Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 208 (2015)

Bipartite Perfect Matching in Pseudo-Deterministic NC

Shafi Goldwasser Ofer Grossman

June 15, 2017

Abstract

We present a pseudo-deterministic NC algorithm for finding perfect matchings in bipartite graphs. Specifically, our algorithm is a randomized parallel algorithm which uses $\text{poly}(n)$ processors, $\text{poly}(\log n)$ depth, $\text{poly}(\log n)$ random bits, and outputs for each bipartite input graph a **unique** perfect matching with high probability. That is, on the same graph it returns the same matching for almost all choices of randomness. As an immediate consequence we also find a pseudo-deterministic NC algorithm for constructing a depth first search (DFS) tree. We introduce a method for computing the union of all min-weight perfect matchings of a weighted graph in RNC and a novel set of weight assignments which in combination enable isolating a unique matching in a graph.

We then show a way to use pseudo-deterministic algorithms to reduce the number of random bits used by general randomized algorithms. The main idea is that random bits can be *reused* by successive invocations of pseudo-deterministic randomized algorithms. We use the technique to show an RNC algorithm for constructing a depth first search (DFS) tree using only $O(\log^2 n)$ bits whereas the previous best randomized algorithm used $O(\log^7 n)$, and a new sequential randomized algorithm for the set-maxima problem which uses fewer random bits than the previous state of the art.

Furthermore, we prove that resolving the decision question $NC = RNC$, would imply an NC algorithm for finding a bipartite perfect matching and finding a DFS tree in NC. This is not implied by previous randomized NC search algorithms for finding bipartite perfect matching, but is implied by the existence of a pseudo-deterministic NC search algorithm.

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.

A distinction of importance to our work is 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. Lovász [18] showed that using randomization, determining the decision problem is reducible to testing that certain integer matrices are non-singular[1]. Since

[1]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) is identically 0.

ISSN 1433-8092

the latter can be done in NC, an RNC algorithm for **deciding** if a perfect matching in a graph exists follows. The **search** version was subsequently shown to be in RNC by Karp, Upfal, and Wigderson [16] via a Monte-Carlo algorithm and by Karloff [15] via a Las-Vegas algorithm.

The next breakthrough was the RNC algorithm of Mulmuley, Vazirani, and Vazirani [19]. They assigned random weights 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 exists. Subsequently, the unique minimum weight perfect matching can be determined in parallel by assigning each edge to a different processor whose task is to essentially determine if the edge participates in the unique min-weight perfect matching. However, we emphasize that for the same graph and different isolating weight assignments it is highly likely that different perfect matchings will be found.

Quite recently, a significant step forward has been made by Fenner, Gurjar, and Thierauf [7] who showed how to remove randomization but increase the number of processors, for both the decision and the search variants of the perfect matching problem in bipartite graphs. They show a quasi-NC algorithm: that is, a deterministic poly$(\log n)$ time algorithm which uses quasi-polynomially many processors.

In a different line of work, initiated by Goldwasser and Gat, [9, 5, 12, 14] the class of search problems which can be solved by *pseudo-deterministic* polynomial time algorithms was introduced – these are probabilistic polynomial-time algorithms for search problems that produce a unique output for each given input except with small probability. That is, they return the same output for all but few of the possible random choices. Algorithms that satisfy the aforementioned condition are named pseudo-deterministic (for a formal definition, see Section 2), as they essentially offer the same functionality as deterministic algorithms: from the point of view of a computationally bounded observer, pseudo-deterministic and deterministic algorithms behave the same, since they always output the same answer.

Efficient pseudo-deterministic algorithms have been shown [5, 9, 14, 12] for several search problems for which no efficient deterministic algorithms are known. These problems include number theoretic search problems [14, 9, 5], multi-variate polynomial non-zero findings [9], and several sub-linear algorithms [12]. The latter work of Goldwasser, Goldreich and Ron [12] shows separations between deterministic, randomized and pseudo-deterministic sub-linear algorithms in accordance with the (asymptotic) number of queries they must require.

The larger question of whether the class of pseudo-deterministic polynomial time search problems is strictly contained in the class of probabilistic polynomial time search problems remains open (see the discussion by Goldreich [11] and in [9, 12] and the discussion below). However, the significance of showing a pseudo-deterministic algorithm in lieu of deterministic ones is amply illustrated by the following observation: If $P = BPP$ then any pseudo-deterministic polynomial time algorithm for a search problem implies a polynomial time deterministic algorithm for the problem.[2] In contrast, the randomized versus deterministic complexity of search problems may not be settled by all proofs of $P = BPP$. That is, certain proofs of $P = BPP$ may not generalize to the search setting. A similar situation emerges for the question of randomized versus deterministic parallel complexity of search problems.

