Dynamic $(1+\epsilon)$-Approximate Matching Size in Truly Sublinear Update Time

Sayan Bhattacharya[*†] Peter Kiss[‡] Thatchaphol Saranurak[§¶]

Abstract

We show a fully dynamic algorithm for maintaining $(1+\epsilon)$-approximate *size* of maximum matching of the graph with n vertices and m edges using $m^{0.5-\Omega_\epsilon(1)}$ update time. This is the first polynomial improvement over the long-standing $O(n)$ update time, which can be trivially obtained by periodic recomputation. Thus, we resolve the value version of a major open question of the dynamic graph algorithms literature (see, e.g., [Gupta and Peng FOCS'13], [Bernstein and Stein SODA'16], [Behnezhad and Khanna SODA'22]).

Our key technical component is the first sublinear algorithm for $(1, \epsilon n)$-approximate maximum matching with sublinear running time on *dense* graphs. All previous algorithms suffered a multiplicative approximation factor of at least 1.499 or assumed that the graph has a very small maximum degree.

[*]University of Warwick s.bhattacharya@warwick.ac.uk

[†]Supported by Engineering and Physical Sciences Research Council, UK (EPSRC) Grant EP/S03353X/1.

[‡]University of Warwick, Max-Planck-Institut für Informatik peter.kiss@warwick.ac.uk

[§]University of Michigan thsa@umich.edu

[¶]Supported by NSF CAREER grant 2238138.

Contents

1 Introduction

We study the dynamic version of the maximum matching problem, a cornerstone of combinatorial optimization [Kuh55, Edm65b, Edm65a]. In the *dynamic matching* problem, the task is to build a data structure that, given a graph G with n vertices and m edges undergoing both edge insertions and deletions, maintains an (approximate) maximum matching of G or, in the value version, just the size of the maximum matching, denoted by $\mu(G)$. The goal is to minimize the *update time* required to update the solution after each edge change.

The first non-trivial algorithm for this problem was by Sankowski [San07] 15 years ago, which exactly maintains the maximum matching *size* using $O(n^{1.495})$ update time, which is recently improved to $O(n^{1.407})$ [BNS19]. Unfortunately, this latter bound is tight under the *hinted OMv* conjecture [BNS19]. Furthermore, in sparse graphs, even $m^{1-o(1)}$ update time is required assuming the k-cycle conjecture [PGVWW20]. These strong conditional lower bounds have shifted the attention of researchers to *approximate* matching. An α-approximate matching is a matching of size at least $\mu(G)/\alpha$. The following has become one of the holy-grail questions in the dynamic graph algorithms and fine-grained complexity communities [ARW17]:

Question 1.1. *Is there a dynamic $(1+\epsilon)$-approximate matching algorithm with polylogarithmic update time for an arbitrarily small constant ϵ?*

The current state of the art is, however, still very far from this goal. A straightforward algorithm with $O(n)$ amortized update time is to simply recompute a $(1+\epsilon)$-approximate matching from scratch in $O(m)$ time [DP14] every after $2\epsilon m/n$ edge updates.[1] Surprisingly, this easy $O(n)$ bound already captures the limitation of all known techniques! An improved algorithm with $O(\sqrt{m})$ update time was given ten years ago by Gupta and Peng [GP13], but it still takes $O(n)$ time in dense graphs. Very recently, Assadi et al. [ABKL23] showed how to obtain $O(n/(\log^* n)^{\Omega(1)})$ update time, but their regularity-lemma-based approach inherently cannot give an improvement larger than a $2^{\Theta(\sqrt{\log n})} = n^{o(1)}$ factor. Until now, no dynamic $(1+\epsilon)$-approximate algorithms can break through the naive $O(n)$ barrier by a polynomial factor.[2]

Intensive research during the last decade instead showed how to speed up update time by relaxing the approximation factor. The influential work by Onak and Rubinfeld [OR10] gave the first dynamic matching algorithm with polylogarithmic update time that maintains a large constant approximate maximum matching. Then, Baswana, Gupta and Sen [BGS11] showed a dynamic maximal matching with logarithmic update time, which gives 2-approximation. A large body of work then refined this result in various directions, including constant update time [Sol16], deamortization [CS18, BFH19, Kis22], and derandomization [BHN16, ACC+18, BK19, Waj20, BK21]. In 2015, Bernstein and Stein [BS15, BS16] showed a novel approach for maintaining a $(3/2+\epsilon)$-approximate matching using $\tilde{O}(m^{1/4}) = \tilde{O}(\sqrt{n})$ update time.[3] Refinement of this approach and new trade-off results with approximation in the range $(3/2, 2)$ were also intensively studied [BLM20, GSSU22, Kis22, BK22, RSW22]. All these techniques, however, seem to get stuck at $(3/2)$-approximation.

Very recently, the above long-standing trade-off was improved by Behnezhad [Beh23] and, independently, by Bhattacharya et al. [BKSW23] via a new connection to sublinear and streaming algorithms. To maintain maximum matching *size*, they gave 1.973-approximation algorithms with polylogarithmic update time, and, on bipartite graphs, Behnezhad [Beh23] pushed it further to $(3/2 - \Omega(1))$-approximation in $\tilde{O}(\sqrt{n})$ update time. While this new connection is very inspiring,

[1] See Appendix C for the proof of this simple algorithm.

[2] In contrast, in the *partially* dynamic setting where graphs undergo edge insertions only or edge deletions only, there are many algorithms with polylogarithmic amortized update time [GRS14, GLS+19, BGS20, JJST22, BKSW23].

[3] We use $\tilde{O}(\cdot)$ to hide polylog(n) factor throughout the paper.

it has been a key open problem [BRRS23] whether non-trivial $(1+\epsilon)$-approximate matching algorithms in dense graphs exist in the sublinear model. Hence, it remains unclear whether an improved dynamic $(1+\epsilon)$-approximation algorithm is possible via this new connection or even possible at all.

Indeed, in this paper, we give the first dynamic $(1+\epsilon)$-approximate matching *size* algorithm that finally improves the $O(n)$ bound by a polynomial factor, formally stated below.

Theorem 1.2. *There is a dynamic $(1+\epsilon)$-approximate matching* size *algorithm with $m^{0.5-\Omega_\epsilon(1)}$ worst-case update time.*

The algorithm is randomized and works against an adaptive adversary with high probability. Moreover, the algorithm maintains $(1+\epsilon)$-approximate matching M of G in the sense that, given a vertex v, it can return a matched edge $(v,v') \in M$ or $\perp$ if $v \notin V(M)$ in $m^{0.5+f(\epsilon)}$ time, where f is an increasing function such that $f(\epsilon) \to 0$ when $\epsilon \to 0$.

It has been asked repeatedly [GP13, BS15, BS16] whether there exists a dynamic $(1+\epsilon)$-approximate matching algorithm with $m^{0.5-\Omega_\epsilon(1)}$ update time. Theorem 1.2 thus gives an affirmative answer to the value version of this open question. Although the matching is not explicitly maintained in Theorem 1.2, it still supports queries whether a vertex is matched or not. The recent algorithms that only maintain the estimate of $\mu(G)$ by [Beh23, BKSW23] inherently cannot support this query.

We obtain Theorem 1.2 by making progress in sublinear algorithms: we show the first sublinear $(1,\epsilon n)$-approximate matching algorithm with truly sublinear time even in dense graphs. Here, an (α,β)-approximate matching means a matching of size at least $\mu(G)/\alpha-\beta$. Given our new sublinear matching algorithm summarized below, Theorem 1.2 follows using known techniques.

Theorem 1.3. *There is a randomized algorithm that, given the adjacency matrix of a graph G, in time $n^{2-\Omega_\epsilon(1)}$ computes with high probability a $(1,\epsilon n)$-approximation $\tilde{\mu}$ of $\mu(G)$.*

After that, given a vertex v, the algorithm returns in $n^{1+f(\epsilon)}$ time an edge $(v,v') \in M$ or $\perp$ if $v \notin V(M)$ where M is a fixed $(1,\epsilon n)$-approximate matching, where f is an increasing function such that $f(\epsilon) \to 0$ when $\epsilon \to 0$.

We note that the additive approximation factor in Theorem 1.3 is unavoidable for sublinear algorithms with access to only the adjacency matrix: checking whether there is zero or one edge requires $\Omega(n^2)$ adjacency matrix queries.

Behnezhad et al. [BRRS23] posted an open question about sublinear matching algorithms as follows "ruling out say a 1.01-approximation in $n^{2-\Omega(1)}$ time would also be extremely interesting."[4]. Since the additive approximation factor is unavoidable for algorithms using the adjacency matrix only, the analogous question becomes whether one can rule out a $(1,n/100)$-approximation in $n^{2-\Omega(1)}$ time. Theorem 1.3 answers this question negatively since we can get arbitrarily good additive approximation in $n^{2-\Omega(1)}$ time.

To put Theorem 1.3 into the larger context of sublinear matching literature, let us discuss its history below. We use Δ and d to denote the maximum and average degree of the graph respectively.

Approximating $\mu(G)$. One of the main goals in this area, initiated by Parnas and Ron [PR07], is to approximate the *size* of maximum matching $\mu(G)$ in sublinear time when given access to the adjacency list and matrix of an input graph. Early research on this topic focused on obtaining $O(1)$ time algorithms when $\Delta = O(1)$. However, these early work [PR07, NO08, YYI12] may require $\Omega(n^2)$ time on general graphs. This drawback was first addressed in [KMNFT20] and [CKK20] (based on [ORRR12]), both of which were then subsumed by the algorithms of Behnezhad [Beh22] that compute a $(2,o(n))$-approximation in $\widetilde{O}(d+1)$ time. His algorithms are near-optimal and settle the problem in the regime of approximation ratio at least 2.

[4] In [RSW22], they use different notation and write 0.99-approximation instead of 1.01.

Subsequent work focuses on optimizing the approximation ratio within $n^{2-\Omega(1)}$ time. To compare with Theorem 1.3, let us discuss only algorithms that use the adjacency matrix. Behnezhad et al. [BRRS23] first broke the 2-approximation barrier by computing a $(2-\Omega_\gamma(1), o(n))$-approximate matching in $\tilde{O}(n^{1+\gamma})$ time. Then $(3/2, \epsilon n)$-approximation algorithms with $n^{2-\Theta(\epsilon^2)}$ time were shown independently in [BKS23, BRR23]. Behnezhad et al. [BRR23] improved this further to $(3/2 - \Omega(1), o(n))$-approximation in $n^{2-\Omega(1)}$ time on bipartite graphs.[5] We summarize the previous work in Table 1.

By the first part of Theorem 1.3, we show that even $(1, \epsilon n)$-approximation is possible in $n^{2-\Omega_\epsilon(1)}$ time. As we mentioned, this result addresses the open question of [RSW22]. It remains very interesting to see an optimal approximation-time trade-off for this problem.

Matching Oracles. In the area of *local computation algorithms* (LCA), initiated by Robinfeld et al. [RTVX11, ARVX12], we want a *matching oracle* for some fixed approximate matching M such that, given any vertex v, return $(v, v') \in M$ or $\perp$ if $v \notin V(M)$. The goal is to optimize the approximation ratio of M and minimize the worst-case query time over *all* vertices. Note that, given a matching oracle for an α-approximate matching, we can compute $(\alpha, \epsilon n)$-approximation of $\mu(G)$ by simply querying the oracle at $O(1/\epsilon^2)$ random vertices. So this is stronger than the previous goal.

The worst-case guarantee over all vertices is stronger than the expected query time for *each* vertex [NO08] or for just a *random* vertex [YYI12, Beh22], which is even weaker. This strong guarantee is useful for bounding the query time of *adaptive queries*, which depend on answers of the previous queries, and is crucial in some applications [LRV22]. Our approach for "boosting" the approximation ratio also requires adaptive queries and hence needs worst-case guarantees.

A long line of work [RTVX11, ARVX12, RV16, LRY15, Gha16, GU19, Gha22] focused on building an oracle for maximal independent sets (which implies a 2-approximate matching oracle) and culminated in an oracle by Ghaffari [Gha22] that uses $\text{poly}(\Delta \log n)$ query time with high probability. Levi et al. [LRY15] also a showed $(1+\epsilon)$-approximate matching oracle with $\Delta^{O(1/\epsilon^2)}\text{polylog}(n)$ query complexity. However, all these algorithms are not sublinear in dense graphs. In this regime, the only non-trivial matching oracle was by Kapralov et al. [KMNFT20] and has $\tilde{O}(\Delta)$ query time, but the approximation ratio is only a large constant and is in expectation. We summarize the previous work in Table 2.

The second part of Theorem 1.3 gives the first non-trivial matching oracle on dense graphs whose multiplicative approximation ratio is a small constant, which is 1 in our case, but we need to pay additive approximation factor.

Summary. Our main result, Theorem 1.2, is the first dynamic $(1+\epsilon)$-approximate matching size algorithm with $m^{0.5-\Omega_\epsilon(1)}$ update time, breaking through the naive yet long-standing $O(n)$ barrier by a polynomial factor. Our key technical component, Theorem 1.3, makes progress in the area of sublinear-time matching algorithms on *dense* graphs. Among algorithms for approximating $\mu(G)$ only, we improve the best approximation ratio from $(3/2-\Omega(1), o(n))$ by [BRR23] to $(1, \epsilon n)$. Among LCAs, it is the first one on dense graphs whose multiplicative approximation is a small constant.

Organization. First, we give an overview of our algorithms in Section 2. Then, we set up notations and give preliminaries in Section 3. In Section 4, we present a key building block which is a matching oracle for an induced graph $G[A]$ where A is unknown to us. Using this, we show in Section 5 how to boost the approximation ratio of any matching oracle. By repeatedly boosting the approximation ratio, we give a $(1, \epsilon n)$-approximate matching oracle (Theorem 1.3) in Section 6. Finally, we combine this oracle with known techniques in dynamic algorithms to Theorem 1.2 in Section 7.

[5] [BRR23] also announced a $\Omega(n^{1.2})$-time lower bound for $(3/2 - \Omega(1), o(n))$-approximation.

Acknowledgement

We would like to thank David Wajc for suggesting the implementation of McGregor's algorithm in the sub-linear model.

2 Technical Overview

Our high-level approach is based on the interconnection between dynamic, sublinear, and streaming algorithms. This connection differs from the ones used in the recent results of [BKSW23, Beh23]. For example, the dynamic $(2-\Omega(1))$-approximate algorithms in [BKSW23, Beh23] are inspired by the two-pass streaming algorithms (e.g. [KMM12]). Then, they use sublinear algorithms [Beh22] to implement this streaming algorithm in the dynamic setting efficiently.[6] In contrast, it is our sublinear algorithm, not dynamic algorithm, that is inspired by the $O(1)$-pass streaming algorithm [McG05]. Below, we explain the overview of our sublinear algorithm, which consists of two key ingredients, and then explain how our dynamic algorithm easily follows.

Ingredient I: Reduction from $(1, \gamma n)$-Approximation to Arbitrarily Bad Approximation.

An initial observation is that the streaming algorithm by McGregor [McG05] can be viewed as the following reduction: one can compute a $(1+\gamma)$-approximate matching by making $O_\gamma(1)$ calls to a subroutine that, given $S \subseteq V$, returns a $O(1)$-approximate matching of the induced subgraph $G[S]$.

We observe that a much weaker subroutine suffices when additive approximation is allowed. Let $\texttt{LargeMatching}(S, \delta)$ be a subroutine that, given $S \subseteq V$ and δ, returns a matching M in $G[S]$ such that if $\mu(G[S]) \geq \delta n$, then $|M| \geq \Omega(\text{poly}(\delta)n)$. Note that the approximation of M can be arbitrarily bad depending of δ. By adapting McGregor's algorithm, we show how to compute a $(1, \gamma n)$-approximate matching using only $t = O_\gamma(1)$ calls to

$$\texttt{LargeMatching}(S_1, \delta_1), \ldots, \texttt{LargeMatching}(S_t, \delta_t)$$

where each δ_i is a small constant depending on γ. This algorithm, denoted by $\texttt{Alg}(\gamma)$, is our *template* algorithm (detailed in Section 5.1), which we will try to implement in the sublinear setting.

Additionally, we observe that each vertex set S_i can be determined in a very local manner. More precisely, a *membership-query* of the form "is a vertex $v \in S_i$?" can be answered by making only $q = O_\gamma(1)$ *matching-queries* of the form "is a vertex $u \in V(M_j)$? if so, return $(u, u') \in M$" where $j < i$ and M_j is the output of $\texttt{LargeMatching}(S_j, \delta_j)$ previously computed.

However, the big challenge in the sublinear model, unlike the streaming model, is that even the weak subroutine like $\texttt{LargeMatching}(\cdot)$ is impossible.[7] Even worse, if we could not compute each matching M_j explicitly for $j < i$, then how can we answer a membership-query whether $v \in S_i$? Note that known sublinear algorithms for estimating the matching size of $G[S]$ are not useful here.

The above obstacle leads us to our second ingredient. We show that at least the oracle version of $\texttt{LargeMatching}(\cdot)$ can be implemented in the sublinear model. Later, we will explain why it is strong enough for implementing the template algorithm $\texttt{Alg}(\gamma)$ in the sublinear model.

[6] The dynamic $(3/2-\Omega(1))$-approximate algorithm in [Beh23] does not have explicit relationship to streaming algorithms. It is obtained using sublinear algorithms to improve the (3/2)-approximation guarantee of the tight instances of EDCS.

[7] Think of a $n \times n$ bipartite graph which consists only of a perfect matching. Using $o(n^2)$ adjacency-matrix queries, it is not possible to out $\Omega(n)$ matching edges in this input instance. The lower bound can be extended even if we allow adjacency-list queries by adding ϵn dummy vertices, each of which connects to every other vertex.

Ingredient II: Large Matching Oracles on Induced Subgraphs.

Suppose that a vertex set $A \subseteq V$ is unknown to us but a membership-query of A, i.e., checking if $v \in A$, can be done in $n^{1+\epsilon}$ time. Given access to the adjacency matrix of G, we show how to construct an oracle $\texttt{LargeMatchingOracle}(A, \delta, \epsilon)$ with the following guarantee:

> Using $\tilde{O}_\delta\left(n^{2-\epsilon}\right)$ preprocessing time, we obtain an oracle that supports matching-queries for a matching M in $G[A]$ with $\tilde{O}_\delta\left(n^{1+g(\epsilon)}\right)$ query time where $\epsilon \leq g(\epsilon) = O(\epsilon)$. If $\mu(G[A]) \geq \delta n$, then $|M| = \Omega(\text{poly}(\delta)n)$ whp.

The main challenge of implementing $\texttt{LargeMatchingOracle}(A, \delta, \epsilon)$ in the sublinear model is that we want to find a large matching on the *induced subgraph* $G[A]$. The challenge comes from possible $\Omega(n^2)$ edges between A and $V \setminus A$, and we must avoid reading these edges to get sublinear time. It turns out that this challenge can be overcome. We use the idea that appeared before in the algorithm of [BRR23] in a different context of estimating $(3/2 - \Omega(1))$-approximation $\mu(G)$ on bipartite graphs. See the details in Section 4.

Given the above two ingredients, we can combine them to get our main results in the sublinear and dynamic settings, as follows.

Result I: $(1, \gamma n)$-Approximate Matching Oracles in $n^{2-\Omega_\gamma(1)}$ Time.

Now, we show how to implement the template algorithm $\texttt{Alg}(\gamma)$ in $n^{2-\Omega_\gamma(1)}$ time. Let $\epsilon \in (0, 1)$ be a small constant where $\lim_{\gamma \to 0} \epsilon = 0$. Let $\epsilon_0 = \epsilon$ and $\epsilon_i = g(\epsilon_{i-1})$ for all $i \in [1, t]$ where g is the function in the guarantee of Ingredient II. So $\epsilon = \epsilon_0 \leq \epsilon_1 \leq \cdots \leq \epsilon_t$ and $\lim_{\gamma \to 0} \epsilon_t = 0$.

