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

Sayan Bhattacharya\textsuperscript{1}
University of Warwick
Coventry, United Kingdom
S.Bhattacharya@warwick.ac.uk

Peter Kiss
University of Warwick
Coventry, United Kingdom
peter.kiss@warwick.ac.uk

Thatchaphol Saranurak\textsuperscript{2}
University of Michigan
Michigan, USA
thsa@umich.edu

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 $m^{0.5+o(1)}\text{update time}$. 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’16]).

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.

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

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.\textsuperscript{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 $O(n/(\log^* n)^{O(1)})$ update time, but their regularity-lemma-based approach inherently cannot give an improvement larger than a $2^{\Omega(\sqrt{\log n})} = n^{o(1)}$ factor. Until now, no dynamic $(1 + \epsilon)$-approximate algorithms can break through the naive $O(n)$ barrier by a polynomial factor.\textsuperscript{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

\textsuperscript{1}Supported by Engineering and Physical Sciences Research Council, UK (EPSRC) Grant EP/S0335330/1

\textsuperscript{2}Supported by NSF CAREER grant 2238138.
2015, Bernstein and Stein [BS15], [BS16] showed a novel approach for maintaining a $(3/2 + \epsilon)$-approximate matching using $O(m^{1/4}) = O(\sqrt{n})$ update time.\footnote{We use $O(\cdot)$ to hide polylog(n) factor throughout the paper.} 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. [BKS23] 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 $O(\sqrt{n})$ update time. While this new connection is very inspiring, 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 I.2.** There is a dynamic $(1+\epsilon)$-approximate matching size algorithm with $n^{0.5-\Omega(1)}$ 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 $v \not\in V(M)$ in $n^{0.5+f(\epsilon)}$ 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 $n^{0.5-\Omega(1)}$ update time. Theorem I.2 thus gives an affirmative answer to the value version of this open question. Although the matching is not explicitly maintained in Theorem I.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], [BKS23] inherently cannot support this query.

We obtain Theorem I.2 by making progress in sublinear algorithms: we show the first sublinear $(1, cn)$-approximate matching algorithm with truly sublinear time even in dense graphs. Here, an $(\alpha, \beta)$-approximate matching means a matching of size at least $\mu(G)/\alpha - \beta$. Given our new sublinear matching algorithm summarized below, Theorem I.2 follows using known techniques.

**Theorem I.3.** There is a randomized algorithm that, given the adjacency matrix of a graph $G$, in time $n^{2-\Omega(1)}$ computes with high probability a $(1, cn)$-approximation $\mu$ 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 $v \not\in V(M)$ where $M$ is a fixed $(1, cn)$-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 I.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.”\footnote{In [RSW22], they use different notation and write $0.99$-approximation instead of 1.01.}. 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 I.3 answers this question negatively since we can get arbitrarily good additive approximation in $n^{2-\Omega(1)}$ time.

To put Theorem I.3 into the larger context of sublinear matching literature, let us discuss its history below. We use $\Delta$ 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 [PRo7], 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], [YY12] 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 $O(d+1)$ time. His algorithms are near-optimal and settle the problem in the regime of approximation ratio at least 2.

Subsequent work focuses on optimizing the approximation ratio within $n^{2-\Omega(1)}$ time. To compare with Theorem I.3, let us discuss only algorithms that use the adjacency matrix. Behnezhad et al. [BRRS23] first...
broke the 2-approximation barrier by computing a \((2 - \Omega(1), o(n))\)-approximate matching in \(O(n^{1+\gamma})\) time. Then \((3/2, \epsilonn)\)-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 Appendix A.

By the first part of Theorem I.3, we show that even \((1, \epsilonn)\)-approximation is possible in \(n^{2-\Omega(1)}\) time. As we mentioned, this result addresses the open question of [RNV22]. 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 \(\alpha\)-approximate matching, we can compute \((\alpha, \epsilonn)\)-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 [YY112], [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 showed a \((1 + \epsilon)\)-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 Appendix A.

The second part of Theorem I.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 I.2, is the first dynamic \((1 + \epsilon)\)-approximate matching size algorithm with \(n^{0.5 - \Omega(1)}\) update time, breaking through the naive yet long-standing \(O(n)\) barrier by a polynomial factor. Our key technical component, Theorem I.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 \((1, \epsilonn)\). 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 II. Then, we set up notations and give preliminaries in Section III. In Section IV, 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 V how to boost the approximation ratio of any matching oracle. By repeatedly boosting the approximation ratio, we give a \((1, \epsilonn)\)-approximate matching oracle (Theorem I.3) in Section VI. Finally, we combine this oracle with known techniques in dynamic algorithms to Theorem I.2 in Section VII.

## II. 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 [BKS23], [Beh23]. For example, the dynamic \((2 - \Omega(1))\)-approximate algorithms in [BKS23], [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 \(O(1)\) 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

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

\(^6\) 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.
LargeMatching\((S, \delta)\) be a subroutine that, given \(S \subseteq V\) and \(\delta\), 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 \(\delta\). By adapting McGregor’s algorithm, we show how to compute a \((1, \gamma n)\)-approximate matching using only \(t = O_\gamma(1)\) calls to

\[
\text{LargeMatching}(S_1, \delta_1), \ldots, \text{LargeMatching}(S_t, \delta_t)
\]

where each \(\delta_i\) is a small constant depending on \(\gamma\).

This algorithm, denoted by \(\text{Alg}(\gamma)\), is our template algorithm (detailed in Section V-A), 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 \(M_i\) is the output of 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 LargeMatching\(()\) 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 LargeMatching\(()\) can be implemented in the sublinear model. Later, we will explain why it is strong enough for implementing the template algorithm \(\text{Alg}(\gamma)\) in the sublinear model.

**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 LargeMatchingOracle\((A, \delta, \epsilon)\) with the following guarantee:

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

\(^7\)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 output \(\Omega(n)\) matching edges in this input instance. The lower bound can be extended even if we allow adjacency-list queries by adding \(\epsilon n\) dummy vertices, each of which connects to every other vertex.

The main challenge of implementing 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 IV.

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 \(n^{2-\Omega(1)}\) time.

Now, we show how to implement the template algorithm \(\text{Alg}(\gamma)\) in \(n^{2-\Omega(1)}(1)\) 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 LargeMatching\((S_i, \delta_i)\) with 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 \(O_\gamma\left(n^{1+\epsilon_{i-1}}\right)\) time and matching-queries for \(M_i\) in \(O_\gamma\left(n^{1+\epsilon_i}\right)\) 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 \(q \cdot O_\gamma\left(n^{1+\epsilon_{i-1}}\right) = O_\gamma\left(n^{1+\epsilon_{i-1}}\right)\).

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

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

To get a \((1, \gamma n)\)-approximate estimate \(\hat{\mu}\) of \(\mu(g)\), we sample \(O(1/\gamma^2)\) vertices and check if they are matched under \(M\). Whp, this is a \((1, O(\gamma n))\)-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 \( n^{2-\Omega(1)} \) 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 \( n^{2-\Omega(1)}/(\gamma^2 n) = n^{1-\Omega(1)} \). In Section VII, we show how to extend this approach to prove Theorem I.2.

### III. Notations and Preliminaries

Unless specified 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 path \( p = (v_0, v_1, \ldots, v_t) \) is an alternating path in \( G \) w.r.t. a matching \( M \subseteq E \) iff \( (v_j, v_{j+1}) \in E \) for all \( j \in [0, t-1] \) 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 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 \( \gamma \) (where \( k \) and \( \gamma \) 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 \( \tilde{O}(\cdot) \) will be used to hide any \( \text{polylog}(n) \) 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 \( A \subseteq V \), an \( A\text{-membership} \) oracle \( \text{mem}_A : V \to \{0, 1\} \) indicates whether \( v \in A \) for any \( v \in V \). That is, we have

\[
\text{mem}_A(v) = 1 \{ v \in A \}.
\]

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

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

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

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

**Concentration Bounds.** We need standard concentration bounds as follows.

**Proposition III.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 \( t > 0 \),

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

**Proposition III.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 \( t \leq \overline{\mu} \),

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

Chernoff bound can be much stronger than Hoeffding bound when \( \mathbb{E}[X] \) has small upper bound. For example, if we applied Proposition III.1 to the setting for Proposition III.2, we would only get that the bound of \( 2 \exp \left( -\frac{2t^2}{2 n \overline{\mu}} \right) \) which is much weaker than \( 2 \exp \left( -\frac{t^2}{3 \overline{\mu}} \right) \) when \( \overline{\mu} \ll n \).

### IV. 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 \( A\text{-membership} \) oracle \( \text{mem}_A \).

**Theorem IV.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 \( A\text{-membership} \) oracle \( \text{mem}_A \) 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 \( \text{match}_M(\cdot) \) for a matching \( M \subseteq G[A] \) of size at least \( \delta_{\text{out}} n \) where \( \delta_{\text{out}} = \delta_{\text{in}}^2/10^8 \) that has \( \tilde{O}((t_A + n)n^{1-\epsilon} + n^{4\epsilon}/\text{poly}(\delta_{\text{in}})) \) worst-case query time. If \( \mu(G[A]) \geq \delta_{\text{min}} n \), then \( \perp \) is not returned. The guarantee holds with high probability.

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

To prove Theorem IV.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 IV.1, the crucial subroutine for later parts of our paper.

This section is organized as follows. In Section IV-A, we show a weaker version of Theorem IV.1 that works well on low degree graphs. We will use this weaker version in the preprocessing step, described in Section IV-B. Then, we complete the query algorithm in Section IV-C.

A. Oracles on Low Degree Graphs

Here, we show a similar result as Theorem IV.1, but it is efficient only when the maximum degree $\Delta$ is small. In particular, the query algorithm makes $n^{O(\epsilon)}$ calls to $\text{mem}_A$ only when $\Delta = n^{O(\epsilon)}$.

Lemma IV.2. Let $G = (V, E)$ be a graph with maximum degree $\Delta$ where $\Delta$ 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 $\text{mem}_A$ with $t_A$ query time. We can construct in $\tilde{O}(t_A \Delta + n + t_A/\Delta/\epsilon^2)$ time a matching oracle $\text{match}_M^\text{low}(\cdot)$ for a $(2, \epsilon n)$-approximate matching $M$ in $G[A]$ that has $\tilde{O}(t_A \Delta + n + t_A/\Delta/\epsilon)$ worst-case query time with high probability.

