries

Given a graph G, we use $V(G)$ to denote its set of vertices and use $E(G)$ to refer to its set of edges. For a vertex $v \in V(G)$, we use $\deg_G(v)$ to denote the degree of the vertex, i.e., the number of edges with one endpoint equal to v. We use $\Delta(G)$ to denote the maximum degree over all vertices in the graph, and $\bar{d}(G)$ to denote the average degree of the graph. Further, we use $\mu(G)$ to denote the size of the maximum matching in G.

Given a graph $G = (V, E)$, and a subset of vertices $A \subseteq V$, $G[A]$ is defined to be the induced subgraph consisting of all edges with both endpoints in A. Further, given disjoint subsets $A, B \subset V$ of vertices, $G[A, B]$ is defined to be the bipartite subgraph of G consisting of all edges between A and B.

Given a matching M in G, an *augmenting path* is a simple path starting and ending at different vertices, such that the first and the last vertices are unmatched in M, and the edges of the path alternate between not belonging to M and belonging to M.

Given a vector b of integer capacities of dimension $|V(G)|$, a *b-matching* in G is a *multi-set F* of edges in G such that each vertex $v \in V$ appears no more than b_v times as an endpoint of an edge in F.

Given a graph G and a permutation π over its edges, $\text{GMM}(G, \pi)$ is used to refer to the unique matching M obtained by the following process. We let M be initialized as empty, and consider the edges of G one by one according to the permutation π. We add an edge to the matching M if none of its endpoints are already matched in M. A random greedy maximal matching, i.e., $\text{RGMM}(G)$ is the matching obtained by picking a permutation π uniformly at random and outputting $\text{GMM}(G, \pi)$.

Proposition 3.1 ([Beh21]). *There exists an algorithm that, given adjacency list access to a graph G of average degree d, for a random vertex v and a random permutation π, determines if v is matched in $\text{GMM}(G, \pi)$ in $\tilde{O}(d)$ expected time.*

Given the problem of maximizing a function $f : D \to \mathbb{R}$ defined over a domain D, with optimal value f^*, an (α, β)-multiplicative-additive approximation of f^* is a solution $s \in D$ such that $(f^*/\alpha) - \beta \leq f(s) \leq f^*$.

4   Algorithm for Bipartite Graphs

We begin by describing our algorithm for bipartite graphs. We focus on implementing an algorithm with a multiplicative-additive approximation guarantee. Also, we assume that we have access to the adjacency list of the graph. These assumptions will help us avoid certain complications and challenges that arise when working with general graphs, the adjacency matrix model, or when trying to obtain a multiplicative approximation guarantee. To lift these assumptions, we can leverage strong tools and methods from the literature, which, with slight modifications, can be applied here. This section contains the main novelties of our approach and proofs. Our algorithm for bipartite graphs can be seen as a translation and implementation of a two-pass streaming algorithm, which we discuss in the next subsection.

4.1 Two-Pass Streaming Algorithm for Bipartite Graphs

Our starting point is the two-pass streaming algorithm which is described in Algorithm 1. This algorithm, or its variations, has appeared in previous works on designing streaming or dynamic algorithms for maximum matching [KN21, BKSW23, Beh23]. In words, the first pass of the algorithm only finds a maximal matching M. In the second pass, the algorithm finds a maximal b-matching B in $G[V(M), V \setminus V(M)]$, where $V(M)$ is the set of vertices matched by M. The capacities of vertices in $V(M)$ and in $V \setminus V(M)$ for the b-matching are k and kb, respectively. Moreover, in the second pass of the algorithm, when an edge (u, v) arrives in the stream, we add multiple copies of the edge to the subgraph B, as long as doing so does not violate the capacity constraints.

Algorithm 1: Two-pass Streaming Algorithm for Bipartite Graphs

---

1 Parameter: let $b = 1 + \sqrt{2}$ and k be an integer larger than $\frac{1}{b\varepsilon^3}$.

2 First Pass: $M \leftarrow$ maximal matching of G.   ▷ Finding maximal matching

3 Second Pass: ▷ Finding b-matching

4 Let $B = \emptyset$.

5 for $(u, v) \in G[V(M), \overline{V(M)}]$ *where* $u \in V(M)$ **do**

6   **while** $\deg_B(u) < k$ *and* $\deg_B(v) < \lceil kb \rceil$ **do**

7 $B \leftarrow B \cup (u, v)$.   ▷ We allow multi edges

8 return $(1 - 1/b) \cdot |M| + 1/(kb) \cdot |B|$.

Intuitively, the algorithm tries to find length-3 augmenting paths using the b-matching that it finds in the second pass. The following lemma shows the approximation guarantee of Algorithm 1.

Lemma 4.1 (Lemma 3.3 in [BKSW23]). *For any $\varepsilon \in (0, 1)$, the output of Algorithm 1 is a $(2 - \sqrt{2} - \varepsilon)$-approximation for maximum matching of G.*

4.2 Sublinear Time Implementation of Algorithm 1

In this section, we demonstrate how to implement a modification of Algorithm 1 in the sublinear time model, as outlined in Section 2.

Sparsification: In order to be able to use two levels of recursive oracle calls, we need to sparsify the graph. We first sample $\widetilde{O}(n\sqrt{n})$ edges and construct a maximal matching on the sampled edges to sparsify the induced subgraph of unmatched vertices. This sparsification step is formalized in Algorithm 2. In Lemma 4.3, we show that if we sample enough edges, then vertices that remain unmatched after this phase have an induced degree of $\sqrt{n}$. This step is very similar to the algorithm of [CKK20, Appendix A] for approximating a maximal matching.

Claim 4.2. *Algorithm 2 runs in $\widetilde{O}(n\sqrt{n})$ time.*

Proof. For each vertex v in the graph, the algorithm samples $\widetilde{O}(\sqrt{n})$ vertices from the adjacency list of the vertex v if it is not matched by the time the algorithm processes the vertex in Line 3. Thus, the total running time is at most $\widetilde{O}(n\sqrt{n})$. ☐

Lemma 4.3. *With high probability, we have $\Delta(G[V \setminus V(M)]) \leq \sqrt{n}$.*

Proof. We will show that for every $v \in V$, the probability that $v \in V \setminus V(M)$ and the degree of v in $G[V \setminus V(M)]$ is larger than $\sqrt{n}$ is at most $1/n^2$. The lemma then follows by a union bound over all $v \in V$.

Algorithm 2: Sparsification of the Induced Subgraph of Unmatched Vertices

1 **Parameter:** let $c = 2\sqrt{n} \cdot \log n$ be the sparsification parameter that controls the number of edges that the algorithm samples.

2 $M \leftarrow \emptyset$

3 **for** $v \in V$ **do**

4   **if** $v \notin V(M)$ **then**

5   Sample c vertices $u_1, \ldots, u_c$ from $N(v)$.

6   **for** $i \leftarrow 1 \ldots c$ **do**

7   **if** $u_i \notin V(M)$ **then**

8 $M \leftarrow M \cup \{(v, u_i)\}$

9   **break;**

10 **return** M

Consider the moment before v is processed. Assume that at this time, still $v \in V \setminus V(M)$ and the degree of v in $G[V \setminus V(M)]$ is larger than $\sqrt{n}$. Then, each of the c samples has a probability of at least $\sqrt{n}/n$ to be one of the unmatched neighbors, in which case v will become matched. Thus the probability that v remains unmatched after it is processed is at most

$$\left(1 - \frac{1}{\sqrt{n}}\right)^c = \left(1 - \frac{1}{\sqrt{n}}\right)^{2\sqrt{n} \log n} \leq \left(\frac{1}{e}\right)^{2 \log n} = \frac{1}{n^2}.$$

$\square$

Augmenting M using nested oracles: Now we are ready to present our sublinear algorithm. After sparsifying the graph by finding a partially maximal matching M, we try to augment M in two different ways that we have outlined in Section 2 and which are formalized in Algorithm 3. See also Figure 1.