We simply replace each call to $\texttt{LargeMatching}(S_i, \delta_i)$ with $\texttt{LargeMatchingOracle}(S_i, \delta_i, \epsilon_{i-1})$. Now, by induction on $i \in [1, t]$, we will show that we can support membership-queries for S_i in $\tilde{O}_\gamma\left(n^{1+\epsilon_{i-1}}\right)$ time and matching-queries for M_i in $\tilde{O}_\gamma\left(n^{1+\epsilon_i}\right)$ time. Let us ignore the base case as it is trivial. For the induction step, we have the following:

1. To answer a membership-query for S_i, the template algorithm only needs to make $q = O_\gamma(1)$ matching-queries to M_j where $j < i$. So the total query time is $q \cdot \tilde{O}_\gamma(n^{1+\epsilon_{i-1}}) = \tilde{O}_\gamma(n^{1+\epsilon_{i-1}})$.

2. To answer a matching-query for M_i, the oracle $\texttt{LargeMatchingOracle}(S_i, \delta_i, \epsilon_{i-1})$ for the matching M_i has query time $\tilde{O}_\gamma\left(n^{1+g(\epsilon_{i-1})}\right) = \tilde{O}_\gamma\left(n^{1+\epsilon_i}\right)$.

The total preprocessing time we need for $\texttt{LargeMatchingOracle}(\cdot)$ to implement all the t rounds is $\sum_{i=1}^{t} \tilde{O}_\gamma\left(n^{2-\epsilon_i}\right) = \tilde{O}_\gamma\left(n^{2-\epsilon}\right) = n^{2-\Omega_\gamma(1)}$. At the end of the last round, we can support matching-queries for the $(1, \gamma n)$-approximate matching M returned by $\texttt{Alg}(\gamma)$ in $\tilde{O}_\gamma(n^{1+\epsilon_t})$ time, where $\lim_{\gamma \to 0} \epsilon_t = 0$.

To get a $(1, \gamma n)$-approximate estimate $\hat{\mu}$ of $\mu(g)$, we sample $\tilde{O}(1/\gamma^2)$ vertices and check if they are matched under M. Whp, this is a $(1, \Theta(\gamma)n)$-approximation of $\mu(G)$ because M is $(1, \gamma n)$-approximate.

Result II: Dynamic $(1 + \gamma)$-Approximate Matching Size.

Our dynamic matching size algorithm now follows from standard techniques. Using the well-known vertex reduction technique (see, for example, Corollary 4.9 of [Kis22]), we can assume that $\mu(G) \geq \gamma n$ at all times. We work in *phases*, where each phase lasts for $\gamma^2 n$ updates. At the start of each phase, we invoke the sublinear algorithm from Result I above, to obtain a $(1, \gamma^2 n)$-approximate

estimate $\hat{\mu}$ of $\mu(G)$, in $n^{2-\Omega_\gamma(1)}$ time. Since $\mu(G) \geq \gamma n$ and since the phase lasts for only $\gamma^2 n$ updates, this $\hat{\mu}$ continues to remain a purely multiplicative $(1+\Theta(\gamma))$-approximate estimate of $\mu(G)$ throughout the duration of the phase. This leads to an amortized update time of $n^{2-\Omega_\gamma(1)}/(\gamma^2 n) = n^{1-\Omega_\gamma(1)}$. In Section 7, we show how to extend this approach to prove Theorem 1.2.

3 Notations and Preliminaries

Unless speficied otherwise, the input graph $G = (V, E)$ will have n nodes and m edges. A *matching* $M \subseteq E$ is a subset of edges that do not share any common endpoint. We use the symbol $\mu(G)$ to denote the size of a maximum matching in G. We say that a pah $p = (v_0, v_1, \ldots, v_i)$ is an *alternating path* in G w.r.t. a matching $M \subseteq E$ iff $(v_j, v_{j+1}) \in E$ for all $j \in [0, i-1]$ and the edges in the path p alternate between being in M and in $E \setminus M$. We say that p is an *augmenting path* in G w.r.t. M iff p is an alternating path whose first and the last edges are both unmatched in M. The *length* of a path is the number of edges in it. We let $V(M)$ denote the set of matched nodes in a matching $M \subseteq E$. Consider any node $v \in V(M)$ and suppose that $(u, v) \in M$. Then we say that u is the *mate* of v in M. Given a subset of nodes $S \subseteq V$, $G[S]$ denotes the subgraph of G induced by S. Given any graph G', the symbol $E(G')$ denotes the set of edges in G'.

Throughout the paper, the symbol $\Theta_{k,\gamma}(1)$ will denote any positive constant that depends only on k and γ (where k and γ are constant parameters whose values will be chosen later on). We analogously use the notation $\Theta_k(1)$ to denote a constant that depends only on k. Finally, the symbol $\tilde{O}(.)$ will be used to hide any polylog(n) factors.

Oracles. We have the adjacency matrix access to the input graph G. Each query takes $O(1)$ time. We do *not* have the adjacency list access to the input graph.

For any vertex set $A \subset V$, an *A-membership* oracle $\texttt{mem}_A : V \to \{0, 1\}$ indicates whether $v \in A$ for any $v \in V$. That is, we have

$$\texttt{mem}_A(v) = \mathbf{1}\{v \in A\}.$$

A *matching oracle* $\texttt{match}_M : V \to \binom{V}{2} \cup \{\perp\}$ *for a matching* M is an oracle that, given a vertex $v \in V$, returns

$$\texttt{match}_M(v) = \begin{cases} (v, v') & (v, v') \in M \\ \perp & v \notin V(M). \end{cases}$$

Similarly, a *mate oracle* $\texttt{mate}_M : V \to V \cup \{\perp\}$ *for a matching* M is an oracle that, given a vertex $v \in V$, returns

$$\texttt{mate}_M(v) = \begin{cases} v' & v \in V(M) \text{ and } (v, v') \in M \\ \perp & v \notin V(M). \end{cases}$$

Concentration Bounds. We need standard concentration bounds as follows.

Proposition 3.1 (Hoeffding bound)**.** *Let $X_1, \ldots, X_n$ be independent random variables such that $a \leq X_i \leq b$. Let $X = \sum_{i=1}^n X_i$. For any $t > 0$,*

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq 2\exp(-\frac{2t^2}{n(b-a)^2}).$$

Proposition 3.2 (Chernoff bound)**.** *Let $X_1, \ldots, X_n$ be independent $\{0, 1\}$-random variables. Let $X = \sum_{i=1}^n X_i$ where $\mathbb{E}[X] \leq \overline{\mu}$ For any $t > 0$ where $t \leq \overline{\mu}$,*

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq 2\exp(-\frac{t^2}{3\overline{\mu}}).$$

Chernoff bound can be much stronger than Hoeffding bound when $\mathbb{E}[X]$ has small upper bound. For example, if we applied Proposition 3.1 to the setting for Proposition 3.2, we would only get that the bound of $2\exp(-\frac{2t^2}{3n})$ which is much weaker than $2\exp(-\frac{t^2}{3\overline{\mu}})$ when $\overline{\mu} \ll n$.

4 Matching Oracles of Induced Subgraphs

In this section, we present the key subroutine of this paper. The goal is to construct a matching oracle for an induced subgraph $G[A]$ but A is unknown to us; we only have access to an A-membership oracle $\texttt{mem}_A$.

Theorem 4.1. *Let $G = (V, E)$ be a graph, $A \subseteq V$ be a vertex set. Suppose that we have access to adjacency matrix of G and an A-membership oracle $\texttt{mem}_A$ with t_A query time. We are given as input $\epsilon > 0$ and $\delta_{\text{in}} > 0$.*

We can preprocess G in $\tilde{O}((t_A + n)(n^{1-\epsilon} + n^{4\epsilon})/\text{poly}(\delta_{\text{in}}))$ time and either return $\perp$ or construct a matching oracle $\texttt{match}_M(\cdot)$ for a matching $M \subset G[A]$ of size at least $\delta_{\text{out}} n$ where $\delta_{\text{out}} = \delta_{\text{in}}^5/10^8$ that has $\tilde{O}((t_A + n)n^{4\epsilon}/\text{poly}(\delta_{\text{in}}))$ worst-case query time. If $\mu(G[A]) \geq \delta_{\text{in}} n$, then $\perp$ is not returned. The guarantee holds with high probability.

The very important property of Theorem 4.1 is that it makes $n^{1-\epsilon}$ oracle calls to $\texttt{mem}_A$ during preprocessing and only $n^{O(\epsilon)}$ calls to $\texttt{mem}_A$ on each query. The rest of this section is devoted for proving Theorem 4.1.

To prove Theorem 4.1, we adapt the technique used inside the algorithm by Behnazhad et al. [BRR23] for $(3/2 - \Omega(1))$-approximating $\mu(G)$ on bipartite graphs. We observe that the idea there has reach beyond $(3/2 - \Omega(1))$-approximation algorithms. The abstraction of that idea leads us to Theorem 4.1, the crucial subroutine for later parts of our paper.

This section is organized as follows. In Section 4.1, we show a weaker version of Theorem 4.1 that works well on low degree graphs. We will use this weaker version in the preprocessing step, described in Section 4.2. Then, we complete the query algorithm in Section 4.3.

4.1 Oracles on Low Degree Graphs

Here, we show a similar result as Theorem 4.1, but it is efficient only when the maximum degree Δ is small. In particular, the query algorithm makes $n^{O(\epsilon)}$ calls to $\texttt{mem}_A$ only when $\Delta = n^{O(\epsilon)}$.

Lemma 4.2. *Let $G = (V, E)$ be a graph with maximum degree Δ where Δ is known and $A \subseteq V$ be a vertex set. Suppose that we have access to adjacency matrix of G and an A-membership oracle $\texttt{mem}_A$ with t_A query time. We can construct in $\tilde{O}((t_A\Delta + n + t_A/\epsilon)\Delta/\epsilon^2)$ time a matching oracle $\texttt{match}_M^{\text{low}}(\cdot)$ for a $(2, \epsilon n)$-approximate matching M in $G[A]$ that has $\tilde{O}(t_A\Delta + n + t_A/\epsilon)\Delta/\epsilon)$ worst-case query time with high probability.*

The proof of Lemma 4.2 is based on the the following $(2, \epsilon n)$-approximate matching oracle given access to adjacency list.

Lemma 4.3. *Given the adjacency lists of a graph $G = (V, E)$ with average degree d and a parameter $\overline{d} \geq d$, we can in $\tilde{O}(\overline{d}/\epsilon^2)$ time construct a matching oracle $\texttt{match}_M(\cdot)$ for a $(2, \epsilon n)$-approximate matching M in G with $\tilde{O}(\overline{d}/\epsilon)$ worst-case query time with high probability.*

Lemma 4.3 is proved by combining an improved analysis of randomized greedy maximal matching of Behnezhad [Beh22] into a framework for constructing an LCA by [LRY15]. We do not claim any novel contribution here and defer the proof to Appendix A.

Now, to prove Lemma 4.2, we need to strengthen Lemma 4.3 in two ways. First, it must work with the adjacency matrix, not the adjacency lists. Second, it must return a large matching of an induced subgraph $G[A]$, not that of G. However, this can be done using a simple simulation.

Proof of Lemma 4.2. Let $\mathcal{A}$ denote the algorithm of Lemma 4.3. We simulate $\mathcal{A}$ on $G[A]$ with parameter $\overline{d} \leftarrow \Delta$ as follows.

Whenever $\mathcal{A}$ needs to sample a vertex, we sample $O(\log(n)/\epsilon)$ vertices in G and call $\texttt{mem}_A$ on each of them. If one of them is in A, then we get a random vertex in $G[A]$. If none of them is in A, then w.h.p. $|A| \leq \epsilon n$. If this ever happens, even an empty matching is a $(2, \epsilon n)$-approximate matching in $G[A]$, and the problem becomes trivial.

Whenever $\mathcal{A}$ needs to make queries to the adjacency list of any vertex v, we can construct the whole adjacency list of v in $G[A]$ by first making n adjacency matrix queries to learn all neighbors of v in G and then makes $\deg(v) \leq \Delta$ oracles calls to $\texttt{mem}_A$ to know which neighbors are in $G[A]$. This takes $O(t_A\Delta + n)$ time. Every other computation can be simulated without the overhead.

Therefore, each step of $\mathcal{A}$ can be simulated with an extra $(t_A \cdot O(\log(n)/\epsilon) + t_A\Delta + n)$ factor. □

4.2 Preprocessing

We describe the preprocessing algorithm in Algorithm 1 with the guarantees summarized in the lemma below.

Lemma 4.4. *In $\tilde{O}((t_A + n)(n^{1-\epsilon} + n^{4\epsilon})/\text{poly}(\delta_{\text{in}}))$ time, Algorithm 1 outputs either $\perp$ (indicating an error) or* the remaining *set $V' \subseteq V$ of vertices together with an explicit matching $M' \subseteq G[V']$ that satisfies one of the following:*

1. *$|M'[A]| \geq 2\delta_{\text{out}} n$, or*
2. *$\mu(G[A \cap V' \setminus V(M')]) \geq 4\delta_{\text{out}} n$ and $G[V' \setminus V(M')]$ has maximum degree at most $n^{2\epsilon}$.*

The algorithm also reports which properties above M' satisfies. If $\mu(G[A]) \geq \delta_{\text{in}} n$, then $\perp$ is not returned with high probability.

In Algorithm 1, the remaining set V' is initialized as V and only shrinks. For convenience, we let $A' := A \cap V'$ and $D' := D \cap V'$ denote the *remaining* alive and dead vertices.

4.2.1 Correctness

In this part, we prove the correctness of Algorithm 1 assuming that it does not return $\perp$. We first show that $\tilde{\mu}_1$ and $\tilde{\mu}_2$ are good approximation of $M^i[A]$ and $\widehat{M^i}$ base on basic there definition and Hoeffding's bound.

Lemma 4.5. *For every i, we have $|M^i[A]| - \delta_{\text{out}} n \leq \tilde{\mu}_1 \leq |M^i[A]|$ w.h.p.*

Proof. The probability that a random edge from M^i is in $G[A]$ is $\frac{|M^i[A]|}{|M^i|}$. So $\mathbb{E}[X] = r_1 \frac{|M^i[A]|}{|M^i|}$ and $|M^i[A]| = \frac{|M^i|\mathbb{E}[X]}{r_1}$. By definition of $\tilde{\mu}_1 = \frac{|M^i|X}{r_1} - \frac{\delta_{\text{out}} n}{2}$ and by Hoeffding bound Proposition 3.1, we have

$$\begin{aligned}
\Pr[\tilde{\mu}_1 < |M^i[A]| - \delta_{\text{out}} n \text{ or } \tilde{\mu}_1 > |M^i[A]|] &= \Pr[\frac{|M^i| \cdot (X - \mathbb{E}[X])}{r_1} > \frac{\delta_{\text{out}} n}{2}] \\
&\leq 2\exp(-\frac{2(\frac{\delta_{\text{out}} n}{2})^2}{r_1(\frac{n}{r_1})^2}) = 2\exp(-\delta_{\text{out}}^2 r_1/2) \\
&\leq 1/n^{10}.
\end{aligned}$$

Algorithm 1 Preprocess G.

$p = 100n^{2-2\epsilon}\log n$, $k = n^{\epsilon}$, $\eta = \delta_{\text{in}}^2\log(n)/10$, $T = 100/\delta_{\text{in}}^2$, $\delta_{\text{out}} = \delta_{\text{in}}^3/10^6 T = \delta_{\text{in}}^5/10^8$.
$r_1 = r_2 = \frac{1000}{\delta_{\text{out}}^2}\log n = \Theta(\frac{\log n}{\delta_{\text{in}}^{10}})$, $r_3 = 1000\delta_{\text{in}}\frac{n}{k}\log n$.
$V' \leftarrow V$.
Repeat the following for T times:

1. Sample kp distinct pairs of vertices from V'. Partition the sampled pairs into $(P^1, \ldots, P^k)$ where each P^i is an ordered list containing p pairs of vertices.
2. For $i \in [k]$
   (a) Let $E^i = \{(u,v) \in P^i \mid (u,v) \in G[V']\}$ be an ordered sublist of P^i containing only pairs which are edges of $G[V']$.
   (b) Let M^i be the greedy maximal matching when scanning E^i in order.
   Case 1:
   (c) Sample r_1 edges from M^i.
   (d) Let X count the sampled edges that are in $G[A]$ (using the oracle $\texttt{mem}_A$)
   (e) Set $\tilde{\mu}_1 = \frac{|M^i|X}{r_1} - \frac{\delta_{\text{out}}n}{2}$.
   (f) If $\tilde{\mu}_1 \geq 2\delta_{\text{out}}n$, then set $M' \leftarrow M^i$ and **report** that M' satisfies Case 1.
   Case 2:
   (g) Let $\widehat{M^i}$ be a $(2, \delta_{\text{out}}n)$-approximate matching in $G[A' \setminus V(M^i)]$ that the matching oracle $\texttt{match}^{\text{low}}_{\widehat{M^i}}(\cdot)$ from Lemma 4.2 respects, given graph $G[V' \setminus V(M^i)]$ with vertex set $A' \setminus V(M^i)$ as input.
   (h) Sample r_2 vertices from $V' \setminus V(M^i)$.
   (i) Let Y count the sampled vertices that are matched in $\widehat{M^i}$ (using the oracle $\texttt{match}^{\text{low}}_{\widehat{M^i}}(\cdot)$).
   (j) Set $\tilde{\mu}_2 = \frac{|V'\setminus V(M^i)|Y}{2r_2} - \frac{\delta_{\text{out}}n}{2}$.
   (k) If $\tilde{\mu}_2 \geq 4\delta_{\text{out}}n$, then set $M' \leftarrow M^i$ and **report** that M' satisfies Case 2.
3. Let $A'_{\text{sp}} \subseteq A'$ be obtained by sampling r_3 vertices from A'.
4. Let $\overline{G} = (V', \cup_{i=1}^k M^i)$.
5. Let $C = \{v \in V' \mid N_{\overline{G}}(v, A'_{\text{sp}}) \geq \eta\}$, i.e., C contains remaining vertices that have at least η neighbors from A'_{sp} in $\overline{G}$.
6. Set $V' \leftarrow V' \setminus C$.

Return $\perp$ (Error).

□

Lemma 4.6. *For every i, we have $|\widehat{M^i}| - \delta_{\text{out}} n \le \tilde{\mu}_2 \le |\widehat{M^i}|$ w.h.p.*

Proof. The probability that a random vertex from $V' \setminus V(M^i)$ is in $V(\widehat{M^i})$ is $\frac{2|\widehat{M^i}|}{|V' \setminus V(M^i)|}$. So $\mathbb{E}[Y] = r_2 \frac{2|\widehat{M^i}|}{|V' \setminus V(M^i)|}$ and $|\widehat{M^i}| = \frac{|V' \setminus V(M^i)|\mathbb{E}[Y]}{2r_2}$. By definition of $\tilde{\mu}_2 = \frac{|V' \setminus V(M^i)|Y}{2r_2} - \frac{\delta_{\text{out}} n}{2}$ and by Hoeffding bound Proposition 3.1, we have

$$\begin{aligned}
\Pr[\tilde{\mu}_2 < |\widehat{M^i}| - \delta_{\text{out}} n \text{ or } \tilde{\mu}_2 > |\widehat{M^i}|] &= \Pr[\frac{|V' \setminus V(M^i)| \cdot (Y - \mathbb{E}[Y])}{2r_2} > \frac{\delta_{\text{out}} n}{2}] \\
&\le 2\exp(-\frac{2(\frac{\delta_{\text{out}} n}{2})^2}{r_2(\frac{n}{2r_2})^2}) = 2\exp(-2\delta_{\text{out}}^2 r_2) \\
&\le 1/n^{10}.
\end{aligned}$$

□

Next, we show the "sparsification" property of randomized greedy maximal matching M^i. That is, $G[V' \setminus V(M^i)]$ has low degree. The idea is that any high-degree vertex v in $G[V' \setminus V(M^i)]$ should not exist because it should have been matched by M^i via one of the sampled edges. The proof is similar to Lemma 3.1 of [BFS12], which considers this sparsification property of the randomized greedy maximal independent set, instead of maximal matching.

Lemma 4.7. *For every i, the maximum degree of $G[V' \setminus V(M^i)]$ is at most $n^{2\epsilon}$ w.h.p.*

Proof. Let us describe an equivalent way to construct M^i. Initialize $M^i = \emptyset$ and then sample p pairs of vertices in V'. For each sampled pair (u, v), if $(u, v) \in G[V']$ and both u and v are not matched by M^i, then we add (u, v) into M^i. At the end of this process, we will show that, for any vertex $v \in V'$, the degree of v in $G[V' \setminus V(M^i)]$ is at most $n^{2\epsilon}$ w.h.p. (we use the convention that if $v \notin V' \setminus V(M^i)$, then the degree v is 0.)

