Electronic Colloquium on Computational Complexity, Report No. 208 (2015)

Perfect Bipartite Matching in Pseudo-Deterministic RNC

Shafi Goldwasser Ofer Grossman

December 24, 2015

Abstract

In this paper we present a pseudo-deterministic RNC algorithm for finding perfect matchings in bipartite graphs. Specifically, our algorithm is a randomized parallel algorithm which uses poly(n) processors, poly$(\log n)$ depth, poly$(\log n)$ random bits, and outputs for each bipartite input graph a **unique** perfect matching with high probability. That is, it returns the same matching for almost all random seeds.

Our work improves upon different aspects of prior work. The celebrated works of Karp, Uval, Wigderson [13] and Mulmuley, Vazirani, Vazirani [15] which find perfect matchings in RNC produce different matchings on different executions. The recent work of Fenner, Gurjar, and Thierauf [7] shows a deterministic parallel algorithm for bipartite perfect matching but requires $2^{\text{poly}(\log n)}$ (quasi-polynomially many) processors, proving that bipartite matching is in quasi-NC. Our algorithm is the first algorithm to return unique perfect matchings with only polynomially many processors.

As an immediate consequence we also find a pseudo-deterministic RNC algorithm for depth first search (DFS).

1 Introduction

Computing a maximum matching in a graph is a paradigm-setting algorithmic problem whose understanding has paved the way to formulating some of the central themes of theoretical computer science.

In particular, Edmonds [6] proposed the definition of tractable polynomial-time solvable problems versus intractable non-polynomial time solvable problems following the study of the graph matching problem versus the graph clique problem. In the context of parallel algorithms, computing a maximum (or possibly perfect) matching is the problem standing at the center of the RNC versus NC question. It is known both in the general and in the bipartite case to be solvable by randomized RNC algorithms but it is unknown if deterministic NC algorithm exist.

One can distinguish between the *decision* version of the perfect matching problem, which asks whether a perfect matching exists, and the *search* version, which asks to return a perfect matching if any exist. The decision problem is equivalent to testing whether the determinant of the Tutte matrix of the graph (or a simplified version of it in the bipartite case [6]) is identically 0. Lovász [14] showed that using randomization, determining the decision problem is reducible to testing that certain integer matrices are non-singular. Since the latter can be done in NC, a a Monte Carlo RNC algorithm for the decision problem follows.

The search version was subsequently shown to be in RNC by Karp, Upfal, and Wigderson [13] via a Monte-Carlo algorithm and by Karloff [12] via a Las-Vegas algorithm. The next breakthrough

ISSN 1433-8092

was the RNC algorithm of Mulmuley, Vazirani, and Vazirani [15]. They assigned random weights from the set $\{1, 2, \ldots, 2|E(G)|\}$ to the edges of the graph and proved the elegant *isolation lemma* which states that with high probability such a random assignment induces (isolates) a unique min-weight perfect matching (if at least one perfect matching exists). The problem of finding the (unique) minimum weight matching can be determined now in parallel by assigning each edge to a different processor whose task is to determine if the edge participates in the unique min-weight matching. We emphasize that although for each isolating weight assignment to the edges, a unique perfect matching will be found, different matching will be found for the same graph for different randomized choices of the weight assignments.

Quite recently, a significant step forward has been made by Fenner, Gurjar, and Thierauf [7] who showed how to remove randomization and obtain a quasi-NC algorithm for both the decision and search perfect matching problems in bipartite graphs. That is, they obtain a deterministic $\mathrm{poly}(\log n)$ depth algorithm which uses quasi-polynomially many processors. An important idea in [7], which served as an inspiration to our work, was to show the existence of weight assignments for which all small (suitably defined) even cycles do not appear in the graph corresponding to the union of all minimal perfect matchings.

In a different line of work, Gat and Goldwasser [8] studied the class of search problems which can be solved by *pseudo-deterministic* polynomial time algorithms – probabilistic (polynomial-time) algorithms for search problems that, with high probability, produce a fixed output for each given input except with negligible error probability. That is, they return the same output for all but few of the possible random seeds. Algorithms that satisfy the aforementioned condition are named pseudo-deterministic, as they essentially offer the same functionality as deterministic algorithms. They are of particular relevance in a distributed or parallel setting since if two parties invoke the algorithm on the same input at different locations (and using different sources of random bits), they are still guaranteed to obtain the same result, except very rarely. The lack of such agreement is a drawback of standard probabilistic search algorithms.

Efficient pseudo-deterministic algorithms have been shown in [5, 8, 11] for several search problems for which no efficient deterministic algorithms are known. These problems include number theoretic search problems (such as finding quadratic non-residues and primitive roots in finite fields), and finding variable settings for polynomial non-identically-zero testing. The existence of pseudo-deterministic algorithms was also studied in the sublinear setting in [9] where separations between deterministic, randomized and pseudo-deterministic sublinear algorithms are shown in accordance with the (asymptotic) number of queries they must require.

A question studied in [9] but which remains largely unresolved is how the class of pseudo-deterministic polynomial time solvable problems relates to various search variants of probabilistic polynomial time. Similarly, an intriguing open question is to determine how the class of pseudo-deterministic parallel algorithms (i.e pseudo-deterministic RNC) relates to RNC (strict containment?) and to NC.