For simplicity, we pretend that $kb \in \mathbb{Z}$ throughout the paper. Since k is an arbitrarily large constant, using kb instead of $\lceil kb \rceil$ leads to an arbitrarily small error in the calculations. We also note that a maximal b-matching can be viewed as maximal matching if we duplicate vertices multiple times.

First, we try to augment the matching by designing a maximal matching oracle for $G[V \setminus V(M)]$ vertices and then another oracle for finding a b-matching between the vertices newly matched using the new oracle and unmatched vertices. Let M' be the maximal matching of $G[V \setminus V(M)]$ that can be obtained by the oracle. We try to augment it with a b-matching B_1.

However, it is also possible that $|M'|$ is small compared to $|M|$, which implies that in the previous case, the b-matching does not help to find many augmenting paths, as the size of the maximal matching that we try to augment is too small. To account for this case, the algorithm also finds a b-matching B_2 between $V(M)$ and $V \setminus V(M)$.

Note that because the algorithm finds the initial matching M explicitly, checking whether a vertex belongs to $V(M)$ or not can be done in $O(1)$ time.

Algorithm 3: Sublinear Time Algorithm for Bipartite Graphs with Access to the Adjacency List (see Figure 1)

1 Run Algorithm 2 with $c = 2\sqrt{n}\log n$ and let M be its output.
2 Let $\mu_{M'}$ and μ_{B_1} be the estimate of the size of a random greedy maximal matching M' in $G[V \setminus V(M)]$ and the estimate of the size of a random greedy maximal b-matching B_1 in $G[V(M'), V \setminus V(M) \setminus V(M')]$ by running Algorithm 4. ▷ Case 1
3 Let $\mu_1 := |M| + (1 - \frac{1}{b})\mu_{M'} + \frac{1}{kb}\mu_{B_1}$. ▷ Case 1
4 Let μ_{B_2} be the estimate of the size of a random greedy maximal b-matching B_2 in $G[V(M), V \setminus V(M)]$ by running Algorithm 5. ▷ Case 2
5 Let $\mu_2 := (1 - \frac{1}{b})|M| + \frac{1}{kb}\mu_{B_2}$. ▷ Case 2
6 **return** $\max(\mu_1, \mu_2)$.

Algorithm 4: Algorithm for the First Case

1 Let $b = 1 + \sqrt{2}$ and k be an integer larger than $\frac{1}{b\varepsilon^3}$.
2 Let π be a random permutation over edges of $G[V \setminus V(M)]$ and let M' be its corresponding random greedy maximal matching.
3 Let $G_1 := G[A, B]$ where $A = V(M')$ and $B = V \setminus V(M) \setminus V(M')$.
4 Let G_1' be a bipartite graph obtained from G_1 by adding k copies of vertices in A and kb copies of vertices in B. Further, if there exists an edge between $u \in A$ and $v \in B$ in G_1, we add edges between all copies of u and v in G_1'.
5 $r \leftarrow 6\log^3 n$.
6 Run the algorithm of Proposition 3.1 for r random vertices and fixed permutation π in $G[V \setminus V(M)]$ and let X_i be the indicator if the i-th vertex is matched.
7 Let $X \leftarrow \sum_{i=1}^r X_i$ and $\mu_{M'} \leftarrow \frac{nX}{2r} - \frac{n}{2\log n}$.
8 Run nested oracles of Proposition 3.1 for r random vertices and fixed permutation π in G_1' and let Y_i be the indicator if the i-th vertex is matched.
9 Let $Y \leftarrow \sum_{i=1}^r Y_i$ and $\mu_{B_1} \leftarrow \frac{nY}{2r} - \frac{n}{2\log n}$.
10 **return** $\mu_{M'}$ and μ_{B_1}.

Algorithm 5: Algorithm for the Second Case

1 Let $b = 1 + \sqrt{2}$ and k be an integer larger than $\frac{1}{b\varepsilon^3}$.
2 Let $G_2 := G[A, B]$ where $A = V(M)$ and $B = V \setminus V(M)$.
3 Let G_2' be a bipartite graph obtained from G_2 by adding k copies of vertices in A and kb copies of vertices in B. Further, if there exists an edge between $u \in A$ and $v \in B$ in G_2, we add edges between all copies of u and v in G_2'.
4 $r \leftarrow 6\log^3 n$.
5 Run the algorithm of Proposition 3.1 for r random vertices and permutations in G_2' and let Z_i be the indicator that shows if the i-th vertex is matched.
6 Let $Z \leftarrow \sum_{i=1}^r Z_i$ and $\mu_{B_2} \leftarrow \frac{nZ}{2r} - \frac{n}{2\log n}$.
7 **return** μ_{B_2}.

Implementation details of the algorithm: There are some technical details in the implementation of the algorithm that are not included in the pseudocode:

- **Access to the adjacency list of an induced subgraph:** Both in Algorithm 4 and Algorithm 5, we run the algorithm of Proposition 3.1 for some induced subgraph of G (for example, line 5 of Algorithm 5). However, Proposition 3.1 works with access to the adjacency list of

the input graph. To address this issue, we leverage an important property of the algorithm in Proposition 3.1, namely that it only needs to find a random neighbor of a given vertex at each step of its execution. Now, whenever the algorithm requires a random neighbor of vertex v in a subgraph H, it queries random neighbors in the original graph G until it finds one that belongs to H. This increases the running time of the algorithm, as it may take $\omega(1)$ time to locate a valid neighbor in H, which we will formally bound in our runtime analysis.

- **Nested oracles in line 8 of Algorithm 4:** Unlike M, we do not explicitly construct the maximal matching M' in Algorithm 4. Moreover, the edges of the subgraph G'_1 connect vertices matched by M' with those that remain unmatched in either M or M'. Hence, to verify whether an edge belongs to G'_1, we need to determine whether its endpoints are matched or unmatched in M' by accessing the algorithm of Proposition 3.1. This again increases the algorithm's runtime, which we will also formally bound in our runtime analysis.

4.3 Analysis of the Approximation Ratio

The following lemma, an analogue of Observation 3.1 in [BKSW23], substantiates the soundness of the estimates μ_1 and μ_2 produced in Algorithm 3.

Lemma 4.4. *Let M, M', B_1 and B_2 be as in the description of Algorithm 3. Then*

- $\mu(G) \geq \mu(M \cup M' \cup B_1) \geq |M| + (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1|$,

- $\mu(G) \geq \mu(M \cup B_2) \geq (1 - \frac{1}{b})|M| + \frac{1}{kb}|B_2|$.