We remark that the study of pseudo-deterministic algorithms seems particularly relevant in

[2]This follows from the characterization of Gat and Goldwasser [9] showing that search problems solvable by polynomial time pseudo-deterministic algorithms are exactly the problems solvable by a polynomial time algorithm with access to a BPP oracle for an analogous decision problem.

the parallel and distributed settings: if two parties invoke a pseudo-deterministic algorithm on the same input, they would be guaranteed to obtain the same result with high probability, regardless of the randomness used. In the distributed setting, pseudo-determinism has been used for scheduling algorithms [10].

1.1 Our Results

In this work we initiate the study of pseudo-deterministic algorithms in the context of NC. In particular, in lieu of deterministic NC algorithms for both the decision and search versions of perfect matching in a graph, we ask the following question: Does a pseudo-deterministic NC algorithm exist for the perfect matching search problem? We settle this question affirmatively for bipartite graphs.

We present a pseudo-deterministic NC algorithm for finding perfect matchings in bipartite graphs. Namely, we present a randomized NC algorithm which on input a bipartite graph G outputs a unique (canonical) perfect matching with high probability, if at least one perfect matching exists. All previous RNC algorithms (including [19]) would output different matchings on different executions.

Theorem 1.1 (Main Theorem)**.** *There exists a pseudo-deterministic NC algorithm that, given a bipartite graph G, returns a perfect matching of G, or states that none exist. The algorithm uses $O(\log^2(n))$ random bits.*

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 NC algorithm for computing depth first search (DFS) in general directed graphs.

Corollary (DFS)**.** *There exists a pseudo-deterministic NC algorithm that, given a directed graph G, returns a depth first search tree of G.*

In Section 5, we show a general method for using pseudo-determinism to reducing the number of random bits used by general randomized algorithms. The main idea is that random bits used to solve problems pseudo-deterministically can later be reused, thus avoiding the need to sample more random bits. We then show applications of the technique, including the following Theorem:

Theorem (DFS With Few Random Bits)**.** *There exists an RNC algorithm that, given a directed graph G, returns a depth first search tree of G using only $O(\log^2(n))$ random bits. Furthermore, the algorithm is pseudo-deterministic.*

Previously, the best known algorithm for computing a DFS in RNC used $O(\log^7(n))$ random bits (and, furthermore, was not pseudo-deterministic).

It is easy to show, as stated below, that the set of problems solvable by NC pseudo-deterministic algorithms are exactly the set of problems solvable by an NC algorithm with an oracle to RNC decision problems. Thus, our main result implies the existence of a deterministic NC algorithm for finding a bipartite perfect matching if $NC = RNC$. We remark that $NC = RNC$ does not generally imply that every *search* problem that is solvable by an RNC algorithm has a deterministic NC solution. In particular, all prior works on the perfect matching search problem in bipartite or general graphs do not imply a deterministic NC solution for the perfect matching search problem,

even if an NC algorithm were found for the decision version of the problem, or more generally if $NC = RNC$.

Lemma (Pseudo-deterministic NC)**.** *The class of search problems with pseudo-deterministic NC algorithms is the class of search problems solvable by an NC machine given access to an oracle for RNC decision problems.*

Combining the above lemma with Theorem 1.1, we prove the following:

Corollary. *If $NC = RNC$, then there exists a deterministic NC algorithm that given a bipartite graph G, outputs a perfect matching of G (or states that none exists).*

Combining with the reduction of Aggarwal, Anderson, and Kao [1], we also obtain the following:

Corollary. *If $NC = RNC$, then given a graph G, there exists an NC algorithm that returns a depth first search tree of G.*

1.2 Pseudo-Determinism and Search vs Decision Derandomization

Understanding the role of randomness in computation is one of the main problems in complexity theory. In the context of *decision* problems the separation between P and BPP indeed captures the gap between randomized and deterministic polynomial time algorithms. However, in the context of *search* problems, it does not. Even if we assume $P = BPP$, there may exist search problems solvable by known randomized polynomial time algorithms which may not succumb to deterministic polynomial time algorithms. In other words, there exist polynomial time search problems whose randomized vs deterministic complexity may not be settled by a proof of $P = BPP$ (it is worth noting that many approaches towards $P = BPP$, such as pseudorandom generators, may generalize to show that search-P = search-BPP, for certain definitions of search-BPP. In particular, any proof of $P = BPP$ strong enough to prove that promise-P = promise-BPP would generalize to the search setting as shown by Goldreich [11]).

For example, the problem of generating primes (given 1^n, output a prime with n bits) has an efficient randomized algorithm. However, even under the assumption $P = BPP$, it is not known if there exists a polynomial time deterministic algorithm. A similar situation is the case for the primitive root problem (given a prime p, output a primitive root modulo p).

In [9], the authors prove that a problem admits a pseudo-deterministic polynomial time algorithm (i.e., a randomized search algorithm which outputs the same result for all but few random seeds, formally defined in Section 2) if and only if it is polynomial time reducible to a decision problem in BPP. Thus, any pseudo-deterministic algorithm one demonstrates for a search problem would immediately provide a derandomized algorithm for the problem if $P = BPP$.