In particular, the following intriguing question emerges. In lieu of deterministic NC algorithms for both the decision and search versions of perfect matching in a graph, can we design a pseudo-deterministic RNC algorithm which finds a perfect matching in a graph? Namely, can we design an RNC algorithm which for each input graph returns the **same (canonical) perfect matching** with high probability?

Our main result affirmatively settles this problem for the case of bipartite graphs.

1.1 Our Results

We present an RNC algorithm for the bipartite perfect matching search problem, which on input a bipartite graph G outputs a unique (canonical) perfect matching with high probability, if at least one perfect matching exists.

Theorem 1.1 (Main Theorem)**.** *There exists a pseudo-deterministic RNC algorithm that, given a bipartite graph G, returns a perfect matching of G, or states that none exist.*

On a bipartite graph G, the algorithm uses polynomially many processors, runs in $\text{poly}(\log n)$ time, and uses $\text{poly}(\log n)$ random bits. This is the first pseudo-deterministic RNC algorithm for bipartite perfect matching. All previous RNC algorithms would output different matchings on different executions whereas our algorithm outputs the same perfect matching on all but a negligible fraction of executions.

Aggarwal, Anderson, and Kao [1] present an RNC algorithm for constructing a depth first search tree for directed graphs. Their algorithm's only use of randomization is to solve bipartite min-weight perfect matching as a subroutine. We can adapt our algorithm to find a unique min-weight perfect matching. Hence, our results imply a pseudo-deterministic RNC algorithm for computing depth first search (DFS) in general directed graphs.

We now present the basic ideas of our solution.

1.2 High Level Ideas of the Solution

Let us assume G is a given bipartite graph and w is a given weight assignment to the edges of G (we will later detail how to construct an appropriate w in NC). We first show how, using randomization, to construct the union of all min-weight perfect matchings of G with respect to w (deterministically, this is not known to be possible):

Lemma (Union of min-weight perfect matchings)**.** *Let $G(V,E)$ be a bipartite graph with weight function w. Let E_1 be the union of all min-weight perfect matchings in G. There exists an RNC algorithm for finding the set E_1.*

The Lemma appears in Section 3 as Lemma 3.3.

We compute the union of min-weight perfect matchings by creating a process, for each edge e_i, whose goal is to determine whether e_i participates in some min-weight perfect matching. To this end, the process creates a new weight assignment w_i which lowers the weight of e_i by a small amount. The new weight assignment is picked so that if e_i is in some w-minimal perfect matching, then e_i must be in all w_i-minimal perfect matchings; whereas if e_i is not in any w-minimal perfect matching, then it must be in none of the w_i-minimal perfect matchings. By finding a (not necessarily unique) w_i-minimal perfect matching (which can be done in RNC using techniques in [15], and is the only randomized step of our algorithm) and checking whether e_i participates in the matching, we can determine whether e_i is in the union of min-weight matchings. We can then return the union of all e_i which are in some min-weight matching.

Constructing the union of min-weight perfect matchings will be an important step in our solution, as it will allow us to prune the graph (removing the edges which participate in no min-weight matching) while maintaining the property that the graph has a perfect matching.

To apply the above procedure so as to effectively reduce the size of the graph, we deterministically construct a set of weight assignments with the property that constructing the union of all min-weight perfect matchings in G with respect to these assignments (by going through the weight assignments in sequence and removing edges in each iteration) leaves G with many vertices of degree at most 2. We can then contract all vertices of degree 2 with their neighbors to get a smaller graph in which we recursively run our algorithm until we remain with only a constant number of vertices. At this point, we can deterministically compute a unique perfect matching in $O(1)$ time. We note that although performing the contraction procedure in NC takes some care, once done properly it is easy to extend a perfect matching in the contracted graph to the original graph.

The construction of weight assignments with the above property proceeds as follows. By a theorem in [2], we learn that if the girth (length of the shortest cycle) of G is at least $4 \log n$, then at least $\frac{1}{10}$ of the vertices have degree at most 2. Therefore, if our weight assignments $w_1, \ldots, w_t$ can make all small cycles disappear (i.e., when we construct the union of w_1-minimal matchings, then construct the union of w_2-minimal matchings on this new graph, etc., then at the end are left with a graph with no small cycles), we will be able to reduce our problem to a smaller graph, contract vertices of degree up to 2, and recurse. As shown in [7], for any weight assignment w every even cycle with nonzero circulation (the sum of the weights of the odd edges of a cycle minus the sum of the weights of the even edges of the cycle) disappears when we look at the union of w-minimal perfect matchings. We thus need to show how to construct a set of weight functions which will ensure that all small (containing fewer than $4 \log n$ vertices) cycles will have nonzero circulation (with respect to at least one of the weight functions).

Lemma (Non-Zero Circulation for Small Cycles)**.** *Let G be a bipartite graph on n vertices. Then, for any number s, one can construct in NC a set of $O(s \log n)$ weight assignments with weights bounded by $O(s \log n)$ such that every cycle of length up to s has nonzero circulation for at least one of the weight assignments.*

This Lemma appears in Section 3 as Lemma 3.2.