Proof. Since G is bipartite, by integrality of the bipartite matching polytope, it is enough to exhibit a fractional matching x of the appropriate value $\sum_e x_e$. For case 1, we set

$$x_e = \begin{cases} 1 & e \in M, \\ 1 - \frac{1}{b} & e \in M', \\ \frac{1}{kb} & e \in B_1. \end{cases}$$

Note that M, M' and B_1 are pairwise disjoint, and B_1 has no edge to $V(M)$. Therefore it is easy to verify that the degree constraints for x are satisfied. For case 2, we similarly set

$$x_e = \begin{cases} 1 - \frac{1}{b} & e \in M, \\ \frac{1}{kb} & e \in B_2. \end{cases}$$

$\square$

The following lemma states the $(2 - \sqrt{2})$-approximation guarantee of the "maximal matching plus b-matching" approach obtained in prior work, for both bipartite and general graphs. We will invoke it for appropriate subgraphs of G to obtain our guarantee.

Lemma 4.5. *Let G' be a graph, M' be any maximal matching in G', and B be a maximal b-matching in $G'[V(M'), V(G') \setminus V(M')]$ for vertex capacities k for vertices in $V(M')$ and kb for vertices in $V(G') \setminus V(M')$, where $k > \frac{1}{b\varepsilon^3}$ and $b = 1 + \sqrt{2}$. Then:*

- *for bipartite G', we have $\mu(M' \cup B) \geq (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B| \geq (2 - \sqrt{2} - \varepsilon)\mu(G')$,*

- *for general G', if B is a* random greedy *maximal b-matching, we still have $\mathbf{E}[\mu(M' \cup B)] \geq (2 - \sqrt{2} - \varepsilon)\mu(G')$.*
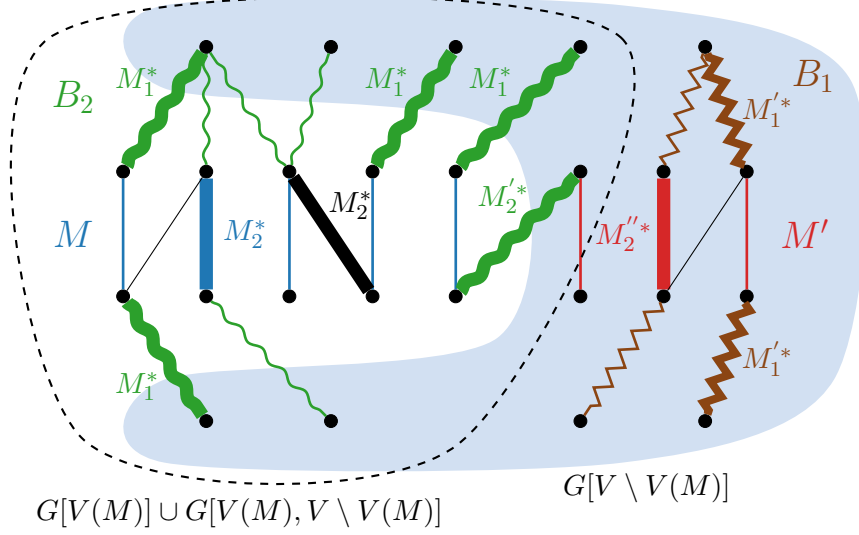
Figure 2: Illustration for the proof of Lemma 4.6. The thick edges belong to a fixed maximum matching M^*. Each of them is labeled with its partition (M_2^*, $M_2'^*$, $M_2''^*$, M_1^*, or $M_1'^*$). The two subgraphs for which we invoke Lemma 4.5 are marked (case 1 – highlighted in light blue, case 2 – dashed line).

Proof. The first statement is the same as Lemma 4.1, and shown as Lemma 3.3 in [BKSW23]. The second statement is shown as Claim 5.5 in [ABR24]. ☐

The following lemma is the crux of our approximation ratio analysis.

Lemma 4.6. *In a bipartite graph G, let M, M', B_1 and B_2 be as in the description of Algorithm 3. Then*

$$\max\left[|M| + (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1|, (1 - \frac{1}{b})|M| + \frac{1}{kb}|B_2|\right] \geq 0.5109 \cdot \mu(G).$$

Proof. Fix a maximum matching M^* in G, and partition its edges as follows (see Figure 2):

- M_2^* are those with both endpoints in $V(M)$,

- $M_2'^*$ are those with one endpoint in $V(M)$ and the other in $V(M')$,

- $M_2''^*$ are those with both endpoints in $V(M')$,

- M_1^* are those with one endpoint in $V(M)$ and the other in $V \setminus V(M) \setminus V(M')$,

- $M_1'^*$ are those with one endpoint in $V(M')$ and the other in $V \setminus V(M) \setminus V(M')$.

Since $M \cup M'$ is maximal, M^* cannot have edges with no endpoints in $V(M) \cup V(M')$. We thus have

$$\mu(G) = |M^*| = |M_1^*| + |M_1'^*| + |M_2^*| + |M_2'^*| + |M_2''^*|. \tag{1}$$

Also, by maximality of M^* and simple counting,

$$|M| = |M_2^*| + \frac{1}{2}|M_1^*| + \frac{1}{2}|M_2'^*|. \tag{2}$$

11

We will use Lemma 4.5 to analyze both cases in Algorithm 3. For case 1, we can use $G' = G[V \setminus V(M)]$; in this graph, M' is a maximal matching, and B_1 is a b-matching as in the statement of Lemma 4.5 (see Figure 2), thus we have

$$(1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1| \geq (2 - \sqrt{2} - \varepsilon)\mu(G[V \setminus V(M)]) \geq (2 - \sqrt{2} - \varepsilon)(|M_1'^*| + |M_2''^*|)$$

since $M_1'^* \cup M_2''^*$ is a matching in $G' = G[V \setminus V(M)]$. With (2) this gives

$$|M| + (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1| \geq |M_2^*| + \frac{1}{2}|M_1^*| + \frac{1}{2}|M_2'^*| + (2 - \sqrt{2} - \varepsilon)(|M_1'^*| + |M_2''^*|). \quad (3)$$

For case 2, we can instead use $G' = G[V(M)] \cup G[V(M), V \setminus V(M)]$; in this graph, M is a maximal matching, and B_2 is a b-matching as in the statement of Lemma 4.5 (see Figure 2), thus

$$(1 - \frac{1}{b})|M| + \frac{1}{kb}|B_2| \geq (2 - \sqrt{2} - \varepsilon)\mu(G') \geq (2 - \sqrt{2} - \varepsilon)(|M_2^*| + |M_2'^*| + |M_1^*|) \quad (4)$$

since $M_2^* \cup M_2'^* \cup M_1^*$ is a matching in G'. Using (3) and (4), we can bound the left-hand side of this lemma's statement as

$$\max[\ldots] \geq \max[|M_2^*| + \frac{1}{2}|M_1^*| + \frac{1}{2}|M_2'^*| + (2 - \sqrt{2} - \varepsilon)(|M_1'^*| + |M_2''^*|),$$
$$(2 - \sqrt{2} - \varepsilon)(|M_2^*| + |M_2'^*| + |M_1^*|)]$$
$$\geq \ldots$$