Our understanding of the randomized complexity for search problems in the parallel setting is quite similar to the polynomial time setting. The NC vs RNC question does not capture the full power of randomization in the parallel setting, since resolving the question for decision problems has no direct bearing on the search-NC vs search-RNC question. We remark that the leading derandomization effort in complexity theory, the so called "hardness vs randomness" paradigm, applies to search problems as well. In a nutshell, the tool of the paradigm is to construct pseudorandom generators or hitting-set generators to derandomize. However, other proofs of $NC = RNC$ may have no direct implication about the relationship of NC and RNC in the context of search.

The existence of a pseudo-deterministic algorithm for a problem has direct bearing on the derandomization question under the assumption $NC = RNC$. We show in Lemma 6.1 that if $NC = RNC$, then the set of problems with NC search algorithms equals the set of problems with NC pseudo-deterministic algorithms (i.e., search algorithms in RNC which output the same solution for all but few random seeds). Our argument is similar to the argument shown in [9] regarding the polynomial time setting.

Viewed in this light, our main theorem is that if $NC = RNC$, then there exists a deterministic NC algorithm for *finding* a perfect matching in a bipartite graph.

While it would be interesting to find general implications about derandomization of search problems under the assumption that decision problems are derandomized, our paper is specifically about the bipartite perfect matching problem.

1.3 High Level Ideas of the Algorithm

We now present an overview of the algorithm and proof of Theorem 1.1.

Let G be the given bipartite graph. At a high level the algorithm will proceed as follows.

1. In deterministic NC we construct a weight assignment w to the edges of G for which the **union graph** of all min-weight perfect matchings with respect to w (i.e., the union of all edges participating in at least one minimum weight matching) is significantly smaller than G.

2. In randomized NC we construct the union graph of all min-weight perfect matchings with respect to w.

3. We repeat steps 1 and 2 above, and every so often we contract some edges of the graph, until we arrive at a graph small enough that we can deterministically construct a perfect matching using brute force.

4. We then "uncontract" to arrive at a matching of the original graph.

Note that the only randomized step of the algorithm is step (2): the construction of the union graph of min-weight perfect matchings. Because the union graph of min-weight perfect matchings is unique (i.e., there is only one union of min-weight perfect matchings), this step of the algorithm is pseudo-deterministic. As all steps in the algorithm are either deterministic or pseudo-deterministic, the resulting algorithm will be in pseudo-deterministic.

Constructing the union of all min-weight perfect matchings of G with respect to w 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 perfect matching) while maintaining the property that the graph has a perfect matching. We will next show how to use randomization to construct the union.

We remark that it remains an open problem to deterministically compute the union of min-weight perfect matchings in NC. Whereas the *analysis* of the union graph with respect to a particular set of weight assignments plays an important role in the recent quasi-NC result of [7], *they do not explicitly construct it* as part of their decision or search algorithms. See Subsection 1.3.1 for a detailed description of the relation of our algorithm to that of [7].

Lemma (Union of min-weight perfect matchings)**.** *Let $G(V, E)$ be a bipartite graph with a polynomially bounded weight assignment w to the edges. 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 3.2. We outline the proof below.

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 any (not necessarily unique) w_i-minimal perfect matching (which can be done in RNC using techniques in [19], 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 with respect to w. We can then return the union of all e_i which are in some min-weight matching.

Recall that the goal of constructing the union graph is to reduce the problem to a smaller graph by removing many edges. 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 at most 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, if it is 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 (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 by contracting vertices of degree up to 2. It was shown in [7] that for any weight assignment w, every 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[3]. We thus need to show how to construct a set of weight functions which will ensure that each small (containing fewer than $4 \log n$ vertices) cycle 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 one can construct in NC a set of $O(\log n)$ weight assignments with weights bounded by $\mathrm{poly}(n)$ such that every cycle of length up to $4 \log n$ has nonzero circulation for at least one of the weight assignments.*

This Lemma appears in Section 3 as Lemma 3.1.

To prove this Lemma we first note that if a cycle of length up to $2k = 4 \log n$ has circulation 0, then the sum of the weights of the odd edges equals the sum of the weights of the even edges.

[3]We note that in [7] the authors do not construct the union of w-minimal perfect matchings, but only use the union in their analysis of the algorithm. The algorithm in this paper, on the other hand, constructs the union of w-minimal perfect matchings.

That means that there are two subsets of $E(G)$ of size up to k that have the same sum of weights. If we could construct weight functions such that no two sets of size up to k have the same sum of weights with respect to all of the weight functions, we will have proved the Lemma.

The idea of the construction in the Lemma's proof is to have $k+1$ weight assignments, and let the mth edge have weight

$$w_i(m) = [m^i]_p$$

with respect to the ith weight function, 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, given the sum of the weights (with respect to each of the w_i) of k elements labeled m_1 through m_k, we can retrieve the sums $\sum_{j=1}^{k} m_j^i \pmod p$, for all $1 \leq i \leq k+1$. Using these sums, we can use Newton's identities to find the minimal polynomial over $\mathbb{F}_p$ with roots $m_1, m_2, \ldots, m_k$, which uniquely determines the set of elements $m_1, m_2, \ldots, m_k$,. 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 (where $k = 2\log n$) are of polynomial size.