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

Lemma IV.3. Given the adjacency lists of a graph $G = (V, E)$ with average degree $d$ and a parameter $\bar{d} \leq d$, we can in $\tilde{O}(\bar{d}/\epsilon^2)$ time construct a matching oracle $\text{match}_M(\cdot)$ for a $(2, \epsilon n)$-approximate matching $M$ in $G$ with $\tilde{O}(\bar{d}/\epsilon)$ worst-case query time with high probability.

Lemma IV.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 IV.2, we need to strengthen Lemma IV.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 IV.2. Let $A$ denote the algorithm of Lemma IV.3. We simulate $A$ on $G[A]$ with parameter $\bar{d} \leftarrow \Delta$ as follows.

Whenever $A$ needs to sample a vertex, we sample $O(\log(n)/\epsilon)$ vertices in $G$ and call $\text{mem}_A$ 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. $\vert A \vert \leq \epsilon n$. If this ever happens, even an empty matching is a $(2, \epsilon n)$-approximate matching in $G[A]$, and the problem becomes trivial.

Whenever $A$ 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 $\text{mem}_A$ 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 $A$ can be simulated with an extra $(t_A \cdot O(\log(n)/\epsilon) + t_A \Delta + n)$ factor. \hfill $\square$

B. Preprocessing

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

Lemma IV.4. In $\tilde{O}((t_A + n)(n^{1-\epsilon} + n^{4\epsilon})/\poly(\delta_{in}))$ time, Algorithm 1 outputs either $\bot$ (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) $|M'[A]| \geq 2\delta_{out} n$, or
2) $\mu(G[A \cap V' \setminus V(M')) \geq 4\delta_{out} n$ and $G[V' \setminus V(M')]$ has maximum degree at most $n^{2\epsilon}$.

The algorithm also reports which properties above $M'$ satisfies. If $\mu(G[A]) \geq \delta_{in} n$, then $\bot$ 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.

1) Correctness : In this part, we prove the correctness of Algorithm 1 assuming that it does not return $\bot$. We first show that $\mu_1$ and $\mu_2$ are good approximation of $M'[A]$ and $\tilde{M}'$ base on basic there definition and Hoeffding’s bound.

Lemma IV.5. For every $i$, we have $|M'[A]| - \delta_{out} n \leq \tilde{\mu}_1 \leq |M'[A]|$ w.h.p.

Proof. The probability that a random edge from $M_i$ is in $G[A]$ is $r_1 (|M'[A]|)$. So $\mathbb{E}[X] = r_1 (|M'[A]|)$ and $|M'[A]| = |M'| \mathbb{E}[X]$. By definition of $\tilde{\mu}_1 = \frac{|M'[A]|}{\epsilon n} - \frac{\delta_{out} n}{2}$ and by Hoeffding bound Proposition III.1, we have

$$\Pr[|\tilde{\mu}_1| < |M'[A]| - \delta_{out} n \text{ or } \tilde{\mu}_1 > |M'[A]|]$$

$$= \Pr\left[\frac{|M'| \cdot (X - \mathbb{E}[X])}{r_1} > \frac{\delta_{out} n}{2}\right]$$

$$\leq 2 \exp\left(-\frac{(2\delta_{out} n)^2}{r_1 \left(\frac{\delta_{out} n}{2}\right)^2}\right) = 2 \exp(-\delta_{out}^2 r_1/2)$$

$$\leq 1/n^{10}.$$

$\square$
Algorithm 1 Preprocess $G$.

- $p = 1000n^{2-2ε} \log n$, $k = n^ε$, $η = \delta \log(n)/10$, $T = 100/\delta n^2$, $δ_{out} = \delta/\log T = \delta/100^k$.
- $r_1 = r_2 = 1000δ n^2 \log n$, $r_3 = 1000δ n^2 \log n$.
- $V' \leftarrow V$.

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}{Case 1:}
- Sample $r_1$ edges from $M^i$.
- Let $X$ count the sampled edges that are in $G[A]$ (using the oracle $mem_1$).
- Set $\bar{µ}_1 = \frac{|M^i|X}{r_1} - \frac{δ_{out}n^2}{2}$.
- If $\bar{µ}_1 \geq 2δ_{out}n^2$, then set $M' \leftarrow M^i$ and report that $M'$ satisfies Case 1.

\Case{2}{Case 2:}
- Let $M^i$ be a $(2, δ_{out}n)$-approximate matching in $G[A' \setminus V(M^i)]$ that the matching oracle $match^{lb}(\cdot)$ from Lemma IV.2 respects, given graph $G[V' \setminus V(M^i)]$ with vertex set $A' \setminus V(M^i)$ as input.
- Sample $r_2$ vertices from $V' \setminus V(M^i)$.
- Let $Y$ count the sampled vertices that are matched in $M^i$ (using the oracle $match^{ub}(\cdot)$).
- Set $\bar{µ}_2 = \frac{|V' \setminus V(M^i)|Y}{2r_2} - \frac{δ_{out}n^2}{2}$.
- If $\bar{µ}_2 \geq 4δ_{out}n^2$, then set $M' \leftarrow M^i$ and report that $M'$ satisfies Case 2.

3) Let $A'_{sp} \subseteq A'$ be obtained by sampling $r_3$ vertices from $A'$.
4) Let $G = (V', \cup_{i=1}^k M^i)$.
5) Let $C = \{v \in V' \mid N_G(v, A'_{sp}) \geq η\}$, i.e., $C$ contains remaining vertices that have at least $η$ neighbors from $A'_{sp}$ in $G$.
6) Set $V' \leftarrow V' \setminus C$.

Return $\bot$ (Error).

Lemma IV.6. For every $i$, we have $|\hat{M}^i| - δ_{out}n \leq \bar{µ}_2 \leq |\hat{M}^i|$ w.h.p.

Proof. The probability that a random vertex from $V' \setminus V(M^i)$ is in $V(\hat{M}^i)$ is $\frac{|V(\hat{M}^i)|}{|V' \setminus V(M^i)|}$. So $\mathbb{E}[Y] = \frac{2|\hat{M}^i|}{2|V' \setminus V(M^i)|}$ and $|\hat{M}^i| = \frac{|V' \setminus V(M^i)|Y}{2r_2}$. By definition of $\bar{µ}_2 = \frac{|V' \setminus V(M^i)|Y}{2r_2} - \frac{δ_{out}n^2}{2}$ and by Hoeffding bound Proposition III.1, we have
\[
\mathbb{P}[\bar{µ}_2 < |\hat{M}^i| - δ_{out}n \text{ or } \bar{µ}_2 > |\hat{M}^i|] = \mathbb{P}\left[\frac{|V' \setminus V(M^i)|Y}{2r_2} - \frac{δ_{out}n^2}{2} > \frac{|V' \setminus V(M^i)|Y}{2r_2} - \frac{δ_{out}n^2}{2}\right] \\
\leq 2 \exp\left(-\frac{2(δ_{out}n)^2}{8\left(\frac{1}{2}\right)^2}\right) = 2 \exp\left(-\frac{2δ_{out}n^2r_2}{2}\right) \\
\leq 1/n^{10}.
\]

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 IV.7. For every $i$, the maximum degree of $G[V' \setminus V(M^i)]$ is at most $n^{2ε}$ 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 $n^{2ε}$ w.h.p. (we use the convention that if $v \notin V' \setminus V(M^i)$, then the degree of $v$ 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 $deg^i(t) = deg_{G[V' \setminus V(M^i)]}(v)$ as the degree of $v$ at time $t$. We want to show that $\mathbb{P}[deg^i(t) > n^{2ε}] \leq 1/n^{10}$ for any $v \in V'$.

Observe that if $deg^i(t) > n^{2ε}$, then $deg^i(t) > n^{2ε}$ for all $t \leq p$. Now, given that $deg^i(t-1) > n^{2ε}$, the probability that $v$ remained at unmatched after time $t$ is
\[
1 - \frac{deg^{i-1}(v)}{|V'|} \leq 1 - \frac{n^{2ε}}{n^2}.
\]
In particular, the probability that $deg^i(t) > n^{2ε}$ is at most $1 - n^{2ε}/n^2$. This implies that the probability that $deg^i(t) >
$n^{2ε}$ is at most 

$$(1 - \frac{n^{2ε}}{n^2})^p \leq \frac{1}{n^{10}}$$

because $p = 100n^{2-2ε}\log n$. 

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

**Corollary IV.8.** If Algorithm 1 returns a matching $M'$, then $M'$ satisfies the guarantees from Lemma IV.4 w.h.p.

**Proof.** If $M'$ is returned under Case 1. Then, by Lemma IV.5, we have $|M'|[A]| \geq \mu_1 \geq 2δ_{out}n$ w.h.p. Otherwise, in Case 2, we have $μ(\mathcal{G}[A \cap V' \setminus V(M')]) \geq |M'| \geq \mu_2 \geq 4δ_{out}n$ by Lemma IV.6 and also maximum degree of $\mathcal{G}[V' \setminus V(M')]$ is at most $n^{2ε}$ by Lemma IV.7 w.h.p.

2) **Termination without Error:** In this part, we show that if $μ(\mathcal{G}[A]) \geq δ_{in}n$, then Algorithm 1 does not return $\perp$ w.h.p. For any graph $G$ and $U_1, U_2 \subseteq V(G)$, we let $\mathcal{G}[U_1, U_2]$ contains all edges of $G$ whose one endpoint is in $U_1$ and another in $U_2$. Note that the induced subgraph $G[U_1] = \mathcal{G}[U_1]$.

Our high-level plan is that we will show that $D'$ decreases its size by $Θ(δ_{in}n^2)$ in each iteration in the repeat loop. So $D'$ must become very small after $T' = Θ(1/δ_{in}^3)$ 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 IV.9) which states that, even if $V'$ keeps shrinking, the maximum matching in $\mathcal{G}[V']$ remains large, $μ(\mathcal{G}[A']) \geq δ_{in}n/2$. We will need this fact throughout the whole argument.

The high-level argument goes as follows. If the algorithm does not return $M'$, then $M'[A']$ is very small for all $i$ and so $\mathcal{G}[A']$ 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 $\mathcal{G}[A']$. Thus, we remove only few vertices from $A'$ and so the size of $μ(\mathcal{G}[A'])$ cannot decrease too much. The formal argument below goes through the set $A'_{sp}$ and the above paragraph gives the right intuition.

