A Local Computation Approximation Scheme to Maximum Matching

Yishay Mansour* and Shai Vardi**

School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. mansour,shaivar1@post.tau.ac.il

Keywords: Local Computation Algortithms, Sublinear Algorithms, Approximation Algorithms, Maximum Matching

Abstract. We present a polylogarithmic local computation matching algorithm which guarantees a $(1-\epsilon)$-approximation to the maximum matching in graphs of bounded degree.

1 Introduction

Finding *matchings* - sets of vertex disjoint edges in a graph - has been an important topic of research for computer scientists for over 50 years. Of particular importance is finding *maximum* matchings - matchings of maximal cardinality. Algorithms that find a maximum matching have many applications in computer science; in fact, their usefulness extends far beyond the boundaries of computer science - to disciplines such as economics, biology and chemistry.

The first works on matching were based on unweighted bipartite graphs (representing problems such as matching men and women). Hall's marriage theorem [6] gives a necessary and sufficient condition for the existence of a perfect matching[1]. The efficient algorithms for the weighted bipartite matching problem date back to the Hungarian method [12,18]. In this work we focus on maximum matchings in general unweighed graphs. Berge [3] proved that a matching is a maximum matching if and only if the graph has no augmenting paths with respect to the matching. Edmonds used augmenting paths to find a maximum matching in his seminal work [5], in which he showed that a maximum matching can be found in polynomial time. Much work on matching been done since (e.g., [7,9,16,17]). Our work uses ideas from Hopcroft and Karp's algorithm for finding maximal matching in bipartite graphs [9], which runs in time $O(n^{2.5})$.

Local computation algorithms (LCAs) [20] consider the scenario in which we must respond to queries (regarding a feasible solution) quickly and efficiently, yet we never need the entire solution at once. The replies to the queries need to be *consistent*; namely, the responses to all possibly queries combine to a single feasible solution. For example, an LCA for matching in a graph G, receives an *edge-query* for an edge $e \in G$ and replies "yes" if and only if e is part of the matching. The replies to all the possible edge queries define a matching in the graph.

In this work we present a *local computation approximation scheme* to maximum matching. Specifically, we present an LCA such that for any $\epsilon > 0$, the edge-query replies comprise a matching that is a $(1-\epsilon)$-approximation to the maximum matching. Our LCA requires $O(\log^3 n)$ space, and with probability at least $1 - 1/n^2$, for any edge-query, it runs in time $O(\log^4 n)$. To the best of our knowledge, this is the first local computation approximation algorithm for a problem which provably does not have an LCA.

Related work. In the distributed setting, Itai and Israeli [10] showed a randomized algorithm which computes a maximal matching (which is a $1/2$-approximation to the maximum matching) and runs in $O(\log n)$ time with high probability. This result has been improved several times since (e.g., [4,8]); of particular relevance is the approximation scheme of Lotker et al. [13], which, for every $\epsilon > 0$, computes a $(1-\epsilon)$-approximation to the maximum matching in $O(\log n)$ time. Kuhn et al., [11] proved that any distributed algorithm, randomized or deterministic, requires (in expectation) $\Omega(\sqrt{\log n / \log\log n})$ time to compute a $\Theta(1)$-approximation to the maximum matching, even if the message size is unbounded.

* Supported in part by the Google Inter-university center for Electronic Markets and Auctions, by a grant from the Israel Science Foundation, by a grant from United States-Israel Binational Science Foundation (BSF), and the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

** Supported in part by the Google Europe Fellowship in Game Theory.

[1] A perfect matching includes all the nodes of a bipartite graph.

Rubinfeld et al., [20] showed how to transform distributed algorithms to LCAs, and gave LCAs for several problems, including maximal independent set and hypergraph 2-coloring. Unfortunately, their method bounds the running time of the transformed algorithm exponentially in the running time of the distributed algorithm. Therefore, distributed algorithms for approximate maximum matching cannot be (trivially) transformed to LCAs using their technique.

Query trees model the dependency of queries on the replies to other queries, and were introduced in the local setting by Nguyen and Onak [19]. If a random permutation of the vertices is generated, and a sequential algorithm is simulated on this order, the reply to a query on vertex v depends only on the replies to queries on the neighbors of v which come before it in the permutation. Alon et al., [2] showed that if the running time of an algorithm is $O(f(n))$, where f is polylogarithmic in n, a $1/n^2$ - almost $f(n)$-independent ordering on the vertices can be generated in time $O(f(n)\log^2 n)$, thus guaranteeing the polylogarithmic space bound of any such algorithm. Mansour et al., [14] showed that the size of the query tree can be bounded, with high probability, by $O(\log n)$, for graphs of bounded degree. They also showed that it is possible to transform many on-line algorithms to LCAs. One of their examples is an LCA for maximal matching, which immediately gives a $1/2$-approximation to the maximum matching. In a recent work, [15], LCAs were presented for mechanism design problems. One of their impossibility results shows that any LCA for maximum matching requires $\Omega(n)$ time.

2 Notation and Preliminaries

2.1 Graph Theory

For an undirected graph $G = (V, E)$, a *matching* is a subset of edges $M \subseteq E$ such that no two edges $e_1, e_2 \in M$ share a vertex. We denote by M^* a matching of maximum cardinality. An *augmenting path* with respect to a matching M is a simple path whose endpoints are *free* (i.e., not part of any edge in the matching M), and whose edges alternate between $E \setminus M$ and M. A set of augmenting paths P is *independent* if no two paths $p_1, p_2 \in P$ share a vertex.

For sets A and B, we denote $A \oplus B \stackrel{def}{=} (A \cup B) \setminus (A \cap B)$. An important observation regarding augmenting paths and matchings is the following.