We now can construct the union of min-weight matchings with respect to w_1 to get a graph G_1. Then, we can construct the union of min-weight matchings in G_1, with respect to w_2, and so on until w_k. 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.

1.3.1 Relation to [7]

It is interesting to compare our work to the work of Fenner, Gurjar, and Thierauf [7] who design a deterministic quasi-NC algorithm for deciding and finding perfect matchings in a bipartite graphs. A main idea of their work was to construct a set of weight assignments and analyze the union of min-weight matchings with respect to these weight assignments. *In their algorithm, the union of min-weight perfect matchings is never explicitly constructed, but is only used in the analysis.* Indeed, it is not known how to deterministically construct the union of min-weight matchings, and our algorithm uses randomness to construct the union of min-weight matchings. Their algorithm to find the matching follows from applying the procedure of Mulmuley et al [19] to the graph with each of the weight functions they construct until an isolating one is found. Furthermore, the weight assignments used by [7] are different from ours. See the discussion in Section 3 after Lemma 3.1 of its relation to Lemma 2.3 of [7].

1.3.2 Organization

In Section 2, we discuss useful definitions and lemmas from prior works. In Section 3, we prove the main lemmas used in the algorithm. In Section 4, we describe the algorithm, and prove its correctness In Section 5, we show a general method for saving random bits using pseudo-determinism, and apply it to save bits in the matching algorithm, as well as in a depth first search algorithm. In Section 6 we show that if $NC = RNC$, then all pseudo-deterministic NC algorithms can be fully derandomized. In the Appendix, we show how to reduce the number of random bits used by our matching algorithm.

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 $(x, y) \in R$.

Definition 2.2 (Pseudo-Deterministic NC)**.** We call an algorithm *pseudo-deterministic NC* if it is in RNC, and is pseudo-deterministic.

We now present some lemmas from previous work.

Lemma 2.3 (Theorem 2 in [19])**.** *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 2.4 (Circulation)**.** Let $G(V, E)$ be a graph with weight assignment w. 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 planar bipartite matching [4] and for a quasi-NC algorithm for bipartite matching [7].

Lemma 2.5 (Lemma 3.2 in [7])**.** *Let G be a bipartite graph, and let C be a cycle in G. Let w be a weight function such that the cycle C has nonzero circulation. Then the graph G_1 obtained by taking the union of all min-weight perfect matchings on G does not contain the cycle C.*

The proof in [7] relies on the matching polytope. We present a combnatorial proof found by Anup Rao, Amir Shpilka, and Avi Wigderson, based on Hall's Theorem:

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 all edges in G'' is lower than the total weight of all edges in G'. We know that G'' is regular of degree d, and therefore is a union of d perfect matchings. If we decompose G'' into d perfect matchings, it is impossible that they all have weight at least M, because G'' had total weight less than Md. 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. □

The following lemma originates in [2].

Lemma 2.6. *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

We present some definitions, as well as key lemmas from previous work, in Section 2.

In [19], 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 (Non-Zero Circulation for Small Cycles.)**.** *Let G be a bipartite graph on n vertices. Then one can construct in NC a set of $O(\log n)$ weight assignments with weights bounded by $\text{poly}(n)$ such that every cycle of length up to $4 \log n$ 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]). At heart, the two Lemmas are quite different, but they seem similar at first glance. 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 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.

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 $S = \{s_1, \ldots, s_m\}$ be the edges of G. Consider the following weight assignments $w_1, w_2, \ldots, w_{2 \log n + 1}$, where we write $w_i(m)$ as shorthand for $w_i(s_m)$:

$$w_i(m) = [m^i]_p,$$

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 $\log n$).

We will show that there exist no two subsets of size up to $2\log n$ which have the same sum of weights. Then, in particular, there exists no cycle of length up to $4\log n$ with circulation 0.

Suppose there exist two distinct subsets of size up to $k = 2\log n$ with equal sums of weights with respect to each of the w_i. 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. This gives us the following equivalences modulo p:

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

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 larger than 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 $\mathbb{F}_p$.

Given the sums of the ith powers of the a_j for i between 1 and $k+1$, Newton's identities 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 distinct subsets of size up to k with the same sum of weights with respect to all of the weight assignments. □

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 3.2 (Union of min-weight perfect matchings)**.** *Let $G(V, E)$ be a bipartite graph with a polynomially bounded weight assignment w to the edges. 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 the weight of 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 2.3 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 w.

