Perfect Matchings in $O(n \log n)$ Time in Regular Bipartite Graphs

Ashish Goel[*]   Michael Kapralov[†]   Sanjeev Khanna[‡]

September 16, 2011

Abstract

In this paper we consider the well-studied problem of finding a perfect matching in a d-regular bipartite graph on $2n$ nodes with $m = nd$ edges. The best-known algorithm for general bipartite graphs (due to Hopcroft and Karp) takes time $O(m\sqrt{n})$. In regular bipartite graphs, however, a matching is known to be computable in $O(m)$ time (due to Cole, Ost, and Schirra). In a recent line of work by Goel, Kapralov, and Khanna the $O(m)$ time bound was improved first to $\tilde{O}\left(\min\{m, n^{2.5}/d\}\right)$ and then to $\tilde{O}\left(\min\{m, n^2/d\}\right)$.

In this paper, we give a randomized algorithm that finds a perfect matching in a d-regular graph and runs in $O(n \log n)$ time (both in expectation and with high probability). The algorithm performs an appropriately truncated *alternating random walk* to successively find augmenting paths. Our algorithm may be viewed as using adaptive uniform sampling, and is thus able to bypass the limitations of (non-adaptive) uniform sampling established in earlier work. Our techniques also give an algorithm that successively finds a matching in the support of a doubly stochastic matrix in expected time $O(n \log^2 n)$, with $O(m)$ pre-processing time; this gives a simple $O(m + mn \log^2 n)$ time algorithm for finding the Birkhoff-von Neumann decomposition of a doubly stochastic matrix.

We show that randomization is crucial for obtaining $o(nd)$ time algorithms by establishing an $\Omega(nd)$ lower bound for deterministic algorithms. We also show that there does not exist a randomized algorithm that finds a matching in a regular bipartite multigraph and takes $o(n \log n)$ time with high probability.

1   Introduction

A bipartite graph $G = (P, Q, E)$ with vertex set $P \cup Q$ and edge set $E \subseteq P \times Q$ is said to be d-regular if every vertex has the same degree d. We use $m = nd$ to denote the number of edges in G and n to represent the number of vertices in P (as a consequence of regularity, P and Q have the same size). Regular bipartite graphs have been studied extensively, in particular in the context of expander constructions, scheduling, routing in switch fabrics, and task-assignment [13, 1, 6].

A regular bipartite graph of degree d can be decomposed into exactly d perfect matchings, a fact that is an easy consequence of Hall's theorem [3][1] and is closely related to the Birkhoff-von Neumann decomposition of a doubly stochastic matrix [2, 16]. Finding a matching in a regular bipartite graph is a well-studied problem, starting with the algorithm of König in 1916 [12], which is now known to run in time $O(mn)$. The well-known bipartite matching algorithm of Hopcroft and Karp [10] can be used to obtain a running time of $O(m\sqrt{n})$. In graphs where d is a power of 2, the following elegant idea, due to Gabow and Kariv [7], leads to an algorithm with $O(m)$ running time. First, compute an Euler tour of the graph (in time $O(m)$) and then follow this tour in an arbitrary direction. Exactly half the edges will go from left to right; these form a regular bipartite graph of degree $d/2$. The total running time $T(m)$ thus follows the recurrence $T(m) = O(m) + T(m/2)$ which yields $T(m) = O(m)$. Extending this idea to the general case proved quite hard, and after a series of improvements (e.g. by Cole and Hopcroft [5], and then by Schrijver [15] to $O(md)$). Cole, Ost, and Schirra [6] gave an $O(m)$ algorithm for the case of general d. Their main interest was in edge coloring of general bipartite graphs, where finding perfect matchings in regular bipartite graphs is an important subroutine.

Recently, Goel, Kapralov, and Khanna [9], gave a sampling-based algorithm that computes a perfect matching in d-regular bipartite graphs in $O(\min\{m, \frac{n^{2.5} \log n}{d}\})$ expected time, an expression that is bounded by $\tilde{O}(n^{1.75})$. The algorithm of [9] uses uniform sampling to reduce the number of edges in the input graph while preserving a perfect matching, and then runs the Hopcroft-Karp algorithm on the sampled graph. The authors of [9] also gave a lower bound of $\tilde{\Omega}\left(\min\{nd, \frac{n^2}{d}\}\right)$ on the running time of an algorithm that uses non-adaptive uniform sampling to reduce the number of edges in the graph as the first step. This lower bound was matched in [8], where the authors use a two stage sampling scheme and a specialized analysis of the runtime of the Hopcroft-Karp algorithm on the sampled graph to obtain a runtime of $\tilde{O}\left(\min\{nd, \frac{n^2}{d}\}\right)$.

For sub-linear (in m) running time algorithms, the exact data model is important. In this paper, as well as in the sub-linear running time algorithms mentioned above, we assume that the graph is presented in the adjacency array format, i.e., for each vertex, its d neighbors are stored in an array. This is the most natural input data structure for our problem. For simple graphs or multigraphs with edge multiplicities bounded above by $\gamma d, \gamma \in (0, 1)$ our algorithms will not make any ordering assumptions within an adjacency array. However, the data layout will be important for multigraphs without any assumptions on edge multiplicities.