**Lemma IV.9.** Suppose $μ(\mathcal{G}[A]) \geq δ_{in}n$. For $τ \in [0, T]$, at the end of the $τ$-th iteration of the repeat loop in Algorithm 1, we have $μ(\mathcal{G}[A']) \geq (1 - \frac{1}{τ^2})δ_{in}n \geq δ_{in}n/2$ w.h.p., if the algorithm does not terminate yet.

**Proof.** We prove by induction on $τ$. For $τ = 0$ (i.e. the beginning the algorithm), the claim holds as $μ(\mathcal{G}[A]) \geq δ_{in}n$. Next, we consider $τ \geq 1$. By induction hypothesis, at the beginning of the $τ$-th iteration, we have $μ(\mathcal{G}[A']) \geq (1 - \frac{1}{τ^2})δ_{in}n \geq δ_{in}n/2$.

At the end of the $τ$-th iteration, since Algorithm 1 did not terminate at Step 2f, by Lemma IV.5, we have, w.h.p., $|M'[A']| \leq 2δ_{out}n + δ_{out}n \leq 3δ_{out}n$ for all $i$. So we have $|E(\mathcal{G}[A'])| \leq 3δ_{out}nκk$ and then the average degree of vertices in $\mathcal{G}[A']$ is at most

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

because $|A'| \geq 2μ(\mathcal{G}[A']) \geq δ_{in}n$.

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

$$E[\text{vol}_{\mathcal{G}[A']}(A'_{sp})] \leq \frac{6δ_{out}kr_3}{δ_{in}} \leq \frac{δ_{in}nlog n}{200T}$$

because $δ_{out} = δ_{in}/10^6$ and $\frac{kr_3}{δ_{in}} = 1000nlog n$. Furthermore, this bound is concentrated. Indeed, since $\text{vol}_{\mathcal{G}[A']}(A'_{sp})$ is a sum of $r_3$ independent random variables whose range is $k$ (since the maximum degree in $\mathcal{G}$ and $\mathcal{G}[A']$ is $k$), by Hoeffding bound Proposition III.1, we have

$$\Pr[|\text{vol}_{\mathcal{G}[A']}(A'_{sp}) - E[\text{vol}_{\mathcal{G}[A']}(A'_{sp})]| > \frac{δ_{in}nlog n}{200T}] \leq 2\exp(-\frac{2r_3nlog n}{200T}) \leq 1/n^{10}. $$

So $\text{vol}_{\mathcal{G}[A']}(A'_{sp}) \leq \frac{δ_{in}nlog n}{1000T}$ w.h.p. Since every vertex $v \in C$ is adjacent to at least $η$ vertices in $A'_{sp}$ in $\mathcal{G}$, we have $|C \cap A'| ≤ δ_{in}n$.

Therefore,

$$|C \cap A'| ≤ \frac{δ_{in}nlog n}{1000T} ≤ \frac{δ_{in}n}{2T}$$

because $η = δ_{in}/10$. This means that we remove at most $\frac{δ_{in}n}{2T}$ 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 $\frac{δ_{in}n}{2T}$. Thus, $μ(\mathcal{G}[A']) \geq (1 - \frac{1}{τ^2})δ_{in}n$ which completes the induction.

Given Lemma IV.9, we will use the following lemma to argue that if Algorithm 1 does not return $M'$, then then $M'$ must match many vertices between $A'$ and $D'$.

**Lemma IV.10.** Suppose $μ(\mathcal{G}[A']) \geq δ_{in}n/2$. For any matching $M$ in $G[V']$, if $|M[A'][A']| > 3δ_{out}n$ and $μ(\mathcal{G}[A' \setminus V(M)]) < 16δ_{out}n$, then $M[A', D'] \geq δ_{in}n/3$.

**Proof.** Let $M^*$ be the maximum matching in $G[A']$ of size at least $δ_{in}n/2$. We partition edges in $M^*$ into two parts: $M^*_1$ and $M^*_2$. For each $(u, v) \in M^*$, we add $(u, v)$ into $M^*_1$ if both $u, v \not\in V(M)$. Otherwise, either $u$ or $v$ are matched by $M$ and we add $(u, v)$ into $M^*_2$. Note that $M^*_2$ is a matching in $M[A' \setminus V(M)]$. So $|M^*_2| < 16δ_{out}n$ and $|M^*_1| ≥ δ_{in}n/2 - 16δ_{out}n$.

Observe that $|V(M^*_1)| ≤ |M[A', D']| + |M[A']|$ 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
because $D' < \delta_{in}/3$, we have $|M[A', D']| \geq \delta_{in}/2 - 16\delta_{out} - 2 \cdot 3\delta_{out} \geq \delta_{in}/3$.

Lemma IV.10 also says that once $D'$ become small enough, Algorithm 1 will not err w.h.p.

**Corollary IV.11.** If $\mu(G[A']) \geq \delta_{in}/2$ and $|D'| < \delta_{in}/3$, then Algorithm 1 will return a matching $M'$ w.h.p.

**Proof.** For any $i$, note that $|M'[A', D']| < |D'| < \delta_{in}/3$. Therefore, the contrapositive of Lemma IV.10, we have that either $|M'[A']| \geq 3\delta_{out} n$ or $\mu(G[\bar{A}' \setminus V(M')]) \geq 10\delta_{out} n$. If $|M'[A']| \geq 3\delta_{out} n$, then $|\tilde{M}_i| \geq 2\delta_{out} n$ w.h.p. by Lemma IV.5. If $\mu(G[\bar{A}' \setminus V(M')]) \geq 10\delta_{out} n$, then $|\tilde{M}_i| \geq \frac{16\delta_{out} n - \delta_{out} n}{2} \geq 7\delta_{out} n$ because $\tilde{M}_i$ is $(2, \delta_{out}/n)$-approximate matching. So $\mu_2 \geq 2\delta_{out}$ w.h.p. by Lemma IV.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 IV.11.

**Lemma IV.12.** Suppose $\mu(G[A']) \geq \delta_{in}/2$. If Algorithm 1 does not terminate until $C$ is computed, then $|D' \cap C| \geq \delta_{in}/106 n$.

There are two main claims in the proof of Lemma IV.12. We suggest the reader to skip the proofs of these claims and see how they are used to prove Lemma IV.12 first.

**Claim IV.13.** $|E(G[A_{sp}', D'])| \geq 100\delta_{in}^2 n \log n$ w.h.p.

**Proof.** At the end of the $\tau$-th iteration, since Algorithm 1 did not terminate at Step 2f nor Step 2k, we have, w.h.p., that $M[A'] < 3\delta_{out} n$ by Lemma IV.5 and $|\tilde{M}'| < 5\delta_{out} n$ by Lemma IV.6. Since $\tilde{M}'$ is a $(2, \delta_{out})$-approximate matching in $G[\bar{A}' \setminus V(M)]$, we have $\mu(G[\bar{A}' \setminus V(M)]) < 10\delta_{out} n$. By Lemma IV.10, we have $|M'[A', D']| \geq \delta_{in}/3$.

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

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

and we have that

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

because $r_3 = 1000\delta_{in}^2 n \log n$. Furthermore, this is concentrated. Indeed, since $|E(G[A_{sp}', D'])|$ is a sum of $r_3$ independent random variable whose range is $k$ (since the maximum degree in $G$ and $G[A']$ is $k$), by Hoeffding bound Proposition III.1, we have

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

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

**Claim IV.14.** For each $v \in D'$, the number of neighbor of $v$ from $A_{sp}'$ in $G$ is at most $2000 \log n$ w.h.p. That is, $|N_G(v, A_{sp}')| \leq 2000 \log n$.

**Proof.** We have

$$\mathbb{E}[|N_G(v, A_{sp}')|] = \sum_{u \in N_{sp}(v, A')} \Pr[u \in A_{sp}'] \leq k \cdot \frac{r_3}{|A'|} \leq 1000 \log n$$

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

$$\Pr[|N_G(v, A_{sp}')| - \mathbb{E}[|N_G(v, A_{sp}')|] > 1000 \log n] \leq 2 \exp\left(-\frac{(1000 \log n)^2}{3 \cdot 1000 \log n}\right) \ll 1/n^{10}.$$ 

So $|N_G(v, A_{sp}')| \leq 2000 \log n$ w.h.p.

Now, let us prove Lemma IV.12 using the above claims.

**Proof of Lemma IV.12.** Observe that

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

where the inequality holds w.h.p. by Claim IV.14. Since $|D' \setminus C| \leq n$ and $\eta = \delta_{in}^2 (\log(n)/10$, we have by Claim IV.13 that

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

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

Finally, we give the conclusion of this part.

**Corollary IV.15.** If $\mu(G[A]) \geq \delta_{in}$, then Algorithm 1 does not return $\bot$ w.h.p.

**Proof.** First, $\mu(G[A]) \geq \delta_{in}$ implies that $\mu(G[A']) \geq \delta_{in}/2$ w.h.p. by Lemma IV.9. So, by Lemma IV.12 $D'$ decreases its size by $\delta_{in}/100$ in each iteration in the repeat loop. Hence, we have that $|D'| \leq \delta_{in}/3$ before $T = 100/\delta_{in}^2$ iterations. Therefore, there is an iteration $\tau \in [1, T]$ where Algorithm 1 will return a matching $M'$ w.h.p. by Corollary IV.11.

*Note that Hoeffding bound Proposition III.1 is not strong enough here.*
3) Preprocessing Time: Consider the \((2, \delta_{\text{out}} n)\)-approximate matching oracle \(\text{match}_{\text{low}}(\cdot)\) in Line 2g, which is given graph \(G[V' \setminus V(M')]\) and vertex set \(A' \setminus V(M')\) as input.

By Lemma IV.7, we can assume w.h.p. that \(G[V' \setminus V(M')]\) has degree at most \(n^{2\epsilon}\). Lemma IV.2 implies the following:

**Proposition IV.16.** Both preprocessing and query time of \(\text{match}_{\text{low}}(\cdot)\) is at most \(\tilde{O}(t_A n^{2\epsilon} + n + t_A/\delta_{\text{out}} n^{2\epsilon} / \delta_{\text{out}}^2) = \tilde{O}((t_A + n)n^{2\epsilon} / \delta_{\text{out}}^3)\) with high probability.

**Lemma IV.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 poly-logarithmic factor. Now, fix one iteration of the repeat-loop.