For $t \in [1, p]$, let M^i_t denote the matching M^i after we sampled the t-th pair. For convenience, we denote $\deg^t(v) = \deg_{G[V' \setminus V(M^i_t)]}(v)$ as the *degree of v at time t*. We want to show that $\Pr[\deg^p(v) > n^{2\epsilon}] \le 1/n^{10}$ for any $v \in V'$.

Observe that if $\deg^p(v) > n^{2\epsilon}$, then $\deg^t(v) > n^{2\epsilon}$ for all $t \le p$. Now, given that $\deg^{t-1}(v) > n^{2\epsilon}$, the probability that v remained at unmatched after time t is

$$1 - \frac{\deg^{t-1}(v)}{\binom{|V'|}{2}} \le 1 - \frac{n^{2\epsilon}}{n^2}.$$

In particular, the probability that $\deg^t(v) > n^\epsilon$ is at most $1 - \frac{n^{2\epsilon}}{n^2}$. This implies that the probability that $\deg^p(v) > n^{2\epsilon}$ is at most

$$(1 - \frac{n^{2\epsilon}}{n^2})^p \le \frac{1}{n^{10}}$$

because $p = 100n^{2-2\epsilon} \log n$. □

From the above lemmas, we can conclude the correctness of the algorithm.

Corollary 4.8. *If Algorithm 1 returns a matching M', then M' satisfies the guarantees from Lemma 4.4 w.h.p.*

Proof. If M' is returned under Case 1. Then, by Lemma 4.5, we have $|M'[A]| \ge \tilde{\mu}_1 \ge 2\delta_{\text{out}} n$ w.h.p. Otherwise, in Case 2, we have $\mu(G[A \cap V' \setminus V(M')]) \ge |M'| \ge \tilde{\mu}_2 \ge 4\delta_{\text{out}} n$ by Lemma 4.6 and also maximum degree of $G[V' \setminus V(M')]$ is at most $n^{2\epsilon}$ by Lemma 4.7 w.h.p. □

4.2.2 Termination without Error

In this part, we show that if $\mu(G[A]) \geq \delta_{\text{in}} n$, then Algorithm 1 does not return $\perp$ w.h.p. For any graph G and $U_1, U_2 \subseteq V(G)$, we let $G[U_1, U_2]$ contains all edges of G whose one endpoint is in U_1 and another in U_2. Note that the induced subgraph $G[U_1] = G[U_1, U_1]$.

Our high-level plan is that we will show that D' decreases its size by $\Theta(\delta_{\text{in}}^2 n)$ in each iteration in the repeat loop. So D' must become very small after $T = \Theta(1/\delta_{\text{in}}^2)$ iterations. But when this happens, we can show that either Case 1 or Case 2 must happen and so the algorithm must terminate without error.

To carry out the above plan, we need a helper lemma (Lemma 4.9) which states that, even if V' keeps shrinking, the maximum matching in $G[V']$ remains large, $\mu(G[A']) \geq \delta_{\text{in}} n/2$. We will need this fact throughout the whole argument.

The high-level argument goes as follows. If the algorithm does not return M', then $M^i[A']$ is very small for all i and so $\overline{G}[A']$ contains few edges. It follows that the set C of removed vertices contains only few vertices in A' because each vertex $v \in C \cap A'$ has high degree of at least η in $\overline{G}[A']$. Thus, we remove only few vertices from A' and so the size of $\mu(G[A'])$ cannot decrease too much. The formal argument below goes through the set A'_{sp} and the above paragraph gives the right intuition.

Lemma 4.9. *Suppose $\mu(G[A]) \geq \delta_{\text{in}} n$. For $\tau \in [0, T]$, at the end of the τ-th iteration of the repeat loop in Algorithm 1, we have $\mu(G[A']) \geq (1 - \frac{\tau}{2T})\delta_{\text{in}} n \geq \delta_{\text{in}} n/2$ w.h.p., if the algorithm does not terminate yet.*

Proof. We prove by induction on τ. For $\tau = 0$ (i.e. the beginning the algorithm), the claim holds as $\mu(G[A]) \geq \delta_{\text{in}} n$. Next, we consider $\tau \geq 1$. By induction hypothesis, at the beginning of the τ-th iteration, we have $\mu(G[A']) \geq (1 - \frac{\tau-1}{2T})\delta_{\text{in}} n \geq \delta_{\text{in}} n/2$.

At the end of the τ-th iteration, since Algorithm 1 did not terminate at Step 2f, by Lemma 4.5, we have, w.h.p., $|M^i[A']| \leq 2\delta_{\text{out}} n + \delta_{\text{out}} n \leq 3\delta_{\text{out}} n$ for all i. So we have $|E(\overline{G}[A'])| \leq 3\delta_{\text{out}} nk$ and then the average degree of vertices in $\overline{G}[A']$ is at most

$$\frac{2|E(\overline{G}[A'])|}{|A'|} \leq \frac{6\delta_{\text{out}} k}{\delta_{\text{in}}}$$

because $|A'| \geq 2\mu(G[A']) \geq \delta_{\text{in}} n$.

Recall that A'_{sp} is obtained by sampling r_3 vertices from A'. So

$$\mathbb{E}[\text{vol}_{\overline{G}[A']}(A'_{\text{sp}})] \leq \frac{6\delta_{\text{out}} k r_3}{\delta_{\text{in}}} \leq \frac{\delta_{\text{in}}^3 n \log n}{200T}$$

because $\delta_{\text{out}} = \delta_{\text{in}}^3/10^6 T$ and $\frac{kr_3}{\delta_{\text{in}}} = 1000 n \log n$. Furthermore, this bound is concentrated. Indeed, since $\text{vol}_{\overline{G}[A']}(A'_{\text{sp}})$ is a sum of r_3 independent random variable whose range is k (since the maximum degree in $\overline{G}$ and $\overline{G}[A']$ is k), by Hoeffding bound Proposition 3.1, we have

$$\Pr[\text{vol}_{\overline{G}[A']}(A'_{\text{sp}}) - \mathbb{E}[\text{vol}_{\overline{G}[A']}(A'_{\text{sp}})] > \frac{\delta_{\text{in}}^3 n \log n}{200T}] \leq 2\exp(\frac{-2(\frac{\delta_{\text{in}}^3 n \log n}{200T})^2}{r_3 k^2}) \ll 1/n^{10}.$$

So $\text{vol}_{\overline{G}[A']}(A'_{\text{sp}}) \leq \frac{\delta_{\text{in}}^3 n \log n}{100T}$ w.h.p. Since every vertex $v \in C$ is adjacent to at least η vertices in A'_{sp} in $\overline{G}$, we have $|C \cap A'|\eta \leq \text{vol}_{\overline{G}[A']}(A'_{\text{sp}})$. Therefore,

$$|C \cap A'| \leq \frac{\delta_{\text{in}}^3 n \log n}{100T\eta} \leq \frac{\delta_{\text{in}} n}{2T}$$

because $\eta = \delta_{\text{in}}^2 \log(n)/10$. This means that we remove at most $\frac{\delta_{\text{in}} n}{2T}$ vertices from A' at the end of the τ-th iteration. So the size of maximum matching in $G[A']$ may decrease by at most $\frac{\delta_{\text{in}} n}{2T}$. Thus, $\mu(G[A']) \geq (1 - \frac{\tau}{2T})\delta_{\text{in}} n$ which completes the induction. □

Given Lemma 4.9, we will use the following lemma to argue that if Algorithm 1 does not return M^i, then then M^i must match many vertices between A' and D'.

Lemma 4.10. *Suppose $\mu(G[A']) \geq \delta_{\text{in}} n/2$. For any matching M in $G[V']$, if $|M[A']| < 3\delta_{\text{out}} n$ and $\mu(G[A' \setminus V(M)]) < 16\delta_{\text{out}} n$, then $M[A', D'] \geq \delta_{\text{in}} n/3$.*

Proof. Let M^* be the maximum matching in $G[A']$ of size at least $\delta_{\text{in}} n/2$. We partition edges in M^* into two parts: M_0^* and M_1^*. For each $(u, v) \in M^*$, we add (u, v) into M_0^* if both $u, v \notin V(M)$. Otherwise, either u or v are matched by M and we add (u, v) into M_1^*. Note that M_0^* is a matching in $G[A' \setminus V(M)]$. So $|M_0^*| < 16\delta_{\text{out}} n$ and $|M_1^*| \geq \delta_{\text{in}} n/2 - 16\delta_{\text{out}} n$.

Observe that $|V(M_1^*)| \leq |M[A', D']| + 2|M[A']|$ because we can charge vertices of $V(M_1^*)$ to either matched edges of $M[A', D']$ or $M[A']$ such that each matched edge in $M[A', D']$ is charged once and each match edges in $M[A']$ is charged at most twice. Since $|M[A']| < 3\delta_{\text{out}} n$, we have $|M[A', D']| \geq \delta_{\text{in}} n/2 - 16\delta_{\text{out}} n - 2 \cdot 3\delta_{\text{out}} n \geq \delta_{\text{in}} n/3$. □

Lemma 4.10 also says that once D' become small enough, Algorithm 1 will not err w.h.p.

Corollary 4.11. *If $\mu(G[A']) \geq \delta_{\text{in}} n/2$ and $|D'| < \delta_{\text{in}} n/3$, then Algorithm 1 will return a matching M' w.h.p.*

Proof. For any i, note that $|M^i[A', D']| < |D'| < \delta_{\text{in}} n/3$. So by the contrapositive of Lemma 4.10, we have that either $|M^i[A']| \geq 3\delta_{\text{out}} n$ or $\mu(G[A' \setminus V(M^i)]) \geq 10\delta_{\text{out}} n$. If $|M^i[A']| \geq 3\delta_{\text{out}} n$, then $\tilde{\mu}_1 \geq 2\delta_{\text{out}} n$ w.h.p. by Lemma 4.5. If $\mu(G[A' \setminus V(M^i)]) \geq 10\delta_{\text{out}} n$, then $|\widehat{M}_i| \geq \frac{16\delta_{\text{out}} n}{2} - \delta_{\text{out}} n \geq 7\delta_{\text{out}} n$ because $\widehat{M}_i$ is $(2, \delta_{\text{out}} n)$-approximate matching. So $\tilde{\mu}_2 \geq 6\delta_{\text{out}}$ w.h.p. by Lemma 4.6. In either cases, so Algorithm 1 must return a matching M' at Line 2f or Line 2k. □

Now, we are ready to show that D' must shrink significantly after each iteration of the repeat loop, which means that there cannot be too many iterations before the algorithm terminate by Corollary 4.11.

Lemma 4.12. *Suppose$\mu(G[A']) \geq \delta_{\text{in}} n/2$. If Algorithm 1 does not terminate until C is computed, then $|D' \cap C| \geq \frac{\delta_{\text{in}}^2}{100} n$.*

There are two main claims in the proof of Lemma 4.12. We suggest the reader to skip the proofs of these claims and see how they are used to prove Lemma 4.12 first.

Claim 4.13. *$|E(\overline{G}[A'_{\text{sp}}, D'])| \geq 100\delta_{\text{in}}^2 n \log n$ w.h.p.*

Proof. At the end of the τ-th iteration, since Algorithm 1 did not terminate at Step 2f nor Step 2k, we have, w.h.p., that $M[A'] < 3\delta_{\text{out}} n$ by Lemma 4.5 and $|\widehat{M}^i| < 5\delta_{\text{out}} n$ by Lemma 4.6. Since $\widehat{M}^i$ is a $(2, \delta_{\text{out}} n)$-approximate matching in $G[A' \setminus V(M)]$, we have $\mu(G[A' \setminus V(M)]) < 16\delta_{\text{out}} n$. By Lemma 4.10, we have $|M^i[A', D']| \geq \delta_{\text{in}} n/3$.

Observe that $|E(\overline{G}[A', D'])| = \sum_i |M^i[A', D']| \geq \delta_{\text{in}} nk/3$ because all M^i are mutually disjoint. Note that $\overline{G}[A', D']$ is a bipartite graph. So the average degree of vertices in A' in $\overline{G}[A', D']$ is

$$\frac{|E(\overline{G}[A', D'])|}{|A'|} \geq \delta_{\text{in}} k/3$$

and we have that

$$\mathbb{E}[|E(\overline{G}[A'_{\mathrm{sp}}, D'])|] \geq \delta_{\mathrm{in}} k r_3/3 \geq 200\delta_{\mathrm{in}}^2 n \log n.$$

because $r_3 = 1000\delta_{\mathrm{in}} \frac{n}{k} \log n$. Furthermore, this is concentrated. Indeed, since $|E(\overline{G}[A'_{\mathrm{sp}}, D'])|$ is a sum of r_3 independent random variable whose range is k (since the maximum degree in $\overline{G}$ and $\overline{G}[A']$ is k), by Hoeffding bound Proposition 3.1, we have

$$\Pr[\big||E(\overline{G}[A'_{\mathrm{sp}}, D'])| - \mathbb{E}[|E(\overline{G}[A'_{\mathrm{sp}}, D'])|]\big| > 100\delta_{\mathrm{in}}^2 n \log n] \leq 2\exp(\frac{-2(100\delta_{\mathrm{in}}^2 n \log n)^2}{r_3 k^2}) \ll 1/n^{10}.$$

So $|E(\overline{G}[A'_{\mathrm{sp}}, D'])| \geq 100\delta_{\mathrm{in}}^2 n \log n$ w.h.p. □

Claim 4.14. *For each $v \in D'$, the number of neighbor of v from A'_{sp} in $\overline{G}$ is at most $2000 \log n$ w.h.p. That is, $|N_{\overline{G}}(v, A'_{\mathrm{sp}})| \leq 2000 \log n$.*

Proof. We have

$$\begin{aligned}\mathbb{E}[N_{\overline{G}}(v, A'_{\mathrm{sp}})] &= \sum_{u \in N_{\overline{G}}(v, A')} \Pr[u \in A'_{\mathrm{sp}}] \\ &\leq k \cdot \frac{r_3}{|A'|} \leq 1000 \log n\end{aligned}$$

because $r_3 k = 1000\delta_{\mathrm{in}} n \log n$ and $|A'| \geq \delta_{\mathrm{in}} n$ since we assume $\mu(G[A']) \geq \delta_{\mathrm{in}} n/2$. Moreover, applying by Chernoff bound Proposition 3.2 where $t = 1000 \log n$ and $\overline{\mu} = 1000 \log n$[8], we have

$$\Pr[|N_{\overline{G}}(v, A'_{\mathrm{sp}}) - \mathbb{E}[N_{\overline{G}}(v, A'_{\mathrm{sp}})| > 1000 \log n] \leq 2\exp(-\frac{(1000 \log n)^2}{3 \cdot 1000 \log n}) \ll 1/n^{10}.$$

So $|N_{\overline{G}}(v, A'_{\mathrm{sp}})| \leq 2000 \log n$ w.h.p. □

Now, let us prove Lemma 4.12 using the above claims.

Proof of Lemma 4.12. Observe that

$$|E(\overline{G}[A'_{\mathrm{sp}}, D'])| = \sum_{v \in D'} |N_{\overline{G}}(v, A'_{\mathrm{sp}})| \leq |D' \cap C| \cdot 2000 \log n + |D' \setminus C|\eta$$

where the inequality holds w.h.p. by Claim 4.14. Since $|D' \setminus C| \leq n$ and $\eta = \delta_{\mathrm{in}}^2 \log(n)/10$, we have by Claim 4.13 that

$$100\delta_{\mathrm{in}}^2 n \log n \leq |D' \cap C| \cdot 2000 \log n + n\delta_{\mathrm{in}}^2 \log(n)/10$$

and so $|D' \cap C| \geq \frac{\delta_{\mathrm{in}}^2}{100} n$ as desired. □

Finally, we give the conclusion of this part.

Corollary 4.15. *If $\mu(G[A]) \geq \delta_{\mathrm{in}} n$, then Algorithm 1 does not return $\perp$ w.h.p.*

Proof. First, $\mu(G[A]) \geq \delta_{\mathrm{in}} n$ implies that $\mu(G[A']) \geq \delta_{\mathrm{in}} n/2$ w.h.p. by Lemma 4.9. So, by Lemma 4.12 D' decreases its size by $\delta_{\mathrm{in}}^2 n/100$ in each iteration in the repeat loop. Hence, we have that $|D'| \leq \delta_{\mathrm{in}} n/3$ before $T = 100/\delta_{\mathrm{in}}^2$ iterations. Therefore, there is an iteration $\tau \in [1, T]$ where Algorithm 1 will return a matching M' w.h.p. by Corollary 4.11. □

[8]Note that Hoeffding bound Proposition 3.1 is not strong enough here.

4.2.3 Preprocessing Time

Consider the $(2, \delta_{\text{out}} n)$-approximate matching oracle $\texttt{match}^{\text{low}}(\cdot)$ in Line 2g, which is given graph $G[V' \setminus V(M^i)]$ and vertex set $A' \setminus V(M^i)$ as input.

By Lemma 4.7, we can assume w.h.p. that $G[V' \setminus V(M^i)]$ has degree at most $n^{2\epsilon}$. Lemma 4.2 implies the following:

Proposition 4.16. *Both preprocessing and query time of* $\texttt{match}^{\text{low}}(\cdot)$ *is at most* $\tilde{O}((t_A n^{2\epsilon} + n + t_A/\delta_{\text{out}})n^{2\epsilon}/\delta_{\text{out}}^2) = \tilde{O}((t_A + n)n^{4\epsilon}/\delta_{\text{out}}^3)$ *with high probability.*

Lemma 4.17. *Algorithm 1 takes* $\tilde{O}((t_A + n)(n^{1-\epsilon} + n^{4\epsilon})/\text{poly}(\delta_{\text{in}}))$ *total running time.*

Proof. We will analyze the total running time for each iteration of the repeat-loop in Algorithm 1. Since there are $T = O(1/\delta_{\text{in}}^2)$ iterations and we assume $\delta_{\text{in}} \geq 1/\text{poly}\log n$, the running time is the same up to polylogarithmic factor. Now, fix one iteration of the repeat-loop.

The total time to compute M^i, for all $i \leq k$, is $\Theta(kp) = \tilde{O}(n^{2-\epsilon})$. For each for-loop iteration, to compute $\tilde{\mu}_1$, we make $\Theta(r_1)$ queries to $\texttt{mem}_A$ taking $\Theta(r_1) \cdot t_A = \tilde{O}(t_A/\delta_{\text{in}}^{10})$ time. To compute $\tilde{\mu}_2$, we make r_2 queries to $\texttt{match}^{\text{low}}(\cdot)$. By Proposition 4.16, this takes time $r_2 \cdot \tilde{O}((t_A + n)n^{4\epsilon}/\delta_{\text{out}}^3) = \tilde{O}((t_A + n)n^{4\epsilon}/\delta_{\text{in}}^{25})$ by Lemma 4.2.

Next, we analyze the time to compute A'_{sp}. Since $|A'| \geq \delta_{\text{in}} n$ w.h.p. by Lemma 4.9, we can sample a random vertex in A' by sampling at most $O(\log n/\delta_{\text{in}})$ times in V' w.h.p. For each sample, we need to make a query to $\texttt{mem}_A$, so we can compute A'_{sp} in time $O(r_3) \cdot O(t_A \log n/\delta_{\text{in}}) = \tilde{O}(t_A n^{1-\epsilon}/\text{poly}(\delta_{\text{in}}))$ because $r_3 = 1000\delta_{\text{in}} \frac{n}{k} \log n$ and $k = n^{\epsilon}$. Once A'_{sp} is computed, we can compute C in $|E(\overline{G})| = \Theta(kp) = \tilde{O}(n^{2-\epsilon})$. To conclude, the total running time in each iteration of the repeat-loop at most

$$\tilde{O}(n^{2-\epsilon} + (t_A + n)n^{4\epsilon} + t_A n^{1-\epsilon})/\text{poly}(\delta_{\text{in}}) = \tilde{O}((t_A + n)(n^{1-\epsilon} + n^{4\epsilon})/\text{poly}(\delta_{\text{in}})).$$

□

The main lemma on preprocessing, Lemma 4.4, is implied by combining Corollary 4.8, Corollary 4.15 and Lemma 4.17

4.3 Query Algorithm

We define our matching oracle $\texttt{match}$ depending on the cases from Lemma 4.4.