Given a partial matching in an undirected graph, an *augmenting path* is a path which starts and ends at an unmatched vertex, and alternately contains edges that are outside and inside the partial matching. Many of the algorithms mentioned above work by repeatedly finding augmenting paths.

1.1   Our Results and Techniques

Our main result is the following theorem.

Theorem 1 *There exists a randomized algorithm for finding a perfect matching in a d-regular bipartite graph $G = (P, Q, E)$ given in adjacency array representation, and takes time $O(n \log n)$ time both in expectation as well as with high probability.*

---

[1]In fact, the first proof of this result was obtain by König in 1916 [12] and predates Hall's theorem.

The algorithm is very simple: the matching is constructed by performing one augmentation at a time, and new augmenting paths are found by performing an *alternating random walk* with respect to the current matching. The alternating random walk on G, defined in Section 2, can be viewed as a random walk on a modified graph that encodes the current matching. The random walk approach may still be viewed as repeatedly drawing a uniform sample from the adjacency array of some vertex v; however this vertex v is now chosen adaptively, thus allowing us to bypass the $\tilde{\Omega}\left(\min\{nd, \frac{n^2}{d}\}\right)$ lower bound on non-adaptive uniform sampling established in [9]. Somewhat surprisingly, we show that the total time taken by these random augmentations can be bounded by $O(n \log n)$ in expectation, only slightly worse than the $\Omega(n)$ time needed to simply output a perfect matching. The proof involves analyzing the hitting time of the sink node in the random walk. It should be noted here that, of course, the $O(m)$ algorithm of Cole-Ost-Schirra is faster than our algorithm when $d = o(\log n)$.

In section 4 we establish that randomization is crucial to obtaining an $o(nd)$ time algorithm, thus showing that the algorithm of [6] is asymptotically optimal in the class of deterministic algorithms. In section 5, we show that no *randomized* algorithm can achieve *high probability* runtime better than $O(n \log n)$ for the case of d-regular bipartite multigraphs.

Our techniques also extend to the problem of finding a perfect matching in the support of a doubly-stochastic matrix, as well as to efficiently compute the Birkhoff-von-Neumann decomposition of a doubly stochastic matrix. The details are given in section 3.1.

Finally, we note that an application of Yao's min-max theorem (see, for instance, [13]) to Theorem 1 immediately yields the following corollary:

Corollary 2 *For any distribution on regular bipartite graphs with $2n$ nodes, there exists a deterministic algorithm that runs in average time $O(n \log n)$ on graphs drawn from this distribution.*

A similar corollary also follows for doubly stochastic matrices.

2   Matchings in d-Regular Bipartite Graphs

2.1   The Basic Algorithm

Let $G = (P, Q, E)$ denote the input d-regular graph and let M be a partial matching in G. We first describe the *alternating random walk* on G with respect to M. We assume that the algorithm has access to the function SAMPLE-OUT-EDGE that takes a vertex $u \in P$ and returns a uniformly random unmatched edge going out of u. The implementation and runtime of SAMPLE-OUT-EDGE depend on the representation of the graph. It is assumed in Theorem 1 and in this section that the graph G does not have parallel edges and is represented in adjacency array format, in which case SAMPLE-OUT-EDGE can be implemented to run in expected constant time (throughout the paper we assume the model in which a random number in the range $1 : n$ can be generated in $O(1)$ time). In Theorem 5, however, a preprocessing step will be required to convert the matrix to an augmented binary search tree, in which case SAMPLE-OUT-EDGE can be implemented to run in $O(\log n)$ time.

The alternating random walk starts at a uniformly random unmatched vertex $u_0 \in P$ and proceeds as follows:

1. Set $v :=$ SAMPLE-OUT-EDGE(u_j);

2. If v is matched, set $u_{j+1} := M(v)$, otherwise terminate.

Note that an augmenting path with respect to M can be obtained from the sequence of steps taken by the alternating random walk by removing possible loops.

We now state a basic version of our algorithm:

Algorithm 1

Input: A d-regular bipartite graph $G = (P, Q, E)$ in adjacency array format.

Output: A perfect matching of G.

1. Set $j := 0$, $M_0 := \emptyset$.

2. Run the alternating random walk starting from a random unmatched vertex in P until it hits an unmatched vertex in Q.

3. Denote the augmenting path obtained by removing possible loops from the sequence of steps taken by the walk by p. Set $M_{j+1} := M_j \Delta p$.

4. Set $j := j + 1$ and go to step 2.

Here for two sets of edges $A, B \subseteq E$ we use the notation $A \Delta B$ to denote the symmetric difference of A and B. In particular, if M is a matching and p is an augmenting path with respect to M, then $M \Delta p$ is the set of edges obtained by augmenting M with p.

We prove in the next section that this algorithm takes $O(n \log n)$ time in expectation. The high probability result is obtained in section 2.3 by performing appropriately truncated random walks in step 2 instead of a single untruncated walk.

2.2   Expected Running Time Analysis

The core of our analysis is the following lemma, which bounds the time that it takes an alternating random walk in G with respect to a partial matching M that leaves $2k$ vertices unmatched to reach an unmatched vertex.