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 minimum 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 2.3) and checking its weight (or 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 Check if G has a matching in RNC using the algorithm of [19].
2 If G does not have a perfect matching, return $\perp$.
3 If $|E(G)| \leq 100$:
4 Find and return a perfect matching of G using brute force.
5 Let $\{w_1, \ldots, w_t\}$ be the set of weight assignments defined in Lemma 3.1 with $t = O(\log n)$.
6 Let $G_0 = G$.
7 For $i = 1, 2, \ldots, t$:
8 Let G_i be the union of w_i-minimal perfect matchings of G_{i-1} (use Lemma 3.2).
9 Contract vertices of degree up to 2 in G_t to create G' (see Section 4.1).
10 Let $M' =$ PERFECT-MATCHING(G').
11 Extend the matching M' in G' to a matching M in G_t (see Section 4.1). 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 11 we can extend the perfect matching M' in G' to a perfect matching M in G_t. This requires analyzing the contraction procedure of step 9. The contraction procedure takes some care, but its ideas are non-central to our proof.

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 (v', u') adjacent to u'. That edge, in the original graph, must either be of the form (v', u_1) or of the form (v', u_2) (note that it cannot be of the form (u', v), with v being the vertex of degree 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 M in G_t from the matching M' in G'. Doing this for multiple vertices in parallel leads to some complications which we elaborate on in Section 4.1.

Note that we can amplify the success probability of step 8 so that the probability of failure is at most $\frac{1}{n}$. Since the step gets executed a total of $O(\log^2 n)$ times ($O(\log n)$ times on each of the $O(\log(n))$ steps of the recursion), by the union bound the probability that step 8 ever fails is at most $\frac{\log^2(n)}{O(n)}$, which can be further amplifies through repetition.

We now argue the algorithm is pseudo-deterministic. We note that randomization is only used in step 8 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 4 takes $O(1)$ time and step 5 is in NC by Lemma 3.1. The number of iterations of the loop in step 7 is of length $O(\log(n)^2)$, by Lemma 3.1, and taking the union of min-weight perfect matchings within the loop in step 8 is in RNC by Lemma 3.2. 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 3.1 and Lemma 2.5, we see that after completing the loop, G_t has no cycles of length up to $4 \log n$. By Lemma 2.6, in step 9 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 10 is $O(\log n)$.

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

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

We note that as a consequence of Lemma 6.1 and Theorem 4.1, if $NC = RNC$ then the bipartite perfect matching search problem can be solved in NC. In the Appendix, we improve upon Theorem 4.1 by showing a pseudo-deterministic NC algorithm for bipartite perfect matching which uses only $O(\log^4 n)$ random bits. In section 5, we further improve upon that by showing a pseudo-deterministic NC algorithm using only $O(\log^2 n)$ random bits.

4.1 Contracting vertices of degree up to 2

As previously mentioned, contracting edges of degree up to 2 in step 9 of the algorithm and extending a matching in step 11 takes some care, yet is non-central to proof. 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 are 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, breaking ties arbitrarily. 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 [13] 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 u', 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 (u, v), where $u \in C$ and $v \notin C$). Note that all edges adjacent to the master vertex u' 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' (i.e., $u \notin V'$, and v is on the same part of the bipartite graph as 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 u'. That edge will have originated from some edge e (if there is more than one such possible e, pick one arbitrarily) adjacent to some $u \in C$. We can remove u (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 u (note that each vertex in $V' \cap C'$ has two neighbors, and since C is a tree, one of them is closer to U than the other. Also, no two vertices in $V' \cap C'$ can have the same neighbor further away from u, since that would imply a cycle in C). This gives us a set of k edges (note that $|V' \cap C'| = 2k$, 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 can find the matching of C' in NC [13].

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.

5 Saving Random Bits Via Pseudo-determinism

In this section, we show a technique for saving random bits using pseudo-determinism. We use the technique to save random bits on various problems. In particular, we present the first DFS algorithm which runs in RNC using only $\log^2(n)$ random bits (the previous best known bound was $\log^7(n)$).

At the high level, the idea of the technique is that random bits used in pseudo-deterministic protocols can be reused. Informally, suppose there exists a randomized algorithm A which, as a subroutine, executes a pseudo-deterministic algorithm B. Then all random bits used to execute B can be reused later in the execution of algorithm A, without significantly lowering the success probability of A. The reasoning is that the state of algorithm A after executing an instance of B does not strongly depend on the randomness used by B (for almost all random strings sampled, the state of A will be the same). Therefore, the random bits used to solve B *look random* to A, since the state of A does not depend on them.

We will demonstrate the technique with a problem we call Bipartite-Matching-Partition. An instance of Bipartite-Matching-Partition receives a regular bipartite graph of degree $d = \text{poly}(\log n)$. The output must be a partition of the edges into d disjoint perfect matchings. We wish to find an RNC algorithm for Bipartite-Matching-Partition which uses as few random bits as possible[4].

We note that by Hall's theorem, every regular bipartite graph has a perfect matching, so the following naive algorithm solves Bipartite-Matching-Partition:

1. Find a perfect matching M in the given graph G.
2. Remove the edges of M from G. Call the new graph G'
3. Recurse by repeating the algorithm on G'.

Since steps (1) and (2) above are in RNC, and the recursion depth of the algorithm is $d = \text{poly}(\log n)$, the above algorithm is indeed in RNC.

One can find a perfect matching in RNC using only $\log^2(n)$ random bits (this was first proven in [7], and we later use the pseudo-deterministic bit-saving technique to give an alternate proof).