Suppose Lemma 4.4 returns M' that satisfies Case 1. Let $M_1 = M'[A]$. By Lemma 4.4, $|M_1| \geq 2\delta_{\text{out}} n$. The algorithm for outputting $\texttt{match}(v)$ with respect to M_1 is described in Algorithm 2. The correctness is straightforward and the worst-case query time is clearly $2t_A + O(1)$.

Algorithm 2 Compute $\texttt{match}(v)$ with respect to M_1.

1. If $v \in V(M')$, let v' be such that $(v, v') \in M'$. Else, return $\perp$.
2. If $\texttt{mem}_A(v), \texttt{mem}_A(v') = 1$, return (v, v'). Else, return $\perp$.

Next, suppose Lemma 4.4 returns M' that satisfies Case 2. Let M_2 be a $(2, \delta_{\text{out}} n)$-approximate matching in $G[A \cap V' \setminus V(M')]$. By Lemma 4.4, $|M_2| \geq \mu(G[A \cap V' \setminus V(M')])/2 - \delta_{\text{out}} n \geq$

$\delta_{\text{out}} n$.[9] The algorithm for outputting $\texttt{match}(v)$ with respect to M_2 is described in Algorithm 3. The correctness is straightforward. Let us analyze the query time. Step 1 takes $t_A + O(1)$ time. Step 2 takes $\tilde{O}((t_A + n)n^{4\epsilon}/\delta_{\text{out}}^3)$ following the same proof as in Proposition 4.16 (the maximum degree of $G[V' \setminus V(M')]$ is at most $n^{2\epsilon}$ w.h.p. by Lemma 4.7).

Algorithm 3 Compute $\texttt{match}(v)$ with respect to M_2.

Let $\texttt{match}_{\text{low}}$ be the $(2, \delta_{\text{out}} n)$-approximate matching oracle from Lemma 4.2 when given graph $G[V' \setminus V(M^i)]$ with vertex set $A \cap V'$ as input.

1. Check if $v \in A \cap V' \setminus V(M')$. If not, return $\perp$.
2. Using the oracle $\texttt{match}_{\text{low}}$, if $v \in V(M_2)$, return $(v, v') \in M_2$. Else, return $\perp$.

In both cases, the matching oracle $\texttt{match}$ respects a matching of size at least $\delta_{\text{out}} n$ and has worst-case query time at most $\tilde{O}((t_A + n)n^{4\epsilon}/\text{poly}(\delta_{\text{in}}))$ w.h.p.

5 Boosting the Approximation Guarantee of a Matching Oracle

Recall the notations from Section 3. Throughout this section, we use the following parameters.

Definition 5.1. *$k \geq 0$ is an integral constant, $\gamma \in (0, 1)$ is a constant, $T = \Theta_{k,\gamma}(1)$ is a sufficiently large integral constant that depends only on k and γ (see Lemma 5.12), and $\epsilon_{in} > 0$ is a sufficiently small constant such that $9^T \cdot \epsilon_{in} < 1/5$.*

We present an algorithm $\texttt{Augment}(G, M^{\text{in}}, k, \gamma, \epsilon_{\text{in}})$, which takes as input: a graph $G = (V, E)$ with n nodes, the parameters $k, \gamma, \epsilon_{\text{in}}$ as in Definition 5.1, and an oracle $\texttt{match}_{M^{\text{in}}}(.)$ for a matching M^{in} in G that has $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{\text{in}}})$ query time. The algorithm either returns an oracle $\texttt{match}_{M^{\text{out}}}(.)$ for a matching M^{out} in G that is obtained by applying a sufficiently large number of length $(2k+1)$-augmenting paths to M^{in}, or it returns FAILURE. We now state our main result in this section.

Theorem 5.2. *Set $\epsilon_{out} := 9^T \cdot \epsilon_{in}$ (see Definition 5.1). Given adjacency-matrix query access to the input graph $G = (V, E)$, the algorithm $\texttt{Augment}(G, M^{in}, k, \gamma, \epsilon_{in})$ runs in $\tilde{O}_{k,\gamma}\left(n^{2-\epsilon_{in}}\right)$ time. Further, either it returns an oracle $\texttt{match}_{M^{out}}(.)$ with query time $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{out}})$, for some matching M^{out} in G of size $|M^{out}| \geq |M^{in}| + \Theta_{k,\gamma}(1) \cdot n$ (we say that it "succeeds" in this case), or it returns* FAILURE. *Finally, if the matching M^{in} admits a collection of $\gamma \cdot n$ many node-disjoint length $(2k+1)$-augmenting paths in G, then the algorithm succeeds whp.*

In Section 5.1, we present a *template algorithm* for the task stated in Theorem 5.2. This is inspired by an algorithm of McGregor [McG05] for computing a $(1+\epsilon)$-approximate matching in the semi-streaming model.While describing the template algorithm, we assume that we are given the matching M^{in} explicitly as part of the input, and that we need to either construct the matching M^{out} or return FAILURE. Note, however, that in the sublinear setting, we cannot assume this.

Subsequently, in Section 5.2, we show how to implement the template algorithm in the sublinear setting under adjacency-matrix queries, which leads to the proof of Theorem 5.2.

Remark on Oracles: Throughout this section, we will treat the oracle $\texttt{match}_M(.)$ as a data structure in the sublinear model, which returns the appropriate answer upon receiving a query. In

[9] In fact, if we define M_2 as $\widehat{M^i}$ from Line 2g in Algorithm 1, we would even have that $|M_2| \geq 4\delta_{\text{in}} n$ w.h.p. But we did use this bound just to avoid white-boxing the preprocessing algorithm and make the presentation of the query algorithm more modular.

contrast, we will treat the oracle $\texttt{mate}_M(.)$ as simply an abstract function, so that $\texttt{mate}_M(v)$ simply denotes the mate of v (if it exists) under M (see Section 3). Note that we can return the value of $\texttt{mate}_M(v)$ by making a single query to $\texttt{match}_M(v)$, without any additional overhead in time.

5.1 A Template Algorithm

We denote the template algorithm simply by $\texttt{Augment-Template}(G, M^{\texttt{in}}, k, \gamma)$, as we do not need the parameter $\epsilon_{\texttt{in}}$ to describe it. The parameter $\epsilon_{\texttt{in}}$ will become relevant only in Section 5.2, when we consider implementing this algorithm in the sublinear setting.

As part of the input to the template algorithm, the n-node graph $G = (V, E)$ and the matching $M^{\texttt{in}}$ are specified explicitly. The algorithm either returns an explicit matching $M^{\texttt{out}}$ in G of size $|M^{\texttt{out}}| \geq |M^{\texttt{in}}| + \Theta_{k,\gamma}(1) \cdot n$ (we say that it "succeeds" in this case), or it returns FAILURE. If $M^{\texttt{in}}$ admits a collection of $\gamma \cdot n$ many node-disjoint length $(2k+1)$-augmenting paths in G, then the template algorithm succeeds whp. This mimics Theorem 5.2. Furthermore, the template algorithm has access to a subroutine $\texttt{LargeMatching}(S, \delta)$, which takes as input a subset of nodes $S \subseteq V$ and a small constant $\delta \in (0, 1)$, and either returns $\perp$ or returns a matching M in $G[S]$ such that $|M| \geq \frac{1}{10^8} \cdot \delta^5 \cdot n$. In addition, if $\mu(G) \geq \delta \cdot n$, then it is guaranteed that $\texttt{LargeMatching}(G, \delta)$ does *not* return $\perp$. This mimics Theorem 4.1, with $\delta_{\texttt{in}} = \delta$.

5.1.1 Algorithm Description

Random partitioning: We start by partitioning the node-set V into $2k+2$ subsets $L_0, \ldots, L_{2k+1}$, as follows. For each $v \in V$, we place the node v into one of the subsets $L_0, \ldots, L_{2k+1}$ chosen uniformly and independently at random. We will refer to the subset L_i as *layer* i of this partition. If $v \in L_i$, then we will write $\ell(v) = i$ and simply say that the node v *belongs to layer* i.

Let p be an augmenting path of length $(2k+1)$ in G w.r.t. $M^{\texttt{in}}$. Assign an arbitrary direction to this path, so that we can write $p = (v_0, v_1, \ldots, v_{2k+1})$ w.l.o.g. Specifically, we have $(v_{2i}, v_{2i+1}) \in E \setminus M^{\texttt{in}}$ for all $i \in [0, k]$, and $(v_{2i-1}, v_{2i}) \in M^{\texttt{in}}$ for all $i \in [1, k]$. We say that the path p *survives* the random partitioning iff $v_i \in L_i$ for all $i \in [0, 2k+1]$.

Lemma 5.3. *Consider any collection $\mathcal{P}$ of node-disjoint length $(2k+1)$-augmenting paths in G w.r.t. $M^{\texttt{in}}$. Let $\mathcal{P}^* \subseteq \mathcal{P}$ denote the subset of paths in $\mathcal{P}$ that survive the random partitioning. If $|\mathcal{P}| \geq \gamma \cdot n$, then $|\mathcal{P}^*| \geq \Theta_{k,\gamma}(1) \cdot n$ whp.*

Proof. Each path $p \in \mathcal{P}$ survives the random partitioning with probability $(2k+2)^{-(2k+2)}$. Since $|\mathcal{P}| \geq \gamma \cdot n$, by linearity of expectation, we get: $\mathbb{E}[|\mathcal{P}^*|] \geq \left((2k+2)^{-(2k+2)}\gamma\right) \cdot n = \Theta_{k,\gamma}(1) \cdot n$. Finally, we note that whether a given path $p \in \mathcal{P}$ survives the random partitioning or not is independent of the fate of the other paths in $\mathcal{P}$. The lemma now follows from a Chernoff bound. □

Motivated by Lemma 5.3, the template algorithm will only attempt to augment $M^{\texttt{in}}$ along those augmenting paths that survive the random partitioning. This leads us to introduce the notion of *layered subgraphs* of G, as described below. Intuitively, although the template algorithm does not know the set $\mathcal{P}^*$ in advance, it can be certain that the sequence of edges in any length $(2k+1)$-augmenting path in $\mathcal{P}^*$ appears in successive layered subgraphs (see Observation 5.6).

Layered subgraphs of G: First, we define a set $V_H \subseteq V$. A node $v \in V$ belongs to V_H iff either

1. $\ell(v) \in \{0, 2k+1\}$ and $\texttt{mate}_{M^{\texttt{in}}}(v) = \perp$, or
2. $\ell(v) = 2j - 1$ for some $j \in [1, k]$ and $\ell\left(\texttt{mate}_{M^{\texttt{in}}}(v)\right) = 2j$, or

3. $\ell(v) = 2j$ for some $j \in [1, k]$ and $\ell\left(\texttt{mate}_{M^{\texttt{in}}}(v)\right) = 2j - 1$.

Given the nodes in V_H, the edge-set $E_H \subseteq E$ is defined as follows. An edge $(u, v) \in E$ belongs to E_H iff $u, v \in V_H$, $|\ell(u) - \ell(v)| = 1$, and either

1. $\min(\ell(u), \ell(v))$ is even and $(u, v) \notin M^{\texttt{in}}$, or
2. $\min(\ell(u), \ell(v))$ is odd and $(u, v) \in M^{\texttt{in}}$

We next define the subgraph $H := (V_H, E_H)$. Finally, for each $i \in [0, 2k]$, let $G_i := (V, E_i)$ be a bipartite subgraph of G, where $E_i := \{(u, v) \in E_H : \ell(u) = i, \ell(v) = i + 1\}$ is the set of edges in E_H between layer i and layer $i + 1$. Note that we have defined the subgraphs $\{G_i\}$ over the entire node-set V, although every edge in these subgraphs has both its endpoints in V_H. This is done to simplify notations, as will become evident when we describe how to implement our algorithm in the sublinear setting. For each $i \in [0, 2k + 1]$, we refer to the nodes in $V_i := L_i \cap V_H$ as being *relevant* for the concerned layer.

We now state some key observations, which immediately follow from the description above.

Observation 5.4. *For all $i \in [1, 2k]$, we have $V_i \subseteq V(M^{\texttt{in}})$.*

Observation 5.5. *For all $i \in [0, k]$, we have $E_{2i} = E\left(G[V_{2i} \cup V_{2i+1}]\right)$. Furthermore, for all $i \in [1, k]$, we have $E_{2i-1} = M^{\texttt{in}} \cap (V_{2i-1} \times V_{2i})$. Thus, if i is even, then G_i consists of all the edges from G that connect two relevant nodes across the concerned layers. In constrast, if i is odd, then G_i consists of the edges from $M^{\texttt{in}}$ that connect two relevant nodes across the concerned layers.*

Observation 5.6. *Consider any augmenting path $p = (v_0, v_1, \ldots, v_{2k+1})$ w.r.t. $M^{\texttt{in}}$ in G that survives the random partitioning. Then we have $(v_i, v_{i+1}) \in E_i$ for all $i \in [0, 2k]$.*

Nested matchings: Fix any $j \in [0, k]$, and for each $i \in [0, j]$ consider a matching $M_{2i} \subseteq E_{2i}$ in G_{2i}. We say that the sequence of matchings $M_0, M_2, \ldots M_{2j}$ is *nested* iff for all $i \in [1, j]$ and all $v \in V(M_{2i}) \cap V_{2i}$, we have $\texttt{mate}_{M^{\texttt{in}}}(v) \in V(M_{2i-2})$.

Observation 5.7. *Consider any sequence of nested matchings $M_0, M_2, \ldots, M_{2k}$. Then there exists a collection of node-disjoint length $(2k + 1)$-augmenting paths of size $|M_{2k}|$ w.r.t. $M^{\texttt{in}}$ in G.*

Proof. Consider any node $v \in V(M_{2k}) \cap V_{2k+1}$. Consider the path $p(v) = (v_0, v_1, \ldots, v_{2k+1})$ in G, which is constructed according to the following procedure.

- $v_{2k+1} \leftarrow v$, and $i \leftarrow 2k$. (Note that v_{2k+1} is at layer $2k + 1$ and is matched under M_{2k}.)
- While $i \geq 0$:
  - If i is even, then $v_i \leftarrow \texttt{mate}_{M_i}(v_{i+1})$.
  - If i is odd, then $v_i \leftarrow \texttt{mate}_{M^{\texttt{in}}}(v_{i+1})$.
  - $i \leftarrow i - 1$.

Since the sequence $M_0, M_2, \ldots, M_{2k}$ is nested, applying Observation 5.4 and Observation 5.5, we can show (by an induction on the number of iterations of the While loop) that $p(v)$ is a length $(2k+1)$-augmenting path in G w.r.t. $M^{\texttt{in}}$. Furthermore, it is easy to see that the paths $\{p(v)\}_{v \in V(M_{2k}) \cap V_{2k+1}}$ constructed in this manner are mutually node-disjoint. This implies the observation. ☐

Important parameters: We fix a constant $\psi \in (0,1)$, which depends on k and γ, i.e., $\psi = \Theta_{k,\gamma}(1)$, and is chosen to be sufficiently small (see Corollary 5.16). Next, for each $i \in [0,k]$, we define:

$$\psi_i := \frac{1}{10^8} \cdot \psi^{5^{4i+3}} \text{ and } \delta_i := \psi^{5^{4i+1}}. \tag{1}$$

Consider any matching M' between the nodes at layers $2i$ and $2i+1$, where $i \in [0,k]$. Intuitively, the parameters ψ_i and δ_i will determine how large M' needs to be so as to make us "happy". Note that the values of δ_i and ψ_i decrease in a doubly exponential manner with i. This fact will be crucially used during the analysis in Section 5.1.2.

A relatively informal summary of the algorithm: Motivated by Observation 5.7, the template algorithm attempts to find a sequence of nested matchings ending at layer $2k$. Specifically, the algorithm runs in *iterations*. At the start of a given iteration, we maintain a sequence of nested matchings $M_0, M_2, \ldots, M_{2i}$ up to some layer $2i$, such that $|M_{2j}| \geq \psi_j \cdot n$ for all $j \in [0,i]$. If $i = k$, then by Observation 5.7 we can already identify a collection of $\psi_k \cdot n = \Theta_{k,\gamma}(1) \cdot n$ many node-disjoint length $(2k+1)$-augmenting paths in G w.r.t. $M^{\texttt{in}}$, and so we just apply those augmenting paths to $M^{\texttt{in}}$ and return the resulting matching $M^{\texttt{out}}$. Henceforth, assume that $i < k$. We classify each node in V_H as either *alive* or *dead* (at the start of the first iteration every node was alive). We also enforce the invariant that all the nodes currently matched in $M_0 \cup M_2 \cup \cdots \cup M_{2i}$ are alive. During the current iteration, we attempt to find a large matching M' between the alive nodes in G_{2i+2}, while ensuring that the sequence $M_0, M_2, \ldots, M_{2i}, M'$ remains nested. Specifically, we make a call to the subroutine $\texttt{LargeMatching}(S, \delta_{i+1})$, for an appropriate $S \subseteq V_{2i+2} \cup V_{2i+3}$. Depending on the outcome of this call, we now fork into one of the following three cases.

Case (a): The call to $\texttt{LargeMatching}(S, \delta_{i+1})$ returns a matching M'. Thus, we are guaranteed that $|M'| \geq \frac{1}{10^8} \cdot (\delta_{i+1})^5 \cdot n \geq \psi_{i+1} \cdot n$. We set $M_{2i+2} := M'$, $i := i+1$, and proceed to the next iteration.

Case (b): The call to $\texttt{LargeMatching}(S, \delta_{i+1})$ returns $\perp$, and $i = -1$ (i.e., the sequence of matchings $M_0, M_2, \ldots, M_{2i}$ was empty). Here, we terminate the template algorithm and return FAILURE.

Case (c): The call to $\texttt{LargeMatching}(S, \delta_{i+1})$ returns $\perp$, and $i \geq 0$. Here, we change the status of all the nodes in $V(M_{2i}) \cap V_{2i+1}$, along with their matched neighbors under $M^{\texttt{in}}$ (who are at layer $2i+2$), from alive to dead. We then delete the matching M_{2i}, set $i := i-1$, and proceed to the next iteration.

We will need some more notations while working with this algorithm in Section 5.2. Accordingly, below we present a more detailed and technical description of the template algorithm, along with these additional notations. While going through the rest of this section, the reader will find it helpful to refer back to the informal description above, whenever necessary.

Iterations: In each iteration $t \geq 1$, we will compute a matching $M^{(t)}$ in the subgraph $G_{\sigma(t)}$, where $\sigma(t) \in \{0, 2, 4, \ldots, 2k\}$. The mapping $\sigma : T \to \{0, 2, \ldots, 2k\}$ will be constructed in an online manner, i.e., we will assign the value $\sigma(t)$ only during the t^{th} iteration. We now describe the state of the algorithm at the end of any given iteration.

At the end of an iteration t, a subset of past iterations $\Lambda^{(t)} \subseteq [t]$ are designated as being *active* w.r.t. t. If $\Lambda^{(t)} \neq \emptyset$, then we write $\Lambda^{(t)} := \left\{\lambda_0^{(t)}, \lambda_1^{(t)}, \ldots, \lambda_{\texttt{stack}(t)}^{(t)}\right\}$, where $\texttt{stack}(t) := \left|\Lambda^{(t)}\right| - 1$ and $\lambda_0^{(t)} < \lambda_1^{(t)} < \cdots < \lambda_{\texttt{stack}(t)}^{(t)}$. The sequence of matchings $M^{\left(\lambda_0^{(t)}\right)}, M^{\left(\lambda_1^{(t)}\right)}, \ldots, M^{\left(\lambda_{\texttt{stack}(t)}^{(t)}\right)}$ corresponds to the sequence $M_0, M_2, \ldots, M_{2i}$ in the discussion immediately after Observation 5.7. Thus, the algorithm satisfies the following invariants.

Invariant 5.8. *We have* $\texttt{stack}(t) \leq k$, *and* $\sigma\left(\lambda_j^{(t)}\right) = 2j$ *for all* $j \in [0, \texttt{stack}(t)]$.

Invariant 5.9. *The sequence of matchings* $M^{\left(\lambda_0^{(t)}\right)}, M^{\left(\lambda_1^{(t)}\right)}, \ldots, M^{\left(\lambda_{\texttt{stack}(t)}^{(t)}\right)}$ *is nested.*

Invariant 5.10. $\left|M^{\left(\lambda_j^{(t)}\right)}\right| \geq \psi_j \cdot n$ *for all* $j \in [0, \texttt{stack}(t)]$.