To prove this Lemma we first note that if a cycle of length up to s has circulation 0, then the sum of the weights of the odd edges equals the sum of the weights of the even edges. That means that there are two subsets of $E(G)$ of size up to $\frac{s}{2}$ that have the same sum of weights. If we could construct a weight function such that no two sets of size up to $\frac{s}{2}$ have the same sum of weights, we will have proved the Lemma. Unfortunately, when we construct such weights, the weights are of quasi-polynomial size:

Lemma (Uniquifying Assignment for small sets)**.** *Let S be a set with $|S| = n$. For any number k, one can construct (in NC) a weight assignment $w : S \to \mathbb{Z}$ with weights bounded by $2^{O(k \log n)}$ such that no two distinct subsets $S_1, S_2 \subset S$ satisfying $|S_1|, |S_2| \leq k$ have the same sum of weights.*

This Lemma appears in Section 3 as Lemma 3.1.

The idea of the construction in the Lemma's proof is to let the mth element have weight

$$w(m) = p^{2k} m + p^{2(k-1)} [m^2]_p + p^{2(k-2)} [m^3]_p + \cdots + k^0 p^0 [m^{k+1}]_p$$

where $[x]_p$ denotes the number between 1 and p which is equal to x modulo p and where p is an arbitrary prime greater than n^2. Then, we can show that given the sum of the weights of k elements labeled m_1 through m_k, we can retrieve the sums $\sum_{i=1}^{k} m_i^j$, for all $1 \leq j \leq k$. Using these sums, we can use Newton's identities to find the minimal polynomial with roots $m_1, m_2, \ldots, m_k$, which uniquely determines the set of elements. Thus, no two distinct subsets of size up to k can have the same sum of weights.

We note that the weights in the Lemma (when $k = 2\log n$) are of quasi-polynomial size, but we want weight functions of polynomial size. To fix this, we note that every cycle C with nonzero circulation in w will have nonzero circulation modulo some small number. Therefore, the weight functions $\{w \pmod{j} : 2 \le j \le t\}$, for $t = O(k \log n)$, are a family of weight functions such that every small cycle will have nonzero circulation modulo at least one of the weight functions..

We now set $s = 4\log n$, and (not in parallel) for each weight assignment $w \pmod{j}$ of the $O(s \log n)$ weight assignments, we update our graph by constructing the union of min-weight matchings with respect to $w \pmod{j}$. When we are done, we have a graph of high girth, so we can contract many vertices of degree up to 2 (recall that a graph of girth greater than $4\log n$ has at least one tenth of its vertices of degree up to 2). We now have a smaller graph, and we recurse, completing the proof's outline.

We note that our solution was inspired by some ideas of Fenner, Gurjar, and Thierauf [7]. They exhibit a set of weight assignments which give non-zero circulation to all small cycles, and consider the union of min-weight matchings and the effect of contracting degree 2 nodes on the size of the graph. However, these ideas are used only as part of their analysis. Their algorithm to find a perfect bipartite graph is entirely different. We remark that it is not known how to construct the union of min-weight matchings deterministically.

2 Background and Preliminaries

We begin with a formal definition of Pseudo-deterministic:

Definition 2.1 (Pseudo-deterministic)**.** An algorithm A for a relation R is *pseudo-deterministic* if there exists some function s such that A, when executed on input x, outputs $s(x)$ with high probability, and s satisfies $(x, s(x)) \in R$.

To contrast the definition with that of a standard randomized algorithm, we note that a standard randomized algorithm may output a different y on different executions, as long as $(x, y) \in R$.

We call an algorithm *pseudo-deterministic RNC* if it is both in RNC, and is pseudo-deterministic.

We now present some lemmas from previous work.

Lemma 2.2 (Theorem 2 in [15])**.** *Given a graph G with a weight function $w : E \to \mathbb{Z}$, with polynomially bounded weights, it is possible to construct a w-minimal perfect matching of G in RNC.*

Definition 2.3 (Circulation)**.** Let $G(V, E)$ be a graph with weight function w. Let C be a cycle in the graph. The *circulation* $c_w(C)$ of an even length cycle $C = (v_1, v_2, \ldots, v_k)$ is defined as the alternating sum of the edge weights of C,

$$c_w(C) = |w(v_1, v_2) - w(v_2, v_3) + w(v_3, v_4) - \cdots - w(v_k, v_1)|.$$

Circulation has been used for an NC algorithm for perfect planer bipartite matching [4] and for a quasi-NC algorithm for bipartite matching [7].

Lemma 2.4 (Lemma 3.4 in [7])**.** *Let G be a bipartite graph. Let w be a weight function such that the cycles $C_1, C_2, \ldots, C_n$ have nonzero circulations. Then the graph G_1 obtained by taking the union of all min-weight perfect matchings on G will not have any of the cycles $C_1, C_2, \ldots, C_n$.*

The proof in [7] is somewhat complicated. We present a simpler proof found by Anup Rao, Amir Shpilka, and Avi Wigderson:

Proof. Let G' be the multigraph obtained by taking the disjoint union of all min-weight perfect matchings (i.e., if an edge e appears in k min-weight perfect matchings of G, then G' contains k copies of e).