We bound the maximum by a weighted average with weights β and $1 - \beta$ for some $\beta \in [0, 1]$ to be determined $(\max(x, y) \geq \beta x + (1 - \beta)y)$:

$$\ldots \geq |M_2^*|(\beta + (2 - \sqrt{2} - \varepsilon)(1 - \beta))$$
$$+ (|M_1^*| + |M_2'^*|)\left(\frac{\beta}{2} + (2 - \sqrt{2} - \varepsilon)(1 - \beta)\right)$$
$$+ (|M_1'^*| + |M_2''^*|)(2 - \sqrt{2} - \varepsilon)\beta$$
$$\geq (|M_2^*| + |M_1^*| + |M_2'^*|)\left(\frac{\beta}{2} + (2 - \sqrt{2} - \varepsilon)(1 - \beta)\right) + (|M_1'^*| + |M_2''^*|)(2 - \sqrt{2} - \varepsilon)\beta$$
$$\overset{(*)}{\geq} (|M_2^*| + |M_1^*| + |M_2'^*|)\gamma + (|M_1'^*| + |M_2''^*|)\gamma$$
$$\overset{(1)}{=} \gamma \cdot \mu(G).$$

We want $(*)$ to hold for some γ (the approximation ratio) as large as possible. For $(*)$ to hold, we need to satisfy:

$$\frac{\beta}{2} + (2 - \sqrt{2} - \varepsilon)(1 - \beta) \geq \gamma,$$
$$(2 - \sqrt{2} - \varepsilon)\beta \geq \gamma.$$

By solving for $\frac{\beta}{2} + (2 - \sqrt{2})(1 - \beta) = (2 - \sqrt{2})\beta$ we get $\beta = \frac{12 + 2\sqrt{2}}{17}$ and $\gamma = \frac{4(5 - 2\sqrt{2})}{17} - O(\varepsilon) > 0.5109$. $\qquad \square$

Lemma 4.7. *Let* $\max(\mu_1, \mu_2)$ *be the output of [Algorithm 3]. With high probability, it holds that*

$$0.5109 \cdot \mu(G) - o(n) \leq \max(\mu_1, \mu_2) \leq \mu(G).$$

The proof of [Lemma 4.7] is routine.

Proof. By [Lemma 4.6], we have

$$0.5109 \cdot \mu(G) \leq \max\left[|M| + (1 - \frac{1}{b})\,\mathbf{E}\,|M'| + \frac{1}{kb}\,\mathbf{E}\,|B_1|, (1 - \frac{1}{b})|M| + \frac{1}{kb}\,\mathbf{E}\,|B_2|\right] \leq \mu(G). \quad (5)$$

Let X_i, Y_i, and Z_i be as defined in [Algorithm 4] and [Algorithm 5]. By the definition of X_i, Y_i, and Z_i we have

$$\mathbf{E}[X_i] = \Pr[X_i = 1] = \frac{2\,\mathbf{E}\,|M'|}{n},$$

$$\mathbf{E}[Y_i] = \Pr[Y_i = 1] = \frac{2\,\mathbf{E}\,|B_1|}{n},$$

$$\mathbf{E}[Z_i] = \Pr[Z_i = 1] = \frac{2\,\mathbf{E}\,|B_2|}{n}.$$

Thus,

$$\mathbf{E}[X] = \frac{2r \cdot \mathbf{E}\,|M'|}{n},$$

$$\mathbf{E}[Y] = \frac{2r \cdot \mathbf{E}\,|B_1|}{n},$$

$$\mathbf{E}[Z] = \frac{2r \cdot \mathbf{E}\,|B_2|}{n}.$$

Then, using Chernoff bound, we obtain

$$\Pr[|X - \mathbf{E}[X]| \geq \sqrt{6\,\mathbf{E}[X]\log n}] \leq 2\exp\left(\frac{6\,\mathbf{E}[X]\log n}{3\,\mathbf{E}[X]}\right) = \frac{2}{n^2},$$

$$\Pr[|Y - \mathbf{E}[Y]| \geq \sqrt{6\,\mathbf{E}[Y]\log n}] \leq 2\exp\left(\frac{6\,\mathbf{E}[Y]\log n}{3\,\mathbf{E}[Y]}\right) = \frac{2}{n^2},$$

$$\Pr[|Z - \mathbf{E}[Z]| \geq \sqrt{6\,\mathbf{E}[Z]\log n}] \leq 2\exp\left(\frac{6\,\mathbf{E}[Z]\log n}{3\,\mathbf{E}[Z]}\right) = \frac{2}{n^2}.$$

Therefore, with a probability of $1 - 2/n^2$,

$$\mu_{M'} = \frac{nX}{2r} - \frac{n}{2\log n} \in \frac{n(\mathbf{E}[X] \pm \sqrt{6\,\mathbf{E}[X]\log n})}{2r} - \frac{n}{2\log n}$$

$$= \mathbf{E}\,|M'| - \frac{n}{2\log n} \pm \sqrt{\frac{3n\,\mathbf{E}\,|M'|\log n}{2r}}$$

$$= \mathbf{E}\,|M'| - \frac{n}{2\log n} \pm \sqrt{\frac{n\,\mathbf{E}\,|M'|}{4\log^2 n}} \qquad \text{(Since } r = 6\log^3 n)$$

$$= \mathbf{E}\,|M'| - \frac{n}{2\log n} \pm \frac{n}{2\log n}.$$

By repeating the same argument for Y and Z, we get

$$\mu_{B_1} \in \mathbf{E}\,|B_1| - \frac{n}{2\log n} \pm \frac{n}{2\log n},$$

$$\mu_{B_2} \in \mathbf{E}\,|B_2| - \frac{n}{2\log n} \pm \frac{n}{2\log n}.$$

Plugging (5) in the bounds obtained above implies

$$\max(\mu_1, \mu_2) = \max\left[|M| + (1-\frac{1}{b})\mu_{M'} + \frac{1}{kb}\mu_{B_1}, (1-\frac{1}{b})|M| + \frac{1}{kb}\mu_{B_2}\right]$$

$$\leq \max\left[|M| + (1-\frac{1}{b})\mathbf{E}\,|M'| + \frac{1}{kb}\mathbf{E}\,|B_1|, (1-\frac{1}{b})|M| + \frac{1}{kb}\mathbf{E}\,|B_2|\right]$$

$$\leq \mu(G).$$

On the other hand, we have

$$\max(\mu_1, \mu_2) = \max\left[|M| + (1-\frac{1}{b})\mu_{M'} + \frac{1}{kb}\mu_{B_1}, (1-\frac{1}{b})|M| + \frac{1}{kb}\mu_{B_2}\right]$$

$$\geq \max\left[|M| + (1-\frac{1}{b})(\mathbf{E}\,|M'| - \frac{n}{\log n}) + \frac{1}{kb}(\mathbf{E}\,|B_1| - \frac{n}{\log n}), (1-\frac{1}{b})|M| + \frac{1}{kb}(\mathbf{E}\,|B_2| - \frac{n}{\log n})\right]$$

$$\geq \max\left[|M| + (1-\frac{1}{b})\mathbf{E}\,|M'| + \frac{1}{kb}\mathbf{E}\,|B_1|, (1-\frac{1}{b})|M| + \frac{1}{kb}\mathbf{E}\,|B_2|\right] - \frac{2n}{\log n}$$

$$\geq 0.5109 \cdot \mu(G) - \frac{2n}{\log n}$$

which completes the proof. $\qquad\square$

4.4   Running Time Analysis

For the analysis of the running time, we use a crucial property of random greedy maximal matching algorithm that was proved recently in [MRTV25].