[4]Bipartite-Matching-Partition can be solved deterministically in NC using a result for deterministically finding a perfect matching in a regular bipartite graph of low degree [17]. However, our algorithm is different from the deterministic one, and serves as a good example for the technique. Our other bit-saving results are not known to have corresponding deterministic algorithms.

Therefore, the above algorithm would use a total of $O(d \log^2(n))$ random bits, which is polylogarithmic, but can be as large as $O(\log^c n)$ for any constant c, depending on the degree d of the input graph.

One idea to save random bits is to try using the same random bits in each iteration of the algorithm. We show that this is possible to do when using a pseudo-deterministic algorithms in step (1). See Remark 5.2 for more information about why pseudo-determinism is necessary.

Our technique improves upon this bound. Suppose there was a pseudo-deterministic algorithm A for bipartite perfect matching which used b random bits to achieve an error probability of $O(1/n)$. Suppose A is used as the routine for the first step of the algorithm for Bipartite-Matching-Partition. We show that our algorithm can reuse the same b bits in each step of the recursion in the algorithm, without significantly lowering the success probability.

Specifically, our algorithm is as follows:

1. Sample a random string s consisting of b random bits.
2. Pseudo-deterministically find a perfect matching M in the given graph G, using the random string s.
3. Remove the edges of M from G. Call the new graph G'
4. Recurse by returning to step (2) with the graph G'.

In the above algorithm, we sampled only b random bits, instead of sampling b random bits in each recursive call.

Note that in this situation, because A is pseudo-deterministic, the graph G_i constructed in the ith iteration of the algorithm is well defined as a function of the input (it does not depend on the random string chosen). By the union bound, the probability that algorithm A fails on any of the G_i is bounded by $\frac{\text{poly}(\log n)}{n}$. Hence, with high probability, the algorithm A succeeds on all of the G_i, and therefore the algorithm as a whole succeeds with high probability. The condition that A is pseudo-deterministic is crucial. Otherwise, it is unclear whether the random bits can be reused (see Remark 5.2).

This results in the following theorem:

Theorem 5.1. *If there exists an RNC psuedo-deterministic algorithm for bipartite perfect matching using $O(b)$ random bits which achieves error probability $1/n$, then there exists an RNC algorithm for Bipartite-Matching-Partition using $O(b)$ random bits.*

We later show a pseudo-deterministic algorithm for search bipartite perfect matching using only $O(\log^2 n)$ random bits, proving that Bipartite-Matching-Partition has an RNC algorithm using $O(\log^2 n)$ random bits.

Remark 5.2. We would like to point out why pseudo-determinism is necessary. *Specifically, we will show why when using a non-pseudo-deterministic algorithm for step 4 of the algorithm, reusing random bits may not work (or is much harder to analyze).* When using a pseudo-deterministic algorithm, the graph G' formed by removing a matching from G is well defined (as a function of G). However, if the algorithm is not pseudo-deterministic, the graph G' is not well defined as a function of G: it depends on the randomness that the algorithm sampled to solve the bipartite matching problem on G. In the pseudo-deterministic case, we can therefore use the union bound

to bound the probability of a set of random bits succeeding on both G and G'. However, in the non-pseudo-deterministic case, we cannot, since G' depends on the randomness used to solve the bipartite matching problem on G. Therefore, in this case, the random bits are not random with respect to G', since G' depends on the random bits chosen.

Remark 5.3. The problem of matching partition can actually be solved in deterministic NC using the algorithm from [17] for finding a perfect matching in a regular graph of polylogarithmic degree. We nevertheless include this example because we find it to be a simple application of the saving bits technique. Our other applications of the saving bits techniques do not have known deterministic algorithms.

5.1 Pseudo-deterministic bipartite perfect matching in RNC with $O(\log^2 n)$ random bits

The algorithm for bipartite matching in the Appendix used $O(\log^4 n)$ random bits. We use our bit-saving technique to improve this to $O(\log^2 n)$. We note that in [8], the authors show an algorithm for search-BPM using $O(\log^2 n)$ bits. However, their algorithm is not pseudo-deterministic.

The improvement is immediate when having the technique in mind.

The matching algorithm from Section A in the appendix proceeds in $O(\log^2 n)$ steps ($O(\log n)$ steps, each of which had $O(\log n)$ weight functions). In each step, a weight assignment was constructed deterministically, and then the union of min-weight perfect matchings was constructed using $O(\log^2 n)$ random bits. This resulted in a total of $O(\log^4 n)$ random bits.