Suppose that there exists a cycle C of nonzero circulation in G'. Then suppose without loss of generality that the sum of weights of the odd edges of C is larger than the sum of the weights of the even edges. Then we remove the odd edges of C from G', and add copies of the even edges of C. Call this new graph G''.

We note that G' is a regular graph since it is the disjoint union of matchings, and matchings are regular graphs of degree 1. We also see that every vertex has the same degree in G'' as in G'. Hence, G'' is regular.

We know that every regular bipartite graph is a union of perfect matchings (to prove this, we can induct on the degree. A regular bipartite graph must satisfy Hall's condition. Therefore, it has a perfect matching, which we can remove. We now obtain a new regular graph of lower degree, which by induction must be a union of perfect matchings).

If we let M be the minimal weight of a matching in G, and we suppose G has d min-weight matchings, then the sum of the weights of edges of G' is Md. However, the total weight of G'' is lower than the total weight of G'. We know that G'' is regular of degree d, and therefore is a union of perfect matchings. If we decompose G'' into d perfect matchings, it is impossible that they all have weight at least M. Therefore, G'' has a matching of weight less than M, which corresponds to a matching of weight less than M in G. This contradicts the assumption that M is the minimal weight of a matching in G. $\square$

The following lemma originates in [2], and is presented in this form in [7].

Lemma 2.5 (Corollary 3.6 in [7]). *Let H be a graph with girth (length of shortest cycle) $g \geq 4\log n$. Then H has average degree < 2.5. In particular, at least $\frac{1}{10}$ (a constant fraction) of the vertices have degree at most 2.*

3 Key Lemmas

Recall that in [15] a weight assignment is chosen at random such that with high probability there is a unique min-weight perfect matching. Our goal will be to deterministically construct weight assignments with similar properties. Specifically, we will construct weight assignments which give nonzero circulation to small cycles.

Lemma 3.1 (Uniquifying assignment for small sets.). *Let S be a set with $|S| = n$. For any number k, one can construct (in NC) a weight assignment $w : S \to \mathbb{Z}$ with weights bounded by $2^{O(k\log n)}$ such that no two distinct subsets $S_1, S_2 \subset S$ satisfying $|S_1|, |S_2| \leq k$ have the same sum of weights.*

We can think about the Lemma as an assignment which isolates all small subsets of S. We will later use this Lemma to construct a weight assignment for the graph G.

Proof. Let $S = \{s_1, \dots, s_n\}$. Consider the following weight assignment, where we write $w(m)$ as shorthand for $w(s_m)$:

$$w(m) = p^{2k}m + p^{2(k-1)}[m^2]_p + p^{2(k-2)}[m^3]_p + \cdots + k^0p^0[m^{k+1}]_p$$

where $[x]_p$ denotes the number between 1 and p which is equal to x modulo p, and where p is an arbitrary prime greater than n^2. We can find such a prime by having n^2 processes each check a different number between n^2 and $2n^2$. Each of these processes initiate $2n^2$ processes which each test divisibility by an integer up to $2n^2$. (Note that this has no implications regarding generating primes in NC since our input is of size n instead of $\log n$).

Suppose there exist two distinct subsets of size up to k with equal sums of weights. We can add zeroes to both subsets such that the sizes of the sets are exactly k. Suppose that the sums of the weights of two subsets $A = \{a_1, a_2, \ldots, a_k\}$ and $B = \{b_1, b_2, \ldots, b_k\}$ are the same. We note that this would imply that the sums of the $p^{2k}m$ terms for both A and B must be equal because the $p^{2k}m$ term is much larger than all other terms (or it is 0). Similarly, the sums of the $p^{2(k-1)}[m^2]_p$ terms must equal and so on for $p^{2(k-i)}[m^{i+1}]_p$ for all i. Therefore, we have the following equivalences modulo p :

$$
\begin{gathered}
a_1 + a_2 + \cdots + a_k \equiv b_1 + b_2 + \cdots + b_k \pmod{p} \\
a_1^2 + a_2^2 + \cdots + a_k^2 \equiv b_1^2 + b_2^2 + \cdots + b_k^2 \pmod{p} \\
\cdots \\
a_1^{k+1} + a_2^{k+1} + \cdots + a_k^{k+1} \equiv b_1^{k+1} + b_2^{k+1} + \cdots + b_k^{k+1} \pmod{p}.
\end{gathered}
$$

We claim that this implies that $A = B$. We note that if $a_i \equiv b_j$ modulo p, then $a_i = b_j$ because p is larger than n^2 which is the maximal size of a_i or b_j. Therefore, it will suffice to show that the set A and the set B are equivalent in $\mathbb{F}_p$.

Newton's identities, given the sums of the ith powers of the a_j for i between 1 and k, uniquely determine the values of the fundamental symmetric polynomials in the a_j. Therefore, Newton's identities also uniquely determine the minimal polynomial which has as roots all of the a_j (with multiplicity). We know that this polynomial will be of degree k and therefore since the b_j share this polynomial, the set of the a_i and the set of the b_j must be equal (they are both the set of roots of the same polynomial), completing the proof that the weight assignment has no two subsets of size up to k with the same sum of weights.