Lemma 3 *Let $G = (P, Q, E)$ be a d-regular bipartite graph and let M be a partial matching that leaves $2k$ vertices unmatched. Then the expected number of steps before the alternating random walk in G reaches an unmatched vertex is at most $4 + 2n/k$.*

Proof:

It will be convenient to use the auxiliary notion of a *matching graph* $H(G, M)$ which will allow us to view alternating random walks in G with respect to M as random walks in $H(G, M)$ starting from a special source node s and hitting a special sink node t. We then get the result by bounding the hitting time from s to t in $H(G, M)$.

The matching graph corresponding to the matching M is defined to be the directed graph H obtained by transforming G as follows:

1. Orient edges of G from P to Q;

2. Add a vertex s connected by d parallel edges to each unmatched node in P, directed out of s;

3. Add a vertex t connected by d parallel edges to each unmatched node in Q, directed into t;

4. Contract each pair $(u, v) \in M$ into a supernode.

The graph H has $n + k + 2$ nodes. Note that for every vertex $v \in H$, $v \neq s, t$ the in-degree of v is equal to its out-degree, the out-degree of s equals dk, as is the in-degree of t. Also, any path from s to t in H gives an augmenting path in G with respect to M. We now concentrate on finding a path from s to t in H.

Construct the graph H^* by identifying s with t in H and adding a self-loop at every vertex (and thus increasing all degrees by 1). Denote the vertex that corresponds to s and t by s^*. Note that H^* is a balanced directed graph, i.e. the out-degree of every vertex is equal to its in-degree. The out-degree of every vertex except s^* is at most $d + 1$, while the out-degree of s^* is $dk + 1$.

Consider the simple random walk in H^* started at s^* where at each step of the random walk, we choose an outgoing edge uniformly at random. We wish to analyze the expected time for a return visit to s^*; this is precisely the expected time to find an augmenting path. We first observe that since H^* is a balanced directed graph, any strongly connected component must have the same number of edges entering and leaving that component. It follows that every strongly connected component is isolated, and hence the Markov chain induced by the above simple random walk is irreducible. Furthermore, the addition of self-loops ensures that the chain is aperiodic. Denote the set of vertices in the strongly connected component of s^* by C.

By the Fundamental theorem of Markov chains (see, e.g. [13], Theorem 6.2), we know that there is a unique stationary distribution, and it is easy to verify that it is given by

$$\pi_u = \frac{\deg(u)}{\sum_{v \in C} \deg(v)}$$

for each $u \in C$. Since the expected return time to s^* is equal to the inverse of the stationary measure of s^*, we get that

$$\frac{1}{\pi_{s^*}} = \frac{\sum_{v \in C} \deg(v)}{\deg(s^*)} \leq \frac{(n + k)(d + 1) + 2(dk + 1)}{kd} \leq 4 + \frac{2n}{k}.$$

∎

We can now prove

Theorem 4 *Algorithm 1 finds a matching in a d-regular bipartite graph $G = (P, Q, E)$ in expected time $O(n \log n)$.*

Proof: By Lemma 3 it takes at most $1 + n/(n - j)$ expected time to find an augmenting path with respect to partial matching M_j. Hence, the expected runtime of the algorithm is bounded by

$$\sum_{j=0}^{n-1} 1 + n/(n - j) = n + nH_n = O(n \log n),$$

where $H(n) := 1 + 1/2 + 1/3 + \ldots + 1/n$ is the n-th Harmonic number. ∎

2.3 Truncated Random Walks and High Probability Analysis

In this section we show how Algorithm 1 can be modified by introducing truncated random walks to obtain a running time of $O(n \log n)$ with high probability.

Algorithm 2

Input: A d-regular bipartite graph $G = (P, Q, E)$ in adjacency array format.

Output: A perfect matching of G.

1. Set $j := 0$, $M_0 := \emptyset$.

2. Repeatedly run alternating random walks for $b_j := 4\left(1 + \frac{n}{n-j}\right)$ steps until a successful run is obtained.

3. Denote the augmenting path obtained by removing possible loops from the sequence of steps taken by the walk by p. Set $M_{j+1} := M_j \Delta p$.

4. Set $j := j + 1$ and go to step 2.

We now analyze the running time of our algorithm, and prove Theorem 1.

Proof of Theorem 1:

We now show that Algorithm 2 takes time $O(n \log n)$ with high probability. First note that by Lemma 3 and Markov's inequality, a truncated alternating random walk in step 2 succeeds with probability at least $1/2$. Let X_j denote the time taken by the j-th augmentation. Let Y_j be independent exponentially distributed with mean $\mu_j := \frac{b_j}{\ln 2}$. Note that

$$\mathbf{Pr}[X_j \geq qb_j] \leq 2^{-q} = \exp\left[-\frac{qb_j \ln 2}{b_j}\right] = \mathbf{Pr}[Y_j \geq qb_j]$$

for all $q > 1$, so

$$\mathbf{Pr}[X_j \geq x] \leq \mathbf{Pr}[Y_j \geq x] \tag{1}$$

for all $x > b_j$. We now prove that $Y := \sum_{0 \leq j \leq n-1} Y_j \leq cn \log n$ w.h.p. for a suitably large positive constant c. Denote $\mu := \mathbf{E}[Y]$. By Markov's inequality, for any $t, \delta > 0$

$$\mathbf{Pr}[Y \geq (1+\delta)\mu] \leq \frac{\mathbf{E}[e^{tY}]}{e^{t(1+\delta)\mu}}.$$