The total time to compute \(M'\), for all \(i \leq k\), is \(\Theta(kp) = \tilde{O}(n^{2\epsilon})\). For each for-loop iteration, to compute \(\mu_1\), we make \(T(1)\) queries to \(\text{mem}_A\) taking \(\Theta(1)\) time. To compute \(\mu_2\), we make \(r_2\) queries to \(\text{match}_{\text{low}}(\cdot)\). By Proposition IV.16, this takes time \(\tilde{O}(k t_A + n)n^{2\epsilon}/\delta_{\text{out}}^3) = \tilde{O}((t_A + n)n^{2\epsilon} / \delta_{\text{in}}^{25})\) by Lemma IV.2.

Next, we analyze the time to compute \(A_{\text{sp}}'\). Since \(|A'| \geq \delta_{\text{in}} n\) w.h.p. by Lemma IV.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 \(\text{mem}_A\), so we can compute \(A_{\text{sp}}'\) in time \(O(r_3) = \tilde{O}(t_A n^{1-\epsilon}/\text{poly}(\delta_{\text{in}}))\) because \(r_3 = 1000 \delta_{\text{in}} n \log n\) and \(k = n^{\epsilon}\). Once \(A_{\text{sp}}'\) is computed, we can compute \(C\) in \(|E(G)| = \Theta(kp) = \tilde{O}(n^{2\epsilon})\). 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 IV.4, is implied by combining Corollary IV.8, Corollary IV.15 and Lemma IV.17.

**C. Query Algorithm**

We define our matching oracle \(\text{match}(\cdot)\) depending on the cases from Lemma IV.4.

Suppose Lemma IV.4 returns \(M'\) that satisfies Case 1. Let \(M_1 = M'[A]\). By Lemma IV.4, \(|M_1| \geq 2\delta_{\text{out}} n\). The algorithm for outputting \(\text{match}(v)\) 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 \(\text{match}(v)\) with respect to \(M_1\).**

1) If \(v \in V(M')\), let \(v'\) be such that \((v,v') \in M'\). Else, return ⊥.

2) If \(\text{mem}_A(v) = \text{mem}_A(v') + 1\), return \((v, v')\). Else, return ⊥.

Next, suppose Lemma IV.4 returns \(M'\) that satisfies Case 2. Let \(M_2\) be a \((2, \delta_{\text{out}} n, t_A n^{2\epsilon})\)-approximate matching in \(G[A \cap V' \setminus V(M')]\). By Lemma IV.4, \(|M_2| \geq \mu(G[A \cap V' \setminus V(M')] / 2 - \delta_{\text{out}} n \geq \delta_{\text{out}} n\). The algorithm for outputting \(\text{match}(v)\) 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 \(O(t_A + n)n^{2\epsilon}/\delta_{\text{out}}^3)\) following the same proof as in Proposition IV.16 (the maximum degree of \(G[V' \setminus V(M')]\) is at most \(n^{2\epsilon}\) w.h.p. by Lemma IV.7).

**Algorithm 3 Compute \(\text{match}(v)\) with respect to \(M_2\).**

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

1) Check if \(v \in A \cap V' \setminus V(M')\). If not, return ⊥.

2) Using the oracle \(\text{match}_{\text{low}}(\cdot)\), if \(v \in V(M_2)\), return \((v, v') \in M_2\). Else, return ⊥.

In both cases, the matching oracle \(\text{match}(\cdot)\) respects a matching of size at least \(\delta_{\text{out}} n\) and has worst-case query time at most \(\tilde{O}(t_A + n)n^{2\epsilon}/\text{poly}(\delta_{\text{in}}))\) w.h.p.

V. BOOSTING THE APPROXIMATION GUARANTEE OF A MATCHING ORACLE

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

**Definition V.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 \(\gamma\) (see Lemma V.12), and \(\epsilon_{\text{in}} > 0\) is a sufficiently small constant such that \(9^\gamma \cdot \epsilon_{\text{in}} < 1/5\).

We present an algorithm

\[\text{Augment}(G, M^{\text{in}}, k, \gamma, \epsilon_{\text{in}}),\]

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

\(^9\)In fact, if we define \(M_2\) as \(\tilde{M}'\) 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.
Theorem V.2. Set \( \epsilon_{in} := 9^T \cdot \epsilon_{in} \) (see Definition V.1). Given adjacency-matrix query access to the input graph \( G = (V, E) \), the algorithm \( \text{Augment}(G, M^{in}, k, \gamma, \epsilon_{in}) \) runs in \( \tilde{O}_k, \gamma (n^{2+\epsilon_{in}}) \) time. Further, either it returns an oracle \( \text{match}_{M^{in}}(\cdot) \) with query time \( \tilde{O}_k, \gamma (n^{1+\epsilon_{in}}) \), 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 V-A, we present a template algorithm for the task stated in Theorem V.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 V-B, we show how to implement the template algorithm in the sublinear setting under adjacency-matrix queries, which leads to the proof of Theorem V.2.

Remark on Oracles: Throughout this section, we will treat the oracle \( \text{match}_M(\cdot) \) as a data structure in the sublinear model, which returns the appropriate answer upon receiving a query. In contrast, we will treat the oracle \( \text{mate}_M(\cdot) \) as simply an abstract function, so that \( \text{mate}_M(v) \) simply denotes the mate of \( v \) (if it exists) under \( M \) (see Section III). Note that we can return the value of \( \text{mate}_M(v) \) by making a single query to \( \text{match}_M(v) \), without any additional overhead in time.

A. A Template Algorithm

We denote the template algorithm simply by \( \text{Augment-Template}(G, M^{in}, k, \gamma) \), as we do not need the parameter \( \epsilon_{in} \) to describe it. The parameter \( \epsilon_{in} \) will become relevant only in Section V-B, 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 \( M^{in} \) are specified explicitly. The algorithm either returns an explicit 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. If \( M^{in} \) 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 V.2. Furthermore, the template algorithm has access to a subroutine \( \text{LargeMatching}(S, \delta) \), 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 \( |M| \geq \frac{1}{|S|} \cdot \delta^5 \cdot n \).

In addition, if \( \mu(G) \geq \delta \cdot n \), then it is guaranteed that \( \text{LargeMatching}(G, \delta) \) does not return \( \perp \). This mimics Theorem IV.1, with \( \delta_{in} = \delta \).

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. \( M^{in} \). 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 \( v_{2i}, v_{2i+1} \in E \setminus M^{in} \) for all \( i \in [0, k] \), and \( v_{2i+1}, v_{2i+2} \in M^{in} \) for all \( i \in [1, k] \). We say that the path \( p \) survives the random partitioning iff \( v_j \in L_i \) for all \( i \in [0, 2k+1] \).

Lemma V.3. Consider any collection \( \mathcal{P} \) of node-disjoint length \((2k + 1)\)-augmenting paths in \( G \) w.r.t. \( M^{in} \). 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: \( \mathbb{E}[|\mathcal{P}^*|] \geq (2k + 2)^{-(2k+2)} \cdot n = \Theta_k, \gamma (1) \cdot n \). 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 V.3, the template algorithm will only attempt to augment \( M^{in} \) 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 V.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 \( \text{mate}_{M^{in}}(v) = \perp \), or
2) \( \ell(v) = 2j - 1 \) for some \( j \in [1, k] \) and \( \ell(\text{mate}_{M^{in}}(v)) = 2j \), or
3) \( \ell(v) = 2j \) for some \( j \in [1, k] \) and \( \ell(\text{mate}_{M^{in}}(v)) = 2j - 1 \).

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

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 \( V_{2i} \). 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 V.4.** For all \( i \in [1, 2k] \), we have \( V_i \subseteq V(M^{i+1}) \).

**Observation V.5.** For all \( i \in [0, k] \), we have \( E_{2i} = E(G[V_{2i} \cup V_{2i+1}]) \). Furthermore, for all \( i \in [1, k] \), we have \( E_{2i-1} = M^{i+1} \cap (V_{2i-1} \times V_{2i}) \). Thus, if \( i \) is even, then \( G_i \) consists of all the edges from \( G \) that connect any two relevant nodes across the concerned layers. In contrast, if \( i \) is odd, then \( G_i \) consists of all the edges from \( M^{i+1} \) that connect any two relevant nodes across the concerned layers.

**Observation V.6.** Consider any augmenting path \( p = (v_0, v_1, \ldots, v_{2k+1}) \) w.r.t. \( M^{i+1} \) 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 \( \text{mate}_{M_{2i}}(v) \in V(M_{2i-2}) \).

**Observation V.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. \( M^{i+1} \) 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 \( v_i \leftarrow \text{mate}_{M_{i}}(v_{i+1}) \).
  - If \( i \) is odd, then \( v_i \leftarrow \text{mate}_{M_{i-1}}(v_{i+1}) \).
  - \( i \leftarrow i + 1 \).

Since the sequence \( M_0, M_2, \ldots, M_{2k} \) is nested, applying Observation V.4 and Observation V.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. \( M^{i+1} \). 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 \( \gamma \), i.e., \( \psi = \Theta_{k, \gamma}(1) \), and is chosen to be sufficiently small (see Corollary V.16). Next, for each \( i \in [0, k] \), we define:

\[
\psi_i := \frac{1}{10^8} \cdot \gamma^{5^i + 3} \quad \text{and} \quad \delta_i := \psi^{5^i + 1}.
\]

Consider any matching \( M' \) between the nodes at layers \( 2i \) and \( 2i + 1 \), where \( i \in [0, k] \). Intuitively, the parameters \( \psi_i \) and \( \delta_i \) determine how large \( M' \) needs to be so as to make us “happy”. Note that the values of \( \delta_i \) and \( \psi_i \) decrease in a doubly exponential manner with \( i \). This fact will be crucially used during the analysis in Section V-A2.

A relatively informal summary of the algorithm: Motivated by Observation V.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 \( |M_{2i}| \geq \psi_j \cdot n \) for all \( j \in [0, i] \). If \( i = k \), then by Observation V.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. \( M^{i+1} \), and so we just apply those augmenting paths to \( M_{2k} \) and return the resulting matching \( M' \). 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 \( M_0 \cup M_2 \cup \ldots \cup M_{2i} \) 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 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 LargeMatching\((S, \delta_{i+1})\) returns a matching \( M' \). Thus, we are guaranteed that \( |M'| \geq \frac{1}{10^8} \cdot \delta_{i+1} \cdot n \geq \psi_{i+1} \cdot n \). We set \( M_{2i+2} := M' \), \( i := i + 1 \), and proceed to the next iteration.