Observation 1 *If M is a matching and P is an independent set of augmenting paths, then $M \oplus P$ is a matching of size $|M| + |P|$.*

A vertex $u \in V$ is a *neighbor* of vertex $v \in V$ if $(u, v) \in E$. Let $N(v)$ denote the set of neighbors of v, i.e., $N(v) = \{u : (v, u) \in E\}$. We assume that we have direct access both to $N(v)$ and to individual edges.

An independent set (IS) is a subset of vertices $W \subseteq V$ with the property that for any $u, v \in W$ we have $(u, v) \notin E$, namely, no two vertices $u, v \in W$ are neighbors in G. The IS is *maximal* (denoted by MIS) if no other vertices can be added to it without violating the independence property.

2.2 Local Computation Algorithms

We use the following model of local computation algorithms (LCAs)[20].[2] A $(t(n), s(n), \delta(n))$ - *local computation algorithm* $\mathcal{LA}$ for a computational problem is a (randomized) algorithm which receives an input of size n, and a query x. Algorithm $\mathcal{LA}$ uses at most $s(n)$ memory, and with probability at least $1 - \delta(n)$, it replies to any query x in time $t(n)$. The algorithm must be *consistent*, that is, the replies to all of the possible queries combine to a single feasible solution to the problem.

[2] Our model differs slightly from the model of [20] in that their model requires that the LCA *always* obeys the time and space bounds, and returns an error with some probability. It is easy to see that any algorithm which conforms to our model can be modified to conform to the model of [20] by forcing it to return an error if the time or space bound is violated.

2.3 Query Trees

Let $G = (V, E)$ be a graph of bounded degree d. A real number $r(v) \in [0, 1]$ is assigned independently and uniformly at random to every vertex v in the graph. We refer to this random number as the *rank* of v. Each vertex in the graph G holds an input $x(v) \in R$, where the range R is some finite set. A randomized Boolean function F is defined inductively on the vertices in the graph such that $F(v)$ is a function of the input $x(v)$ at v as well as the values of F at the neighbors w of v for which $r(w) < r(v)$.

We would like to upper bound the number of queries that are needed to be made vertices in the graph in order to compute $F(v_0)$ for any vertex $v_0 \in G$. We turn to the simpler task of bounding the size of a certain d-regular tree, which is an upper bound on the number of queries. Consider an infinite d-regular tree $\mathcal{T}$ rooted at v_0. Each node w in $\mathcal{T}$ is assigned independently and uniformly at random a distinct real number $r(w) \in [0, 1]$. For every node $w \in \mathcal{T}$ other than v_0, let $\texttt{parent}(w)$ denote the parent node of w. We grow a (possibly infinite) subtree T of $\mathcal{T}$ rooted at v as follows: a node w is in the subtree T if and only if $\texttt{parent}(w)$ is in T and $r(w) < r(\texttt{parent}(w))$. We keep growing T in this manner such that a node $w' \in T$ is a leaf node in T if the ranks of its d children are all larger than $r(w')$. We call the random tree T constructed in this way a *query tree* and we denote by $|T|$ the random variable that corresponds to the size of T. Note that $|T|$ is an upper bound on the number of queries.

If the reply to a query q depends (only) on the replies to a set of queries, Q, we call Q the set of *relevant* queries with respect to q.

2.4 Random Orders

Let $[n]$ denote the set $\{1, \ldots, n\}$.

A distribution $D : \{0, 1\}^n \to \mathbb{R}^{\geq 0}$ is *k-wise independent* if, when D is restricted to any index subset $S \subset [n]$ of size at most k, the induced distribution over S is the uniform distribution.

A random ordering $D_{\mathbf{r}}$ induces a probability distribution over permutations of $[n]$. It is said to *ϵ-almost k-wise independent* if for any subset $S \subset [n]$ of size at most k, the variation distance between the distribution induced by $D_{\mathbf{r}}$ on S and a uniform permutation over S is at most ϵ. We use the following Theorem from [2].

Theorem 2 ([2]). *Let $n \geq 2$ be an integer and let $2 \leq k \leq n$. Then there is a construction of $\frac{1}{n^2}$-almost k-wise independent random ordering over $[n]$ whose seed length is $O(k \log^2 n)$.*

We provide a short, intuitive explanation of the construction. We can construct n k-wise independent random variables $Z = (z_1, \ldots, z_n)$, using a seed of length $k \log n$ (see [1]). We generate $4 \log n$ independent copies of k-wise independent random variables, $Z_1, \ldots Z_{4 \log n}$. For $i \in [n]$, taking the i-th bit of each $Z_j, 1 \leq j \leq 4 \log n$ makes for a random variable $r(i) \in \{0, 1\}^{4 \log n}$, which can be expressed as an integer in $\{0, 1, \ldots, n^4 - 1\}$. The order is induced by r (u comes before v in the order if $r(u) < r(v)$). The probability that there exists $u, v \in [n]$ such that $r(u) = r(v)$ is at most $1/n^2$, hence the ordering is $1/n^2$-almost k-wise independent.

3 Approximate Maximum Matching

We present a local computation approximation scheme for maximum matching: We show an LCA that, for any $\epsilon > 0$, computes a maximal matching which is a $(1 - \epsilon)$-approximation to the maximum matching.

Our main result is the following theorem:

Theorem 3. *Let $G = (V, E)$ be a graph of bounded degree d. Then there exists an $(O(\log^4 n), O(\log^3 n), 1/n)$ - LCA that, for every $\epsilon > 0$, computes a maximal matching which is a $(1 - \epsilon)$-approximation to the maximum matching.*

Our algorithm is, in essence, an implementation of the abstract algorithm of Lotker et al., [13]. Their algorithm, relies on several interesting results due to Hopcroft and Karp [9]. First, we briefly recount some of these results, as they are essential for the understanding of our algorithm.