We note that the weights are bounded by $p^{2k+2} = 2^{O(k \log n)}$. □

If a cycle of length s has circulation 0, then there are two distinct subsets of size $s/2$ that have the same sum of weights (namely, the sum of the weights of the cycle's odd edges equals the sum of the weights of the cycle's even edges). Therefore, the above Lemma implies that we can construct weight assignment for G with weights bounded by $2^{O(s \log n)}$ such that all cycles of length up to s have nonzero circulation.

Lemma 3.2 (Nonzero circulation for small cycles.)**.** *Let G be a bipartite graph on n vertices. Then for any number s, one can construct in NC a set of $O(s \log n)$ weight assignments with weights bounded by $O(s \log n)$ such that every cycle of length up to s has nonzero circulation for at least one of the weight assignments.*

We would like to point out the differences between this Lemma and Lemma 2.3 of [7] (which originates in [3]). Lemma 2.3 of [7] proves that for any number t, one can construct a set of $O(n^2 t)$ weight assignments with weights bounded by $O(n^2 t)$, such that for any set of t cycles, one of the weight assignments gives nonzero circulation to each of the t cycles.

Since the number of length s cycles is at most $\frac{n!}{(n-s)!} \leq n^s$, their theorem implies a set of $O(n^{s+2})$ weight assignments with weights bounded by $O(n^{s+2})$ (note that this is quasi-polynomial for $s = 4\log n$, which will be our setting of parameters) such that at least one of the weight assignments gives non-zero circulation to all small cycles.

Proof. We begin with our weight assignment from Lemma 3.1 with $k = \lfloor s/2 \rfloor$. We consider the weight assignment modulo small numbers, i.e., the weight functions $\{w \pmod{j} | 2 \leq j \leq t\}$ for some appropriately chosen t. (The idea here is to pick t so that if a cycle C has nonzero circulation $c_w(C)$, then there must exist some $j \leq t$ such that $c_w(C) \not\equiv 0 \pmod{j}$.)

We note that if the lemma does not hold then there exists a cycle C of nonzero circulation such that $c_w(C) \equiv 0 \pmod{j}$, for all j between 1 and t. Therefore,

$$\operatorname{lcm}(2, 3, \ldots, t) | c_w(C_i).$$

The right is bounded above by $2^{O(s \log n)}$. In [16] we learn that $\operatorname{lcm}(2, 3, \ldots, t) > 2^t$ for sufficiently large t, so letting $t = O(s \log n)$ makes it so a cycle with nonzero circulation with respect to w is guaranteed to have nonzero circulation with respect to $w \pmod{j}$ for some $2 \leq j \leq t$.

Therefore, we have $O(s \log n)$ total weight assignments with weights bounded by $O(s \log n)$ such that every cycle of length up to s has nonzero circulation in at least one weight assignment. □

The following lemma shows that in *RNC* we can construct the union of min-weight perfect matchings of a graph G with a weight assignment w.

Lemma 3.3 (Union of min-weight perfect matchings)**.** *Let $G(V, E)$ be a bipartite graph with weight function w. Let E_1 be the union of all min-weight perfect matchings in G. There exists an RNC algorithm for finding the set E_1.*

The idea behind the proof is that for each edge e_i, we run a process whose goal is to tell whether e_i is part of a min-weight perfect matching. To do so, the process creates a new weight function which lowers e_i so that if e_i was in a min-weight perfect matching, under the new weight assignment e_i is in *every* min-weight perfect matching (but if e_i was not in any min-weight perfect matching, it should still not be in any min-weight matching). Then, we use Lemma 2.2 to find a min-weight perfect matching, and we check if e_i is in the matching. e_i will be in the matching if and only if it is part of a min-weight matching with respect to the original weight function.

Proof. For each edge $e_i \in E$, consider the weight function w_i defined by

$$w_i(e_j) = \begin{cases} 2w(e_j) - 1 & \text{if } i = j \\ 2w(e_j) & \text{if } i \neq j. \end{cases}$$

Suppose that M is the minimal weight for a matching with respect to w. Then with respect to w_i, the min-weight matching will have weight $2M$ if e_i is in no w-minimal matching. Otherwise, the min-weight matching will have weight $2M - 1$. By finding a w_i-minimal perfect matching (which we can do in *RNC* by Lemma 2.2) and checking whether e_i participates in the matching, we can determine whether e_i is in a w-minimal matching.

Note that this is highly parallelizable: we can run the above for each edge in parallel. Then, we return the set of all e_i which are part of some w-minimal matching. □

4 The Algorithm

We now put everything together to construct an algorithm:

Perfect-Matching(G)

1 If $|E(G)| \leq 100$:

2 Find and return a perfect matching of G using brute force.

3 Let $\{w_1, \ldots, w_t\}$ be the set of weight assignments defined in Lemma 3.2 with $s = 4 \log n$.

4 Let $G_0 = G$.

5 For $i = 1, 2, \ldots, t$:

6 Let G_i be the union of w_i-minimal perfect matchings of G_{i-1} (use Lemma 3.3).

7 Contract vertices of degree up to 2 in G_t to create G' (see Appendix).

8 Let $M' =$ Perfect-Matching(G').

9 Extend the matching M' in G' to a matching M in G_t (see Appendix). Return M.

We first argue that the algorithm returns a perfect matching with high probability. To do so, we first note that since G_t and G have the same vertices, it is enough to find a perfect matching on G_t. It is therefore enough to show that G' has a perfect matching, and that in step 9 we can extend the perfect matching M' in G' to a perfect matching M in G_t. This requires analyzing the contraction step of step 7. The contraction procedure takes some care, but is generally uninteresting and non-central to our proof, so we explain it in the appendix for completion.