Case (b): The call to LargeMatching\((S, \delta_{i+1})\) returns \( \bot \), 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 LargeMatching\((S, \delta_{i+1})\) returns \( \bot \), 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 $M^{(i)}$ (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 V-B. 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 $\Lambda(t) := \{\lambda_0(t), \lambda_1(t), \ldots, \lambda_{\text{stack}(t)}(t)\}$, where $\text{stack}(t) := |\Lambda(t)| - 1$ and $\lambda_0(t) < \lambda_1(t) < \cdots < \lambda_{\text{stack}(t)}(t)$. The sequence of matchings $M(\lambda_0(t)), M(\lambda_1(t)), \ldots, M(\lambda_{\text{stack}(t)}(t))$ corresponds to the sequence $M_0, M_2, \ldots, M_{2i}$ in the discussion immediately after Observation V.7. Thus, the algorithm satisfies the following invariants.

**Invariant V.8.** We have stack$(t) \leq k$, and $\sigma \left( \lambda_j(t) \right) = 2j$ for all $j \in [0, \text{stack}(t)]$.

**Invariant V.9.** The sequence of matchings $M(\lambda_0(t)), M(\lambda_1(t)), \ldots, M(\lambda_{\text{stack}(t)}(t))$ is nested.

**Invariant V.10.** $\left| M(\lambda_j(t)) \right| \geq \psi_j \cdot n \quad \text{for all } j \in [0, \text{stack}(t)]$.

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 V.11.** $A \supseteq V(M(\lambda_j(t))) \quad \text{for all } j \in [0, \text{stack}(t)]$.

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 $i = \text{stack}(t - 1)$. 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 $M(\lambda_0(t)), M(\lambda_1(t)), \ldots, M(\lambda_k(t))$ is nested. Based on this sequence, we identify a collection of $\{M(\lambda_i(t))\}$ many node-disjoint augmenting paths w.r.t. $M^{in}$ in $G$, augment $M^{in}$ along those paths (see Observation V.7), and return the resulting matching $M^{out}$. Accordingly, from now on we assume that $i \leq k - 1$.

During iteration $t$, we will attempt to find a large matching $M^{t}$ 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 $M^{t}$ 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: $C_{2i+2}$ equals $A_{2i+2}$ if $i = -1$ and $v \in A_{2i+2}$: mate$_{M} (v) \in V(\bigcup_{\lambda \in \Lambda(t)} M(\lambda))$, if $i \geq 0$.

We now call the subroutine $\text{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$, 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 V.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 $\text{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$ returns a matching $M^{t}$. Thus, we are guaranteed that $|M^{t}| \geq \frac{1}{3^i} \cdot (\delta_{i+1})^5 \cdot n \geq \psi_{i+1} \cdot n$. We set $M(t) := M^{t}$, $\Lambda(t) := \Lambda(t-1) \cup \{t\}$ and stack$(t) := \text{stack}(t-1)+1$. This implies that $\lambda_j(t) := \lambda_j(t-1)$ for all $j \in [0, \text{stack}(t-1)]$, and $\lambda_{\text{stack}(t)}(t) := t$. 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 $\text{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$ returns $\perp$, and $i = -1$. Here, the algorithm terminates and returns $\text{FAILURE}$. Case (c): The call to $\text{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_{i+1})$ 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 $M^{in}$ (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 stack$(t) := \text{stack}(t-1) - 1$. We now return to the end of the $t^{th}$ iteration.
Thus, during 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 V.8, V.9, V.10 and V.11 continue to hold at the end of each iteration \( t \).

2) **Analysis:** In this section, we analyze the template algorithm, and prove the following lemma.

**Lemma V.12.** The algorithm \( \text{Augment-Template}(G, M^{in}, k, \gamma) \) runs for at most \( T = \Theta(k, \gamma(1)) \) iterations. It either returns a matching \( M^{out} \) in \( G \) of size \( |M^{out}| \geq |M^{in}| + \Theta(k, \gamma(1)) \cdot n \) (we say that the algorithm “succeeds” in this case), or it returns \( \text{FAILURE} \). Furthermore, if \( M^{in} \) admits a collection \( \gamma \cdot n \) 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 V.14).

**Claim V.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 V.10 implies that

\[
V \left( M^{(\lambda^{(t-1)})} \right) \cap V_{2i+1} \geq \psi_i \cdot n.
\]

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 V.14.** The algorithm \( \text{Augment-Template}(G, M^{in}, k, \gamma) \) 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 \( \text{FAILURE} \), 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:

\[
T = T_f + T_b + T_0 \leq (k + 1) \cdot (T_b + T_0) + (k + 1) + T_0 = \Theta(k) \cdot T_b \leq \Theta(k) \cdot \frac{1}{\psi_i} = \Theta(k, \gamma(1)).
\]

The second inequality follows from Claim V.13, and the last equality follows from (1).

We now move on to showing that if \( M^{in} \) 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 \( P \) denote a maximum-sized collection of node-disjoint length \( (2k + 1) \)-augmenting paths in \( G \) w.r.t. \( M^{in} \). Let \( P^* \subseteq P \) be the subset of paths in \( P \) that survive the random partitioning. If \( |P| \geq \gamma \cdot n \), then Lemma V.3 guarantees that whp:

\[
|P^*| \geq \Theta(k, \gamma(1)) \cdot n.
\]

At any point in time during the execution of the algorithm \( \text{Augment-Template}(G, M^{in}, k, \gamma) \), we say that a path \( p \in P^* \) is alive if all the nodes on \( p \) are alive, and we say that the path \( p \) is dead otherwise. Let \( P_A \subseteq P^* \) and \( P_D = P^* \setminus P_A \) respectively denote the set of alive and dead paths at any point in time. Just before the start of iteration \( 1 \), we have \( P_A = P^* = P_D = \emptyset \).

Subsequently, a path \( p \in P^* \) 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 V.15.** During a backtracking iteration for layer \( 2i \), where \( i \in [0, k - 1] \), at most \( \delta_i \cdot n \) many paths in \( P^* \) moves from \( P_A \) to \( P_D \).

**Proof.** Let \( t \geq 1 \) denote a backtracking iteration for layer \( 2i \). During iteration \( t \), the algorithm calls \( \text{LargeMatching}(C_{2i+2} \cup A_{2i+3}, \delta_i+1) \), 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 \( \text{LargeMatching}(\ldots) \) would not have returned \( \perp \).

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

**Corollary V.16.** Let \( \psi = \Theta(k, \gamma(1)) \) be a sufficiently small constant depending on \( k \) and \( \gamma \), and suppose
that (2) holds. Then throughout the entire duration of the algorithm, we have:

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

**Proof.** From (1), Claim V.13 and Claim V.15, we infer that:

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

(3)

Now, since we can set $$\psi$$ to be any sufficiently small constant value depending on $$k$$ and $$\gamma$$, 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 V.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:

$$\mu(G') < \delta_0 \cdot n.$$  

(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 $$\mathcal{P}_t^*$$. Thus, from Corollary V.16, we get:

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

(5)

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

**Lemma V.18.** Suppose that we can access the input graph $$G$$ only via adjacency-matrix queries, and we have an oracle match$$_{M^\text{in}}()$$ with $$O_{k,\gamma}(n^{1+\epsilon})$$ query time. Then we can implement each iteration $$t \geq 0$$ of the algorithm Augment-Template$$((G, M^\text{in}, k, \gamma))$$, as described in Section V-A, in $$O_{k,\gamma}(n^{2-\epsilon_t})$$ 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 match$$_{M^\text{out}}()$$ for the matching $$M^t$$, that has a query time of $$O_{k,\gamma}(n^{1+\epsilon})$$.
- An oracle alive$$_{\theta_{k,\gamma}}()$$ that has a query time of $$O_{k,\gamma}(n^{1+\epsilon})$$. 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 stack$$(t)$$, and the contents of the set $$\Lambda(t)$$.

**Corollary V.19.** Let $$\epsilon_{\text{out}} := 9T \cdot \epsilon_{\text{in}}$$, where $$T = \Theta_{k,\gamma}(1)$$ is the maximum possible number of iterations of the template algorithm (see Lemma V.12). Then it takes $$O_{k,\gamma}(n^{2-\epsilon_{\text{out}}})$$ time to implement the template algorithm, under adjacency-matrix query access to $$G$$. Further, if the template algorithm does not return **FAILURE** at the end of our implementation we have an oracle match$$_{M^\text{out}}()$$ for the matching $$M^\text{out}$$ returned by it, with query time $$O_{k,\gamma}(n^{1+\epsilon_{\text{out}}})$$.

**Proof.** By Lemma V.18, each iteration $$t$$ of the template algorithm can be implemented in time $$O_{k,\gamma}(n^{2-\epsilon_{t-1}}) = O_{k,\gamma}(n^{2-\epsilon_{t-1}})$$, since $$\epsilon_{\text{in}} \leq \epsilon_{t-1}$$, thus, the total time taken to implement the template algorithm is at most $$O_{k,\gamma}(T \cdot n^{2-\epsilon_{\text{out}}}) = O_{k,\gamma}(n^{2-\epsilon_{\text{out}}})$$.

Suppose that the template algorithm terminates at the end of iteration $$t$$, and returns a matching $$M^\text{out}$$. Then, at the end of iteration $$t$$ of our sublinear implementation, the situation is as follows.
\[ \sigma(t) = k, \text{ and } \Lambda(t) = \{ \lambda(t)^0, \lambda(t)^1, \ldots, \lambda(t)^{k} \} \text{, where} \]
\[ \sigma \left( \lambda(t)^j \right) = 2j \text{ for each } j \in [0, k] \text{ (see Invariant V.8).} \]

The sequence of matchings in \( \Lambda(t) \) is nested (see Invariant V.9). Thus, from this sequence of nested matchings we can extract a set of at least \( \left| M(\lambda(t)^k) \right| \) many node-disjoint length \((2k + 1)\)-augmenting paths w.r.t. \( M^{in} \) in \( G \) (see Observation V.7). The template algorithm obtains the matching \( M^{out} \) by applying these augmenting paths to \( M^{in} \). In our sublinear implementation of the template algorithm, however, we can access each matching \( m \in \Lambda(t) \) only via an oracle \( \text{match}_m(.) \), which has a query time of at most \( \hat{O}_{k, \gamma}(n^{1+e_t}) \) (see (6) and Lemma V.18). Furthermore, we can access the matching \( M^{in} \) only via the oracle \( \text{match}_{M^{in}}(.) \), which also has a query time of at most \( \hat{O}_{k, \gamma}(n^{1+e_t}) = \hat{O}_{k, \gamma}(n^{1+e_{out}}) \), where the last equality holds since \( e_t \leq e_T = e_{out} \). This concludes the proof.