3.1 Distributed Maximal Matching

While the main result of Hopcroft and Karp [9] is an improved matching algorithm for bipartite graphs, they show the following useful lemmas for general graphs. The first lemma shows that if the current matching has augmenting paths of length at least ℓ, then using a maximal set of augmenting paths of length ℓ will result in a matching for which the shortest augmenting path is strictly longer than ℓ. This gives a natural progression for the algorithm.

Lemma 4. *[9] Let $G = (V, E)$ be an undirected graph, and let M be some matching in G. If the shortest augmenting path with respect to M has length ℓ and Φ is a maximal set of independent augmenting paths of length ℓ, the shortest augmenting path with respect to $M \oplus \Phi$ has length strictly greater than ℓ.*

The second lemma shows that if there are no short augmenting paths then the current matching is approximately optimal.

Lemma 5. *[9] Let $G = (V, E)$ be an undirected graph. Let M be some matching in G, and let M^* be a maximum matching in G. If the shortest augmenting path with respect to M has length $2k - 1 > 1$ then $|M| \geq (1 - 1/k)|M^*|$.*

Lotker et al., [13] gave the following abstract approximation scheme for maximal matching in the distributed setting.[3] Start with an empty matching. In stage $\ell = 1, 3, \ldots, 2k - 1$, add a maximal independent collection of augmenting paths of length ℓ. For $k = \lceil 1/\epsilon \rceil$, by Lemma 5, we have that the matching M_ℓ is a $(1 - \epsilon)$-approximation to the maximum matching.

In order to find such a collection of augmenting paths of length ℓ, we need to define a conflict graph:

Definition 6. *[13] Let $G = (V, E)$ be an undirected graph, let $M \subseteq E$ be a matching, and let $\ell > 0$ be an integer. The ℓ-conflict graph with respect to M in G, denoted $C_M(\ell)$, is defined as follows. The nodes of $C_M(\ell)$ are all augmenting paths of length ℓ, with respect to M, and two nodes in $C_M(\ell)$ are connected by an edge if and only if their corresponding augmenting paths intersect at a vertex of G.*[4]

We present the abstract distributed algorithm of [13], **AbstractDistributedMM**.

Algorithm 1 - **AbstractDistributedMM** - Abstract distributed algorithm with input $G = (V, E)$ and $\epsilon > 0$

1: $M_{-1} \leftarrow \emptyset$ ▷ M_{-1} is the empty matching
2: $k \leftarrow \lceil 1/\epsilon \rceil$
3: **for** $\ell \leftarrow 1, 3, \ldots, 2k - 1$, **do**
4: Construct the conflict graph $C_{M_{\ell-2}}(\ell)$
5: Let $\mathcal{I}$ be an MIS of $C_{M_{\ell-2}}(\ell)$
6: Let $\Phi(M_{\ell-2})$ be the union of augmenting paths corresponding to $\mathcal{I}$
7: $M_\ell \leftarrow M_{\ell-2} \oplus \Phi(M_{\ell-2})$ ▷ M_ℓ is matching at the end of phase ℓ
8: **end for**
9: Output M_ℓ ▷ M_ℓ is a $(1 - \frac{1}{k+1})$-approximate maximum matching

Note that for M_ℓ, the minimal augmenting path is of length at least $\ell+2$. This follows since $\Phi(M_{\ell-2})$ is a maximal independent set of augmenting paths of length ℓ. When we add $\Phi(M_{\ell-2})$ to $M_{\ell-2}$, to get M_ℓ, by Lemma 4 all the remaining augmenting paths are of length at least $\ell + 2$ (recall that augmenting paths have odd lengths).

Lines 4 - 7 do the task of computing M_ℓ as follows: the conflict graph $C_{M_{\ell-2}}(\ell)$ is constructed and an MIS, $\Phi(M_{\ell-2})$, is found in it. $\Phi(M_{\ell-2})$ is then used to augment $M_{\ell-2}$, to give M_ℓ.

We would like to simulate this algorithm locally. Our main challenge is to simulate Lines 4 - 7 without explicitly constructing the entire conflict graph $C_{M_{\ell-2}}(\ell)$. To do this, we will simulate an on-line MIS algorithm.

[3] This approach was first used by Hopcroft and Karp in [9]; however, they only applied it efficiently in the bipartite setting.

[4] Notice that the nodes of the conflict graph represent *paths* in G. Although it should be clear from the context, in order to minimize confusion, we refer to a vertex in G by *vertex*, and to a vertex in the conflict graph by *node*.

3.2 Local Simulation of the On-Line Greedy MIS Algorithm

In the on-line setting, the vertices arrive in some unknown order, and **GreedyMIS** operates as follows: Initialize the set $I = \emptyset$. When a vertex v arrives, **GreedyMIS** checks whether any of v's neighbors, $N(v)$, is in I. If none of them are, v is added to I. Otherwise, v is not in I. (The pseudocode for **GreedyMIS** can be found in the full version of the paper.)

In order to simulate **GreedyMIS** locally, we first need to fix the order (of arrival) of the vertices, π. If we know that each query depends on at most k previous queries, we do not need to explicitly generate the order π on all the vertices (as this would take at least linear time). By Theorem 2, we can produce a $\frac{1}{n^2}$-almost-k-wise independent random ordering on the edges, using a seed, s, of length $O(k \log^2 n)$.

Technically, this is done as follows. Let r be a function $r : (v, s) \to [cn^4]$, for some constant c.[5] The vertex order π is determined as follows: vertex v appears before vertex u in the order π if $r(v, s) < r(u, s)$. Let $G' = (V', E')$ be the subgraph of G induced by the vertices $V' \subseteq V$; we denote by $\pi(G', s)$ the partial order of π on V'. Note that we only need to store s in the memory: we can then compute, for any subset V', the induced order of their arrival.