Proposition 4.8 (Lemma 5.14 of [MRTV25])**.** *Let $Q(v)$ be the expected number of times that the oracle queries an adjacent edge of v if we start the oracle calls from a random vertex, for a random permutation over the edges of the graph G when running random greedy maximal matching. It holds that $Q(v) = \widetilde{O}(\deg_G(v)/|V(G)|)$.*

Proposition 4.9 (Corollary 5.15 of [MRTV25])**.** *Let $T(v)$ be the expected time needed to return a random neighbor of vertex v. Then, the expected time to run the random greedy maximal matching oracle for a random vertex and a random permutation in graph G is $\sum_{v\in V(G)} \widetilde{O}(T(v) \cdot \deg_G(v)/|V(G)|)$.*

Lemma 4.10. *Algorithm 4 runs in $\widetilde{O}(\bar{d}(G) \cdot \sqrt{n})$ time in expectation.*

Proof. Without loss of generality, we assume that $|V \setminus V(M)| = \Omega(n)$; otherwise, Algorithm 3 does not need to execute Algorithm 4, as the estimate from Algorithm 5 suffices.

The proof consists of two parts: first we show that line 6 of the algorithm can be implemented in $\widetilde{O}(\bar{d}(G))$ expected time, and then we prove that line 8 of the algorithm can be implemented in $\widetilde{O}(\bar{d}(G) \cdot \sqrt{n})$ expected time.

14

For the first part, let $G'' = G[V \setminus V(M)]$ and let v be a vertex in G''. In the adjacency list of v in the original graph, which contains $\deg_G(v)$ elements, only $\deg_{G''}(v)$ of them are neighbors in G''. Thus, to find a neighbor in G'', we need to randomly sample $T(v) = \deg_G(v)/\deg_{G''}(v)$ elements from v's adjacency list in expectation. Consequently, using Proposition 4.9, the expected time required to execute the random greedy maximal matching oracle for a randomly chosen vertex and permutation in G'' is

$$\sum_{v \in V(G'')} \widetilde{O}\left(\frac{T(v) \cdot \deg_{G''}(v)}{|V(G'')|}\right) = \sum_{v \in V(G'')} \widetilde{O}\left(\frac{(\deg_G(v)/\deg_{G''}(v)) \cdot \deg_{G''}(v)}{|V(G'')|}\right)$$

$$= \sum_{v \in V(G'')} \widetilde{O}\left(\frac{\deg_G(v)}{|V(G'')|}\right) = \widetilde{O}\left(\frac{|E(G)|}{|V(G'')|}\right) = \widetilde{O}\left(\frac{|E(G)|}{|V(G)|}\right) = \widetilde{O}(\bar{d}(G)).$$

In the algorithm, we choose a permutation π and run the random greedy maximal matching for $r = \widetilde{O}(1)$ randomly chosen vertices. By linearity of expectation, the expected running time of the first part is $\widetilde{O}(\bar{d}(G))$.

We prove the second part in a few steps. As a first step, for simplicity, assume that we have access to the adjacency list of G_1' and there is no need to run nested oracles. Then, using Proposition 3.1, the expected running time of line 8 is bounded by $\widetilde{O}(\bar{d}(G_1'))$ for $r = \widetilde{O}(1)$ random permutations and random vertices.

Now, suppose that we have access to the adjacency list of $G'' = G[V \setminus V(M)]$ instead of G_1'. By Proposition 4.9, the total expected runtime of the outer (b-matching) oracle is

$$\widetilde{O}\left(\sum_{v \in G_1'} T_{\text{outer}}(v) \cdot \deg_{G_1'}(v)/|V(G_1')|\right),$$

where $T_{\text{outer}}(v)$ is the expected time needed to find a random neighbor of v in G_1'. To find such a neighbor, we sample neighbors w of v in G'' and check whether w is matched in M' using the inner oracle. The expected number of these checks until a vertex w with $(v, w) \in E(G_1')$ is found is $\widetilde{O}(\deg_{G''}(v)/\deg_{G_1'}(v))$. The expected cost of each check is $\mathbf{E}_{w:(v,w) \in E(G'')}[\text{cost}_{\text{inner}}(w)]$, where $\text{cost}_{\text{inner}}(w)$ is the runtime of invoking the inner oracle to check if w is matched in M'. Putting this together, the expected total runtime is

$$\widetilde{O}\left(\frac{\sum_{v \in G_1'} T_{\text{outer}}(v) \cdot \deg_{G_1'}(v)}{|V(G_1')|}\right) = \widetilde{O}\left(\frac{1}{|V(G_1')|} \sum_{v \in G_1'} \frac{\deg_{G''}(v)}{\deg_{G_1'}(v)} \cdot \frac{\sum_{w:(v,w) \in E(G'')} \text{cost}_{\text{inner}}(w)}{\deg_{G''}(v)} \cdot \deg_{G_1'}(v)\right)$$

$$= \widetilde{O}\left(\frac{1}{|V(G_1')|} \sum_{w \in G''} \text{cost}_{\text{inner}}(w) \cdot \deg_{G''}(w)\right)$$

$$= \widetilde{O}\left(\sum_{w \in G''} \frac{\text{cost}_{\text{inner}}(w)}{|V(G'')|} \cdot \deg_{G''}(w)\right)$$

$$\overset{\textit{Lemma 4.3}}{=} \widetilde{O}\left(\sqrt{n} \cdot \mathbf{E}_{w \in G''}[\text{cost}_{\text{inner}}(w)]\right)$$

$$\overset{\textit{Proposition 4.9}}{=} \widetilde{O}\left(\sqrt{n} \cdot \sum_{v \in G''} \deg_{G''}(v) \cdot T_{\text{inner}}(v) \cdot \frac{1}{|V(G'')|}\right)$$

where $T_{\text{inner}}(v)$ is the expected time needed to find a random neighbor of v in G''. Under our assumption of having access to the adjacency list of G'', $T_{\text{inner}}(v) = O(1)$ and thus we can finish bounding with $\widetilde{O}(\sqrt{n} \cdot \bar{d}(G'')) = \widetilde{O}(\sqrt{n} \cdot \bar{d}(G))$.

Now we lift the assumption of having access to the adjacency list of G''. This means that, similarly as in the first part, we need to sample $T_{\text{inner}}(v) = \deg_G(v)/\deg_{G''}(v)$ vertices from the adjacency list of v in expectation (checking if a vertex is matched in M is done in $O(1)$ time). Then the total bound becomes

$$\widetilde{O}\left(\sqrt{n} \cdot \sum_{v \in G''} \deg_{G''}(v) \cdot \frac{\deg_G(v)}{\deg_{G''}(v)} \cdot \frac{1}{|V(G'')|}\right) = \widetilde{O}\left(\sqrt{n} \cdot \frac{|E(G)|}{|V(G'')|}\right) = \widetilde{O}(\sqrt{n} \cdot \bar{d}(G)).$$

Also $T_{\text{outer}}(v)$ increases by $\deg_G(v)/\deg_{G_1'}(v)$, which increases the total runtime only by

$$\widetilde{O}\left(\frac{1}{|V(G_1')|} \sum_{v \in G_1'} \frac{\deg_G(v)}{\deg_{G_1'}(v)} \cdot \deg_{G_1'}(v)\right) = \widetilde{O}(\bar{d}(G)).$$

$\square$

Lemma 4.11. *Algorithm 5 runs in $\widetilde{O}(\bar{d}(G))$ time in expectation.*

Proof. It is enough to show that it takes $\widetilde{O}(\bar{d}(G))$ time to run the algorithm of Proposition 3.1 (the b-matching oracle) for each sampled vertex and permutation, as we have $r = \widetilde{O}(1)$. We repeat the same arguments as in the proof of Lemma 4.10.