Also, for any j, and for $t < 1/\mu_j$, we have

$$\mathbf{E}[e^{tY_j}] = \frac{1}{\mu_j}\int_0^\infty e^{tx}e^{-x/\mu_j}dx = \frac{1}{1-t\mu_j}.$$

The two expressions above, along with the fact that the Y_j's are independent, combine to give:

$$\mathbf{Pr}[Y \geq (1+\delta)\mu] \leq \frac{e^{-t(1+\delta)\mu}}{\prod_{j=0}^{n-1}(1-t\mu_j)}. \tag{2}$$

Observe that μ_{n-1} is the largest of the μ_j's. Assume that $t = \frac{1}{2\mu_{n-1}}$, which implies that $(1-t\mu_j) \geq e^{-t\mu_j \ln 4}$. Plugging this into equation 2, we get:

$$\mathbf{Pr}[Y \geq (1+\delta)\mu] \leq e^{-(1+\delta-\ln 4)\mu/(2\mu_{n-1})}. \tag{3}$$

Further observe that $\mu = \frac{4}{\ln 2}\sum_{j=0}^{n-1}(1 + n/(n-j)) = \frac{4}{\ln 2}n(1 + H_n) \geq \mu_{n-1}H(n)$, where $H(n) := 1 + 1/2 + 1/3 + \ldots + 1/n$ is the n-th Harmonic number. Since $H(n) \geq \ln n$, we get our high probability result:

$$\mathbf{Pr}[Y \geq (1+\delta)\mu] \leq n^{-(1+\delta-\ln 4)/2}. \tag{4}$$

Since $\mu = O(n \log n)$, this completes the proof of Theorem 1. ∎

3 Matchings in Doubly-Stochastic Matrices and Regular Bipartite Multigraphs

3.1 Doubly-Stochastic Matrices

We now apply techniques of the previous section to the problem of finding a perfect matching in the support of an $n \times n$ doubly stochastic matrix $\mathbf{M}$ with m non-zero entries. A doubly-stochastic matrix can be viewed as a regular graph, possibly with parallel edges, and we can thus use the same algorithm and analysis as above, provided that SAMPLE-OUT-EDGE can be implemented efficiently. We start by describing a simple data structure for implementing SAMPLE-OUT-EDGE. For each vertex v, we store all the outgoing edges from v in a balanced binary search tree, augmented so that each node in the search tree also stores the weight of all the edges in its subtree. Since inserts into, deletes from, and random samples from this augmented tree all take time $O(\log n)$, we obtain a running time of $O(n \log^2 n)$ for finding a matching in the support of a doubly stochastic matrix.

Superficially, it might seem that initializing the balanced binary search trees for each vertex takes total time $\Theta(m \log n)$. However, note that there is no natural ordering on the outgoing edges from a vertex, and we can simply superimpose the initial balanced search tree for a vertex on the adjacency array for that vertex, assuming that the underlying keys are in accordance with the (arbitrary) order in which the edges occur in the adjacency array. We have proved

Theorem 5 *Given an $n \times n$ doubly-stochastic matrix M with m non-zero entries, one can find a perfect matching in the support of M in $O(n \log^2 n)$ expected time with $O(m)$ preprocessing time.*

In many applications of Birkhoff von Neumann decompositions (e.g. routing in network switches [4]), we need to find one perfect matching in a single iteration, and then update the weights of the matched edges. In such applications, each iteration can be implemented in $O(n \log^2 n)$ time (after initial $O(m)$ preprocessing time), improving upon the previous best known bound of $O(mb)$ where b is the bit precision.

The complete Birkhoff-von Neumann decomposition can be computed by subtracting an appropriately weighted matching matrix from $\mathbf{M}$ every time a matching is found, thus decreasing the number of nonzero entries of $\mathbf{M}$. Note that the augmented binary search tree can be maintained in $O(\log n)$ time per deletion. This yields

Corollary 6 *For any $k \geq 1$, there exists an $O(m + kn \log^2 n)$ expected time algorithm for finding k distinct matchings (if they exist) in the Birkhoff-von-Neumann decomposition of an $n \times n$ doubly stochastic matrix with m non-zero entries.*

3.2 Regular Bipartite Multigraphs

For regular bipartite multigraphs with edge multiplicities at most $d/2$, Algorithm 2 still takes time at most $O(n \log n)$ with high probability, since SAMPLE-OUT-EDGE can be implemented by sampling the adjacency list of the appropriate vertex in G until we find an unmatched edge. Each sample succeeds with probability at least $1/2$ since the matched edge can have multiplicity at most $d/2$. Here, we assume that an edge with multiplicity k occurs k times in the adjacency arrays of its endpoints.

We also note that our algorithm can be implemented to run in $O(n \log n)$ time without any assumptions on multiplicities if the data layout is as follows. For each vertex we have an adjacency array with edges of multiplicity k appearing as contiguous blocks of length k. Also, each element in the adjacency array is augmented with the index of the beginning of the block corresponding to its edge and the index of the end of the block. Assuming this data layout, SAMPLE-OUT-EDGE can be implemented in $O(1)$ expected time regardless of edge multiplicities: it is sufficient to sample locations outside the block corresponding to the currently matched edge.