For each layer $i \in [0, 2k+1]$, the set of relevant nodes V_i is partitioned into two subsets: A_i and D_i. We refer to the nodes in A_i as *alive*, and the nodes in D_i as *dead*. We let $A := \bigcup_{i=0}^{2k+1} A_i$ and $D := \bigcup_{i=0}^{2k+1} D_i$ respectively denote the set of all alive and dead nodes, across all the layers. The next invariant states that every matched node in an active iteration is alive.

Invariant 5.11. $A \supseteq V\left(M^{\left(\lambda_j^{(t)}\right)}\right)$ *for all* $j \in [0, \texttt{stack}(t)]$.

At the start of the first iteration (when $t = 1$), every relevant node is alive (i.e., $A_i = V_i$ and $D_i = \emptyset$ for all $i \in [0, 2k+1]$). Subsequently, over time the status of a relevant node can only change from being alive to being dead, but *not* the other way round. Thus, with time, the set D keeps growing, where the set A keeps shrinking. We now explain how to implement a given iteration t.

Implementing iteration t: Let $i = \texttt{stack}(t-1)$. If $\Lambda^{(t-1)} = \emptyset$, then we set $i = -1$. If $i = k$, then there will be no more iterations, i.e., the algorithm will last for only $t-1$ iterations. In this scenario, we know that the sequence of matchings $M^{\left(\lambda_0^{(t-1)}\right)}, M^{\left(\lambda_1^{(t-1)}\right)}, \cdots, M^{\left(\lambda_k^{(t-1)}\right)}$ is nested. Based on this sequence, we identify a collection of $|M^{\left(\lambda_k^{(t-1)}\right)}|$ many node-disjoint augmenting paths w.r.t. $M^{\texttt{in}}$ in G, augment $M^{\texttt{in}}$ along those paths (see Observation 5.7), and return the resulting matching $M^{\texttt{out}}$. Accordingly, from now on we assume that $i \leq k-1$.

During iteration t, we will attempt to find a large matching M' in G_{2i+2} between two sets of nodes: A_{2i+3} and C_{2i+2}. Recall that A_{2i+3} denotes the alive nodes at layer $2i+3$. We refer to C_{2i+2} as the set of *candidate nodes* for iteration t. Intuitively, we pick as many nodes from V_{2i+2} into the set C_{2i+2} as possible, subject to two constraints: (i) if we append M' at the end of the sequence of matchings from the currently active iterations, then the resulting sequence will continue to remain nested, and (ii) the nodes in C_{2i+2} are currently alive. This leads us to the following definition.

$$C_{2i+2} := \begin{cases} \left\{ v \in A_{2i+2} : \texttt{mate}_{M^{\texttt{in}}}(v) \in V\left(M^{\left(\lambda_i^{(t-1)}\right)}\right) \right\} & \text{if } i \geq 0; \\ A_{2i+2} & \text{else if } i = -1. \end{cases}$$

We now call the subroutine $\texttt{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$, in an attempt to obtain a large matching in $G[C_{2i+2} \cup A_{2i+3}] = G_{2i+2}[C_{2i+2} \cup A_{2i+3}]$. The last equality holds because of Observation 5.5, and since $C_{2i+2} \subseteq V_{2i+2}$ and $A_{2i+3} \subseteq V_{2i+3}$. We set $\sigma(t) := 2i+2$. Now, we fork into one of the following three cases.

Case (a): The call to $\texttt{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$ returns a matching M'. Thus, we are guaranteed that $|M'| \geq \frac{1}{10^8} \cdot (\delta_{i+1})^5 \cdot n \geq \psi_{i+1} \cdot n$. We set $M^{(t)} := M'$, $\Lambda^{(t)} := \Lambda^{(t-1)} \cup \{t\}$ and $\texttt{stack}(t) := \texttt{stack}(t-1) + 1$. This implies that $\lambda_j^{(t)} := \lambda_j^{(t-1)}$ for all $j \in [0, \texttt{stack}(t-1)]$, and $\lambda_{\texttt{stack}(t)}^{(t)} := t$. Henceforth, we refer to this iteration t as a *forwarding iteration* at layer $(2i+2)$. We now move on to the next iteration $(t+1)$.

Case (b): The call to $\texttt{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$ returns $\perp$, and $i = -1$. Here, the algorithm terminates and returns FAILURE.

Case (c): The call to $\texttt{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$ returns $\perp$, and $i \geq 0$. Here, we set $M^{(t)} := \emptyset$. We also change the status of all the nodes in C_{2i+2}, along with their matched neighbors under $M^{\texttt{in}}$ (who are at layer $2i+1$), from alive to dead, and respectively move these nodes from A_{2i+1} to D_{2i+1} and from A_{2i+2} to D_{2i+2}. Next, we set $\Lambda^{(t)} := \Lambda^{(t-1)} \setminus \{\lambda_i^{(t-1)}\}$ and $\texttt{stack}(t) := \texttt{stack}(t-1) - 1$. This implies that $\lambda_j^{(t)} := \lambda_j^{(t-1)}$ for all $j \in [0, \texttt{stack}(t)]$. Henceforth, we refer to this iteration t as a *backtracking iteration* for layer $2i$. We now move on to the next iteration $(t+1)$.

Remark: From the above description of the template algorithm, it immediately follows that Invariants 5.8, 5.9, 5.10 and 5.11 continue to hold at the end of each iteration t.

5.1.2 Analysis

In this section, we analyze the template algorithm, and prove the following lemma.

Lemma 5.12. *The algorithm* $\texttt{Augment-Template}(G, M^{\texttt{in}}, k, \gamma)$ *runs for at most* $T = \Theta_{k,\gamma}(1)$ *iterations. It either returns a matching* $M^{\texttt{out}}$ *in* G *of size* $|M^{\texttt{out}}| \geq |M^{\texttt{in}}| + \Theta_{k,\gamma}(1) \cdot n$ *(we say that the algorithm "succeeds" in this case), or it returns* FAILURE. *Furthermore, if* $M^{\texttt{in}}$ *admits a collection of* $\gamma \cdot n$ *many node-disjoint length* $(2k+1)$*-augmenting paths in* G*, then the algorithm succeeds whp.*

We start by focusing on bounding the number of iterations (see Corollary 5.14).

Claim 5.13. *There can be at most* $1/(\psi_i)$ *backtracking iterations for layer* $2i$*, where* $i \in [0, k-1]$.

Proof. Consider any backtracking iteration t for layer $2i$. Then we have $\sigma(t-1) = i$, and Invariant 5.10 implies that

$$\left| V\left(M^{\left(\lambda_i^{(t-1)}\right)}\right) \cap V_{2i+1} \right| = \left| M^{\left(\lambda_i^{(t-1)}\right)} \right| \geq \psi_i \cdot n.$$

Thus, during iteration t, at least $\psi_i \cdot n$ nodes at layer $(2i+1)$ change their status from alive to dead. Since there are at most n nodes at layer $(2i+1)$, such an event can occur at most $1/(\psi_i)$ times. □

Corollary 5.14. *The algorithm* $\texttt{Augment-Template}(G, M^{\texttt{in}}, k, \gamma)$ *has at most* $\Theta_{k,\gamma}(1)$ *iterations.*

Proof. Let T_f, T_b and T_0 respectively denote the total number of forwarding iterations across all layers, the total number of backtracking iterations across all layers, and the total number of iterations across all layers that are neither forwarding nor backtracking.We have $T_0 = 1$ if the template algorithm returns FAILURE, and $T_0 = 0$ otherwise.

We now observe that: there cannot exist a sequence of more than $(k+1)$ consecutive forwarding iterations, for otherwise, the $(k+2)^{th}$ forwarding iteration in this sequence would have to take place at a layer $\geq (2k+2)$, which does not exist. Hence, we have: $T_f \leq (k+1) \cdot (T_b + T_0) + (k+1)$, and the total number of iterations is bounded by:

$$T = T_f + T_b + T_0 \leq (k+1) \cdot (T_b + T_0) + (k+1) + T_b + T_0 = \Theta(k) \cdot T_b \leq \Theta(k) \cdot \sum_{i=0}^{k-1} \frac{1}{\psi_i} = \Theta_{k,\gamma}(1).$$

The second inequality follows from Claim 5.13, and the last equality follows from (1). □

We now move on to showing that if $M^{\texttt{in}}$ admits a collection $\gamma \cdot n$ many node-disjoint length $(2k+1)$-augmenting paths in G, then the template algorithm succeeds whp. Towards this end, let $\mathcal{P}$ denote a maximum-sized collection of node-disjoint length $(2k+1)$-augmenting paths in G

w.r.t. $M^{\texttt{in}}$. Let $\mathcal{P}^* \subseteq \mathcal{P}$ be the subset of paths in $\mathcal{P}$ that survive the random partitioning. If $|\mathcal{P}| \geq \gamma \cdot n$, then Lemma 5.3 guarantees that whp:

$$|\mathcal{P}^*| \geq \Theta_{k,\gamma}(1) \cdot n. \tag{2}$$

At any point in time during the execution of the algorithm `Augment-Template`$(G, M^{\texttt{in}}, k, \gamma)$, we say that a path $p \in \mathcal{P}^*$ is *alive* if *all* the nodes on p are alive, and we say that the path p is *dead* otherwise. Let $\mathcal{P}^*_A \subseteq \mathcal{P}^*$ and $\mathcal{P}^*_D = \mathcal{P}^* \setminus \mathcal{P}^*_A$ respectively denote the set of alive and dead paths at any point in time. Just before the start of iteration 1, we have $\mathcal{P}^*_A = \mathcal{P}^*$ and $\mathcal{P}^*_D = \emptyset$. Subsequently, a path $p \in \mathcal{P}^*$ can change its status from alive to dead only during a backtracking iteration (note that this change occurs in only one direction, i.e., a dead path will never become alive). The next claim upper bounds the number of such changes.

Claim 5.15. *During a backtracking iteration for layer $2i$, where $i \in [0, k-1]$, at most $\delta_{i+1} \cdot n$ many paths in $\mathcal{P}^*$ moves from $\mathcal{P}^*_A$ to $\mathcal{P}^*_D$.*

Proof. Let $t \geq 1$ denote a backtracking iteration for layer $2i$. During iteration t, the algorithm calls `LargeMatching`$(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$, which returns $\perp$. Consider the subgraph $G' = G[C_{2i+2} \cup A_{2i+3}]$. We have: $\mu(G') < \delta_{i+1} \cdot n$, for otherwise the call to `LargeMatching`$(.,.)$ would not have returned $\perp$.

Just before iteration t, let $\mathcal{P}' \subseteq \mathcal{P}^*_A$ denote the subset of paths in $\mathcal{P}^*_A$ that pass through some node in C_{2i+2}. Only the paths in $\mathcal{P}'$ move from $\mathcal{P}^*_A$ to $\mathcal{P}^*_D$ at the end of iteration t. We can, however, form a matching in G' which contains one edge from each path in $\mathcal{P}'$. Hence, we have $|\mathcal{P}'| \leq \mu(G') < \delta_{i+1} \cdot n$. This concludes the proof of the claim. □

Corollary 5.16. *Let $\psi = \Theta_{k,\gamma}(1)$ be a sufficiently small constant depending on k and γ, and suppose that (2) holds. Then throughout the entire duration of the algorithm, we have:*

$$|\mathcal{P}^*_A| \geq |\mathcal{P}^*| - \sum_{i=0}^{k-1} \frac{\delta_{i+1}}{\psi_i} \cdot n \geq \delta_0 \cdot n.$$

Proof. From (1), Claim 5.13 and Claim 5.15, we infer that:

$$|\mathcal{P}^*_A| \geq |\mathcal{P}^*| - \sum_{i=0}^{k-1} \frac{\delta_{i+1}}{\psi_i} \cdot n \geq |\mathcal{P}^*| - k \cdot (10^8 \psi) \cdot n. \tag{3}$$

Now, since we can set ψ to be any sufficiently small constant value depending on k and γ, and since $\delta_0 \leq \psi$ according to (1), from (2) we get: $|\mathcal{P}^*| - k \cdot (10^8 \psi) \cdot n \geq \delta_0 \cdot n$. This concludes the proof. □

Corollary 5.17. *If (2) holds, then the algorithm does* not *return* FAILURE.

Proof. For contradiction, suppose that the algorithm returns FAILURE at the end of an iteration t.

Let $i = \sigma(t-1)$. Since the algorithm returns FAILURE after iteration t, we must have $i = -1$. Furthermore, during iteration t, the call to `LargeMatching`$(C_0 \cup A_1, \delta_0)$ must have returned $\perp$. Let $G' = G[C_0 \cup A_1]$. It follows that:

$$\mu(G') < \delta_0 \cdot n. \tag{4}$$

Next, observe that $C_0 = A_0$. Hence, just before the start of iteration t, we could have formed a matching in G' by taking the first edge of each path in $\mathcal{P}^*_A$. Thus, from Corollary 5.16, we get:

$$\mu(G') \geq |\mathcal{P}^*_A| \geq \delta_0 \cdot n. \tag{5}$$

However, both (4) and (5) cannot simultaneously be true. This leads to a contradiction. □

Note that if the template algorithm does *not* return FAILURE, then it necessarily returns a matching $M^{\texttt{out}}$ of size $|M^{\texttt{out}}| \geq |M^{\texttt{in}}| + \psi_k \cdot n$ (this holds because of Invariant 5.9, Invariant 5.10 and Observation 5.7). Finally, recall that $\psi_k = \Theta_{k,\gamma}(1)$ as per (1). Lemma 5.12 now follows from Corollary 5.14, Lemma 5.3 and Corollary 5.17.

5.2 Implementation in Sublinear Models

In this section, we show how to implement the template algorithm from Section 5.1, when we are allowed access to the input graph G only via adjacency-matrix queries. Throughout this section, we use the following parameters (recall Definition 5.1).

$$\epsilon_0 := \epsilon_{\texttt{in}}, \text{ and } \epsilon_t := 9 \cdot \epsilon_{t-1} \text{ for all } t \in [1, T]. \tag{6}$$

In Section 5.1.1, the template algorithm starts with iteration $t = 1$. Here, we use the phrase "iteration $t = 0$" to refer to the scenario just before the start of the first iteration. Towards this end, for consistency of notations, we define $\epsilon_{-1} := 2$, $M^{(0)} := M^{\texttt{in}}$, $\sigma(0) := \perp$, $\texttt{stack}(0) := -1$ and $\Lambda^{(0)} := \emptyset$. Further, we define an oracle $\texttt{alive}_0(v)$ that is supposed to return YES if v is alive at the end of iteration 0 (i.e., just before the start of iteration 1), and return NO otherwise.

The rest of this section is organized as follows. Lemma 5.18 shows how to implement each iteration of the template algorithm, under adjacency-matrix query access to the input graph G. Its proof appears at the end of this section. Theorem 5.2 now follows from Lemma 5.12 and Corollary 5.19.

Lemma 5.18. *Suppose that we can access the input graph G only via adjacency-matrix queries, and we have an oracle $\texttt{match}_{M^{in}}(.)$ with $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{in}})$ query time. Then we can implement each iteration $t \geq 0$ of the algorithm $\texttt{Augment-Template}(G, M^{in}, k, \gamma)$, as described in Section 5.1, in $\tilde{O}_{k,\gamma}(n^{2-\epsilon_{t-1}})$ time. Furthermore, if the concerned iteration t does not result in the algorithm returning* FAILURE*, then we can ensure that we have access to the following data structures at the end of iteration t.*

- *An oracle $\texttt{match}_{M^{(t)}}(.)$ for the matching $M^{(t)}$, that has a query time of $\tilde{O}_{k,\gamma}(n^{1+\epsilon_t})$.*
- *An oracle $\texttt{alive}_t(.)$ that has a query time of $\tilde{O}_{k,\gamma}(n^{1+\epsilon_t})$. When queried with a node $v \in V$, this oracle returns* YES *if v is alive at the end of iteration t, and returns* NO *otherwise.*
- *The values $\sigma(t)$ and $\texttt{stack}(t)$, and the contents of the set $\Lambda^{(t)}$.*

Corollary 5.19. *Let $\epsilon_{out} := 9^T \cdot \epsilon_{in}$, where $T = \Theta_{k,\gamma}(1)$ is the maximum possible number of iterations of the template algorithm (see Lemma 5.12). Then it takes $\tilde{O}_{k,\gamma}(n^{2-\epsilon_{in}})$ time to implement the template algorithm, under adjacency-matrix query access to G. Further, if the template algorithm does not return* FAILURE*, then at the end of our implementation we have an oracle $\texttt{match}_{M^{out}}(.)$ for the matching M^{out} returned by it, with query time $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{out}})$.*

Proof. By Lemma 5.18, each iteration t of the template algorithm can be implemented in time $\tilde{O}_{k,\gamma}(n^{2-\epsilon_{t-1}}) = \tilde{O}_{k,\gamma}(n^{2-\epsilon_{\texttt{in}}})$, since $\epsilon_{\texttt{in}} \leq \epsilon_{t-1}$. Thus, the total time taken to implement the template algorithm is at most $\tilde{O}_{k,\gamma}(T \cdot n^{2-\epsilon_{\texttt{in}}}) = \tilde{O}_{k,\gamma}(n^{2-\epsilon_{\texttt{in}}})$.

Suppose that the template algorithm terminates at the end of iteration t, and returns a matching $M^{\texttt{out}}$. Then, at the end of iteration t of our sublinear implementation, the situation is as follows.

$\sigma(t) = k$, and $\Lambda^{(t)} = \left\{\lambda_0^{(t)}, \lambda_1^{(t)}, \cdots, \lambda_k^{(t)}\right\}$,where $\sigma\left(\lambda_j^{(t)}\right) = 2j$ for each $j \in [0, k]$ (see Invariant 5.8). The sequence of matchings in $\Lambda^{(t)}$ is nested (see Invariant 5.9). Thus, from this sequence of nested matchings we can extract a set of at least $\left|M^{\left(\lambda_k^{(t)}\right)}\right|$ many node-disjoint length $(2k+1)$-augmenting paths w.r.t. $M^{\texttt{in}}$ in G (see Observation 5.7). The template algorithm obtains the matching $M^{\texttt{out}}$ by applying these augmenting paths to $M^{\texttt{in}}$. In our sublinear implementation of the template algorithm, however, we can access each matching $M \in \Lambda^{(t)}$ only via an oracle $\texttt{match}_M(.)$, which has a query time of at most $\tilde{O}_{k,\gamma}(n^{1+\epsilon_t})$ (see (6) and Lemma 5.18). Furthermore, we can access the matching $M^{\texttt{in}}$ only via the oracle $\texttt{match}_{M^{\texttt{in}}}(.)$, which also has a query time of at most $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{\texttt{in}}}) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_t})$.

We now show how to answer a query to the oracle $\texttt{match}_{M^{\texttt{out}}}(v)$. The key observation is this:

Let $E^* := (M^{\texttt{in}} \cap E_H) \bigcup_{M \in \Lambda^{(t)}} M$ (see the discussion on layered subgraphs in Section 5.1.1). Then the graph $G^* = (V, E^*)$ consists of a collection of node-disjoint alternating paths w.r.t. $M^{\texttt{in}}$. We say that a path in G^* is *complete* iff it has one endpoint at layer 0 and the other endpoint at layer $(2k+1)$. Now, a node $v \in V$ is matched in $M^{\texttt{out}}$ iff: either $v \in V(M^{\texttt{in}})$, or $v \notin V(M^{\texttt{in}})$ and v is the starting/end point of a complete path in G^*.

Using this observation, we now describe how to answer queries of the form: "Is $\texttt{match}_{M^{\texttt{out}}}(v) = \perp$ for a given node $v \in V$?". To answer such a query, we apply the procedure below.

If $\texttt{match}_{M^{\texttt{in}}}(v) \neq \perp$, then we return that $\texttt{match}_{M^{\texttt{out}}}(v) \neq \perp$. Else if $\texttt{match}_{M^{\texttt{in}}}(v) = \perp$ and $\ell(v) \notin \{0, 2k+1\}$, then we return that $\texttt{match}_{M^{\texttt{out}}}(v) = \perp$. Finally, if $\texttt{match}_{M^{\texttt{in}}}(v) = \perp$ and w.l.o.g. $\ell(v) = 0$, then we perform the following steps.