When simulating **GreedyMIS** on the conflict graph $C_M(\ell) = (V_{C_M}, E_{C_M})$, we only need a subset of the nodes, $V' \subseteq V_{C_M}$. Therefore, there is no need to construct $C_M(\ell)$ entirely; only the relevant subgraph need be constructed. This is the main observation which allows us to bound the space and time required by our algorithm.

3.3 LCA for Maximal Matching

We present our algorithm for maximal matching - **LocalMM**, and analyze it. (The pseudocode for **LocalMM** can be found in the full version of the paper.) In contrast to the distributed algorithm, which runs iteratively, **LocalMM** is recursive in nature. In each iteration of **AbstractDistributedMM**, a maximal matching M_ℓ, is computed, where M_ℓ has no augmenting path of length less than ℓ. We call each such iteration a *phase*, and there are a total of k phases: $1, 3, \ldots 2k-1$. To find out whether an edge $e \in E$ is in M_ℓ, we recursively compute whether it is in $M_{\ell-2}$ and whether it is in $\Phi(M_{\ell-2})$, a maximal set of augmenting paths of length ℓ. We use the following simple observation to determine whether $e \in M_\ell$. The observation follows since $M_\ell \leftarrow M_{\ell-2} \oplus \Phi(M_{\ell-2})$.

Observation 7 *$e \in M_\ell$ if and only if it is in either in $M_{\ell-2}$ or in $\Phi(M_{\ell-2})$, but not in both.*

Recall that **LocalMM** receives an edge $e \in E$ as a query, and outputs "yes/no". To determine whether $e \in M_{2k-1}$, it therefore suffices to determine, for $\ell = 1, 3, \ldots, 2k-3$, whether $e \in M_\ell$ and whether $e \in \Phi(M_\ell)$.

We will outline our algorithm by tracking a single query. (The initialization parameters will be explained at the end.) When queried on an edge e, **LocalMM** calls the procedure IsInMatching with e and the number of phases k. For clarity, we sometimes omit some of the parameters from the descriptions of the procedures.

Procedure IsInMatching determines whether an edge e in in the matching M_ℓ. To determine whether $e \in M_\ell$, IsInMatching recursively checks whether $e \in M_{\ell-2}$, by calling IsInMatching$(\ell - 2)$, and whether e is in some path in the MIS $\Phi(M_{\ell-2})$ of $C_{M_{\ell-2}}(\ell)$. This is done by generating all paths p of length ℓ that include e, and calling IsPathInMIS(p) on each. IsPathInMIS(p) checks whether p is an augmenting path, and if so, whether it in the independent set of augmenting paths. By Observation 7, we can compute whether e is in M_ℓ given the output of the calls.

Procedure IsPathInMIS receives a path p and returns whether the path is in the MIS of augmenting paths of length ℓ. The procedure first computes all the relevant augmenting paths (relative to p) using RelevantPaths. Given the set of relevant paths (represented by nodes) and the intersection between them (represented by edges) we simulate **GreedyMIS** on this subgraph. The resulting independent set is a set of independent augmenting paths. We then just need to check if the path p is in that set.

[5] Alternately, we sometimes view r as a function $r : (v, s) \to [0, 1]$: Let r' be a function $r' : (v, s) \to [cn^4]$, and let $f : [cn^4] \to [0, 1]$ be a function that maps each $x \in [cn^4] - \{1\}$ uniformly at random to the interval $((x-1)/cn^4, x/cn^4]$, and f maps 1 uniformly at random to the interval $[0, 1/cn^4]$. Then set $r(v, s) = f(r'(v, s))$.

Procedure RELEVANTPATHS receives a path p and returns all the relevant augmenting paths relative to p. The procedure returns the subgraph of $C_{M_{\ell-2}}(\ell), C = (V_C, E_C)$, which includes p and all the relevant nodes. These are exactly the nodes needed for the simulation of **GreedyMIS**, given the order induced by seed s_ℓ. The set of augmenting paths V_C is constructed iteratively, by adding an augmenting path q if it intersects some path $q' \in V_C$ and arrives before it (i.e., $r(q, s_\ell) < r(q', s_\ell)$). In order to determine whether to add path q to V_C, we need first to test if q is indeed a valid augmenting path, which is done using ISANAUGMENTINGPATH.

Procedure ISANAUGMENTINGPATH tests if a given path p is an augmenting path. It is based on the following observation.

Observation 8 *For any graph $G = (V, E)$, let M be a matching in G, and let $p = e_1, e_2, \ldots, e_\ell$ be a path in G. Path p is an augmenting path with respect to M if and only if all odd numbered edges are not in M, all even numbered edges are in M, and both the vertices at the ends of p are free.*

Given a path p of length ℓ, to determine whether $p \in C_{M_{\ell-2}}(\ell)$, ISANAUGMENTINGPATH$(\ell)$ determines, for each edge in the path, whether it is in $M_{\ell-2}$, by calling ISINMATCHING$(\ell-2)$. It also checks whether the end vertices are free, by calling Procedure ISFREE(ℓ), which checks, for each vertex, if any of its adjacent edges are in $M_{\ell-2}$. From Observation 8, ISANAUGMENTINGPATH(ℓ) correctly determines whether p is an augmenting with respect to $M_{\ell-2}$.

We end by describing the initialization procedure INITIALIZE, which is run only once, during the first query. The procedure sets the number of phases to $\lceil 1/\epsilon \rceil$. It is important to set a different seed s_ℓ for each phase ℓ, since the conflict graphs are unrelated (and even the size of the description of each node, a path of length ℓ, is different). The lengths of the k seeds, $s_1, s_3, \ldots, s_{2k-1}$, determine our memory requirement.