Let v be a vertex in the graph $G_2' = G[V(M), V \setminus V(M)]$. Note that in the adjacency list of v in the original graph, which contains $\deg_G(v)$ elements, only $\deg_{G_2'}(v)$ of the elements are neighbors in G_2'. Thus, we need to randomly sample $T(v) = \deg_G(v)/\deg_{G_2'}(v)$ elements of the adjacency list of v to find a neighbor in G_2'. Recall that we can check whether a vertex is matched in M in $O(1)$ time. Therefore, using Proposition 4.9, the expected time needed to run the random greedy maximal matching for a random vertex and permutation in G_2' is

$$\sum_{v \in V(G_2')} \widetilde{O}\left(\frac{T(v) \cdot \deg_{G_2'}(v)}{|V(G_2')|}\right) = \sum_{v \in V} \widetilde{O}\left(\frac{(\deg_G(v)/\deg_{G_2'}(v)) \cdot \deg_{G_2'}(v)}{n}\right)$$

$$= \sum_{v \in V} \widetilde{O}\left(\frac{\deg_G(v)}{n}\right) = \widetilde{O}\left(\frac{|E(G)|}{n}\right) = \widetilde{O}(\bar{d}(G)).$$

$\square$

Lemma 4.12. *Algorithm 3 runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

Proof. By Claim 4.2, the sparsification step takes $\widetilde{O}(n\sqrt{n})$ time. Also, by Lemmas 4.10 and 4.11, Algorithms 4 and 5 run in $\widetilde{O}(\bar{d}(G) \cdot \sqrt{n})$ and $\widetilde{O}(\bar{d}(G))$ expected time, respectively. To establish a high probability bound on the time complexity, we execute $O(\log n)$ instances of the algorithm in parallel and halt as soon as the first one completes. By applying Markov's inequality, we deduce that each individual instance terminates within $\widetilde{O}(n\sqrt{n})$ time with constant probability. As a result, with high probability, at least one of these instances finishes within $\widetilde{O}(n\sqrt{n})$ time. This concludes the proof. $\square$

Now we are ready to prove the final theorem of this section.

Theorem 4.13. *There exists an algorithm that, given access to the adjacency list of a bipartite graph, estimates the size of the maximum matching with a multiplicative-additive approximation factor of $(0.5109, o(n))$ and runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

Proof. By Lemma 4.7, Algorithm 3 achieves multiplicative-additive approximation of $(0.5109, o(n))$. Moreover, the running time of Algorithm 3 is $\widetilde{O}(n\sqrt{n})$ with high probability by Lemma 4.12. $\square$

5 Algorithm for General Graphs

The following is an analogue of Lemma 4.6 for general graphs.

Lemma 5.1. *In a general graph G, let M, M', B_1 and B_2 be as in the description of Algorithm 3. Then*

$$(1 - \varepsilon) \max \left[|M| + \mathbf{E}[\mu(M' \cup B_1)], \mathbf{E}[\mu(M \cup B_2)] \right] \geq 0.5109 \cdot \mu(G).$$

Proof. The proof proceeds as that of Lemma 4.6, with the following modifications:

- we invoke the second rather than the first part of Lemma 4.5 to obtain analogues of inequalities (3) and (4), i.e., instead of lower-bounding $|M| + (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1|$, we lower-bound $|M| + \mathbf{E}[\mu(M' \cup B_1)]$, and instead of lower-bounding $(1 - \frac{1}{b})|M| + \frac{1}{kb}|B_2|$, we lower-bound $\mathbf{E}[\mu(M \cup B_2)]$,

- the $(1 - \varepsilon)$ factor on the left-hand side of the statement can be folded into the $O(\varepsilon)$ term in γ at the end of the proof, for small enough ε.

$\square$

In this section, we show how to extend our algorithm to work for general graphs. The main difference between bipartite and general graphs is that the estimate based on the sizes of $|M|$, $|M'|$, $|B_1|$, and $|B_2|$ is insufficient to achieve a 0.5109 approximation guarantee. In Lemma 5.1, we show that we can achieve this approximation ratio by estimating $\mu(M \cup B_2)$ and $\mu(M' \cup B_1)$. More formally, to produce μ_1 and μ_2 in Algorithm 3, we use $|M| + \mu(M' \cup B_1)$ and $\mu(M \cup B_2)$, respectively. Also, Algorithm 4 and Algorithm 5 output $\mu(M' \cup B_1)$ and $\mu(M \cup B_2)$, respectively.

In both Algorithm 4 and Algorithm 5 we have access to oracles that can return whether a vertex is matched in M', B_1, or B_2 for a fixed permutation π. These oracles can also be used to return the edge of the matching if the vertex is matched, which is a corollary of Proposition 3.1 since the algorithm of Proposition 3.1 can be also used to return the edge of the matching.

Lemma 5.2. *For a vertex v, there exists an algorithm that returns the edges of v in M' and B_1 in $\widetilde{O}(\bar{d}(G) \cdot \sqrt{n})$ expected time.*

Proof. The proof follows from Lemma 4.10 and the fact that the algorithm in Proposition 3.1 can return the edge of the matched vertex in the same running time. $\square$

Lemma 5.3. *For a vertex v, there exists an algorithm that returns the edges of v in M and B_2 in $\widetilde{O}(\bar{d}(G))$ expected time.*

Proof. If v is matched in M, we can return the edge in $O(1)$ time since we have access to this maximal matching. The rest of the proof follows from Lemma 4.11 and the fact that the algorithm in Proposition 3.1 can return the edge of the matched vertex in the same running time. $\square$

Local computation algorithms (LCA) is a model of computation, also motivated by large data sets, in which the algorithm is not expected to produce the entire output at once. Instead, the algorithm is queried for parts of the output, and must produce a consistent and approximately optimal output. We use the following local computation algorithm (LCA) by [LRY17] to design our algorithm.

Proposition 5.4 ([LRY17])**.** *There exists a $(1 - \varepsilon)$-approximate local computation algorithm for maximum matching of graph G in $\widetilde{O}(\Delta(G)^{1/\varepsilon^2})$ time with access to the adjacency list of G.*

Now we prove the main technical part of this section that can be used to estimate $\mu(M' \cup B_1)$ and $\mu(M \cup B_2)$.