It remains to note that when the size of the support of the set of edges, which we denote by m_s, is small, then the data representation used in finding a matching in a doubly-stochastic matrix can be used to find a matching in time $O(m_s + n \log^2 n)$. It is interesting to compare this runtime to the result of [6]. The runtime of their matching algorithm is stated as $O(m + n \log^3 d) = O(m)$, but it is easy to see that it can be implemented to run in $O(m_s + n \log^3 d)$ time.

Remark 7 *Our algorithm can be used to obtain a simple algorithm for edge-coloring bipartite graphs with maximum degree d in time $O(m \log n)$ (slightly worse than the best known $O(m \log d)$ dependence obtained in [6, 11, 14]). In the first step one reduces the problem to that on a regular graph with $O(m)$ edges as described in [6] (note that parallel edges may emerge at this point). The lists of neighbors of every vertex of the graph can then be arranged in a data structure that supports sampling and deletion in $O(1)$ amortized time. It then remains to find matchings repeatedly, taking $O(n \log n)$ time per matching. This takes $O(nd \log n) = O(m \log n)$ time overall.*

One simple approach to implementing sampling and deletion in $O(1)$ amortized time is to use the data layout outlined above: for each vertex we have an adjacency array with edges of multiplicity k appearing as contiguous blocks of length k. Also, each element in the adjacency array is augmented with the index of the beginning of the block corresponding to its edge and the index of the end of the block. Deletion is performed as follows. When an edge is deleted, its block is marked for deletion (it is sufficient to store a corresponding flag at the beginning of each contiguous block), but is only removed when the number of elements marked for deletion (counting multiplicities) exceeds half of the current size of the adjacency array, in which case the whole array is rearranged, removing the marked elements. Note that until the number of elements in the deleted blocks exceeds half of the current size of the array, sampling can be performed in $O(1)$ expected time. On the other hand, since rearrangement of the array takes linear time and is only performed when at least half of the elements are marked for deletion, the amortized cost of deletion is $O(1)$.

4   An $\Omega(nd)$ Lower Bound for Deterministic Algorithms

In this section, we will prove

Theorem 8 *For any $1 \le d < n/12$, there exists a family of d-regular graphs on which any deterministic algorithm for finding a perfect matching requires $\Omega(nd)$ time.*

We will show that for any positive integer d, any deterministic algorithm to find a perfect matching in a d-regular bipartite graph requires $\Omega(nd)$ probes, even in the adjacency array representation, where the ordering of edges in an array is decided by an adversary. Specifically, for any positive integer d, we construct a family $\mathcal{G}(d)$ of simple d-regular bipartite graphs with $O(d)$ vertices each that we refer to as *canonical* graphs. A canonical bipartite graph $G(P \cup \{t\}, Q \cup \{s\}, E) \in \mathcal{G}(d)$ is defined as follows. The vertex set $P = P_1 \cup P_2$ and $Q = Q_1 \cup Q_2$ where $|P_i| = |Q_i| = 3d$ for $i \in \{1, 2\}$. The vertex s is connected to an arbitrary set of d distinct vertices in P_1 while the vertex t is connected to an arbitrary set of d distinct vertices in Q_2. In addition, G contains a matching M' of size d that connects a subset $Q_1' \subseteq Q_1$ to a subset $P_2' \subseteq P_2$, where $|Q_1'| = |P_2'| = d$. The remaining edges in E connect vertices in P_i to Q_i for $i \in \{1, 2\}$ so as to satisfy the property that the degree of each vertex in G is exactly d. It suffices to show an $\Omega(d^2)$ lower bound for graphs drawn from $\mathcal{G}(d)$ since we can take $\Theta(n/d)$ disjoint copies of canonical graphs to create a d-regular graph on n vertices.

Overview: Let $\mathcal{D}$ be a deterministic algorithm for finding a perfect matching in graphs drawn from $\mathcal{G}(d)$. We will analyze a game between the algorithm $\mathcal{D}$ and an adaptive adversary $\mathcal{A}$ whose goal is to maximize the number of edges that $\mathcal{D}$ needs to examine in order to find a perfect matching. In order to find a perfect matching, the algorithm $\mathcal{D}$ must find an edge in M', since s must be matched to a vertex in P_1, and thus in

turn, some vertex in Q_1 must be matched to a vertex in P_2. We will show that the adversary $\mathcal{A}$ can always force $\mathcal{D}$ to examine $\Omega(d^2)$ edges in G before revealing an edge in M'. The specific graph $G \in \mathcal{G}(d)$ presented to the algorithm depends on the queries made by the algorithm $\mathcal{D}$. The adversary adaptively answers these queries while maintaining at all times the invariant that the partially revealed graph is a subgraph of some graph $G \in \mathcal{G}(d)$. The cost of the algorithm is the number of edge locations probed by it before $\mathcal{A}$ reveals an edge in M' to $\mathcal{D}$.

In what follows, we assume that the adversary reveals s, t and the partition of remaining vertices into P_i, Q_i for $1 \leq i \leq 2$, along with all edges from s to P_1 and all edges from t to Q_2, to the deterministic algorithm $\mathcal{D}$ at the beginning. The algorithm pays no cost for this step.