3.4 Bounding the Complexity

In this section we prove Theorem 3. We start with the following observation:

Observation 9 *In any graph $G = (V, E)$ with bounded degree d, each edge $e \in E$ can be part of at most $\ell(d-1)^{\ell-1}$ paths of length ℓ. Furthermore, given e, it takes at most $O(\ell(d-1)^{\ell-1})$ time to find all such paths.*

Proof. Consider a path $p = (e_1, e_2, \ldots, e_\ell)$ of length ℓ. If p includes the edge e, then e can be in one of the ℓ positions. Given that $e_i = e$, there are at most $d-1$ possibilities for e_{i+1} and for e_{i-1}, which implies at most $(d-1)^{\ell-1}$ possibilities to complete the path to be of length ℓ. □

Observation 9 yields the following corollary.

Corollary 10. *The ℓ-conflict graph with respect to any matching M in $G = (V, E)$, $C_M(\ell)$, consists of at most $\ell(d-1)^{\ell-1}|E| = O(|V|)$ nodes, and has maximal degree at most $d(\ell+1)\ell(d-1)^{\ell-1}$.*

Proof. (For the degree bound.) Each path has length ℓ, and therefore has $\ell+1$ vertices. Each vertex has degree at most d, which implies $d(\ell+1)$ edges. Each edge is in at most $\ell(d-1)^{\ell-1}$ paths. □

Our main task will be to compute a bound on the number of recursive calls. First, let us summarize a recursive call. The only procedure whose runtime depends on the order induced by s_ℓ is RELEVANTPATHS, which depends on the number of vertices V_C (which is a random variable depending of the seed s_ℓ). To simplify the notation we define the random variable $X_\ell = d(\ell+1)\ell(d-1)^{\ell-1}|V_C|$. Technically, **GreedyMIS** also depends on V_C, but its running time is dominated by the running time of RELEVANTPATHS.

Calling procedure	Called Procedures
ISINMATCHING(ℓ)	$1\times$ ISINMATCHING$(\ell-2)$ and $\ell(d-1)^{\ell-1}\times$ ISPATHINMIS(ℓ)
ISPATHINMIS(ℓ)	$1\times$ RELEVANTPATHS(ℓ) and $1\times$ **GreedyMIS**
RELEVANTPATHS(ℓ)	$X_\ell\times$ ISANAUGMENTINGPATH(ℓ)
ISANAUGMENTINGPATH(ℓ)	$\ell\times$ ISINMATCHING$(\ell-2)$ and $2\times$ ISFREE(ℓ)
ISFREE(ℓ)	$(d-1)\times$ ISINMATCHING$(\ell-2)$

From the table, it is easy to deduce the following proposition.

Proposition 11. IsAnAugmentingPath(ℓ) *generates at most* $\ell + 2(d-1)$ *calls to* IsInMatching($\ell - 2$)*, and therefore at most* $(\ell + 2d - 2) \cdot \ell(d-1)^{\ell-1}$ *calls to* IsPathInMIS($\ell - 2$)*.*

We would like to bound X_ℓ, the number of calls to IsAnAugmentingPath(ℓ) during a single execution of IsPathInMIS(G, p, ℓ, S). We require the following theorem, the proof of which appears in Section 4.

Theorem 12. *For any infinite query tree* T *with bounded degree* d*, there exists a constant* c*, which depends only on* d*, such that for any large enough* $N > 0$*,*

$$Pr[|T| > N] \leq e^{-cN}.$$

As a query tree T of bounded degree $D = d(\ell+1)\ell(d-1)^{\ell-1}$ is an upper bound to X_ℓ (by Corollary 10, D is an upper bound on the degree of $C_{M_{\ell-2}}(\ell)$), we have the following corollary to Theorem 12.

Corollary 13. *There exists an absolute constant* c*, which depends only on* d*, such that for any large enough* $N > 0$*,*

$$Pr[X_\ell > N] \leq e^{-cN}.$$

Denote by f_ℓ the number of calls to IsAnAugmentingPath(ℓ) during one execution of **LocalMM**. Let $f = \sum_{\ell=1}^{2k-1} f_\ell$.[6] The base cases of the recursive calls **LocalMM** makes are IsAnAugmentingPath(1) (which always returns TRUE). As the execution of each procedure of **LocalMM** results in at least one call to IsAnAugmentingPath, f (multiplied by some small constant) is an upper bound to the total number of computations made by **LocalMM**.

We state the following proposition, the proof of which appears in Section 4.

Proposition 14 *Let* W_i *be a random variable. Let* $z_1, z_2, \ldots z_{W_i}$ *be random variables, (some possibly equal to* 0 *with probability* 1*). Assume that there exist constants* c *and* μ *such that for all* $1 \leq j \leq W_i$*,* $Pr[z_j \geq \mu N] \leq e^{-cN}$*, for all* $N > 0$*. Then there exist constants* μ_i *and* c_i'*, which depend only on* d*, such that for any* $q_i > 0$*,*

$$Pr[\sum_{j=1}^{W_i} z_j \geq \mu_i q_i | W_i \leq q_i] \leq e^{-c_i' q_i}.$$

Using Proposition 14, we prove the following:

Proposition 15. *For every* $1 \leq \ell \leq 2k-1$*, there exist constants* μ_ℓ *and* c_ℓ*, which depend only on* d *and* ϵ*, such that for any large enough* $N > 0$

$$Pr[f_\ell > \mu_\ell N] \leq e^{-c_\ell N}.$$

Proof. The proof is by induction. For the base of the induction, we have, from Corollary 13, that there exists an absolute constant c_{2k-1}, which depends only on d, such that for any large enough $N > 0$, $Pr[X_{2k-1} > N] \leq e^{-c_{2k-1}N}$. Assume that the proposition holds for $\ell = 2k-1, 2k-3, \ldots \ell$, and we show that it holds for $\ell - 2$.