The main idea behind the contraction step is that if we contract both edges adjacent to a vertex of degree 2 and find a matching in the new contracted graph, it is easy to turn a perfect matching in the contracted graph to a perfect matching in the original graph. If v is the vertex of degree 2, and its two neighbors are u_1 and u_2, then once we contract the three vertices we can call the new vertex u'. A perfect matching in the contracted graph will have an edge adjacent to u'. That edge must either be of the form (v', u_1) or of the form (v', u_2). Suppose without loss of generality that the edge is (v', u_1). Then we can add the edge (v, u_2) to the matching to form a perfect matching of M in G_t from the matching M' of G. Doing this for multiple vertices in parallel leads to some complications which we elaborate on in the appendix.

Note that we can amplify the success probability of step 6 so that the probability of failure is at most $\frac{1}{n^4}$. Since the step gets executed a total of $O(t \log n) < O(n^3)$ times (t times on each of the $O(\log(n))$ steps of the recursion), by the union bound the probability that step 6 ever fails is at most $\frac{1}{O(n)}$.

We now argue the algorithm is pseudo-deterministic. We note that randomization is only used in step 6 to construct the union of min-weight matchings. We use the randomization in the following context: given a weight assignment on a graph, construct the union of min-weight perfect matchings of the graph. Since this has a unique correct answer, correctness implies uniqueness. Therefore, our algorithm returns the same output with high probability, and is therefore pseudo-deterministic.

We will now show the algorithm lies in RNC. We note that step 2 takes $O(1)$ time and step 3 is in NC by Lemma 3.2. The number of iterations of the loop in step 5 is of length $O(\log(n)^2)$, by Lemma 3.2, and taking the union of min weight perfect matchings within the loop in step 6 is in RNC by Lemma 3.3. Note that if G_{i-1} has a perfect matching, then so does G_i, since G_i is a non-empty union of perfect matchings of G_{i-1}. Therefore, the loop iterations can be performed in RNC.

By Lemma 3.2 and Lemma 2.4, we see that after completing the loop, G_t has no cycles of length up to $4\log n$. By Lemma 2.5, in step 7 we contract a constant fraction of the vertices, so G' has a constant fraction of the number of vertices of G_t. Therefore, the number of recursive calls of step 8 is $\log n$.

This completes the algorithm's analysis, proving the following theorem:

Theorem 4.1. *There exists a pseudo-deterministic RNC algorithm that, given a bipartite graph G on n vertices, returns a perfect matching of G, or states that none exist.*

5 Using Fewer Random Bits

In this section, we will construct a pseudo-deterministic RNC algorithm for the bipartite perfect matching search problem which uses only $\text{poly}(\log n)$ random bits.

Our algorithm is based on our previous pseudo-deterministic RNC algorithm. We note that in our previous algorithm, the only use of randomization was to solve the following subproblem: given a graph G, a weight assignment w with polynomially bounded weights, and an edge e, output whether the edge e is part of a max-weight perfect matching (we note that we can talk about max-weight matchings even though earlier we talked about min-weight matchings because we can define a new weight function $w'(e_i) = \max_{e_j} w(e_j) - w(e_i)$ such that all w-minimal matchings are w'-maximal). We will show how to solve this with $\text{poly}(\log(n))$ random bits.

Let $M = \max_{x \in E} w(x)$. Consider the weight assignment w_e defined by

$$w_e(e') = \begin{cases} w(e') + (nM + 1) & \text{if } e' = e \\ w(e') & \text{otherwise.} \end{cases}$$

If there exists a perfect matching containing e, then all max-weight perfect matchings with respect to w_e will contain e. We note that if e is part of a max-weight matching with respect to w, then the max-weight matching with respect to w_e will have weight $W + nM + 1$ where W is the weight of the max-weight perfect matching with w. On the other hand, if e is not not a part of a max-weight matching with respect to w, then the max-weight matching with respect to w_e will have weight at most $(W-1) + (nM+1) = W + nM$. We will detect this difference by constructing a matrix and calculating its determinant.

Consider the following matrix, where the a_{ij} and z will be defined later.

$$A_e(i,j) = \begin{cases} z^{w_e(v_i,u_j)} a_{ij} & \text{if } (u_i, v_j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

We can set z to be much larger than the a_{ij}. For example, we can set $z = n^{n^2} \max_{i,j} |a_{ij}|^n$ (note that z has polynomially many bits, so we are still able to compute the determinant in NC). We can write the determinant as

$$\det(A_e) = \sum_{S \text{ a perfect matching in } G} sgn(S) z^{w_e(S)} \prod_{e \in S} a_e.$$

We see that because we picked z to be so large, each term where S a max-weight matching will be larger than the sum of all terms with non-max-weight matchings. Then, assuming that the terms with z^{W_e} (where W_e is the weight of a max-weight matching with respect to w_e) do not cancel, we can recover W_e from the determinant by finding the largest n such that $z^n \leq 2|\det(A_e)|$.