**Proof of Lemma V.18:** We prove the lemma by induction on \( t \).

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

We already have the oracle \( \text{match}_{M^{in}}(.) \) with query time \( \hat{O}_{k, \gamma}(n^{1+e_0}) \), since \( e_0 = e_{in} \) and \( M^{(0)} = M^{in} \). We set \( \sigma(t) \leftarrow \bot, \text{stack}(t) \leftarrow -1 \) and \( \Lambda^{(0)} \leftarrow \emptyset \). We now claim that we already have the oracle \( \text{alive}_0(.) \). 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 \( \text{alive}_0(v) \), we can determine whether \( v \) is in \( V_H \) by checking the value of \( \ell(v) \), setting \( u \leftarrow \text{mate}_{M^{(0)}}(v) \), and then checking the value of \( \ell(u) \) if \( u \neq \perp \). Thus, answering a query to the oracle \( \text{alive}_0(.) \) takes \( \hat{O}_{k, \gamma}(n^{1+e_0}) \) time. So, we can implement iteration 0 in \( O(1) \) time, and Lemma V.18 holds for \( t = 0 \).

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

We assume that Lemma V.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 V.A1) under adjacency-matrix query access to \( G \). Let \( i = \text{stack}(t - 1) \). 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 \( \text{LargeMatching}(C_{2t+2} \cup A_{2t+3}, \delta_{t+1}) \). We first show that we can efficiently query whether or not a given node \( v \) belongs to the set \( C_{2t+2} \cup A_{2t+3} \). Subsequently, we split up our implementation of iteration \( t \) into two steps, as described below.

**Claim V.20.** Given any node \( v \in V \), we can determine if \( v \in C_{2t+2} \) in \( \hat{O}_{k, \gamma}(n^{1+e_{t-1}}) \) time.

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

1. \( \ell(v) \neq 2i + 2 \). Here, we return that \( v \notin C_{2t+2} \).
2. \( \ell(v) = 2i + 2 \) and \( \text{alive}_{t-1}(v) = \text{No} \). Here, we also return that \( v \notin C_{2t+2} \).
3. \( \ell(v) = 2i + 2, \text{alive}_{t-1}(v) = \text{Yes}, \text{ and } i = -1 \). Here, we return that \( v \in C_{2t+2} \).
4. \( \ell(v) = 2i + 2, \text{alive}_{t-1}(v) = \text{Yes} \) and \( i \geq 0 \). Here, we first set \( u_v \leftarrow \text{mate}_{M^{(t)}}(v) \), which would also allow us to answer the query \( \text{match}_{M^{out}}(v) \).
takes $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}}) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$. Next, we call $\text{match}_{M}(s^{(t-1)}(u_{v}))$, 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 $\text{match}_{M}(s^{(t-1)}(u_{v}) \neq \perp$.

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 V.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 $\text{alive}_{t-1}(v)$, which takes $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$ time. The corollary now follows from Claim V.20. $\square$

**Step I: Constructing the oracle $\text{match}_{M(t)}()$.** Armed with Corollary V.21, we mimic the call to LargeMatching($C_{2i+2} \cup A_{2i+3} \cup \delta_{i+1}$) in the template algorithm, by invoking Theorem IV.1 with $A = C_{2i+2} \cup A_{2i+3}, \delta_{i+1} = \delta_{i+1}, \epsilon = 2 \cdot \epsilon_{i-1}$ and $t_{A} = \tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$. If Theorem IV.1 returns $\perp$, then we set $M(t) = \emptyset$, and the trivial oracle $\text{match}_{M(t)}()$ has $O(1) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_{t}})$ query time. Otherwise, Theorem IV.1 returns an oracle $\text{match}_{M(t)}()$ for a matching $M$, and we set $M(t) = M$. By (6) and Theorem IV.1, this oracle $\text{match}_{M(t)}()$ has query time:

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

Finally, from (6) and Theorem IV.1, we infer that overall Step I takes time:

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

In the above derivation, the second equality holds since $\epsilon = 2\epsilon_{t-1} \leq 97\epsilon_{t-1} \leq 1/5$ (see (6) and Definition V.1), whereas the third equality holds since $t_{A} = \tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}})$.

**Step II: Determining $\sigma(t), \text{stack}(t), \Lambda(t)$, and the oracle $\text{alive}_{t}()$.** 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 IV.1 returned an oracle $\text{match}_{M(t)}()$ 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 $\Lambda(t) := \Lambda(t-1) \cup \{i\}$ and $\text{stack}(t) := \text{stack}(t-1) + 1$. Now, we observe that the set of alive nodes does not change during such a forwarding iteration, and so we already have the oracle $\text{alive}_{t}(v)$, because $\text{alive}_{t}(v) = \text{alive}_{t-1}(v)$ for all $v \in V$. Accordingly, the oracle $\text{alive}_{t}(v)$ 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 IV.1 returned $\perp$, and $i = -1$. Here, the algorithm terminates and returns $\text{FAILURE.}$ Case (c): In Step I, the invocation of Theorem IV.1 returned $\perp$, and $i \geq 0$. This will be referred to as a backtracking iteration at layer $2i$. In this case, we set $\Lambda(t) := \Lambda(t-1) \setminus \{i\}$ and $\text{stack}(t) := \text{stack}(t-1) - 1$. Now, we observe that due to iteration $t$, only the nodes in $C_{2i+2}$ and their matched neighbors under $M^{(i)}$ (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 $\text{alive}_{t}(v)$, in $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t}})$ time, as follows.

We first check the value of $\ell(v)$, query $\text{alive}_{t-1}(v)$ and $\text{match}_{M^{(i)}(v)}$, and determine whether or not $v \in C_{2i+2}$ by invoking Claim V.20. Overall, this takes $\tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}}) + \tilde{O}_{k,\gamma}(n^{1+\epsilon_{t-1}}) = \tilde{O}_{k,\gamma}(n^{1+\epsilon_{t}})$ time. Next, we consider three cases.

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

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

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

To summarize, Step I takes $\tilde{O}_{k,\gamma}(n^{2-\epsilon_{t-1}})$ time, whereas Step II takes only $O(k) = \tilde{O}_{k,\gamma}(1)$ time. Furthermore, at the end of Step II we have all the desired data structures for iteration $t$, and both the oracles $\text{match}_{M^{(i)}(v)}$ and $\text{alive}_{t}(v)$ have a query time of at most $\tilde{O}(n^{1+\epsilon_{t}})$. Finally, Theorem IV.1 ensures that w.p., 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 V-A1, and how the subroutine LargeMatching($S, \delta$) is defined in the second paragraph of Section V-A). This concludes the proof of Lemma V.18.
VI. (1, εn)-APPROXIMATE MATCHING ORACLE

In this section, starting from an empty matching, we repeatedly apply Theorem V.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 I.3.

Theorem VI.1. Let γ, ε′ ∈ (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 ε″ ∈ (0, ε′), there exists an algorithm which: In $O_\gamma(n^2 - \epsilon')$ time, returns a matching $\mathcal{M}()$ for a $(1, 3\gamma n)$-approximate matching $M$ in $G$, where the oracle $\mathcal{M}(\cdot)$ has $\tilde{O}_\gamma(n^{1+\epsilon''})$ query time.

Corollary VI.2. Given adjacency-matrix query access to an $n$-node graph $G$ and any constant $\gamma \in (0, 1)$, in $O_\gamma(n^{2 - \epsilon'})$ 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 $\gamma$.

Proof. First, we apply Theorem VI.1, with $\epsilon = \epsilon'$, to get the oracle $\mathcal{M}(\cdot)$ in $O_\gamma(n^{2 - \epsilon'})$ time. Next, we compute $\tilde{O}_\gamma(n^{1+\epsilon''})$ estimate $\hat{\mu}$ of $|M|$ by sampling, uniformly at random, a set $S$ of $O_\gamma(1)$ nodes from $V$ and querying $\mathcal{M}(v)$ for each node $v \in S$. This takes $O_\gamma(n^{1+\epsilon''})$ time. The last inequality holds since $\epsilon = \epsilon'$ and $\epsilon''$ 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 VI.1

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

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

\begin{algorithm}
\caption{Near-optimal-matching-oracle $(G = (V, E), \gamma)$}
Choose $\epsilon' \in (0, 1)$ to be a sufficiently small constant. \hspace{1cm} $\gamma_{in} \leftarrow \epsilon'$, $k \leftarrow \lceil 1/\gamma \rceil$, $M^\text{in} \rightarrow \emptyset$. \hspace{1cm} $\tau \leftarrow$ TRUE.
\textbf{While} $\tau =$ TRUE: \hspace{1cm} // Start of a new round
\hspace{1cm} 1) $\tau \leftarrow$ FALSE.
\hspace{1cm} 2) For $i = 0$ to $k$: \hspace{1cm} \text{See Theorem V.2}
\hspace{1cm} a) $Z \leftarrow \text{Augment}(G, M^\text{in}, i, \gamma^2, \epsilon_{in})$. \hspace{1cm} \text{See Theorem V.2}
b) If $Z \neq$ FAILURE, then
\hspace{1.5cm} • Suppose that $Z = (M^\text{out}, \ell^\text{out})$.
\hspace{1.5cm} • $M^\text{in} \leftarrow M^\text{out}$, $\epsilon_{in} \leftarrow \ell^\text{out}$.
\hspace{1.5cm} • $\tau \leftarrow$ TRUE.
\hspace{1cm} $M \leftarrow M^\text{in}$, $\epsilon'' \leftarrow \ell^\text{out}$.
\textbf{Return} the oracle $\mathcal{M}(\cdot)$, which has query time $\tilde{O}_\gamma(n^{1+\epsilon''})$.
\end{algorithm}

Claim VI.3. Algorithm 4 runs for $\tilde{O}_\gamma(1)$ rounds, and makes $\tilde{O}_\gamma(1)$ calls to $\text{Augment}(\cdot)$.