Queries: Whenever the algorithm $\mathcal{D}$ probes a new location in the adjacency array of some vertex $u \in P \cup Q$, we will equivalently view $\mathcal{D}$ as making a query $\mathcal{Q}(u)$ to the adversary $\mathcal{A}$, in response to which the adversary outputs a vertex v that had not been yet revealed as being adjacent to u.

Subgraphs consistent with canonical graphs: Given a bipartite graph $G'(P \cup \{t\}, Q \cup \{s\}, E')$, we say that a vertex $u \in P \cup Q$ is *free* if its degree in G' is strictly smaller than d. We now identify sufficient conditions for a partially revealed graph to be a subgraph of some canonical graph in $\mathcal{G}(d)$.

Lemma 9 *Let $G_r(P \cup \{t\}, Q \cup \{s\}, E_r)$ be any simple bipartite graph such that*

(a) *the vertex s is connected to d distinct vertices in P_1 and the vertex t is connected to d distinct vertices in Q_2,*

(b) *all other edges in G_r connect a vertex in P_i to a vertex in Q_i for some $i \in \{1, 2\}$,*

(c) *degree of each vertex in G_r is at most d, and*

(d) *at least $\frac{5d}{2}$ vertices each in both Q_1 and P_2 have degree strictly less than $\frac{d}{5}$.*

Then for any pair u, v of free vertices such that $u \in P_i$ and $v \in Q_i$ for some $i \in \{1, 2\}$, and $(u, v) \notin E_r$, there exists a canonical graph $G(P \cup \{t\}, Q \cup \{s\}, E) \in \mathcal{G}(d)$ such that that $E_r \cup (u, v) \subseteq E$.

Proof: Let $G'(P \cup \{t\}, Q \cup \{s\}, E')$ be the graph obtained by adding edge (u, v) to G_r, that is, $E' = E_r \cup \{(u, v)\}$. Since u and v are free vertices, all vertex degrees in G' remain bounded by d. We now show how G' can be extended to a d-regular canonical graph.

We first add to G' a perfect matching M' of size d connecting an arbitrary set of d *free* vertices in Q_1 to an arbitrary set of d *free* vertices in P_2. This is feasible since G' has at least $\frac{5d}{2}$ free vertices each in both Q_1 and P_2. In the resulting graph, since the total degree of all vertices in P_i is same as the total degree of all vertices in Q_i, we can repeatedly pair together a vertex of degree less than d in P_i with a vertex of degree less than d in Q_i until degree of each vertex becomes exactly d, for $i \in \{1, 2\}$. Let E'' be the set of edges added to $G' \cup M'$ in this manner, and let G'' be the final graph. The graph G'' satisfies all properties of a canonical graph in the family $\mathcal{G}(d)$ except that it may not be a simple graph. We next transform G'' into a simple d-regular graph by suitably modifying edges in E''.

Given any graph $H(V_H, E_H)$, we define

$$\Phi(H) = \sum_{(x,y) \in V_H \times V_H} \max\{0, \eta(x, y) - 1\},$$

where $\eta(x, y)$ denotes the number of times the edge (x, y) appears in H. Note that $\Phi(H) = 0$ iff H is a simple graph. Consider any edge (u, v) that has multiplicity more than one in G''. It must be that $(u, v) \in E''$ since G' is a simple graph. Assume w.l.o.g. that $u \in P_1$ and $v \in Q_1$. Let $X \subset P_1$ and $Y \subset Q_1$ respectively denote the set of vertices adjacent to v and u in G''. Using condition (d) on the graph G_r, we know that

8

$$|E'' \cap (P_1 \times Q_1)| \geq \left(\frac{5d}{2}\right)\left(\frac{4d}{5}+1\right) - (d+1) > 2d^2.$$

Since $|X| < d$ and $|Y| < d$, it follows that there must exist an edge $(u', v') \in E'' \cap (P_1 \times Q_1)$ such that $u' \notin X$ and $v' \notin Y$. We can thus replace edges $\{(u, v), (u', v')\}$ in E'' with edges $\{(u, v'), (u', v)\}$ without violating the d-regularity condition. It is easy to verify that the exchange reduces $\Phi(G'')$ by at least one, and that all edges involved in the exchange belong to the set E''. We can thus repeat this process until the graph G'' becomes simple, and hence a member of the family $\mathcal{G}(d)$. ∎

Adversary strategy: For each vertex $u \in P \cup \{t\}, Q \cup \{s\}$, the adversary $\mathcal{A}$ maintains a list $N(u)$ of vertices adjacent to u that have been so far revealed to the algorithm $\mathcal{D}$. Wlog we can assume that the algorithm $\mathcal{D}$ never queries a vertex u for which $|N(u)| = d$. At any step of the game, we denote by G_r the graph formed by the edges revealed thus far. We say the game is in *evasive* mode if the graph G_r satisfies the condition (a) through (d) of Lemma 9, and is in *non-evasive* mode otherwise. Note that the game always starts in the evasive mode, and then switches to non-evasive mode.

When the game is in the evasive mode, in response to a query $\mathcal{Q}(u)$ by $\mathcal{D}$ for some free vertex $u \in P_i$ ($i \in \{1,2\}$), $\mathcal{A}$ returns an arbitrary free vertex $v \in Q_i$ such that $v \notin N(u)$. The adversary then adds v to $N(u)$ and u to $N(v)$. Similarly, when $\mathcal{D}$ asks a query $\mathcal{Q}(u)$ for some free vertex $u \in Q_i$ ($i \in \{1,2\}$), $\mathcal{A}$ returns an arbitrary free vertex $v \in P_i$ such that $v \notin N(u)$. It then adds v to $N(u)$ and u to $N(v)$ as above.