Lemma 5.5. *Let H be a subgraph of graph G. Suppose that H is the union of a constant number of random greedy maximal matching on different subsets of vertices. Also, we have oracle access to edges of random greedy maximal matching. We can query a vertex to obtain the matching edge of vertex v in $\widetilde{O}(T)$ expected time. Moreover, the maximum degree of H is constant. Then, there exists a $(1 - \varepsilon)$-approximate algorithm that estimates the size of the maximum matching of H in $\widetilde{O}(T)$ expected time.*

Proof. We run the algorithm of Proposition 5.4. Each time the algorithm visits a new vertex u, we first query all the constant number of oracles for random greedy maximal matching to get the adjacency list of u in H. If the algorithm in Proposition 5.4 made uniform queries to the oracle, then we could conclude the proof since $\Delta(H)$ and ε are constant. However, note that queries to the oracles are not independent and we have an expected time guarantee on the running time of the oracles. So it is possible that the way that the algorithm of Proposition 5.4 works might result in making biased queries to oracles.

Let $\mathcal{L}(u, \pi)$ be the set of vertices the algorithm of Proposition 5.4 visits when we run the algorithm for vertex u and we use permutation π for random greedy maximal matchings of the subgraph H. Let $F(u, \pi)$ be the running time of the oracle for vertex u and permutation π. Also, let $L(u, \pi)$ be the running time of the algorithm of Proposition 5.4 on vertex u using permutation π for random greedy maximal matchings. Hence, we have

$$\mathbf{E}_{u,\pi}[F'(u, \pi)] = \sum_{\pi} \sum_{u} \sum_{v \in \mathcal{L}(u,\pi)} \frac{\mathbf{E}[F(v, \pi)]}{n \cdot |E(G)|!}$$

Further, the algorithm in Proposition 5.4 explores the local neighborhood of a vertex to answer each query. Therefore, if two vertices w and z are at a distance greater than $\Delta(H)^{1/\varepsilon^3}$, the algorithm does not visit z when answering a query for w. Thus,

$$\mathbf{E}_{u,\pi}[F'(u, \pi)] \leq \sum_{\pi} \sum_{v} \Delta(H)^{\left(\Delta(H)^{1/\varepsilon^3}\right)} \cdot \frac{\mathbf{E}[F(v, \pi)]}{n \cdot |E(G)|!}$$

$$= \Delta(H)^{\left(\Delta(H)^{1/\varepsilon^3}\right)} \cdot \sum_{\pi} \sum_{v} \frac{\mathbf{E}[F(v, \pi)]}{n \cdot |E(G)|!}$$

$$\leq \widetilde{O}(T),$$

where the first inequality follows by the fact that each vertex has at most $\Delta(H)^{\left(\Delta(H)^{1/\varepsilon^3}\right)}$ neighbors in H within a distance of $\Delta(H)^{1/\varepsilon^3}$, and the last inequality because $\Delta(H)$ and ε are constants. Therefore, even with the biased queries, we can implement Proposition 5.4 on subgraph H in $\widetilde{O}(T)$ time, which completes the proof. □

Lemma 5.6. *There exists an algorithm that outputs a $(1 - \varepsilon)$-approximate estimation of the value of $\mu(M \cup B_2)$ in $\widetilde{O}(\bar{d}(G))$ expected time.*

Proof. The proof follows by plugging Lemma 5.3 into Lemma 5.5 and running the algorithm for r samples. □

Lemma 5.7. *There exists an algorithm that outputs a $(1 - \varepsilon)$-approximate estimation of the value of $\mu(M' \cup B_1)$ in $\widetilde{O}(\bar{d}(G) \cdot \sqrt{n})$ expected time.*

Proof. The proof follows by plugging Lemma 5.2 into Lemma 5.5 and running the algorithm for r samples. $\qquad\square$

Theorem 5.8. *There exists an algorithm that, given access to the adjacency list of a graph, estimates the size of the maximum matching with a multiplicative-additive approximation factor of $(0.5109, o(n))$ and runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

Proof. By Lemma 5.1 and a Chernoff bound similar to Lemma 4.7, we achieve a multiplicative-additive approximation of $(0.5109, o(n))$. Moreover, the expected running time is $\widetilde{O}(n\sqrt{n})$ by Claim 4.2, Lemma 5.6, and Lemma 5.7. To establish a high probability bound on the time complexity, we execute $O(\log n)$ instances of the algorithm in parallel and halt as soon as the first one completes. By applying Markov's inequality, we deduce that each individual instance terminates within $O(n\sqrt{n})$ time with constant probability. As a result, with high probability, at least one of these instances finishes within $O(n\sqrt{n})$ time. This concludes the proof. $\qquad\square$

6 Multiplicative Approximation

In this section, we show that we can achieve a multiplicative approximation guarantee by slightly increasing the number of samples in Algorithm 4 and Algorithm 5. First, we prove a simple lower bound for the size of the maximum matching of a graph based on its maximum and average degree.

Claim 6.1. *For any graph G, it holds that $\mu(G) \geq n\bar{d}(G)/(4\Delta(G))$.*

Proof. Any graph with a maximum degree of $\Delta(G)$ can be colored greedily using $2\Delta(G)$ colors. Furthermore, the edges of each color create a matching. Thus, we have $\mu(G) \geq |E(G)|/(2\Delta(G))$. Combining with $|E(G)| = n\bar{d}(G)/2$, we can conclude the proof. $\qquad\square$

The goal is to obtain a multiplicative approximation guarantee of $(0.5109 - \varepsilon)\mu(G)$. It is important to note that if any of $|M|$, $|M'|$, $|B_1|$, or $|B_2|$ is not a constant fraction of the others, it can be omitted from the equation in the statement of Lemma 4.6 without affecting the approximation by more than a function of ε. Thus, without loss of generality, we can assume that $|M| = \Omega(\mu(G))$, $|M'| = \Omega(\mu(G))$, $|B_1| = \Omega(\mu(G))$, and $|B_2| = \Omega(\mu(G))$. Consequently, using Claim 6.1 and as an application of Chernoff bound, we can use $\widetilde{O}_\varepsilon(\Delta(G)/\bar{d}(G))$ samples in Algorithm 4 and Algorithm 5 to obtain multiplicative estimation of $\mu_{M'}$, μ_{B_1}, and μ_{B_2}.

By Lemma 4.10, Algorithm 4 runs in $\widetilde{O}(\bar{d}(G) \cdot \sqrt{n})$ time when we have $r = \widetilde{O}(1)$ samples. By increasing the number of samples to $\widetilde{O}_\varepsilon(\Delta(G)/\bar{d}(G))$, the running time of Algorithm 4 increases to $\widetilde{O}(\Delta(G) \cdot \sqrt{n})$. Moreover, by Lemma 4.11, Algorithm 5 runs in $\widetilde{O}(\bar{d}(G))$ time when we have $r = \widetilde{O}(1)$ samples. By increasing the number of samples to $\widetilde{O}_\varepsilon(\Delta(G)/\bar{d}(G))$, the running time of Algorithm 5 increases to $\widetilde{O}(\Delta(G))$. Therefore, the total running time of the algorithm is within $\widetilde{O}(n\sqrt{n})$.

Finally, we can obtain the degree of each vertex in the graph using binary search. Therefore, we can assume that we have access to $\Delta(G)$ and $\bar{d}(G)$ by spending $\widetilde{O}(n)$ time. Thus we get:

Theorem 1.1. *There exists an algorithm that, given access to the adjacency list of a graph, estimates the size of the maximum matching with a multiplicative approximation factor of 0.5109 and runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

7 Algorithm with Access to the Adjacency Matrix

In this section, we use a simple reduction to show that with a small modification, our algorithm can be adapted to the setting where we have access to the graph's adjacency matrix. A slightly different version of this kind of reduction appeared in previous works on sublinear time algorithms for maximum matching [Beh21, BRRS23].