Now that we know W_e for every edge e, we can find the maximum of the set $\{W_e : e \in E\}$. The e_i such that W_{e_i} is maximal are the edges which are part of a max-weight perfect matching with w. This set is the union of max-weight perfect matchings, as we wished.

Therefore, it will suffice to find a_{ij} so that the terms with z^{W_e} do not cancel. This is exactly the same as finding a_{ij} such that the matrix

$$A'_e(i,j) = \begin{cases} a_{ij} & \text{if } (u_i, v_j) \text{ in a max-weight matching with } w_e \\ 0 & \text{otherwise} \end{cases}$$

has nonzero determinant.

In section 5 of [7], there is a randomized construction for the a_{ij} such that for each graph G' which has a perfect matching, the matrix

$$A'_{G'}(i,j) = \begin{cases} a_{ij} & \text{if } (u_i, v_j) \in E(G') \\ 0 & \text{otherwise} \end{cases}$$

has nonzero determinant with high probability. (Note that the a_{ij} do not depend on G'. This is important because we don't actually know that the set of edges in a max-weight matching with w_e.)

Because the construction uses $\mathrm{poly}(\log n)$ random bits and can achieve $\frac{1}{n^3}$ probability of failure, we can use the the same values of a_{ij} for all e, and by the union bound the probability that any failures occur is still small: at most $\frac{1}{n}$. Therefore, we can solve the subproblem using $\mathrm{poly}(\log(n))$ bits:

Theorem 5.1 (Main Theorem). *There exists a pseudo-deterministic RNC algorithm that, given a bipartite graph G on n vertices, returns a perfect matching of G, or states that none exist. The algorithm uses only* $\mathrm{poly}(\log(n))$ *random bits.*

6 Discussion

The above implies a pseudo-deterministic *RNC* algorithm for depth first search, another problem in *RNC* that is not known to be in *NC*. This result follows immediately from [1], where an *RNC* algorithm for DFS is presented, and the only use of randomization is in a subroutine for finding a min-weight perfect matching in a weighted bipartite graph.

We can adapt our algorithm to bipartite maximum matching. Given a bipartite graph G, we add edges such that we have a complete graph, and give weight 1 to each edge of G and weight 0 to each edge not in G. Now, we take the union of max-weight matchings. We know that any matching on this graph will have the same maximal weight (Lemma 3.2 in [7]). We now pseudo-deterministically find a perfect matching in this new graph, and restrict it to G to output a maximum matching.

The above also implies pseudo-deterministic *RNC* algorithms for some network flow problems such as max-flow approximation, which was shown in [17] to be *NC*-reducible to maximum bipartite matching.

Acknowledgments

Thanks to Anup Rao for communicating to us his proof with Amir Shpilka and Avi Wigderson of Lemma 2.4 (also Lemma 3.4 in [7]).

References

[1] Alok Aggarwal, Richard J Anderson, and M-Y Kao. Parallel depth-first search in general directed graphs. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 297–308. ACM, 1989.

[2] Noga Alon, Shlomo Hoory, and Nathan Linial. The moore bound for irregular graphs. *Graphs and Combinatorics*, 18(1):53–57, 2002.

[3] Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM Journal on Computing*, 24(5):1036–1050, 1995.

[4] Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47(3):737–757, 2010.

[5] Bart de Smit and Hendrik W Lenstra. Standard models for finite fields. *Handbook of finite fields, Discrete Mathematics and Its Applications. CRC Press, Hoboken, NJ*, 2013.

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

[7] Stephen Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-*nc*. *ECCC*, 9th November 2015. http://eccc.hpi-web.de/report/2015/177/.

[8] Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 18, page 136, 2011.

[9] Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 127–138. ACM, 2013.

[10] Dima Yu Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in nc. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 166–172. IEEE, 1987.

[11] Ofer Grossman. Finding primitive roots pseudo-deterministically. *ECCC*, 23rd December 2015. http://eccc.hpi-web.de/report/2015/207/.

[12] Howard J Karloff. A las vegas rnc algorithm for maximum matching. *Combinatorica*, 6(4):387–391, 1986.

[13] Richard M Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random nc. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 22–32. ACM, 1985.

[14] László Lovász. On determinants, matchings, and random algorithms. In *FCT*, volume 79, pages 565–574, 1979.

[15] Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1987.

[16] Mohan Nair. On chebyshev-type inequalities for primes. *American Mathematical Monthly*, pages 126–129, 1982.

[17] Maria Serna and Paul Spirakis. Tight rnc approximations to max flow. In *STACS 91*, pages 118–126. Springer, 1991.

Appendix: Contracting vertices of degree up to 2

As previously mentioned, contracting edges of degree up to 2 in step 7 of the algorithm and extending a matching in step 9 takes some care. We explain the details of the procedure below.

The main idea to note is if we contract both edges adjacent to a vertex of degree 2 and find a matching in the new contracted graph, it is easy to turn a perfect matching in the contracted graph to a perfect matching in the original graph. Doing this for multiple vertices in parallel leads to some complications which we elaborate on below.

Let G_t be a bipartite graph on n vertices which is a non-empty union of perfect matchings. Assume at least one-tenth of its vertices of degree at most 2. We will construct a new bipartite graph G' with at most $\frac{39n}{40}$ vertices which contains at least one perfect matching, and such that if we find a perfect matching in G', we can use it to find a perfect matching in G_t in NC.