As the game transitions from evasive to non-evasive mode, Lemma 9 ensures existence of a canonical graph $G \in \mathcal{G}(d)$ that contains the graph revealed by the adversary thus far as a subgraph. The adversary answers all subsequent queries by $\mathcal{D}$ in a manner that is consistent with the edges of G. The lemma below shows that the simple adversary strategy above forces $\Omega(d^2)$ queries before the evasive mode terminates.

Lemma 10 *The algorithm makes $\Omega(d^2)$ queries before the game enters non-evasive mode.*

Proof: The adversary strategy ensures that conditions (a) through (c) in Lemma 9 are maintained at all times as long as the game is in the evasive mode. So we consider the first time that condition (d) is violated. Since each query answered by the adversary in the evasive mode contributes 1 to the degree of exactly one vertex in $Q_1 \cup P_2$, $\mathcal{A}$ always answers at least $\Omega(d^2)$ queries before the number of vertices with degree less than $\frac{d}{5}$ falls below $\frac{5d}{2}$ in either Q_1 or P_2. The lemma follows. ∎

Since A can not discover an edge in M' until the game enters the non-evasive mode, we obtain the desired lower bound of $\Omega(d^2)$.

5 An $\Omega(n \log n)$ High Probability Lower Bound

In this section we give a lower bound on the running time of any randomized algorithm for finding a perfect matching in d-regular bipartite multigraphs, even with edge multiplicities bounded above by $d/2$:

Theorem 11 *Let A be any randomized algorithm that finds a matching in a d-regular bipartite multigraph with n nodes and edge multiplicities bounded above by $d/2$. Then there exists a family of graphs for which A probes at least $(\gamma/64)n \ln n$ locations in the input adjacency arrays with probability at least $n^{-\gamma}$.*

We first reiterate that even though the algorithm obtained in section 2 is stated for simple graphs, the same runtime analysis applies for multigraphs as long as edge multiplicities are bounded above by $d/2$. The restriction on maximum edge multiplicity is necessary to ensure that SAMPLE-OUT-EDGE takes $O(1)$ time in expectation. In this section we show that every algorithm that finds a matching in a d-regular multigraph (even with edge multiplicities bounded above by $d/2$) probes at least $(\gamma/64)n \ln n$ locations in the input

adjacency arrays with probability at least $n^{-\gamma}$ (on some fixed family of distributions). Thus, the lower and upper bounds are tight in this particular case. The lower bound instances use $d = \Theta(n)$.

It is also interesting to contrast Theorem 11 with the result of [17], which shows that sampling a constant number of edges incident to every vertex of a *complete* bipartite graph yields a subgraph that contains a perfect matching with high probability, i.e. the sampling complexity is $O(n)$ even if a high probability result is desired. The lower bound on the randomized algorithm is not as comprehensive as the deterministic lower bound: it holds only for very specific values of d (specifically, $d = \Theta(n)$), it bounds the "with high probability"-running time as opposed to the expected running time, and it works for multi-graphs. Obtaining tight upper and lower bounds for the entire range of parameters and for expected running time remains an interesting open problem.

We first introduce the following problem, which we will refer to as BIPARTITE-DISCOVERY(d).

Definition 12 *(BIPARTITE-DISCOVERY(d)) Let $G = (P, Q, E)$ be a bipartite multigraph with $|P| = 4d$ and $|Q| = d$. The set of edges $E(G)$ is constructed as follows. For each $u \in Q$ choose d neighbors in P uniformly at random with replacement. A node $u^* \in Q$ is then marked as special, and edges incident to the special node are referred to as special. The graph G is presented in adjacency array format with edges appearing in random order in adjacency lists. When an algorithm A queries a neighbor of a vertex $u \in Q$ or $v \in P$, an incident edge is returned uniformly at random among the yet undiscovered edges. The location of the edge in the adjacency arrays of both endpoints is revealed to A, i.e. it is no longer considered undiscovered when any of its endpoints is queried. The algorithm is not allowed to query the special node directly.*

Algorithm A solves BIPARTITE-DISCOVERY(d) if it finds an edge incident to the special node. The cost of A is defined as the number of queries that it makes before discovering an edge to the special node.

We show the following:

Lemma 13 *Any algorithm that solves BIPARTITE-DISCOVERY(d) makes at least $(\gamma/2)d \ln d$ queries with probability at least $d^{-\gamma}$ for any $\gamma > 0$.*

Proof: Suppose that the algorithm has discovered J edges of G. Then the probability of the next query not yielding a special edge is at least $\frac{d^2 - d - J}{d^2 - J}$, independent of the actual set of edges of G that have already been discovered. Hence, the probability of not discovering a special edge after $J < d^2/3$ queries is at least

$$\prod_{j=0}^{J} \frac{d^2 - d - j}{d^2 - j} \geq \prod_{j=0}^{J} \frac{2d^2/3 - d}{2d^2/3} \geq e^{-2J/d}$$

for sufficiently large d. Hence, we have that the probability of not finding a special edge after $(\gamma/2)d \ln d$ queries is at least $d^{-\gamma}$. ∎

We now give a reduction from BIPARTITE-DISCOVERY(d) to the problem of finding a matching in a regular bipartite multigraph with edge multiplicities bounded by $d/2$:

Proof of Theorem 11:

Let A be an algorithm that finds a matching in a regular bipartite multigraph with edge multiplicities bounded above by $d/2$ and makes fewer than $(\gamma/64)n \ln n$ queries with probability at least $1 - n^{-\gamma}$ on every such graph. We will give an algorithm A' that solves BIPARTITE-DISCOVERY(d) and makes fewer than $(\gamma/2)d \ln d$ queries with probability strictly larger than $1 - d^{-\gamma}$.