- $v_0 \leftarrow v$.
- **For** $i = 1$ to $2k+1$:
  - **If** i is odd, **then** $v_i \leftarrow \texttt{mate}_{M^{\left(\lambda_{(i-1)/2}^{(t)}\right)}}(v_{i-1})$.
  - **Else if** i is even, **then** $v_i \leftarrow \texttt{mate}_{M^{\texttt{in}}}(v_{i-1})$.
  - **If** $v_i = \perp$, **then** return that $\texttt{match}_{M^{\texttt{out}}}(v) = \perp$.
- Return that $\texttt{match}_{M^{\texttt{out}}}(v) \neq \perp$.

It is easy to verify that the above procedure correctly returns whether or not $\texttt{match}_{M^{\texttt{out}}}(v) = \perp$. We can extend this procedure in a natural manner, which would also allow us to answer the query $\texttt{match}_{M^{\texttt{out}}}(v)$. To summarize, we can answer a query $\texttt{match}_{M^{\texttt{out}}}(v)$ by making at most one call to each of the oracles $\texttt{match}_M(.)$, for $M \in \Lambda^{(t)}$, and at most $\Theta(k)$ calls to the oracle $\texttt{match}_{M^{\texttt{in}}}(.)$. Each of these oracle calls take at most $\tilde{O}_{k,\gamma}(n^{1+\epsilon_t})$ time, as $\epsilon_{t'} \leq \epsilon_t$ for all $t' \in [1, t]$. Since $\left|\Lambda^{(t)}\right| = k$, the oracle $\texttt{match}_{M^{\texttt{out}}}(.)$ has a query time of $\tilde{O}_{k,\gamma}(k \cdot n^{1+\epsilon_t}) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_t}) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_{\texttt{out}}})$, where the last equality holds since $\epsilon_t \leq \epsilon_T = \epsilon_{\texttt{out}}$. This concludes the proof. □

Proof of Lemma 5.18

We prove the lemma by induction on t.

Base case $(t = 0)$**:**

We already have the oracle $\texttt{match}_{M^{(0)}}(.)$ with query time $\tilde{O}_{k,\gamma}(n^{1+\epsilon_0})$, since $\epsilon_0 = \epsilon_{\texttt{in}}$ and $M^{(0)} = M^{\texttt{in}}$. We set $\sigma(t) \leftarrow \perp$, $\texttt{stack}(t) \leftarrow -1$ and $\Lambda^{(0)} \leftarrow \emptyset$. We now claim that we already have the oracle $\texttt{alive}_0(.)$. This is because a node $v \in V$ is alive just before the start of iteration 1 if and only if $v \in V_H$. Furthermore, given a query $\texttt{alive}_0(v)$, we can determine whether v is in V_H by checking the value of $\ell(v)$, setting $u \leftarrow \texttt{mate}_{M^{(0)}}(v)$, and then checking the value of $\ell(u)$ if $u \neq \perp$. Thus, answering a query to the oracle $\texttt{alive}_0(.)$ takes $\tilde{O}_{k,\gamma}(n^{1+\epsilon_0})$ time. So, we can implement iteration 0 in $O(1)$ time, and Lemma 5.18 holds for $t = 0$.

Inductive case $(t \geq 1)$**:**

We assume that Lemma 5.18 holds for all $t' \in [0, t-1]$, and that we have access to the data structures constructed during all these past iterations. We now focus on implementing the current iteration t (see Section 5.1.1) under adjacency-matrix query access to G. Let $i = \texttt{stack}(t-1)$. If $i = k$, then the algorithm would terminate after iteration $(t-1)$. Henceforth, we assume that $i \in [-1, k-1]$. In the current iteration t, the template algorithm wants to first compute a matching $M^{(t)}$ by calling the subroutine $\texttt{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$. We first show that we can efficiently query whether or not a given node in V belongs to the set $C_{2i+2} \cup A_{2i+3}$. Subsequently, we split up our implementation of iteration t into two steps, as described below.

Claim 5.20. *Given any node $v \in V$, we can determine if $v \in C_{2i+2}$ in $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$ time.*

Proof. We first check the value of $\ell(v)$ and call $\texttt{alive}_{t-1}(v)$. Now, we consider the following cases.

(i) $\ell(v) \neq 2i+2$. Here, we return that $v \notin C_{2i+2}$.

(ii) $\ell(v) = 2i+2$ and $\texttt{alive}_{t-1}(v) = \textsc{No}$. Here, we also return that $v \notin C_{2i+2}$.

(iii) $\ell(v) = 2i+2$, $\texttt{alive}_{t-1}(v) = \textsc{Yes}$, and $i = -1$. Here, we return that $v \in C_{2i+2}$.

(iv) $\ell(v) = 2i+2$, $\texttt{alive}_{t-1}(v) = \textsc{Yes}$ and $i \geq 0$. Here, we first set $u_v \leftarrow \texttt{mate}_{M^{\texttt{in}}}(v)$, which takes $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{\texttt{in}}}) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$ time. Next, we call $\texttt{match}_{M\left(\lambda_i^{(t-1)}\right)}(u_v)$, which also takes at most $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$ time. Finally, we return that $v \in C_{2i+2}$ iff $\texttt{match}_{M\left(\lambda_i^{(t-1)}\right)}(u_v) \neq \perp$.

The correctness of the above procedure follows from the definition of the set C_{2i+2}. Furthermore, the preceding discussion implies that this procedure overall takes at most $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$ time. ☐

Corollary 5.21. *Given any $v \in V$, we can determine if $v \in C_{2i+2} \cup A_{2i+3}$ in $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$ time.*

Proof. We can determine if $v \in A_{2i+3}$ by checking the value of $\ell(v)$ and making a query $\texttt{alive}_{t-1}(v)$, which takes $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$ time. The corollary now follows from Claim 5.20. ☐

Step I: Constructing the oracle $\texttt{match}_{M^{(t)}}(.)$. Armed with Corollary 5.21, we mimic the call to $\texttt{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$ in the template algorithm, by invoking Theorem 4.1 with $A = C_{2i+2} \cup A_{2i+3}$, $\delta_{\texttt{in}} = \delta_{i+1}$, $\epsilon = 2 \cdot \epsilon_{t-1}$ and $t_A = \tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$.[10] If Theorem 4.1 returns $\perp$, then

[10] The reader should keep in mind that in the current section (Section 6), we are using the symbol A to denote the set of alive nodes across all the layers. This is different from the way the symbol A is being used in the statement of Theorem 4.1, where it refers to any arbitrary subset of nodes.

we set $M^{(t)} := \emptyset$, and the trivial oracle $\texttt{match}_{M^{(t)}}(.)$ has $O(1) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_t})$ query time. Otherwise, Theorem 4.1 returns an oracle $\texttt{match}_M(.)$ for a matching M, and we set $M^{(t)} := M$. By (6) and Theorem 4.1, this oracle $\texttt{match}_{M^{(t)}}(.)$ has query time:

$$\tilde{O}\left(\frac{(t_A+n)\cdot n^{4\epsilon}}{\text{poly}(\delta_{i+1})}\right) = \tilde{O}_{k,\gamma}\left((t_A+n)\cdot n^{4\epsilon}\right) = \tilde{O}_{k,\gamma}\left(n^{1+\epsilon_{t-1}+4\epsilon}\right) = \tilde{O}_{k,\gamma}\left(n^{1+\epsilon_t}\right).$$

Finally, from (6) and Theorem 4.1, we infer that overall Step I takes time:

$$\begin{aligned}\tilde{O}\left(\frac{(t_A+n)\cdot(n^{1-\epsilon}+n^{4\epsilon})}{\text{poly}(\delta_{i+1})}\right) &= \tilde{O}_{k,\gamma}\left((t_A+n)\cdot(n^{1-\epsilon}+n^{4\epsilon})\right)\\ &= \tilde{O}_{k,\gamma}((t_A+n)\cdot n^{1-\epsilon})\\ &= \tilde{O}_{k,\gamma}\left(n^{2+\epsilon_{t-1}-\epsilon}\right)\\ &= \tilde{O}_{k,\gamma}\left(n^{2-\epsilon_{t-1}}\right).\end{aligned}$$

In the above derivation, the second equality holds since $\epsilon = 2\epsilon_{t-1} \leq 9^T\epsilon_{\texttt{in}} \leq 1/5$ (see (6) and Definition 5.1), whereas the third equality holds since $t_A = \tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$.

Step II: Determining $\sigma(t), \texttt{stack}(t)$, $\Lambda^{(t)}$, and the oracle $\texttt{alive}_{(t)}(.)$. We set $\sigma(t) \leftarrow 2i+2$. We now fork into one of the following three cases.

Case (a) In Step I, the invocation of Theorem 4.1 returned an oracle $\texttt{match}_M(.)$ for a matching M, and we set $M^{(t)} := M$. This will be referred to as a forwarding iteration at layer $2i+2$. In this case, we set $\Lambda^{(t)} \leftarrow \Lambda^{(t-1)} \cup \{t\}$ and $\texttt{stack}(t) \leftarrow \texttt{stack}(t-1)+1$. Now, we observe that the set of alive nodes does not change during such a forwarding iteration, and so we already have the oracle $\texttt{alive}_t(.)$, because $\texttt{alive}_t(v) = \texttt{alive}_{(t-1)}(v)$ for all $v \in V$. Accordingly, the oracle $\texttt{alive}_t(.)$ has query time $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}}) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_t})$.

Case (b): In Step I, the invocation of Theorem 4.1 returned $\perp$, and $i = -1$. Here, the algorithm terminates and returns FAILURE.

Case (c): In Step I, the invocation of Theorem 4.1 returned $\perp$, and $i \geq 0$. This will be referred to as a backtracking iteration at layer $2i$. In this case, we set $\Lambda^{(t)} \leftarrow \Lambda^{(t-1)} \setminus \{\lambda_i^{(t-1)}\}$ and $\texttt{stack}(t) \leftarrow \texttt{stack}(t-1)-1$. Now, we observe that due to iteration t, only the nodes in C_{2i+2} and their matched neighbors under $M^{\texttt{in}}$ (who are at layer $2i+1$), change their status from alive to dead. The status of every other node remains unchanged. Thus, we can answer a query $\texttt{alive}_t(v)$, in $\tilde{O}_{k,\gamma}(n^{1+\epsilon_t})$ time, as follows.

We first check the value of $\ell(v)$, query $\texttt{alive}_{t-1}(v)$ and $\texttt{match}_{M^{\texttt{in}}}(v)$, and determine whether or not $v \in C_{2i+2}$ by invoking Claim 5.20. Overall, this takes $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}}) + \tilde{O}_{k,\gamma}(n^{1+\epsilon_{\texttt{in}}}) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_t})$ time. Next, we consider three cases.

(i) $\ell(v) \notin \{2i+1, 2i+2\}$. Here, we return $\texttt{alive}_t(v) = \texttt{alive}_{(t-1)}(v)$.

(ii) $\ell(v) = 2i+2$. Here, if $v \in C_{2i+2}$ then we return $\texttt{alive}_t(v) = \text{NO}$; otherwise we return $\texttt{alive}_t(v) = \texttt{alive}_{t-1}(v)$.

(iii) $\ell(v) = 2i+1$. Here, we set $u_v \leftarrow \texttt{mate}_{M^{\texttt{in}}}(v)$. Now, if $u_v \in C_{2i+2}$ then we return $\texttt{alive}_t(v) = \text{NO}$; otherwise we return $\texttt{alive}_t(v) = \texttt{alive}_{t-1}(v)$.

To summarize, Step I takes $\tilde{O}_{k,\gamma}(n^{2-\epsilon_{t-1}})$ time, whereas Step II takes only $O(k) = O_{k,\gamma}(1)$ time. Furthermore, at the end of Step II we have all the desired data structures for iteration t, and both

the oracles $\texttt{match}_{M^{(t)}}(.)$ and $\texttt{alive}_t(.)$ have a query time of at most $\tilde{O}(n^{1+\epsilon_t})$. Finally, Theorem 4.1 ensures that whp, the way we decide whether we are in case (a), case (b) or case (c) is consistent with the choice made by the template algorithm in the same scenario (see the discussion on "implementing iteration t" in Section 5.1.1, and how the subroutine $\texttt{LargeMatching}(S, \delta)$ is defined in the second paragraph of Section 5.1). This concludes the proof of Lemma 5.18.

6 $(1, \epsilon n)$-Approximate Matching Oracle

In this section, starting from an empty matching, we repeatedly apply Theorem 5.2 to obtain our main results in the sublinear setting. They are summarized in the theorem and the corollary below, which are restatements of Theorem 1.3.

Theorem 6.1. *Let $\gamma, \epsilon'' \in (0,1)$ be any two small constants, and let G be the input graph with n nodes which we can access via adjacency-matrix queries. Then for a sufficiently small constant $\epsilon' \in (0, \epsilon'')$, there exists an algorithm which: In $\tilde{O}_\gamma(n^{2-\epsilon'})$ time, returns an oracle $\texttt{match}_M(.)$ for a $(1, 3\gamma n)$-approximate matching M in G, where the oracle $\texttt{match}_M(.)$ has $\tilde{O}_\gamma(n^{1+\epsilon''})$ query time.*

Corollary 6.2. *Given adjacency-matrix query access to an n-node graph G and any constant $\gamma \in (0,1)$, in $\tilde{O}_\gamma(n^{2-\epsilon})$ time we can return a $(1, 4\gamma n)$-approximation to the value of $\mu(G)$, whp. Here, $\epsilon \in (0,1)$ is a sufficiently small constant depending on γ.*

Proof. First, we apply Theorem 6.1, with $\epsilon = \epsilon'$, to get the oracle $\texttt{match}_M(.)$ in $\tilde{O}_\gamma(n^{2-\epsilon})$ time. Note that M is a $(1, 3\gamma n)$-approximate matching in $G = (V, E)$. Using Chernoff bound, we now compute a $(1, \gamma n)$-approximate estimate $\hat{\mu}$ of $|M|$ by sampling, uniformly at random, a set S of $\tilde{O}_\gamma(1)$ nodes from V and querying $\texttt{match}_M(v)$ for each node $v \in S$. This takes $\tilde{O}_\gamma(n^{1+\epsilon''}) = \tilde{O}_\gamma(n^{2-\epsilon})$ time. The last inequality holds since $\epsilon = \epsilon'$ and ϵ'' are chosen to be sufficiently small, so that $1 + \epsilon'' \leq 2 - \epsilon$. It is now easy to observe that $\hat{\mu}$ is a $(1, 4\gamma n)$-approximation to the value of $\mu(G)$. □

Proof of Theorem 6.1

Algorithm 4 contains the relevant pseudocode. We slightly abuse the notation in step 2-(b) of Algorithm 4, when we write $Z = (M^{\texttt{out}}, \epsilon_{\texttt{out}})$. Here, we essentially mean that $\texttt{Augment}(G, M^{\texttt{in}}, i, \gamma^2, \epsilon_{\texttt{in}})$ returns the oracle $\texttt{match}_{M^{\texttt{out}}}(.)$ with query time $\tilde{O}_\gamma(n^{1+\epsilon_{\texttt{out}}})$. Similarly, in step 2-(b), when we write $M^{\texttt{in}} \leftarrow M^{\texttt{out}}$, this means that henceforth we will refer to the oracle $\texttt{match}_{M^{\texttt{out}}}(.)$ as $\texttt{match}_{M^{\texttt{in}}}(.)$.

The idea behind Algorithm 4 is simple and intuitive. We start by initializing $M^{\texttt{in}} \leftarrow \emptyset$, $k \leftarrow \lceil 1/\gamma \rceil$ and $\epsilon_{\texttt{in}} \leftarrow \epsilon'$, where $\epsilon' \in (0,1)$ is a sufficiently small constant. At this point, we trivially have the oracle $\texttt{match}_{M^{\texttt{in}}}(.)$ with query time $\tilde{O}_\gamma(n^{1+\epsilon_{\texttt{in}}})$. The algorithm now runs in *rounds*. In each round, it repeatedly tries to augment the matching $M^{\texttt{in}}$ along small-length augmenting paths, by successively calling $\texttt{Augment}(G, M^{\texttt{in}}, i, \gamma^2, \epsilon_{\texttt{in}})$ for $i \in [0, k]$. Whenever a call to $\texttt{Augment}(\cdot)$ succeeds, the algorithm feeds its output into the next call to $\texttt{Augment}(\cdot)$. The algorithm terminates whenever it encounters a round where every call to $\texttt{Augment}(\cdot)$ returns FAILURE.

Claim 6.3. *Algorithm 4 runs for $\tilde{O}_\gamma(1)$ rounds, and makes $\tilde{O}_\gamma(1)$ calls to $\texttt{Augment}(\cdot)$.*

Proof. Say that a given round of Algorithm 4 is *successful* iff during that round: for some $i \in [0, k]$, the call to $\texttt{Augment}(G, M^{\texttt{in}}, i, \gamma^2, \epsilon_{\texttt{in}})$ succeeded (see Theorem 5.2 and step 2-(a) of Algorithm 4). By Theorem 5.2, each time a call to $\texttt{Augment}(G, M^{\texttt{in}}, i, \gamma^2, \epsilon_{\texttt{in}})$ succeeds, it increases the size of the matching $M^{\texttt{in}}$ (see step 2-(b) of Algorithm 4) by at least $\Theta_\gamma(1) \cdot n$. Since $\mu(G) \leq n$, such an event can occur at most $\Theta_\gamma(1)$ times. Finally, each round of Algorithm 4 consists of $(k+1) = \Theta_\gamma(1)$ calls to $\texttt{Augment}(\cdot)$, and all but the last round is successful. This implies the claim. □

Algorithm 4 Near-optimal-matching-oracle $(G = (V, E), \gamma)$.

Choose $\epsilon' \in (0, 1)$ to be a sufficiently small constant.
$\epsilon_{\texttt{in}} \leftarrow \epsilon'$, $k \leftarrow \lceil 1/\gamma \rceil$, $M^{\texttt{in}} \leftarrow \emptyset$.
$\tau \leftarrow$ TRUE.
While $\tau =$ TRUE**:** // Start of a new round

1. $\tau \leftarrow$ FALSE.
2. **For** $i = 0$ to k:
   (a) $Z \leftarrow \texttt{Augment}(G, M^{\texttt{in}}, i, \gamma^2, \epsilon_{\texttt{in}})$. // See Theorem 5.2
   (b) **If** $Z \neq$ FAILURE, **then**
   - Suppose that $Z = (M^{\texttt{out}}, \epsilon^{\texttt{out}})$.
   - $M^{\texttt{in}} \leftarrow M^{\texttt{out}}$, $\epsilon_{\texttt{in}} \leftarrow \epsilon_{\texttt{out}}$.
   - $\tau \leftarrow$ TRUE.

$M \leftarrow M_{\texttt{in}}$, $\epsilon'' \leftarrow \epsilon_{out}$.
Return the oracle $\texttt{match}_M(\cdot)$**, which has query time** $\tilde{O}_\gamma(n^{1+\epsilon''})$**.**

Claim 6.4. *Suppose that at the start of a given round of Algorithm 4, there exists a collection of at least $\gamma^2 \cdot n$ many node-disjoint length $(2i+1)$-augmenting paths w.r.t. $M^{\texttt{in}}$ in G, for some $i \in [0, k]$. Then whp, Algorithm 4 does* not *terminate at the end of the given round.*

Proof. If there exists some $j \in [0, i-1]$ such that the call to $\texttt{Augment}(G, M^{\texttt{in}}, j, \gamma^2, \epsilon_{\texttt{in}})$ succeeds during the given round, then it immediately implies the claim (since we would have $\tau =$ TRUE when the round ends and so the **While** loop in Algorithm 4 will run for at least one more iteration).