Proof. Say that a given round of Algorithm 4 is successful if during that round: for some $i \in [0, k]$, the call to $\text{Augment}(G, M^\text{in}, i, \gamma^2, \epsilon_{in})$ succeeded (see Theorem V.2 and step 2-(a) of Algorithm 4). By Theorem V.2, each time a call to $\text{Augment}(G, M^\text{in}, i, \gamma^2, \epsilon_{in})$ succeeds, it increases the size of the matching $M^\text{in}$ (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 $\text{Augment}(\cdot)$, and all but the last round is successful. This implies the claim. □

Claim VI.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. $M^\text{in}$ 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 $\text{Augment}(G, M^\text{in}, j, \gamma^2, \epsilon_{in})$ succeeds during the given round, then it immediately implies the claim (since we would have $\tau = \text{TRUE}$ 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 $\text{Augment}(G, M^\text{in}, j, \gamma^2, \epsilon_{in})$ returns FAILURE, and hence the matching $M^\text{in}$ 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 $M^\text{in}$ still admits a collection of at least $\gamma^2 \cdot n$ many node-disjoint length $(2i + 1)$-
augmenting paths in $G$. Thus, by Theorem V.2, the call to $\text{Augment}(G, M^{in}, i, \gamma^2, \epsilon_{in})$ succeeds whp. So, it follows that Algorithm 4 does not end after the given round, whp.

\begin{proof}
Let $M^*$ be a maximum matching in $G$. By Claims VI.3 and VI.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 $M^{in} \cup M^*$.

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

Since Algorithm 4 makes only constantly many calls to $\text{Augment}()$, we can choose $\epsilon^* > 0$ to be sufficiently small so as to guarantee that $0 < \epsilon^* \ll 1$ (see Claim VI.3 and Theorem V.2). Further, during the execution of Algorithm 4, each call to $\text{Augment}()$ takes $O_{\epsilon}(n^{2-\epsilon}) = \tilde{O}(n^{2-\epsilon})$ time. Theorem VI.1 now follows from Claim VI.3 and Corollary VI.5.

\section{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 VII.1 is a restatement of Theorem I.2.

\begin{theorem}
There is a dynamic $(1+\epsilon)$-approximate matching size algorithm with $m^{0.5-\Omega(\epsilon)}$ 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 match$_M()$ with query time $\tilde{O}(m^{0.5+\epsilon})$ (for a small constant $\epsilon > 0$ which depends on $\epsilon$), where $M$ is a $(1+\epsilon)$-approximate maximum matching of $G$.

To highlight the main idea behind the proof of Theorem VII.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 $\epsilon n$ updates. At the start of a phase, we compute a $(1+\epsilon)$-approximate estimate $\mu^*$ of $\mu(G)$, by invoking Corollary VI.2, in $O_{\epsilon}(n^{2-\epsilon})$ time. Since $\mu(G) = \Omega(n)$, the value of $\mu^*$ 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: $O_{\epsilon}(n^{2-\epsilon}) / (\epsilon n) = \tilde{O}(n^{1-\epsilon})$, 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.

\begin{proof}[Proof of Theorem VII.1]
For ease of exposition, we first focus on proving an amortized update time bound. We start by recalling a useful technique for sparsiﬁng $G$, which allows us to assume that $\mu(G) = \Omega(n)$.

\textbf{Constructions:} Consider a function $\phi : V \rightarrow V_\phi$ which maps every node in $V$ to some element in the set $V_\phi$. We say that $\phi$ is a contraction of $G$ if $|V_\phi| \leq |V|$. 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 $G_\phi$, 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 VII.2 in Appendix A.

\begin{lemma}
There exists a dynamic algorithm $A$ with $O(1)$ worst-case update time, which maintains: a set of $K = O(1)$ 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 $(1-\epsilon) \cdot \mu(G) \leq \mu(G_{\phi_{i^*}}) \leq \mu(G)$.

\end{lemma}

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

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.

\textbf{Type-I Phase:} At the start of a type-I phase, we have $\tilde{\mu} \geq E[|E|]^{0.5+\epsilon_0}$. Let $m_{\text{init}}$ denote the value of $|E|$ at the start of the phase. Then the phase will last for the next $\epsilon \cdot (m_{\text{init}})^{0.5+\epsilon_0}$ 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 $O_{\epsilon}(m_{\text{init}})$ time [DP14]. Define $\mu^* = |M|$. Throughout
the phase, $\mu^*$ continues to remain a $(1+2\epsilon)$-approximate estimate of $\mu(G)$, and we continue to output the same value $\mu^*$. This leads to an amortized update time of:

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

The last equality holds since $|E| = m = \Theta(m_{\text{init}})$ 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 $\text{match}_M(\cdot)$, 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 $\mu_{\text{init}}$ and $m_{\text{init}}$ respectively denote the value of $\hat{\mu}$ and $|E|$ at the start of the phase. The phase will last for the next $\epsilon \mu_{\text{init}}$ 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 $\hat{\mu}^i$ of $\mu(G_{\phi_i})$. We obtain $\mu^*_i$ by invoking Corollary VI.2 on $G_{\phi_i}$, with $\gamma = 2\epsilon^{11}$. This takes time:

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

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

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

As $\gamma = 2\epsilon^2$, we infer that $\mu^*_j$ is a purely multiplicative $(1+\Theta(\epsilon))-\text{approximate estimate of } \mu(G)$. We continue to output the same value $\mu^*_j$ 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 $\epsilon \mu_{\text{init}}$ updates. Accordingly, this leads to an amortized update time of:

$$\tilde{O}_\epsilon \left( \left( \hat{\mu}_{\text{init}}^{2-\epsilon^*} \right) \right) = \frac{\tilde{O}_\epsilon \left( \hat{\mu}_{\text{init}}^{1-\epsilon^*} \right)}{\epsilon \mu_{\text{init}}}.$$ 

The last equality holds because we can ensure that $\epsilon_0$ is sufficiently small compared to $\epsilon^*$ (which, in turn, depends on $\epsilon$), and since $|E| = m = \Theta(m_{\text{init}})$ throughout the duration of the phase.

Because of (7), at the start of the phase we can construct the oracle $\text{match}_M(\cdot)$ by invoking Theorem VI.1 on the graph $G_{\phi_j}$. Over the $\epsilon \mu_{\text{init}}$ 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 $\mu_{\text{init}}$ 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 $O(\epsilon \cdot \mu_{\text{init}})$ updates. Furthermore, the total computational work done by the algorithm in both types of phases is upper bounded by $\tilde{O}(\hat{\mu}_{\text{init}}^{2-\epsilon^*})$ 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 $A(G_i)$ stand for the output of the previously described algorithm initialized on $G_i$. Let $\hat{\mu}_{\text{init}}$ stand for the value of $\mu_{\text{init}}$ at the beginning of phase $i$. We will now describe the behaviour of the worst-case update time algorithm.

The improved algorithm similarly initializes its output to be $A(G_1)$. Throughout the first three phases it does not alter its output and during the first two phases it doesn’t complete any background computation. During phase $i$ for $i > 2$ the algorithm calculates $A(G_{i-2})$ 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 its output to be $A(G_{i-2})$.

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 $A(G_{i-2})$ takes time proportional to $\tilde{O}(\hat{\mu}_{\text{init}}^{2-\epsilon^*})$ and is distributed over $\epsilon \cdot \hat{\mu}_{\text{init},i}$ updates. As during a phase $\mu(G)$ may change by at most a $1+O(\epsilon)$ multiplicative factor we must have that $\hat{\mu}_{\text{init},i} = \Theta(\hat{\mu}_{\text{init},i-2})$. The amortized implementation amortizes the work of computing $A(G_{i-2})$ over $\epsilon \cdot \hat{\mu}_{\text{init},i-2}$ 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.

**VIII. Acknowledgements**

We would like to thanks David Wajc for introducing us to the possibility that McGregor’s algorithm [McG05] can be adapted in the dynamic setting.
REFERENCES


APPENDIX

In this section, we prove Lemma IV.3.

Preliminaries on Randomized Greedy Matching. A greedy maximal matching in G with respect to an edge
permutation $\pi$, denoted by $M = GMM(G, \pi)$ is a maximal matching obtained by scanning through edges with ordering defined by $\pi$, and for each edge $e$, include the edge $e$ into the matching if both of its end point are not matched. The matching oracle $\mathtt{match}_M(v)$ of Lemma IV.3 simply returns $\mathtt{VO}(v)$ where the vertex oracle $\mathtt{VO}$ and the edge oracle $\mathtt{EO}$ are defined in [Beh22] follows.

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

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 $\text{EO}(e_i, u_i, \pi) = \text{TRUE}$, then return $(v, u_i)$.
3) return $\bot$.

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

1) if $\text{EO}(e, u, \pi)$ 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 $\pi(e_1) < \pi(e) < \cdots < \pi(e_k)$.
3) for $i = 1, \ldots, k$: if $\text{EO}(e_i, w_i, \pi) = \text{TRUE}$, then return $\text{FALSE}$.
4) return $\text{TRUE}$.

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

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

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

Given access to adjacency list, we can execute $\text{VO}(v, \pi)$ 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 $\pi$ 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 $\ell \geq \alpha \cdot \frac{\overline{d}}{\epsilon}$. We have by Markov’s inequality that

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

For any edge permutation $\pi$, let $f(\pi) = \Pr_{v \sim \pi}[T(v, \pi) > \ell]$ measure the fraction of vertices such that randomized greedy matching w.r.t. $\pi$ makes many recursive calls exceeding the threshold $\ell$. 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, $\Pr_{v, \pi}[T(v, \pi) > \ell] \geq \Pr_{v, \pi}[T(v, \pi) > \ell | \pi \text{ is not great}] > (\epsilon/2) \cdot (1/4)$ 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** $\text{TestPerm}(\pi)$

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 $\tilde{f} = X/r$. \hspace{1cm} 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 $\pi$ is great, then $\text{TestPerm}(\pi)$ returns “yes” with high probability. If $\pi$ is bad, then $\text{TestPerm}(\pi)$ returns “no” with high probability.

**Proof.** If $\pi$ is great but $\text{TestPerm}(\pi)$ 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 $X \sim \mathbb{E}[X] > r\epsilon/4$.

Applying Chernoff bound Proposition III.2 with $t = r\epsilon/4$ and $\overline{\mu} = r\epsilon/2$ 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 $\pi$ is bad but $\text{TestPerm}(\pi)$ 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

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

\[\square\]
Now, we are ready to prove Lemma IV.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 \( \text{TestPerm}(\pi_i) \) returns "yes", then set \( \pi^* \leftarrow \pi_i \).