Consider an instance $G = (P, Q, E)$ of BIPARTITE-DISCOVERY(d). Algorithm A' first checks if the degrees of all nodes in P are smaller than $d/2$. If there exists a node with degree strictly larger than $d/2$, A' queries all edges of all vertices in P and thus finds a special edge in at most $2d^2$ queries. Note that since the expected degree of vertices in P is $d/4$, the probability of this happening is at most e^{-d} for sufficiently large d by an application of the Chernoff bound with a union bound over vertices of P.

Now suppose that degrees of all nodes in P are at most $d/2$. A' adds a set of $3d$ vertices Q' to the Q side of the partition of G and connects nodes in Q' to nodes in P to ensure that the degree of every vertex in P and Q' is exactly d (it can be shown using an argument similar to the one in the proof of Lemma 9 that this can be done without introducing double edges). Denote the resulting regular multigraph by $G^+ = (P, Q \cup Q', E \cup E')$. Note that G^+ has $4d$ vertices in each part, and one vertex in the Q part of the bipartition is marked special together with its d adjacent edges. Now A' constructs the final graph by putting together two copies of G^+. In particular, we denote by G^- a mirrored copy of G^+, i.e. $G^- = (Q \cup Q', P, E \cup E')$, and finally denote by G^* the graph obtained by taking the union of G^+ and G^-, removing the two special nodes and identifying special edges in G^+ with special edges in G^-. Note that any matching in G^* contains a special edge, so algorithm A necessarily finds a special edge. Note that a query to an adjacency list in G^+ or G^- can be answered by doing at most one query on G. The number of vertices in each bipartition of G^* is $8d - 1$ and the degree of each node is d.

By assumption, algorithm A does not make more than $(\gamma'/64)n \ln n$ queries with probability at least $1 - n^{-\gamma'}$ for any $\gamma' > 0$. Setting $n = 8d - 1$ and $\gamma' = 2\gamma$, we get that A does not need more than $(2\gamma/64)8d \ln(8d) \le (\gamma/2)d \ln d$ queries with probability at least $1 - (8d - 1)^{-2\gamma} \ge 1 - d^{-2\gamma}$ for sufficiently large d. Hence, we conclude that A' probes at most $(\gamma/2)d \ln d$ locations with probability at least $1 - d^{-2\gamma} + e^{-d} > 1 - d^{-\gamma}$, contradicting Lemma 13. $\blacksquare$

6 Acknowledgements

References

[1] G. Aggarwal, R. Motwani, D. Shah, and A. Zhu. Switch scheduling via randomized edge coloring. *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 502 – 512, 2003.

[2] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A*, 5:147–151, 1946.

[3] B. Bollobas. *Modern graph theory*. Springer, 1998.

[4] C. Chang, D. Lee, and Y. Jou. Load balanced birkhoff-von neumann switches, part i: one-stage buffering. *Computer Communications*, 25(6):611–622, 2002.

[5] R. Cole and J.E. Hopcroft. On edge coloring bipartite graphs. *SIAM J. Comput.*, 11(3):540–546, 1982.

[6] R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21(1):5–12, 2001.

[7] H.N. Gabow and O. Kariv. Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM J. Comput.*, 11(1):117–129, 1982.

[8] A. Goel, M. Kapralov, and S. Khanna. Perfect matchings in $\tilde{O}(n^{1.5})$ time in regular bipartite graphs. *http://arxiv.org/abs/0902.1617v2*, 2009.

[9] A. Goel, M. Kapralov, and S. Khanna. Perfect matchings via uniform sampling in regular bipartite graphs. *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 11–17, 2009.

[10] J.E. Hopcroft and R.M. Karp. An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[11] Ajai Kapoor and Romeo Rizzi. Edge-coloring bipartite graphs. *J. Algorithms*, 34(2):390–396, 2000.

[12] D. König. Uber graphen und ihre anwendung auf determinententheorie und mengenlehre. *Math. Annalen*, 77:453–465, 1916.

[13] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[14] Romeo Rizzi. Finding 1-factors in bipartite regular graphs and edge-coloring bipartite graphs. *SIAM J. Discrete Math.*, 15(3):283–288, 2002.

[15] A. Schrijver. Bipartite edge coloring in $O(\Delta m)$ time. *SIAM J. on Comput.*, 28:841–846, 1999.

[16] J. von Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. *Contributions to the optimal assignment problem to the Theory of Games*, 2:5–12, 1953.

[17] D. W. Walkup. Matchings in random regular bipartite graphs. *Discrete Math*, 31:59–64, 1980.