Let $b_\ell = (\ell + 2d - 2) \cdot \ell(d-1)^{\ell-1}$. From Proposition 11, we have that each call to IsAnAugmentingPath(ℓ) generates at most b_ℓ calls to IsPathInMIS($\ell - 2$), and hence $b_\ell \cdot X_{\ell-2}$ calls to IsAnAugmentingPath($\ell - 2$). From Corollary 13, we have that there exists an absolute constant c, which depends only on d, such that for any large enough $N > 0$,

$$Pr[X_{\ell-2} > N] \leq e^{-cN}.$$

Setting $W_\ell = b_\ell f_\ell$, $f_{\ell-2} = \sum_{j=1}^{W_i} z_j$, $q_i = b_\ell \mu_\ell y_\ell$, and $\mu_i = \mu_{\ell-2}/b_\ell\mu_\ell$, and letting $c_i' = c_\ell'/b_\ell\mu_\ell$ in Proposition 14 implies the following:

$$Pr[f_{\ell-2} > \mu_{\ell-2} y_\ell | f_\ell \leq \mu_\ell y_\ell] \leq e^{-c_\ell' y_\ell}. \tag{1}$$

[6] For all even ℓ, let $f_\ell = 0$.

We have

$$\begin{aligned} Pr[f_{\ell-2} > \mu_{\ell-2}N] =& Pr[f_{\ell-2} > \mu_{\ell-2}N | f_\ell \leq \mu_\ell N] \cdot Pr[f_\ell \leq \mu_\ell N] \\ &+ Pr[f_{\ell-2} > \mu_{\ell-2}N | f_\ell > \mu_\ell N] \cdot Pr[f_\ell > \mu_\ell N] \\ \leq& Pr[f_{\ell-2} > \mu_{\ell-2}N | f_\ell \leq \mu_\ell N] + Pr[f_\ell > \mu_\ell N] \\ \leq& e^{-c'_\ell N} + e^{-c_\ell N} \\ =& e^{-c_{\ell-2}N}, \end{aligned} \tag{2}$$

where Inequality 2 stems from Inequality 1 and the induction hypothesis. □

Taking a union bound over all k levels immediately gives

Lemma 16. *There exists a constant c, which depends only on d and ϵ, such that*

$$Pr[f > c \log n] \leq 1/n^2.$$

Proof (Proof of Theorem 3). Using Lemma 16, and taking a union bound over all possible queried edges gives us that with probability at least $1 - 1/n$, **LocalMM** will require at most $O(\log n)$ queries. Therefore, for each execution of **LocalMM**, we require at most $O(\log n)$-independence for each conflict graph, and therefore, from Theorem 2, we require $\lceil 1/\epsilon \rceil$ seeds of length $O(log^3 n)$, which upper bounds the space required by the algorithm. The time required is upper bound by the time required to compute $r(p)$ for all the required nodes in the conflict graphs, which is $O(\log^4 n)$. □

4 Combinatorial Proofs

We want to bound the total number of queries required by Algorithm **LocalMM**.

Let T be a d-regular query tree. As in [2,14], we partition the interval [0,1] into $L \geq d+1$ sub-intervals: $I_i = (1 - \frac{i}{L+1}, 1 - \frac{i-1}{L+1}]$, for $i = 1, 2, \cdots, L$ and $I_{L+1} = [0, \frac{1}{L+1}]$. We refer to interval I_i as *level* i. A vertex $v \in T$ is said to be on level i if $r(v) \in I_i$. Assume the worst case, that for the root of the tree, v_0, $r(v_0) = 1$. The vertices on level 1 form a tree T_1 rooted at v_0. Denote the number of (sub)trees on level i by t_i. The vertices on level 2 will form a forest of subtrees $\{T_2^{(1)}, \cdots, T_2^{(t_2)}\}$, where the total number of subtrees is at most the sum of the number of children of all the vertices in T_1. Similarly, the vertices on level $i > 1$ form a forest of subtrees $F_i = \{T_i^{(1)}, \cdots T_i^{(t_i)}\}$. Note that all these subtrees $\{T_i^{(j)}\}$ are generated independently by the same stochastic process, as the ranks of all of the nodes in T are i.i.d. random variables. Denote $f_i = |F_i|$, and let $Y_i = \sum_{j=1}^{i} f_j$. Note that F_{i+1} can consist of at most Y_i subtrees.

We prove the following theorem.

Theorem 12. *For any infinite query tree T with bounded degree d, there exists a constant c, which depends only on d, such that for any large enough $N > 0$,*

$$Pr[|T| \geq N] \leq e^{-cN}.$$

We require the following Lemma from [14].

Lemma 17 ([14]). *Let $L \geq d+1$ be a fixed integer and let T be the d-regular infinite query tree. Then for any $1 \leq i \leq L$ and $1 \leq j \leq t_i$, there is an absolute constant c, which depends only on d, such that for all $N > 0$,*

$$\Pr[|T_i^{(j)}| \geq N] \leq e^{-cN}.$$

We first prove the following proposition:

Proposition 18. *For any infinite query tree T with bounded degree d, there exist constants μ_1 and c_1, which depend only on d, such that for any $1 \leq i \leq L-1$, and any $y_i > 0$,*

$$Pr[f_{i+1} \geq \mu_1 y_i | Y_i = y_i] \leq e^{-c_1 y_i}.$$

Proof. Fix $Y_i = y_i$. Let $\{z_1, z_2, \dots z_{y_i}\}$ be integers such that $\forall 1 \leq i \leq y_i, z_i \geq 0$ and let $x_i = \sum_{i=1}^{y_i} z_i$. By Lemma 17, the probability that F_{i+1} consists exactly of trees of size $(z_1, z_2, \dots z_{y_i})$ is at most $\prod_{i=1}^{y_i} e^{-cz_i} = e^{-cx_i}$. There are $\binom{x_i+y_i}{y_i}$ vectors that can realize x_i.[7] We want to bound $Pr[f_{i+1} = \mu y_i | Y_i = y_i]$ for some large enough constant $\mu > 0$. Letting $x_i = \mu y_i$, we bound it as follows:

$$\begin{aligned}
Pr[f_{i+1} = x_i | Y_i = y_i] &\leq \binom{x_i + y_i}{y_i} e^{-x_i} \\
&\leq \left(\frac{e \cdot (x_i + y_i)}{y_i}\right)^{y_i} e^{-cx_i} \\
&= \left(\frac{e \cdot (\mu y_i + y_i)}{y_i}\right)^{y_i} e^{-c\mu y_i} \\
&= (e \cdot (1+\mu))^{y_i} e^{-c\mu y_i} \\
&= e^{y_i(-c\mu + \ln(1+\mu) + 1)} \\
&\leq e^{-c'\mu y_i},
\end{aligned}$$

for some constant $c' > 0$. It follows that

$$\begin{aligned}
Pr[f_{i+1} \geq \mu y_i | Y_i = y_i] &\leq \sum_{k=\mu y_i}^{\infty} e^{-c'k} \\
&\leq e^{-c_1 y_i},
\end{aligned}$$

for some constant $c_1 > 0$. □

Proposition 18 immediately implies the following corollary.

Corollary 19. *For any infinite query tree T with bounded degree d, there exist constants μ and c, which depend only on d, such that for any $1 \leq i \leq L-1$, and any y_i,*

$$Pr[f_{i+1} \geq \mu y_i | Y_i \leq y_i] \leq e^{-cy_i}.$$

Corollary 19, which is about query trees, can be restated as follows: let $W_i = Y_i$, $q_i = y_i$ and $\sum_{i=1}^{W_i} z_i = f_{i+1}$. Furthermore, let $c_i' = c_1$ and $\mu_i' = \mu_1$ for all i. This notation yields the following proposition, which is unrelated to query trees, and which we used in Section 3:

Proposition 14 *Let W_i be a random variable. Let $z_1, z_2, \dots z_{W_i}$ be random variables (some possibly equal to 0 with probability 1). Assume that there exist constants c and μ such that for all $1 \leq j \leq W_i$, $Pr[z_j \geq \mu N] \leq e^{-cN}$, for all $N > 0$. Then there exist constants μ_i and c_i', which depend only on d, such that for any $q_i > 0$,*

$$Pr[\sum_{j=1}^{W_i} z_j \geq \mu_i q_i | W_i \leq q_i] \leq e^{-c_i' q_i}.$$

We need one more proposition before we can prove Theorem 12. Notice that $f_1 = |T_1|$.

Proposition 20. *For any infinite query tree T with bounded degree d, for any $1 \leq i \leq L$, there exist constants μ_i and c_i, which depend only on d, such that for and any $N > 0$,*

$$Pr[f_i \geq \mu_i N] \leq e^{-c_i N}.$$

[7] This can be thought of as y_i separators of x_i elements.

The proof is similar to the proof of Proposition 15. We include it for completeness.

Proof. The proof is by induction on the levels $1 \leq i \leq L$, of T.

For the base of the induction, $i = 1$, by Lemma 17, we have that there exist some constants μ_1 and c_1 such that

$$Pr[f_1 \geq \mu_1 N] \leq e^{-c_1 N},$$

as $f_1 = |T_1|$.

For the inductive step, we assume that the proposition holds for levels $1, 2, \ldots, i-1$, and show that it holds for level i.

$$\begin{aligned} Pr[f_i \geq \mu_i N] =& Pr[f_i \geq \mu_i N | Y_{i-1} < \mu_{i-1} N] \cdot Pr[Y_{i-1} < \mu_{i-1} N] \\ &+ Pr[f_i \geq \mu_i N | Y_{i-1} \geq \mu_{i-1} N] \cdot Pr[Y_{i-1} \geq \mu_{i-1}] \\ \leq& Pr[f_i \geq \mu_i N | Y_{i-1} < \mu_{i-1} N] + Pr[Y_{i-1} \geq \mu_{i-1}] \\ \leq& e^{-cN} + e^{-c_{i-1} N} \qquad (3) \\ \leq& e^{-c_i N}, \end{aligned}$$

for some constant c_i. Inequality 3 stems from Corollary 19 and the inductive hypothesis. □

We are now ready to prove Theorem 12.

Proof (Proof of Theorem 12). We would like to bound $Pr[|T| = \sum_{i=1}^{L} f_i \geq \mu N]$. From Proposition 20, we have that for $1 \leq i \leq L$,

$$Pr[f_i \geq \mu_i N] \leq e^{-c_i N}.$$

A union bound on the L levels gives the required result. □

References

1. Noga Alon, László Babai, and Alon Itai. A fast and simple randomized algorithm for the maximal independent set problem. *Journal of Algorithms*, 7:567–583, 1986.
2. Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proc. 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1132–1139, 2012.
3. Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842–844, 1957.
4. Andrzej Czygrinow and Michal Hanckowiak. Distributed algorithm for better approximation of the maximum matching. In *COCOON*, pages 242–251, 2003.
5. Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
6. Philip Hall. On representatives of subsets. *J. London Math. Soc.*, 10(1):26–30, 1935.
7. Nicholas Harvey. Algebraic structures and algorithms for matching and matroid problems. In *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 531–542, 2006.
8. Jaap-Henk Hoepman, Shay Kutten, and Zvi Lotker. Efficient distributed weighted matchings on trees. In *SIROCCO*, pages 115–129, 2006.
9. John E. Hopcroft and Richard M. Karp. An $N^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.
10. Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Inf. Process. Lett.*, 22(2):77–80, 1986.
11. Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *Proc. 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 980–989, 2006.
12. Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
13. Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. In *Proc. 20th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 129–136, 2008.

14. Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In *Proc. 39th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 653–664, 2012.

15. Yishay Mansour and Shai Vardi. Local algorithmic mechanism design. Under submission elsewhere.

16. Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proc. 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.

17. Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *FOCS*, pages 248–255, 2004.

18. James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

19. Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 327–336, 2008.

20. Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Proc. 2nd Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011.

A Pseudocode for Algorithm GreedyMIS

Algorithm 2 - **GreedyMIS** - On-line MIS algorithm with input $G = (V, E)$ and vertex permutation π

$I \leftarrow \emptyset$ ▷ I is a set of independent vertices.
Let $\pi = (v_1, v_2, \ldots, v_n)$.
for $i = 1$ to n **do**
    if $\forall u \in N(v_i)$, $u \notin I$ **then**
        $I \leftarrow I \cup \{v_i\}$.
    end if
end for
Output I ▷ I is an MIS.

B Pseudocode for Algorithm LocalMM

Algorithm 3 - **LocalMM** - LCA for MM with input $G = (V, E)$, $e \in E$ and $\epsilon > 0$

Global $\mathcal{S} = \emptyset$ ▷ $\mathcal{S}$ is the set of seeds

procedure MAIN(G, e, ϵ)
    if this is the first execution of Algorithm **LocalMM then**
        $(\mathcal{S}, k) \leftarrow$ INITIALIZE(G, ϵ)
    end if
    Return ISINMATCHING($G, e, 2k - 1, \mathcal{S}$).
end procedure

Algorithm 4 Auxiliary procedures

procedure INITIALIZE(G, ϵ) ▷ This is run only at the first execution
  $k \leftarrow \lceil 1/\epsilon \rceil$.
  for $\ell = 1, 3, \ldots 2k-1$ **do**
    Generate a seed s_ℓ of length $O(\log^3 n)$. ▷ s_ℓ is a seed for a random ordering π_ℓ on all possible paths of length ℓ in G.
  end for
  $\mathcal{S} = \bigcup_\ell \mathcal{S}_\ell$.
  Return $(\mathcal{S}, k)$.
end procedure

procedure ISINMATCHING(G, e, ℓ, $\mathcal{S}$)
  if $\ell = -1$ **then** ▷ The empty matching
    Return false.
  end if
  b_1 = ISINMATCHING(G, e, $\ell - 2$, $\mathcal{S}$).
  $b_2 =$ false.
  $P = \{p \in G : e \in p \wedge |p| = \ell\}$
  for all $p \in P$ **do**
    if ISPATHINMIS(G, p, ℓ, $\mathcal{S}$) **then**
      $b_2 =$ true.
    end if
  end for
  Return $b_1 \oplus b_2$.
end procedure

procedure ISPATHINMIS(G, p, ℓ, $\mathcal{S}$)
  $C \leftarrow$ RELEVANTPATHS(G, p, ℓ, $\mathcal{S}$). ▷ C is a subgraph of $C_{M_{\ell-2}}(\ell)$
  $I \leftarrow$ **Greedy MIS** $(C, \pi(C, s_\ell))$
  $b = (v \in I)$
  Return b
end procedure

procedure ISFREE(G, v, ℓ, $\mathcal{S}$) ▷ Checks that a vertex is free
  IsFreeVertex = true.
  for all $u \in N(v)$ **do** ▷ All edges touching v
    if ISINMATCHING($G, (u, v), \ell - 2, \mathcal{S}$) **then**
      IsFreeVertex = false.
    end if
  end for
  Return IsFreeVertex.
end procedure

Algorithm 5 More auxiliary procedures

procedure RELEVANTPATHS(G, p, ℓ, $\mathcal{S}$)
  Initialize $C = (V_C, E_C) \leftarrow (\emptyset, \emptyset)$.
  if ISANAUGMENTINGPATH($G, p, \ell, \mathcal{S}$) **then**
    $V_C = \{p\}$.
  else
    Return C.
  end if
  while $\exists p \in V_C : (p, p') \in E_C, r_\ell(p', s_\ell) < r_\ell(p, s_\ell)$ **do**
    if ISANAUGMENTINGPATH($G, p', \ell, \mathcal{S}$) **then**
      $V_C \leftarrow p'$
      for all $p'' \in N(p')$ **do** ▷ Edges between p' and vertices in V_C
        if $p'' \in V_C$ **then**
          $E_C \leftarrow (p', p'')$.
        end if
      end for
    end if
  end while
  Return C.
end procedure

procedure ISANAUGMENTINGPATH(G, p, ℓ, $\mathcal{S}$) ▷ Checks that p is an augmenting path.
  If $\ell = 1$ return TRUE. ▷ all edges are augmenting paths of the empty matching
  Let $p = (e_1, e_2, \ldots, e_\ell)$, with end vertices $v_1, v_{\ell+1}$.
  IsPath = true.
  for $i = 1$ to ℓ **do**
    if $i \pmod 2 = 0$ **then** ▷ All even numbered edges should be in the matching
      if $\neg$ISINMATCHING($G, e_i, \ell - 2, \mathcal{S}$) **then**
        IsPath = false.
      end if
    end if
    if $i \pmod 2 = 1$ **then** ▷ No odd numbered edges should be in the matching
      if ISINMATCHING($G, e_i, \ell - 2, \mathcal{S}$) **then**
        IsPath = false.
      end if
    end if
  end for
  if ($\neg$ISFREE($G, v_1, \ell, \mathcal{S}$)$\vee$ ($\neg$ISFREE($G, v_{\ell+1}, \ell, \mathcal{S}$) **then**
    IsPath = false. ▷ The vertices at the end should be free
  end if
  Return IsPath.
end procedure