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

**Query.** Given a vertex \( v \), we simply execute \( \text{VO}(v, \pi^*) \) except that we return \( \perp \) if it makes more than \( \ell \) recursive calls. If \( \text{VO}(v, \pi^*) = (v', \pi^*) \) returns a matched edge, to make sure that the answers on \( v \) and \( v' \) are consistent, we also call \( \text{VO}(v', \pi^*) \). If it turns out that \( \text{VO}(v', \pi^*) \) makes more than \( \ell \) recursive calls, then we return \( \perp \). Otherwise, \( \text{VO}(v', \pi^*) \) must also return \( (v, \pi^*) \) and then 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 \( \pi^* \) is good, i.e., \( |\pi^*| = \Pr_{v \sim V}[T(v, \pi^*) > \ell] \leq \epsilon \).

So

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

This completes the correctness of Lemma IV.3. It remains to analyze the running time.

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

This algorithm description is analogous to the algorithm in [Kis22] which extends previous algorithms from [AKL19], [Beh23], [BDH20], [BKW23] 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 \( \alpha = \tilde{O}(1) \)-approximate estimate \( \hat{\mu} \) of \( \mu(G) \). We make \( \tilde{O}(1) \) guesses of \( \mu(G) \), i.e., \( \hat{\alpha}^1, \hat{\alpha}^2, \ldots, \hat{\alpha}^\beta \). For \( \mu(G) \) guess \( \hat{\alpha}^i \) we will define \( T = \lceil \ln(n)/512 \rceil = \tilde{O}(1) \) contractions of \( V \phi_j : j \in [T] \) and contracted graphs \( G_{\phi_j} \). If \( \hat{\mu} \in [\hat{\alpha}^i, \hat{\alpha}^{i+1}] \) we define \( \mu(G) \) guess \( \alpha^i \) to be the accurate guess at the given time if guess \( \alpha^i \) is currently the accurate guess then the algorithm will maintain that i) for all \( j \in [T] \) we have that \( |V_{\phi_j}| = \Theta(\frac{\mu(G)}{\epsilon^i}) \) and ii) there exists some \( j \in [T] \) such that \( (1 - \epsilon) \cdot \mu(G) \leq \mu(G_{\phi_j}) \leq \mu(G) \).

We will now define how \( \phi_{\phi} \) is generated. We define a set of vertices \( |V_{\phi_j}| = \frac{8 \cdot \alpha^{i+1}}{\epsilon^i} \) and map vertices of \( V \) to \( V_{\phi_j} \) 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 \( G_{\phi_j} \) we must have that \( \mu(G_{\phi_j}) \leq \mu(G) \). Furthermore, it remains to show that if \( \alpha^i \) is the currently accurate guess of \( \mu(G) \) then there exists some \( j \in [T] \) such that \( (1 - \epsilon) \cdot \mu(G) \leq \mu(G_{\phi_j}) \).

Let’s assume that \( \alpha^i \) 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 \( \left( \frac{n}{2\mu(G)} \right) \leq \frac{n^2 \mu(G)}{\exp(\ln(n) \cdot 2\mu(G))} \) such choices of \( S \). Fix some \( j \in [T] \). For all \( v \) vertices of \( V_{\phi_j} \) define the event \( X^v_j \) to be the indicator variable of the event \( \phi^{-1}(v) \cap S \neq \emptyset \) and define \( X_j = \sum_{v \in V_{\phi_j}} X^v_j \).

**Claim A.4.** For a fixed \( j \) events \( X^v_j \) are negatively associated random variables.

The proof of Claim A.4 appears in the appendix of [Kis22]. Let \( \beta = \alpha^{i+1}/\mu(G) \in [1, \alpha] \). We first lower bound the expectation of \( X_j \):

\[
\mathbb{E}[X^v_j] = 1 - \Pr[S \cap V^j_v = \emptyset] \tag{10}
\]

\[
= 1 - \left( 1 - \frac{1}{|V_{\phi_j}|} \right)^{2\mu(G)} \tag{11}
\]

\[
= 1 - \left( 1 - \frac{\epsilon}{8 \cdot \mu(G) \cdot \beta} \right)^{2\mu(G)} \tag{12}
\]

\[
\geq 1 - \exp \left( -\frac{\epsilon}{4 \cdot \beta} \right) \tag{13}
\]

\[
\geq \epsilon \cdot (1 - \epsilon/(8 \cdot \beta)) \tag{14}
\]

Inequality 14 holds for small values of \( \epsilon \). Therefore,

\[
\mathbb{E}[X_j] \geq |V_{\phi_j}| \cdot \frac{\epsilon \cdot (1 - \epsilon/(8 \cdot \beta))}{4 \cdot \beta} \geq 2\mu(G) \cdot (1 - \epsilon/(8 \cdot \beta))
\]

Now we apply Chernoff’s bound on the sum of negatively associated random variables \( X_j \) to get:
\begin{align*}
\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 (\frac{8}{5})^2}{2} \right) \\
&\leq \exp \left( -\frac{\mu(G) \cdot \epsilon^2}{64} \right) 
\end{align*}

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

\begin{align*}
\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 \left( -16 \ln(n) \cdot \mu(G) \right) 
\end{align*}

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 \( \phi_j \) 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 \( \phi_j(V[M^*]) \geq 2 \mu(G) \cdot (1 - \epsilon/4) \) (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 \( V_{\phi_j} \) by \( \phi_j \) 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 \( V_{\phi_j} \) by \( \phi_j \) amongst other endpoints of \( \mu(G) \) hence \( \mu(G_{\phi_j}) \geq \mu(G) \cdot (1 - \epsilon) \).

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 \( \phi_j \) are fixed at initialization. The task of the algorithm is to maintain maximum matching size estimate \( \mu(G) \) 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 \( \tilde{O}(1) \).

Consider the following folklore algorithm for explicitly maintaining a matching: recompute a \( (1 + \epsilon) \)-approximate matching \( M \) from scratch in \( \tilde{O}(m \epsilon^{-1} \log \epsilon^{-1}) \) time [DP14] every after \( cm/2n \) 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 \( O(m \epsilon^{-1} \log \epsilon^{-1}) = O(n \epsilon^{-2} \log \epsilon^{-1}) = O(n) \). 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

\[ |M| \geq \mu(G)/(1 + \epsilon - \epsilon m/2n) \geq \mu(G)/(1 + O(\epsilon)) \]

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 \( 2\Delta \) edges from the graph. Therefore, \( m \leq \mu(G) \cdot 2\Delta \leq \mu(G) \cdot 2n \).
<table>
<thead>
<tr>
<th>Guarantee</th>
<th>Approx</th>
<th>Time</th>
<th>Approx</th>
<th>Time</th>
<th>Approx</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[PR07], [MR09]</td>
<td>(2, εn)</td>
<td>$\Delta^O(\log(\Delta/\epsilon))$</td>
<td>(2, εn)</td>
<td>$2^O(\Delta/\epsilon^2)$</td>
<td>(1, εn)</td>
<td>$2^\Delta^O(\epsilon^2)$</td>
</tr>
<tr>
<td>[NO08]</td>
<td>(2, εn)</td>
<td>$2^{O(\Delta^2/\epsilon^2)}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[YYI09]</td>
<td>(2, εn)</td>
<td>$\Delta^4/\epsilon^2$</td>
<td>(1, εn)</td>
<td>$\Delta^O(1/\epsilon^2)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ORRR12], [CKK20]</td>
<td>(2, εn)</td>
<td>$(d + 1)\Delta/\epsilon^2$</td>
<td>(2, εn)</td>
<td>$n\sqrt{n}/\epsilon^2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[BRRS23]</td>
<td>$2 - \frac{\Omega(1)}{\epsilon^2}, o(n))$</td>
<td>$(d + 1)\Delta^+$</td>
<td>$2 - \frac{\Omega(1)}{\epsilon^2}$</td>
<td>$n + \Delta/\epsilon^2$</td>
<td>(2, εn)</td>
<td>$n/\epsilon^4$</td>
</tr>
<tr>
<td>[BKS23], [BRR23]</td>
<td>(1.5, εn)</td>
<td>$n\Delta^{1-\Omega(\epsilon^2)}$</td>
<td>1.5 + ε</td>
<td>$n\Delta^{1-\Omega(\epsilon^2)}$</td>
<td>(1.5, εn)</td>
<td>$n^{2-\Omega(\epsilon^2)}$</td>
</tr>
<tr>
<td>[BRR23]</td>
<td>(1.5 – $\Omega(1), o(n)$)</td>
<td>$n^{2-\Omega(1)}$</td>
<td>1.5 – $\Omega(1)$</td>
<td>$n^{2-\Omega(1)}$</td>
<td>(1.5 – $\Omega(1), o(n)$)</td>
<td>$n^{2-\Omega(1)}$</td>
</tr>
<tr>
<td>Our bipartite graph only</td>
<td>(1, εn)</td>
<td>$n^{2-\Omega(1)}$</td>
<td>(1, εn)</td>
<td>$n^{2-\Omega(1)}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE I**

Summary of sublinear-time algorithms for estimating the size of maximum matching. We omit polylog($n/\epsilon$) factors. $\Delta$ and $d$ denote the maximum and average degree of the graph, respectively.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Approximation</th>
<th>Query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[PR07], [MR09]</td>
<td>$2 + \epsilon$</td>
<td>$\Delta^O(\log(\Delta/\epsilon))$</td>
</tr>
<tr>
<td>[RTVX11], [ARVX12]</td>
<td>2</td>
<td>$2^O(\Delta)$</td>
</tr>
<tr>
<td>[RV16]</td>
<td>2</td>
<td>$2^O(\Delta)$</td>
</tr>
<tr>
<td>[LRY15]</td>
<td>$2 + \epsilon$</td>
<td>$\Delta^O(\log^2 \Delta)$</td>
</tr>
<tr>
<td>[Gha16]</td>
<td>2</td>
<td>$\Delta^O(\log^2 \Delta)$</td>
</tr>
<tr>
<td>[GU19]</td>
<td>2</td>
<td>$\Delta^O(\log \Delta)$</td>
</tr>
<tr>
<td>[Gha22]</td>
<td>2</td>
<td>$\Delta^O(\log \Delta)$</td>
</tr>
<tr>
<td>[KMNFT20]</td>
<td>$O(1)$ in expectation</td>
<td>$\Delta$</td>
</tr>
<tr>
<td>Our</td>
<td>(1, εn)</td>
<td>$n^{2-\Omega(1)}$</td>
</tr>
</tbody>
</table>

**TABLE II**

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