Since finding the union of min-weight perfect matchings is a pseudo-deterministic process (there is only one set which is the union of min-weight perfect matchings, so any algorithm which achieves correctness is also pseudo-deterministic), we can use the same $O(\log^2 n)$ random bits in each of the $\log n$ iterations to achieve a bound of $O(\log^2 n)$ random bits. We get the following theorem:

Theorem 5.4 (Main Theorem)**.** *There exists a pseudo-deterministic NC algorithm that, given a bipartite graph G, returns a perfect matching of G, or states that none exist. The algorithm uses $O(\log^2(n))$ random bits.*

We note that the algorithm of Karloff [15] of a Las Vegas algorithm for Bipartite Perfect Matching uses randomization only to solve Bipartite Perfect Matching as a sub-problem. We can use the same random bits whenever executing a bipartite perfect matching protocol, and therefore we have the following corollary:

Theorem 5.5 (Las Vegas Pseudo-deterministic bipartite perfect matching)**.** *There exists a pseudo-deterministic RNC algorithm for search-BPM which requires only $O(\log^2 n)$ random bits. If there is no perfect matching, the algorithm outputs a proof that there is no matching (which is verifiable by a deterministic NC algorithm).*

Combining Theorem 5.4 with Theorem 5.1, we get

Corollary 5.6 (Bipartite-Matching-Partition)**.** *There exists an RNC algorithm for Bipartite-Matching-Partition using $O(\log^2 n)$ random bits.*

5.2 Depth First Search in RNC

In [1], a deterministic NC reduction from DFS to bipartite matching is shown. The algorithm queries an oracle for bipartite perfect matching $O(\log^5 n)$ times. Therefore without using the pseudo-deterministic bit-saving technique, a naive algorithm for DFS would require $O(\log^7 n)$ random bits. However, if we use a pseudo-deterministic algorithm to solve the instances of perfect matching, and we reuse the random bits, we get an RNC algorithm for constructing a DFS using $O(\log^2 n)$ random bits.

Formally, we have:

Theorem 5.7 (Depth-First-Search)**.** *Given a directed graph G, one can construct a depth-first-search tree in RNC using $O(\log^2 n)$ random bits.*

5.3 Set Maxima

In this section, we demonstrate that our technique is also useful in the sequential setting, contrasting our other examples which involved the parallel setting. Set Maxima is a problem related to the minimum spanning tree problem, which is defined as follows. We are given a set system (χ, S) where χ is a set of n totally ordered elements and $S = \{S_1, \ldots, S_m\}$ is a collection of subsets of χ. The problem is to determine $\{\max S_i\}_{1 \le i \le m}$.

In [20], Pettie and Ramachandran show an algorithm using $O(n \log\left(\frac{m+n}{n}\right))$ expected comparisons[5] (which is optimal) and $O(\log n \log\log\log n 2^{\log^* n})$ random bits. Using our technique, we improve the bound to $O(\log n \log\log\log n \log^* n)$ random bits using the same number of expected comparisons. It remains open to fully derandomize the algorithm, or to improve this bound on the number of random bits.

The algorithm of [20] proceeds recursively, where the recursive depth of the algorithm is $O(\log^* n)$. However, because the set-maxima problem has a unique correct solution, any algorithm for set-maxima is pseudo-deterministic (this is not an "interesting" case of pseudo-determinism, because there is only one correct answer, not multiple). Therefore, we can use the same random bits for each of the recursive calls, instead of sampling new bits at each recursive call. So, instead of sampling $O(\log n \log\log\log n)$ new bits for each node in the tree, we sample $O(\log n \log\log\log n)$ bits for each level of the tree, reducing the number of random bits needed to $O(\log n \log\log\log n \log^* n)$. Formally, we have

Theorem 5.8 (Set Maxima)**.** *Set Maxima can be solved optimally, with expected $O(n \log \frac{m+n}{n})$ comparisons, while using $O(\log n \log\log\log n \log^* n)$ random bits.*

5.4 Pseudo-deterministic polynomial identity testing with fewer random bits

In [9], the authors show a pseudo-deterministic algorithm for polynomial identity testing: given a nonzero multivariate polynomial $P(x_1, x_2, \ldots, x_n)$, they pseudo-deterministically find an input $a_1, a_2, \ldots, a_n$ such that $P(a_1, a_2, \ldots, a_n) \neq 0$. We show how to solve the same problem pseudo-deterministically using fewer random bits.

Recall that the algorithm in [9] used randomization only to check if polynomials are zero or not. They did so up to $O(nd)$ times, where n is the number of variables, and d is the degree of the polynomial. Therefore, the total number of random bits they used is $O(ndB(n,d))$, where $B(n,d)$

[5] We note that we measure the complexity in terms of the number of comparisons, and not in terms of time

is the number of random bits used to check if a polynomial of degree d of n variables is zero, with error probability $\frac{1}{O(n^2 d)}$.

Using the bit-saving method, we can reduce the number of random bits to $O(B(n,d))$ (the number of bits required to run only a constant number of polynomial identity tests), beating the previous bound by a factor of nd (we can repeat our algorithm a constant number of times to achieve the same error probability as the algorithm of [9]).

6 The RNC vs. NC question and Pseudo-Determinism

In [9] the authors prove that the set of problems solvable by polynomial time pseudo-deterministic algorithms os exactly the set of problems solvable by a polynomial time algorithm using a decision-BPP oracle. We prove the analogous lemma for pseudo-deterministic NC with the same technique.