It is important to note that obtaining a constant-factor multiplicative approximation is impossible when the algorithm only has access to the adjacency matrix of the graph. This is because if the graph is guaranteed to contain either a single edge or be completely empty, any algorithm would require $\Omega(n^2)$ adjacency matrix queries to distinguish between these two cases. Consequently, we allow the algorithm to have an additive error of $o(n)$ in addition to the multiplicative approximation ratio.

We build an auxiliary graph H with the following vertex set and edge set:

- **Vertex set:** $V(H)$ contains $n + 2$ disjoint sets of n vertices V_1, V_2, and $U_1, \ldots, U_n$. Each V_i is a copy of the vertices of the original graph. Each U_i contains $n \log^2 n$ vertices.

- **Edge set:** For each vertex $v \in V_1$, the i-th neighbor of v is the i-th vertex in V_1 if $(v, i) \in E(G)$, and otherwise it is the i-th vertex in V_2. Similarly, for each vertex $v \in V_2$, the i-th neighbor of v is i-th vertex in V_2 if $(v, i) \in E(G)$, and otherwise it is the i-th vertex in V_1. Also, each $v \in V_2$ is connected to all vertices of U_v. As a result, the degree of each vertex in $U_1 \cup U_2 \cup \ldots \cup U_n$ is 1, the degree of each vertex in V_2 is n, and the degree of each vertex in V_2 is $n + n \log^2 n$. Therefore, we have $\Delta(H) = \widetilde{O}(n)$.

Because of the way we constructed the graph, it is not hard to see that we can find the i-th neighbor of the adjacency list of each vertex in H using only a single query to the adjacency matrix of G.

Observation 7.1. *For each vertex v in graph H, the i-th neighbor of H can be found using at most a single adjacency matrix query in G.*

Proof. If $i > \deg_H(v)$, we can simply certify this since the degree of each vertex in H is determined by the vertex set to which v belongs. If $v \in U_j$, then the only neighbor of v is j in V_2. If $v \in V_1$, we make an adjacency matrix query for (v, i), and based on the result, the i-th neighbor is either vertex i in V_1 or in V_2. A similar procedure applies for $v \in V_2$ when $i \leq n$. If $i > n$ for $v \in V_2$, we return the $(i - n)$-th vertex in U_v. $\square$

Modification to the algorithm: Since the graph contains $\widetilde{\Theta}(n^2)$ vertices, we cannot afford to apply the sparsification step to all vertices. However, vertices in $U_1 \cup \ldots \cup U_n$ have degree 1. Therefore, we apply the sparsification step only to vertices in V_1 and V_2. Since we have $|V_1| + |V_2| = 2n$, we can apply the sparsification for these sets in $\widetilde{O}(n\sqrt{n})$ time. We first iterate over the vertices in V_2 and apply the sparsification step, and then we apply it to the vertices in V_1. This ordering ensures that most vertices in V_2 get matched to vertices in $U_1 \cup \ldots \cup U_n$ in this step, which is desirable for our application.

Claim 7.2. *After the sparsification step, each vertex in V_2 is matched by M with high probability. Moreover, at most $n / \log n$ vertices in V_2 are matched to vertices in $V_1 \cup V_2$ with high probability.*

Proof. Note that we first process the vertices in V_2. At the time we process a vertex $v \in V_2$, it has at least $n \log^2 n$ neighbors due to its neighbors in U_v. Since, after sparsification, each vertex must

20

have a degree of $\widetilde{O}(\sqrt{n})$ with high probability by Lemma 4.3, all vertices in V_2 will be matched with high probability.

Additionally, at the time we process a vertex $v \in V_2$, it has $n \log^2 n$ unmatched neighbors in U_v. On the other hand, it has at most n neighbors in the rest of the graph. Thus, with a probability of at most $1/\log^2 n$, it gets matched to a vertex outside U_v. Therefore, in expectation, at most $n/\log^2 n$ vertices in V_2 are matched to vertices in $V_1 \cup V_2$. Using the Chernoff bound, we can conclude that at most $n/\log n$ vertices in V_2 are matched to vertices in $V_1 \cup V_2$ with high probability. □

Equipped with this reduction, we can now simply run the rest of the algorithm for vertices in V_1. The only difference is that we exclude the edges of M that lie between V_2 and $U_1 \cup \ldots \cup U_n$ in the estimation. Additionally, in the final estimation, the algorithm returns the previous estimate minus $n/\log n$, accounting for the vertices in V_2 that are not matched within $V_1 \cup V_2$, which introduces an additional $o(n)$ additive error. Thus we obtain:

Theorem 1.2. *There exists an algorithm that, given access to the adjacency matrix of a graph, estimates the size of the maximum matching with a multiplicative-additive approximation factor of $(0.5109, o(n))$ and runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

References

[ABR24]   Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. Fully dynamic matching: $(2 - \sqrt{2})$-approximation in polylog update time. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3040–3061. SIAM, 2024.

[ARVX12]   Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-Efficient Local Computation Algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1132–1139, 2012.

[Beh21]   Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 873–884. IEEE, 2021.

[Beh23]   Soheil Behnezhad. Dynamic algorithms for maximum matching size. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 129–162. SIAM, 2023.

[BKS23a]   Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1 + \varepsilon)$-approximate matching size in truly sublinear update time. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1563–1588. IEEE, 2023.

[BKS23b]   Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Sublinear algorithms for $(1.5+\varepsilon)$-approximate matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 254–266, 2023.

[BKSW23]   Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In Nikhil

Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 100–128. SIAM, 2023.

[BRR23a]   Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Local computation algorithms for maximum matching: New lower bounds. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2322–2335. IEEE, 2023.

[BRR23b]   Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 267–280, 2023.

[BRR24]   Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Approximating maximum matching requires almost quadratic time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 444–454, 2024.

[BRRS23]   Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. Beating greedy matching in sublinear time. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3900–3945. SIAM, 2023.

[CKK20]   Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear algorithms and lower bounds for metric tsp cost estimation. *arXiv preprint arXiv:2006.05490*, 2020.

[Gha22]   Mohsen Ghaffari. Local computation of maximal independent set. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 438–449, 2022.

[KMNFT20]   Michael Kapralov, Slobodan Mitrović, Ashkan Norouzi-Fard, and Jakab Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1753–1772. SIAM, 2020.

[KN21]   Christian Konrad and Kheeran K. Naidu. On two-pass streaming algorithms for maximum bipartite matching. In Mary Wootters and Laura Sanita, 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, 2021.

[LRY17]   Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, 77(4):971–994, 2017.

[MRTV25]   Sepideh Mahabadi, Mohammad Roghani, Jakub Tarnawski, and Ali Vakilian. Sublinear Metric Steiner Tree via Improved Bounds for Set Cover. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*, volume 325 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 74:1–74:24, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[NO08]   Huy N Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *2008 49th annual IEEE symposium on foundations of computer science*, pages 327–336. IEEE, 2008.

[PR07]   Michal Parnas and Dana Ron. Approximating the Minimum Vertex Cover in Sublinear Time and a Connection to Distributed Algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007.

[RTVX11]   Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 223–238, 2011.

[YYI09]   Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 225–234. ACM, 2009.