For the rest of the proof assume that during the given round, for all $j \in [0, i-1]$ the call to $\texttt{Augment}(G, M^{\texttt{in}}, j, \gamma^2, \epsilon_{\texttt{in}})$ returns FAILURE, and hence the matching $M^{\texttt{in}}$ does not change during iterations $j = 0$ to $i-1$ of the **For** loop. Accordingly, at the start of the concerned iteration i of the **For** loop, the matching $M^{\texttt{in}}$ still admits a collection of at least $\gamma^2 \cdot n$ many node-disjoint length $(2i+1)$-augmenting paths in G. Thus, by Theorem 5.2, the call to $\texttt{Augment}(G, M^{\texttt{in}}, i, \gamma^2, \epsilon_{inp})$ succeeds whp. So, it follows that Algorithm 4 does not end after the given round, whp. □

Corollary 6.5. *When Algorithm 4 terminates, whp $M^{\texttt{in}}$ is a $(1, 3\gamma n)$-approximate matching in G.*

Proof. Let M^* be a maximum matching in G. By Claims 6.3 and 6.4, the following holds whp when the algorithm terminates: For all $i \in [0, k]$, there exists at most $\gamma^2 \cdot n$ many length-$(2i+1)$ augmenting paths in $M^{\texttt{in}} \cup M^*$.

As $k = \lceil 1/\gamma \rceil$, the augmenting paths in $M^{\texttt{in}} \cup M^*$ that are of length $\leq 2k+1$ contribute at most $(k+1) \cdot \gamma^2 n \leq 2\gamma n$ extra edges to M^* compared to $M^{\texttt{in}}$. On the other hand, augmenting paths $M^{\texttt{in}} \cup M^*$ that are of length $> (2k+1)$ contribute at most $\frac{(k+2)-(k+1)}{k+1} \cdot |M^{\texttt{in}}| \leq \gamma \cdot |M^{\texttt{in}}| \leq \gamma n$ extra edges to M^* compared to $M^{\texttt{in}}$. Thus, we get: $|M^*| \leq |M^{\texttt{in}}| + 3\gamma n$. □

Since Algorithm 4 makes only constantly many calls to $\texttt{Augment}(\cdot)$, we can choose $\epsilon' > 0$ to be sufficiently small so as to guarantee that $0 < \epsilon'' \ll 1$ (see Claim 6.3 and Theorem 5.2). Further, during the execution of Algorithm 4, each call to $\texttt{Augment}(\cdot)$ takes $\tilde{O}_\gamma(n^{2-\epsilon_{\texttt{in}}}) = \tilde{O}_\gamma(n^{2-\epsilon'})$ time. Theorem 6.1 now follows from Claim 6.3 and Corollary 6.5.

7 Dynamic $(1+\epsilon)$-Approximate Matching Size

We now prove our main result in the dynamic setting; as summarized in the theorem below. Note that Theorem 7.1 is a restatement of Theorem 1.2.

Theorem 7.1. *There is a dynamic $(1+\epsilon)$-approximate matching* size *algorithm with $m^{0.5-\Omega_\epsilon(1)}$ worst-case update time, where m is the number of edges in the dynamic input graph $G=(V,E)$ with n nodes. The algorithm is randomized and works against an adaptive adversary whp. Moreover, the algorithm maintains an oracle $\texttt{match}_M(.)$ with query time $\tilde{O}(m^{0.5+\epsilon'})$ (for a small constant $\epsilon'>0$ which depends on ϵ), where M is a $(1+\epsilon)$-approximate maximum matching of G.*

To highlight the main idea behind the proof of Theorem 7.1, first we recall that using techniques presented in a series of papers [AKL19, Beh23, BDH20, BKSW23, Kis22], we can assume: $\mu(G)=\Omega(n)$ throughout the sequence of updates. Accordingly, consider the following dynamic matching size algorithm, which runs in phases, where each phase lasts for ϵn updates. At the start of a phase, we compute a $(1+\epsilon)$-approximate estimate μ^* of $\mu(G)$, by invoking Corollary 6.2, in $\tilde{O}_\epsilon(n^{2-\epsilon})$ time. Sine $\mu(G)=\Omega(n)$, the value of μ^* continues to remain a $(1+\Theta(\epsilon))$-approximate estimate of $\mu(G)$ throughout the duration of the phase. This already leads to an amortized update time of: $\tilde{O}_\epsilon(n^{2-\epsilon})/(\epsilon n)=\tilde{O}_\epsilon(n^{1-\epsilon})$, which is sublinear in n. We now show how to extend this idea to get an update time that is sublinear in $\sqrt{m}$, and how to answer queries in $m^{0.5+\epsilon'}$ time.

Proof of Theorem 7.1

For ease of exposition, we first focus on proving an amortized update time bound. We start by recalling a useful technique for sparsifying G, which allows us to assume that $\mu(G)=\Omega(n)$.

Contractions: Consider a function $\phi: V\to V_\phi$ which maps every node in V to some element in the set V_ϕ. We say that ϕ is a *contraction* of G iff $|V_\phi|\leq|V|$. Define the multiset of edges $E_\phi:=\{(u,v)\in E:\phi(u)\neq\phi(v)\}$, and consider the multigraph $G_\phi:=(V_\phi,E_\phi)$. From every matching in G_ϕ, we can recover a matching in G of same size. Hence, we have: $\mu(G_\phi)\leq\mu(G)$.

The next theorem follows immediately from past work on the maximum matching problem across a range of computational models [AKL19, Beh23, BDH20, BKSW23, Kis22]. For the sake of completeness, however, we outline the proof of Lemma 7.2 in Appendix B.

Lemma 7.2. *There exists a dynamic algorithm $\mathcal{A}$ with $\tilde{O}(1)$ worst-case update time, which maintains: a set of $K=\tilde{O}(1)$ contractions $\{\phi_1,\ldots,\phi_K\}$ of G, the corresponding graphs $\{G_{\Phi_1},\ldots,G_{\Phi_K}\}$, and a subset $I\subseteq[1,K]$. Throughout the sequence of updates (whp against an adaptive adversary) the algorithm ensures that: (i) $|V_{\phi_i}|=\Theta\left(\frac{\mu(G)}{\epsilon}\right)$ for all $i\in I$, and (ii) there is an index $i^*\in I$ such that $(1-\epsilon)\cdot\mu(G)\leq\mu(G_{\phi_{i^*}})\leq\mu(G)$.*

Description of our dynamic algorithm: We maintain a $(2+\epsilon)$-approximate estimate $\hat{\mu}$ of $\mu(G)$, in $\tilde{O}(1)$ worst-case update time, using an existing deterministic dynamic matching algorithm as a subroutine [BCH17]. We also use the algorithm $\mathcal{A}$, as in Theorem 7.2, as a subroutine. Let $\epsilon_0\in(0,1)$ be a sufficiently small constant, depending on ϵ.

Our dynamic algorithm partitions the update sequence into *phases*. We now explain how the algorithm works during a given phase, which can be of two types.

Type-I Phase: At the start of a type-I phase, we have $\hat{\mu}\geq|E|^{0.5+\epsilon_0}$. Let $m_{\texttt{init}}$ denote the value of $|E|$ at the start of the phase. Then the phase will last for the next $\epsilon\cdot(m_{\texttt{init}})^{0.5+\epsilon_0}$ updates. At the start of the phase, we call an existing static algorithm to compute a $(1+\epsilon)$-approximate maximum

matching M of G, which takes $O_\epsilon(m_{\texttt{init}})$ time [DP14]. Define $\mu^* = |M|$. Throughout the phase, μ^* continues to remain a $(1+2\epsilon)$-approximate estimate of $\mu(G)$, and we continue to output the same value μ^*. This leads to an amortized update time of:

$$\frac{O_\epsilon(m_{\texttt{init}})}{\epsilon \cdot (m_{\texttt{init}})^{0.5+\epsilon_0}} = O_\epsilon\left((m_{\texttt{init}})^{0.5-\epsilon_0}\right) = m^{0.5-\Omega_\epsilon(1)}.$$

The last equality holds since $|E| = m = \Theta(m_{\texttt{init}})$ throughout the phase. We can ensure that throughout the phase, the algorithm explicitly maintains M, which remains a $(1+2\epsilon)$-approximate maximum matching of G. This gives us the matching oracle $\texttt{match}_M(.)$, with constant query time.

Type-II Phase: At the start of a type-II phase, we have $\hat{\mu} < |E|^{0.5+\epsilon_0}$. Let $\hat{\mu}_{\texttt{init}}$ and $m_{\texttt{init}}$ respectively denote the value of $\hat{\mu}$ and $|E|$ at the start of the phase. The phase will last for the next $\epsilon\hat{\mu}_{\texttt{init}}$ updates. Hence, $\mu(G)$ can change by at most a multiplicative $(1+\epsilon)$ factor during the phase.

At the start of the phase, for each $i \in I$, we find a $(1, 4\gamma)$-approximate estimate μ_i^* of $\mu(G_{\phi_i})$. We obtain μ_i^* by invoking Corollary 6.2 on G_{ϕ_i}, with $\gamma = \epsilon^2$.[11] This takes time:

$$\tilde{O}\left(|V_{\phi_i}|^{2-\epsilon^*}\right) = \tilde{O}_\epsilon\left((\mu(G))^{2-\epsilon^*}\right) = \tilde{O}_\epsilon\left((\hat{\mu}_{\texttt{init}})^{2-\epsilon^*}\right),$$

where $\epsilon^* \in (0,1)$ is a sufficiently small constant depending on ϵ. Since $|I| = \tilde{O}(1)$, overall we spend $\tilde{O}_\epsilon\left((\hat{\mu}_{\texttt{init}})^{2-\epsilon^*}\right)$ time to compute μ_i^* for all $i \in I$. Next, in $|I| = \tilde{O}(1)$ time, we find an index $j \in I$ which maximizes the value μ_j^*. From Theorem 7.2, it follows that:

$$(1-\epsilon)\cdot\mu(G) - \Theta(\gamma)\cdot\Theta\left(\frac{\mu(G)}{\epsilon}\right) \leq \mu_j^* \leq \mu(G). \tag{7}$$

As $\gamma = \epsilon^2$, we infer that μ_j^* is a purely multiplicative $(1+\Theta(\epsilon))$-approximate estimate of $\mu(G)$. We continue to output the same value μ_j^* throughout the phase, since we have already observed that during the phase $\mu(G)$ changes by at most a multiplicative $(1+\epsilon)$ factor.

The phase lasts for $\epsilon\hat{\mu}_{\texttt{init}}$ updates. Accordingly, this leads to an amortized update time of:

$$\frac{\tilde{O}_\epsilon\left((\hat{\mu}_{\texttt{init}})^{2-\epsilon^*}\right)}{\epsilon\hat{\mu}_{\texttt{init}}} = \tilde{O}_\epsilon\left((\hat{\mu}_{\texttt{init}})^{1-\epsilon^*}\right) = \tilde{O}_\epsilon\left((m_{\texttt{init}})^{0.5+\epsilon_0-\epsilon^*}\right) = m^{0.5-\Omega_\epsilon(1)}.$$

The last equality holds because we can ensure that ϵ_0 is sufficiently small compared to ϵ^* (which, in turn, depends on ϵ), and since $|E| = m = \Theta(m_{\texttt{init}})$ throughout the duration of the phase.

Because of (7), at the start of the phase we can construct the oracle $\texttt{match}_M(.)$ by invoking Theorem 6.1 on the graph $G_{\phi_{j^*}}$. Over the $\epsilon\hat{\mu}_{\texttt{init}}$ edge updates of the phase, M continues to remain a $(1+O(\epsilon))$-approximate maximum matching in G. Finally, to maintain the oracle under edge insertions/deletions during the phase, we simply ignore edge deletions and assume that if a vertex is matched by a deleted edge of M then it is unmatched.

Improving the update time bound to worst-case: Recall that $\hat{\mu}_{init}$ denotes the value of $\hat{\mu}$ at the beginning of a phase. As observed after the initialization of a phase the output maintained by the algorithm remains $(1+O(\epsilon))$-approximate for the next $O(\epsilon\cdot\hat{\mu}_{init})$ updates. Furthermore, the total computational work done by the algorithm in both types of phases is upper bounded by $\tilde{O}(\hat{\mu}_{init}^{2-\epsilon^*})$ for some small constant $\epsilon^* \in (0,1)$ depending on the parameters of the algorithm. Let G_i stand for the state of the input graph at the beginning of phase i and let $\mathcal{A}(G_i)$ stand for the output

[11] It is trivial to verify that Corollary 6.2 holds even when applied on a multigraph.

of the previously described algorithm initialized on G_i. Let $\hat{\mu}_{init,i}$ stand for the value of $\hat{\mu}_{init}$ at the beginning of phase i. We will now describe the behaviour of the worst-case update time algorithm.

The improved algorithm similarly initializes it's output to be $\mathcal{A}(G_1)$. Throughout the first three phases it does not alter it's output and during the first two phases it doesn't complete any background computation. During phase i for $i > 2$ the algorithm calculates $\mathcal{A}(G_{i-2})$ distributing the work evenly throughout the phase. Note that by phase i the algorithm has complete knowledge of G_{i-2}. At the end of the same phase it switches it's output to be $\mathcal{A}(G_{i-2})$.

As the algorithm only outputs a matching size estimate and an oracle and not an actual matching this switch is done in constant time. Computing $\mathcal{A}(G_{i-2})$ takes time proportional to $\tilde{O}(\hat{\mu}_{init,i-2}^{2-\epsilon^*})$ and is distributed over $\epsilon \cdot \hat{\mu}_{init,i}$ updates. As during a phase $\mu(G)$ may change by at most a $1 + O(\epsilon)$ multiplicative factor we must have that $\hat{\mu}_{init,i} = \Theta(\hat{\mu}_{init,i-2})$. The amortized implementation amortizes the work of computing $\mathcal{A}(G_{i-2})$ over $\epsilon \cdot \hat{\mu}_{init,i-2}$ updates. This implies that the worst-case update time guarantee of the delayed rebuild based algorithm matches the amortized versions update time within a constant factor.

References

[ABKL23] Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. On regularity lemma and barriers in streaming and dynamic matching. In *STOC*, 2023. ↑1

[ACC+18] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 79:1–79:16, 2018. ↑1

[AKL19] Sepehr Assadi, Sanjeev Khanna, and Yang Li. The stochastic matching problem with (very) few queries. *ACM Trans. Economics and Comput.*, 7(3):16:1–16:19, 2019. ↑28, ↑37

[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*, pages 1132–1139. SIAM, 2012. ↑3, ↑40

[ARW17] Amir Abboud, Aviad Rubinstein, and Ryan Williams. Distributed pcp theorems for hardness of approximation in p. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 25–36. IEEE, 2017. ↑1

[BCH17] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in $O(1)$ amortized update time. In *Proceedings of the 19th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 86–98, 2017. ↑28

[BDH20] Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Stochastic matching with few queries: (1-ϵ) approximation. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1111–1124. ACM, 2020. ↑28, ↑37

[Beh22] Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 873–884. IEEE, 2022. ↑2, ↑3, ↑4, ↑7, ↑35, ↑39

[Beh23] Soheil Behnezhad. Dynamic algorithms for maximum matching size. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page To appear in, 2023. ↑1, ↑2, ↑4, ↑28, ↑37

[BFH19] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1899–1918, 2019. ↑1

[BFS12] Guy E Blelloch, Jeremy T Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 308–317, 2012. ↑10

[BGS11] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Proceedings of the 52nd Symposium on Foundations of Computer Science (FOCS)*, pages 383–392, 2011. ↑1

[BGS20] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*, pages 1123–1134, 2020. ↑1

[BHN16] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 398–411, 2016. ↑1

[BK19] Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $O(1/\epsilon^2)$ amortized update time. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1872–1885, 2019. ↑1

[BK21] Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. In *Proceedings of the 48th International Colloquium on Automata, Languages and Programming (ICALP)*, 2021. ↑1

[BK22] Soheil Behnezhad and Sanjeev Khanna. New trade-offs for fully dynamic matching via hierarchical edcs. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3529–3566, 2022. ↑1

[BKS23] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Sublinear algorithms for $(1.5+\epsilon)$-approximate matching. In *STOC*, 2023. ↑3, ↑39

[BKSW23] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 100–128. SIAM, 2023. ↑1, ↑2, ↑4, ↑28, ↑37

[BLM20] Soheil Behnezhad, Jakub Łącki, and Vahab Mirrokni. Fully dynamic matching: Beating 2-approximation in Δ^ϵ update time. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2492–2508, 2020. ↑1

[BNS19] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*, pages 456–480, 2019. ↑1

[BRR23] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. In *STOC*, 2023. ↑3, ↑5, ↑7, ↑39

[BRRS23] Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. Beating greedy matching in sublinear time. *arXiv preprint arXiv:2206.13057*, 2023. To appear at SODA'23. ↑2, ↑3, ↑39

[BS15] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 167–179. Springer, 2015. ↑1, ↑2

[BS16] Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 692–711. SIAM, 2016. ↑1, ↑2

[CKK20] Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear algorithms and lower bounds for metric tsp cost estimation. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. ↑2, ↑39

[CS18] Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial barrier for worst-case time bounds. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 33:1–33:14, 2018. ↑1

[DP14] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1, 2014. ↑1, ↑29, ↑39

[Edm65a] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965. ↑1

[Edm65b] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965. ↑1

[Gha16] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 270–277. SIAM, 2016. ↑3, ↑40

[Gha22] Mohsen Ghaffari. Local computation of maximal independent set. *arXiv preprint arXiv:2210.01104*, 2022. Announced at FOCS'22. ↑3, ↑40

[GLS+19] Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1+\epsilon)$-approximate incremental matching in constant deterministic amortized time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1886–1898. SIAM, 2019. ↑1

[GP13] Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *Proceedings of the 54th Symposium on Foundations of Computer Science (FOCS)*, pages 548–557, 2013. ↑1, ↑2

[GRS14] Manoj Gupta, Venkatesh Raman, and SP Suresh. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 29, pages 227–239, 2014. ↑1

[GSSU22] Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzrad. Maintaining an edcs in general graphs: Simpler, density-sensitive and with worst-case time bounds. *Proceedings of the 5th Symposium on Simplicity in Algorithms (SOSA)*, pages 12–23, 2022. ↑1

[GU19] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1636–1653. SIAM, 2019. ↑3, ↑40

[JJST22] Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Regularized box-simplex games and dynamic decremental bipartite matching. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2022. ↑1

[Kis22] Peter Kiss. Improving update times of dynamic matching algorithms from amortized to worst case. *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 94:1–94:21, 2022. ↑1, ↑5, ↑28, ↑37, ↑38

[KMM12] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Proceedings of the 15th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 231–242, 2012. ↑4

[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. ↑2, ↑3, ↑40

[Kuh55] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. ↑1

[LRV22] Jane Lange, Ronitt Rubinfeld, and Arsen Vasilyan. Properly learning monotone functions via local correction. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 75–86. IEEE, 2022. ↑3

[LRY15] Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 59–61, 2015. ↑3, ↑7, ↑40

[McG05] Andrew McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 170–181. 2005. ↑4, ↑15

[MR09] Sharon Marko and Dana Ron. Approximating the distance to properties in bounded-degree and general sparse graphs. *ACM Transactions on Algorithms (TALG)*, 5(2):1–28, 2009. ↑40

[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. ↑2, ↑3, ↑39

[OR10] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 457–464, 2010. ↑1

[ORRR12] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1123–1131. SIAM, 2012. ↑2, ↑39

[PGVWW20] Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 153–166, 2020. ↑1

[PR07] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007. ↑2, ↑39, ↑40

[RSW22] Mohammad Roghani, Amin Saberi, and David Wajc. Beating the folklore algorithm for dynamic matching. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 111:1–111:23, 2022. ↑1, ↑2, ↑3

[RTVX11] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. *arXiv preprint arXiv:1104.1377*, 2011. ↑3, ↑40

[RV16] Omer Reingold and Shai Vardi. New techniques and tighter bounds for local computation algorithms. *Journal of Computer and System Sciences*, 82(7):1180–1200, 2016. ↑3, ↑40

[San07] Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 118–126, 2007. ↑1

[Sol16] Shay Solomon. Fully dynamic maximal matching in constant update time. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016. ↑1

[Waj20] David Wajc. Rounding dynamic matchings against an adaptive adversary. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 194–207, 2020. ↑1

[YYI09] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum˜ matchings. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 225–234, 2009. ↑39

[YYI12] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM Journal on Computing (SICOMP)*, 41(4):1074–1093, 2012. ↑2, ↑3

A Matching Oracles on Low Degree Graphs

In this section, we prove Lemma 4.3.

Preliminaries on Randomized Greedy Matching. A greedy maximal matching in G with respect to an edge permutation π, denoted by $M = GMM(G, \pi)$ is a maximal matching obtained by scanning through edges with ordering defined by π, and for each edge e, include the edge e into the matching if both of its end point are not matched. The matching oracle $\texttt{match}_M(v)$ of Lemma 4.3 simply returns $\mathsf{VO}(v)$ where the vertex oracle VO and the edge oracle EO are defined in [Beh22] follows.