First, we check if G_t has more than $\frac{n}{20}$ vertices of degree 1. If so, we can remove each such vertex and its neighbor, since we know that the edges adjacent to a vertex of degree 1 must be in the matching. This gives us the desired G'. We note that this G' will have at most $\frac{19n}{20}$ vertices, and that a perfect matching M' of G' can be turned into a perfect matching M of G_t by simply adding the edges of vertices of degree one in G_t to M'.

Otherwise (if fewer than $\frac{n}{20}$ vertices are of degree 1), since at least one-tenth of the vertices of G_t are of degree up to 2, we know that at least $\frac{n}{20}$ vertices are of degree exactly 2 (note that G_t is a non-empty union of perfect matchings, so it has no vertices of degree 0). We check each side of the bipartite graph, and pick the side with more vertices of degree 2. This side must have at least $\frac{n}{40}$ vertices of degree 2. We let V' be this set of degree 2 vertices which lie on one side G_t.

Consider the set of all edges e that are adjacent to a vertex in V'. These edges create a subgraph H of G_t. Note that H is bipartite, and that one of its parts contains only vertices of degree 2. This part consists exactly of the vertices in V'.

It is worth noting that we can explicitly construct each connected component C of H in NC. To do so, we first construct H. Then, given a vertex v, we can find all vertices in its connected component by testing ST connectivity (which is in NL, and therefore in NC) between v and each other vertex u (in parallel for all u). Now, we have the set of vertices of C. We can complete the construction of C by checking for any two vertices of C whether they are connected with an edge in G_t, and if so add an edge between them in C.

We note that each connected component C of H with k vertices in V' will have either $2k$ or $2k+1$ vertices in total (there must be at least $2k$ vertices because G_t satisfies Hall's condition, as G_t is a union of perfect matchings. There cannot be more than $2k+1$ vertices in C because there are exactly $2k$ edges in C, and C is connected). We will use a different procedure to deal with connected components of even size and of odd size.

Case 1: C is of even size: Suppose there is a connected component C with $2k$ vertices. We will find a matching of the connected component. Note that any perfect matching of G_t must have a matching of C as a submatching, since the k vertices of C which are in V' have k neighbors in total (namely, the other k vertices of C). Therefore, every matching of G_t can be separated into a matching of C and a matching of $V(G_t) \setminus C$. Therefore, if we find a matching of C, it can be extended to a matching of G_t. We know that C has at most 1 cycle, since it has $2k$ vertices, $2k$ edges, and is connected. Therefore, C has at most 2 matchings. It follows that we can find a perfect matching of the connected component in NC, since in [10] the authors prove that one can find a perfect matching in NC if the number of perfect matching is polynomial in n. We can thus eliminate all vertices which participate in connected components of H of even size. Note that we can run the above in parallel for all connected components of even size.

Case 2: C is of odd size: We describe the procedure in terms of a single connected component. In the algorithm itself we do this for all odd connected components in parallel. Let C be a connected component of H with $2k+1$ vertices. We note that the component is connected and has $2k$ edges, and is therefore a tree. We contract the connected component into a vertex v', and call the contracted graph G' (specifically, we create one "master vertex" for the connected component C. The edges of the master vertex include all edges of the form (v, u), where $v \in C$ and $u \notin C$). Note that all edges adjacent to the master vertex v' must connect to the same side of the bipartite graph. Namely, all such edges must connect to the same side as the vertices in V'. Therefore, the contracted graph is bipartite as well.

We note that any perfect matching in G_t must turn into a perfect matching in G' when the connected component is contracted. Therefore, G' has at least one perfect matching. Also, note that G' had all of the vertices in V' either contracted, or eliminated since they participated in a connected component of H of even size. Therefore, G' has at most $n - \frac{n}{40} = \frac{39n}{40}$ vertices.

We now describe how to turn a matching in G' to a matching in G_t. When we receive a matching on G', the matching will contain exactly one edge e' adjacent to the master vertex v'. That edge will have originated from some edge e adjacent to some $v \in C$. We can remove v (along with its edges) from C to get C'. The graph C' will have exactly $2k$ vertices and no cycles (since C was a tree).

Also, C' must have a perfect matching. Specifically, each vertex in $V' \cap C'$ can be matched with its neighbor that is further away from v (note that each vertex in $V' \cap C'$ has two neighbors, and since C is a tree, one of them is closer to v than the other. Also, no two vertices in $V' \cap C'$ can have the same neighbor further away from v, since that would imply a cycle in C). This gives us a set of k edges (note that $|V' \cap C'| = k$, since $|V' \cap C| = 2k+1$, and the vertex we removed from C to get C' was not on the same side of the bipartite graph as V') which are disjoint, which is a perfect matching of C'. Therefore, C' must have a perfect matching. The matching must be unique because C' has no cycles. Therefore, we we can find the matching of C' in NC [10].

When we add the edges of the matching of C' to the matching M' of G', and also add the matchings of even-sized connected components of H, we get a perfect matching of G, completing the analysis.

ECCC ISSN 1433-8092
http://eccc.hpi-web.de