Algorithm 5 $\mathsf{VO}(v, \pi)$

1. Let $e_1 = (v, u_1), \ldots, e_k = (v, u_k)$ be the edges incident to v where $\pi(e_1) < \cdots < \pi(e_k)$.
2. **for** $i = 1, \ldots, k$: if $\mathsf{EO}(e_i, u_i, \pi) = \mathsf{TRUE}$, then **return** (v, u_i).
3. **return** $\perp$

Algorithm 6 $\mathsf{EO}(e, u, \pi)$

1. **if** $\mathsf{EO}(e, u, \pi)$ is computed, then return the computed answer.
2. Let $e_1 = (u, w_1), \ldots, e_k = (u, w_k)$ be the edges incident to u where $\pi(e_i) < \pi(e)$ and $\pi(e_1) < \cdots < \pi(e_k)$.
3. **for** $i = 1, \ldots, k$: if $\mathsf{EO}(e_i, w_i, \pi) = \mathsf{TRUE}$, then **return** FALSE.
4. **return** TRUE.

Let $T(v, \pi)$ denote the number of recursive calls $\mathsf{EO}(\cdot, \cdot, \pi)$ over the course of answering $\mathsf{VO}(v, \pi)$. The main theorem of [Beh22] is as follows.

Lemma A.1 (Theorem 3.5 of [Beh22]). *Let v be a random vertex and π be a random permutation over edges, independent from v.*

$$\mathbb{E}_{v,\pi}[T(v, \pi)] = \alpha \triangleq O(d \log n).$$

Given access to adjacency list, we can execute $\mathsf{VO}(v, \pi)$ using $O(T(v, \pi)\Delta)$ time straightforwardly. But Behnezhad [Beh22] also showed that we can think of $T(v, \pi)$ as the running time, given access to the adjacency lists:

Lemma A.2 (Lemma 4.1 of [Beh22]). *Let v be an arbitrary vertex in a graph $G = (V, E)$. There is an algorithm that draws a random permutation π over E, and determines whether v is matched in $GMM(G, \pi)$ in time $\tilde{O}(T(v, \pi) + 1)$ having query access to the adjacency lists. The algorithm succeeds w.h.p.*

Basic Properties of Randomized Greedy Matching. Recall that $\overline{d}$ is the given parameter where $\overline{d} \geq d$. We set the *threshold* $\ell = \Theta(\overline{d}\log(n)/\epsilon)$ such that $\ell \geq \alpha \cdot \frac{8}{\epsilon}$. We have by Markov's inequality that

$$\Pr_{v,\pi}[T(v,\pi) > \ell] \leq \epsilon/8. \tag{8}$$

For any edge permutation π, let $f(\pi) = \Pr_{v\sim V}[T(v,\pi) > \ell]$ measure the fraction of vertices such that randomized greedy matching w.r.t. π makes many recursive calls exceeding the threshold ℓ. We say that $\pi \in \Pi$ is *great* if $f(\pi) \leq \epsilon/2$. Observe that

$$\Pr_{\pi}[\pi \text{ is great}] \geq 1/4 \tag{9}$$

Otherwise, $\Pr_{v,\pi}[T(v,\pi) > \ell] \geq \Pr_{v,\pi}[T(v,\pi) > \ell \mid \pi \text{ is not great}] \Pr_{\pi}[\pi \text{ is not great}] > (\epsilon/2)\cdot(1/4)$ which contradicts Equation (8). We also say that $\pi \in \Pi$ is *good* if $f(\pi) \leq \epsilon$, otherwise we say that it is *bad*.

Algorithm 7 TestPerm(π)

1. Sample $r = 1000\log(n)/\epsilon$ vertices independently: $v_1, \ldots, v_r$.
2. Let $X = |i \mid \{T(v_i,\pi) > \ell\}|$ and $\tilde{f} = X/r$.
3. If $\tilde{f} \leq \frac{3}{4}\epsilon$, return "yes". Otherwise, return "no".

A simple procedure in Algorithm 7 accepts a great permutation and rejects a bad permutation with high probability.

Lemma A.3. *If π is great, then* TestPerm(π) *returns "yes" with high probability. If π is bad, then* TestPerm(π) *returns "no" with high probability.*

Proof. If π is great but TestPerm(π) returns "no", then we have $f(\pi) \leq \epsilon/2$ but $\tilde{f} > 3\epsilon/4$. Since $\mathbb{E}[X] = \epsilon \cdot f(\pi)$ and $X = \epsilon \cdot \tilde{f}$, we have

$$X - \mathbb{E}[X] > r\epsilon/4.$$

Applying Chernoff bound Proposition 3.2 with $t = r\epsilon/4$ and $\overline{\mu} = r\epsilon/2 \geq \mathbb{E}[X]$, we have

$$\Pr[X - \mathbb{E}[X] > r\epsilon/4] \leq \exp(-\frac{(r\epsilon/4)^2}{3\overline{\mu}}) \leq \exp(-\frac{r\epsilon}{24}) \leq 1/n^{10}.$$

If π is bad but TestPerm(π) returns "yes", then we have $f(\pi) \geq \epsilon$ but $\tilde{f} < 3\epsilon/4$. This means that $X < \frac{3}{4}\mathbb{E}[X]$. Applying the standard Chernoff bound, i.e.,$\Pr[X < (1-\delta)\mathbb{E}[X]] \leq \exp(-\frac{\delta^2\mathbb{E}[X]}{2})$ for $\delta \in [0,1]$, we have

$$\Pr[X < \frac{3}{4}\mathbb{E}[X]] \leq \exp(-\frac{(\frac{1}{4})^2\mathbb{E}[X]}{2}) \leq \exp(-\frac{r\epsilon}{32}) \leq 1/n^{10}.$$

□

Now, we are ready to prove Lemma 4.3.

Preprocessing. The preprocessing algorithm is as follows. First, independently sample $O(\log n)$ random edge-permutations $\pi_1, \ldots, \pi_{O(\log n)}$. For any i, if TestPerm(π_i) returns "yes", then set $\pi^* \leftarrow \pi_i$.

We claim that π^* is good w.h.p. Recall that each π_i is great with probability at least $1/4$ by Equation (9). So w.h.p. one of the permutation π_i is great and so, by Lemma A.3, TestPerm(π_i) must return "yes" w.h.p. Moreover, also by Lemma A.3, the returned permutation $\pi^*$is not bad w.h.p. That is, π^* is good.

Query. Given a vertex v, we simply execute $\mathsf{VO}(v, \pi^*)$ except that we return $\perp$ if it makes more than ℓ recursive calls. If $\mathsf{VO}(v, \pi^*) = (v, v')$ returns a matched edge, to make sure that the answers on v and v' are consistent, we also call $\mathsf{VO}(v', \pi^*)$. If it turns out that $\mathsf{VO}(v', \pi^*)$ makes more than ℓ recursive calls, then we return $\perp$. Otherwise, $\mathsf{VO}(v', \pi^*)$ must also return (v, v') and then we return (v, v').

By construction, we obtain a matching oracle whose answers are consistent w.r.t. some fixed matching M. If $\ell = \infty$, then M would be a normal randomized greedy maximal matching of size $|M| \geq \mu(G)/2$. This might not be true in reality as we set $\ell = \Theta(d \log(n)/\epsilon)$. But since π^* is good, i.e., $f(\pi^*) = \Pr_{v \sim V}[T(v, \pi) > \ell] \leq \epsilon$. So

$$|M| \geq \mu(G)/2 - \epsilon n.$$

This completes the correctness of Lemma 4.3. It remains to analyze the running time.

Time analysis. Step Item 1 of TestPerm$(\cdot)$ takes $O(r)$ time as we just sample r vertices in G. Step Item 2 takes time $r \cdot \tilde{O}(\ell)$ time where the factor $\tilde{O}(\ell)$ is by Lemma A.2. Since $r = \Theta(\log(n)/\epsilon)$ and $\ell = \Theta(\overline{d} \log(n)/\epsilon)$, each TestPerm takes $\tilde{O}(\overline{d}/\epsilon^2)$. We call TestPerm $O(\log n)$ times. So the total preprocessing time is $\tilde{O}(d/\epsilon^2)$. For the query time, we run VO and makes $O(\ell)$ recursive calls. Therefore, the total query time is $\tilde{O}(\ell) = \tilde{O}(d/\epsilon)$ time by Lemma A.2.

B Vertex Reduction for Dynamic Matching: Proof of Lemma 7.2

This algorithm description is analogous to the algorithm in [Kis22] which extends previous algorithms from [AKL19, Beh23, BDH20, BKSW23] to function against an adaptive adversary. Assume we are given $G = (V, E)$. Using a $\tilde{O}(1)$ worst-case update time deterministic algorithm from literature we maintain an $\alpha = O(1)$-approximate estimate $\hat{\mu}$ of $\mu(G)$. We make $\tilde{O}(1)$ guesses of $\mu(G)$, $1, \alpha, \alpha^2, \ldots, \alpha^k$. For $\mu(G)$ guess α^i we will define $T = \frac{\ln(n) \cdot 512}{\epsilon^2} = \tilde{O}(1)$ contractions of V $\phi_j^i : j \in [T]$ and contracted graphs $G_{\phi_j^i}$. If $\hat{\mu} \in [\alpha^i, \alpha^{i+1})$ we define $\mu(G)$ guess α^i to be the accurate guess at the given time. If guess α^i is currently the accurate guess then the algorithm will maintain that i) for all $j \in [T]$ we have that $|V_{\phi_j^i}| = \Theta(\frac{\mu(G)}{\epsilon})$ and ii) there exists some $j \in [T]$ such that $(1 - \epsilon) \cdot \mu(G) \leq \mu(G_{\phi_j^i}) \leq \mu(G)$.

We will now define how ϕ_j^i is generated. We define a set of vertices $|V_{\phi_j^i}| = \frac{8 \cdot \alpha^{i+1}}{\epsilon}$ and map vertices of V to $V_{\phi_j^i}$ uniformly at random. Note that property i) holds as for the accurate guess we must have that $\mu(G) = \Theta(\alpha^i)$. From the definition of vertex contractions for any contracted graph $G_{\phi_j^i}$ we must have that $\mu(G_{\phi_j^i}) \leq \mu(G)$. Therefore, it remains to show that if α^i is the currently accurate guess of $\mu(G)$ then there exists some $j \in [T]$ such that $(1 - \epsilon) \cdot \mu(G) \leq \mu(G_{\phi_j^i})$.

Let's assume that α^i is the currently accurate guess of $\mu(G)$. Note that this implies that $\mu(G)/\alpha \leq \alpha^i \leq \mu(G) \cdot \alpha$. Let S be an arbitrary subset of V of size $2 \cdot \mu(G)$ representing the possible endpoints of a maximum matching. There can be $\binom{n}{2\mu(G)} \leq n^{2 \cdot \mu(G)} \leq \exp(\ln(n) \cdot 2\mu(G))$ such chooses of S.Fix some $j \in [T]$. For all v vertices of $V_{\phi_j^i}$ define the event X_j^v to be the indicator variable of the event $\phi^{-1}(v) \cap S \neq \emptyset$ and define $\bar{X}_j = \sum_{v \in V_{\phi_j^i}} X_j^v$.

Claim B.1. *For a fixed j events X_j^v are negatively associated random variables.*

The proof of Claim B.1 appears in the appendix of [Kis22]. Let $\beta = \alpha^{i+1}/\mu(G) \in [1, \alpha]$. We first lower bound the expectation of $\hat{X}_j$:

$$\begin{aligned}
\mathbb{E}[X_i^j] &= 1 - \Pr[S \cap V_i^j = \emptyset] \\
&= 1 - \left(1 - \frac{1}{|V_{\phi_j^i}|}\right)^{2\mu(G)} \\
&= 1 - \left(1 - \frac{\epsilon}{8 \cdot \mu(G) \cdot \beta}\right)^{2\mu(G)} \\
&\geq 1 - \exp\left(-\frac{\epsilon}{4 \cdot \beta}\right) \\
&\geq \frac{\epsilon \cdot (1 - \epsilon/(8 \cdot \beta))}{4 \cdot \beta} \qquad (10)
\end{aligned}$$

Inequality 10 holds for small values of ϵ. Therefore,

$$\mathbb{E}[\bar{X}_j] \geq |V_{\phi_i^j}| \cdot \frac{\epsilon \cdot (1 - \epsilon/(8 \cdot \beta))}{4 \cdot \beta} \geq 2\mu(G) \cdot (1 - \epsilon/(8 \cdot \beta))$$

.

Now we apply Chernoff's bound on the sum of negatively associated random variables $\bar{X}_j$ to get that:

$$\begin{aligned}
\Pr\left(\bar{X}_j \leq 2\mu(G) \cdot (1 - \frac{\epsilon}{4 \cdot \beta})\right) &\leq \Pr\left(\bar{X}_j \leq \mathbb{E}[\bar{X}_j] \cdot (1 - \frac{\epsilon}{8}\right) \\
&\leq \exp\left(-\frac{\mathbb{E}[\bar{X}_j] \cdot \left(\frac{\epsilon}{8}\right)^2}{2}\right) \\
&\leq \exp\left(-\frac{\mu(G) \cdot \epsilon^2}{64}\right)
\end{aligned}$$

Recall that we construct $T = \frac{\ln(n) \cdot 512}{\epsilon^2}$ contracted $G_{\phi_i^j}$ for $\mu(G)$ guess α^i.

$$\Pr\left(\min_{j \in [T]} \bar{X}_j \leq 2 \cdot \mu(G) \cdot (1 - \frac{\epsilon}{4})\right) \leq \left(1 - \exp\left(-\frac{\mu(G) \cdot \epsilon^2}{64}\right)\right)^T \leq \exp(-16 \ln(n) \cdot \mu(G))$$

Further recall that S may be selected at most $\exp(2 \ln(n) \cdot \mu(G))$ different ways. Hence, taking a union bound over the possible choices of S we can say that regardless how S was chosen there is a vertex contraction ϕ_j^i such that $\bar{X}_j \geq 2 \cdot \mu(G) \cdot (1 - \epsilon/4)$. Fix any maximum matching M^* of G. By this argument we know that with high probability there must be some $j \in [T]$ such that $\phi_j^i(V[M^*]) \geq 2\mu(G) \cdot (1 - \epsilon/4)$ (here $V[M^*]$ stands for the set of endpoints of M^*). This implies that there might be at most $\mu(G) \cdot \epsilon/2$ vertices of $V[M^*]$ which are mapped not mapped to a unique vertex of $V_{\phi_j^i}$ by ϕ_j^i amongst other vertices of $V[M^*]$. In turn this implies that $\mu(G) \cdot (1 - \epsilon)$ edges of M^* have both their endpoints mapped to unique vertices of $V_{\phi_j^i}$ by ϕ_j^i amongst other endpoints of $\mu(G)$ hence $\mu(G_{\phi_i^j}) \geq \mu(G) \cdot (1 - \epsilon)$.

Observe that this argument holds regardless of the choice of M^* the statement remains true as G undergoes updates even when the updates are made by an adaptive adversary. The contractions ϕ_j^i are fixed at initialization. The task of the algorithm is to maintain maximum matching size estimate $\mu(\hat{G})$ and hence maintain the accurate guess of $\mu(G)$ and to update the contracted graphs. Each contracted graph may undergoes a single update per update to G and there are $\tilde{O}(1)$ contracted graphs. All parts included the worst-case update time of the algorithm is $\tilde{O}(1)$.

C Dynamic $(1+\epsilon)$-Approximate Matching in $O(n)$ Update Time

Consider the following folklore algorithm for explicitly maintaining a matching: recompute a $(1+\epsilon)$-approximate matching M from scratch in $O(m\epsilon^{-1}\log\epsilon^{-1})$ time [DP14] every after $\epsilon m/2n$ edge updates. Before recomputation, if any edge of M is deleted from the graph, we delete it from M.

The amortized update time is clearly $\frac{O(m\epsilon^{-1}\log\epsilon^{-1})}{\epsilon m/2n} = O(n\epsilon^{-2}\log\epsilon^{-1}) = O(n)$. Also, we have $|M| \geq \mu(G)/(1+\epsilon) - \epsilon m/2n$ where the term $\epsilon m/2n$ is because we might decrease the size of M by $\epsilon m/2n$ before we recompute a new $(1+\epsilon)$-approximate matching. But since $\mu(G) \geq m/2n$, we have

$$|M| \geq \mu(G)/(1+\epsilon) - \epsilon\mu(G) \geq \mu(G)/(1+O(\epsilon))$$

implying that M is always a $(1+O(\epsilon))$-approximate matching.

To see why $\mu(G) \geq m/2n$, consider the process where we repeatedly choose an edge e and delete both endpoints of e from the graph until no edge is left. Since the set of deleted edges forms a matching, we may repeat at most $\mu(G)$ times. Also, each deletion removes at most 2Δ edges from the graph. Therefore, $m \leq \mu(G)\cdot 2\Delta \leq \mu(G)\cdot 2n$.

D Tables

Model	Adjacency List		Adjacency List		Adjacency Matrix	
Guarantee	Approx	Time	Approx	Time	Approx	Time
[PR07]	$(2,\epsilon n)$	$\Delta^{O(\log(\Delta/\epsilon))}$				
[NO08]	$(2,\epsilon n)$	$2^{O(\Delta)}/\epsilon^2$				
	$(1,\epsilon n)$	$2^{\Delta^{O(1/\epsilon)}}$				
[YYI09]	$(2,\epsilon n)$	Δ^4/ϵ^2				
	$(1,\epsilon n)$	$\Delta^{O(1/\epsilon^2)}$				
[ORRR12, CKK20]	$(2,\epsilon n)$	$(d+1)\Delta/\epsilon^2$			$(2,\epsilon n)$	$n\sqrt{n}/\epsilon^2$
[Beh22]	$(2,\epsilon n)$	$(d+1)/\epsilon^2$	$2+\epsilon$	$n+\Delta/\epsilon^2$	$(2,\epsilon n)$	n/ϵ^3
[BRRS23]	$(2-\frac{1}{2^{O(1/\gamma)}}, o(n))$	$(d+1)\Delta^\gamma$	$2-\frac{1}{2^{O(1/\gamma)}}$	$n+\Delta^{1+\gamma}$	$(2-\frac{1}{2^{O(1/\gamma)}}, o(n))$	$n^{1+\gamma}$
[BKS23, BRR23]	$(1.5,\epsilon n)$	$nd^{1-\Omega(\epsilon^2)}$	$1.5+\epsilon$	$n\Delta^{1-\Omega(\epsilon^2)}$	$(1.5,\epsilon n)$	$n^{2-\Omega(\epsilon^2)}$
[BRR23] bipartite graph only	$(1.5-\Omega(1), o(n))$	$n^{2-\Omega(1)}$	$1.5-\Omega(1)$	$n^{2-\Omega(1)}$	$(1.5-\Omega(1), o(n))$	$n^{2-\Omega(1)}$
Our					$(1,\epsilon n)$	$n^{2-\Omega_\epsilon(1)}$

Table 1: Summary of sublinear-time algorithms for estimating the size of maximum matching. We omit polylog(n/ϵ) factors. Δ and d denote the maximum and average degree of the graph, respectively.

Reference	Approximation	Query time
[PR07, MR09]	$2+\epsilon$	$\Delta^{O(\log(\Delta/\epsilon))}$
[RTVX11, ARVX12]	2	$\Delta^{O(\Delta \log \Delta)}$
[RV16]	2	$2^{O(\Delta)}$
[LRY15]	2	$\Delta^{O(\log^2 \Delta)}$
	$2+\epsilon$	Δ^4
	$1+\epsilon$	$\Delta^{O(1/\epsilon^2)}$
[Gha16]	2	$\Delta^{O(\log \Delta)}$
[GU19]	2	$\Delta^{O(\log \log \Delta)}$
[Gha22]	2	$\Delta^{O(1)}$
[KMNFT20]	$O(1)$ in expectation	Δ
Our	$(1, \epsilon n)$	$n^{2-\Omega_\epsilon(1)}$

Table 2: Summary of local computation algorithms for matching oracles. We omit $\text{polylog}(n/\epsilon)$ factors. All 2-approximation algorithms actually compute a maximal independent set. Δ and d denote the maximum and average degree of the graph, respectively.