Lemma 6.1. *The class of search problems with pseudo-deterministic NC algorithms is the class of search problems solvable by an NC machine given access to an oracle for RNC decision problems.*

Proof. First, we show that an NC algorithm with an oracle for RNC decision problems has a corresponding pseudo-deterministic NC algorithm. Consider an NC algorithm A which uses an oracle for RNC. We can simulate A by another algorithm B which runs RNC algorithms instead of querying the oracle. B will output unique solutions since, with high probability, all of its runs of RNC algorithms will return the same solution on each execution (since in the case of decision problems, correctness implies uniqueness).

We now show that a pseudo-deterministic NC algorithm B has a corresponding NC algorithm A that uses an RNC oracle. On input x, we let A create a process A_i for each output bit $B(x)_i$ of $B(x)$. Note that determining $B(x)_i$ is an RNC decision problem, so A_i can find $B(x)_i$ using its oracle. Then, the algorithm A combines all of the bits $B(x)_i$ to output $B(x)$. □

Note that the above Lemma implies that the class of problems with pseudo-deterministic NC algorithms equals the class of problems with NC search algorithms if and only if $NC = RNC$.

As a consequence of the Lemma, Theorem 4.1 implies the following:

Corollary 6.2. *If $NC = RNC$, then given a bipartite graph G with at least one perfect matching, there exists a deterministic NC algorithm that outputs a perfect matching of G.*

7 Discussion

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 (the symmetric difference of two matchings of different weights will contain a cycle of non-zero circulation). 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 NC algorithms for some network flow problems such as max-flow approximation, which was shown in [21] to be NC-reducible to maximum bipartite matching. In addition, as an immediate corollary we get a pseudo-deterministic algorithm for max

flow, where capacities are expressed in unary (this problem was shown to be NC-reducible to bipartite perfect matching in [16])

It remains open to find a pseudo-deterministic NC algorithm for perfect matching in general (non-bipartite) graphs.

In the context of polynomial time pseudo-determinism, there are many fundamental problems with polynomial time randomized algorithms where the existence of pseudo-deterministic polynomial time algorithms remains open. These problems include generating primes (given 1^n, output a prime with n bits); given a prime p and 1^d, finding an irreducible degree d polynomial over $\mathbb{F}_p$; and finding a primitive root modulo p given a prime p and the factorization of $p-1$. We hope for more progress towards finding pseudo-deterministic polynomial time algorithms for these problems.

Acknowledgments

Thanks to Anup Rao for communicating to us his proof with Amir Shpilka and Avi Wigderson of Lemma 2.5 (also Lemma 3.2 in [7]). We are very grateful to Oded Goldreich for many helpful discussions on this paper and on the connections between the questions of decision versus search derandomization and pseudo-determinism.

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. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2016, pages 754–763, New York, NY, USA, 2016. ACM.

[8] 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/.

[9] 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.

[10] Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 3–12. ACM, 2015.

[11] Oded Goldreich. In a world of P= BPP. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 191–232. Springer, 2011.

[12] 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.

[13] 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.

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

[15] Howard J Karloff. A Las Vegas RNC algorithm for maximum matching. *Combinatorica*, 6(4):387–391, 1986.

[16] 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.

[17] Raghav Kulkarni. A new nc-algorithm for finding a perfect matching in d-regular bipartite graphs when d is small. In *Italian Conference on Algorithms and Complexity*, pages 308–319. Springer, 2006.

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

[19] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[20] Seth Pettie and Vijaya Ramachandran. Randomized minimum spanning tree algorithms using exponentially fewer random bits. *ACM Transactions on Algorithms (TALG)*, 4(1):5, 2008.

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

A Using Fewer Random Bits

We note that this section is about constructing an algorithm for bipartite matching using fewer random bits. For the section on techniques for saving random bits using pseudo-deterministic algorithms, see Section 5.

In this section, we will construct a pseudo-deterministic NC algorithm for the bipartite perfect matching search problem which uses only $\log^4 n$ random bits.

We remark that this is the first known RNC algorithm for *search* bipartite perfect matching which uses only poly($\log n$) random bits (while [7] has such a result, their result did not appear in the first version of the paper). In a followup to their original paper and the first posting of this paper, [7] showed an algorithm using $\log^2(n)$ random bits. In section 5 we also show an RNC algorithm using $\log^2(n)$ random bits (which was discovered after the algorithm from [7] which achieves $\log^2(n)$ random bits). We note that our algorithms are pseudo-deterministic, while the algorithms in [7] are not.

Our algorithm is based on our previous pseudo-deterministic NC algorithm of Section 4. 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 poly($\log(n)$) random bits.

Theorem A.1. *There exists a pseudo-deterministic NC algorithm that, given a bipartite graph G, returns a perfect matching of G, or states that none exist. The algorithm uses $O(\log^4 n)$ random bits.*

Proof. 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 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, since every matching in the union of max-weight matchings has the same weight (if there are two matchings of different weights, their symmetric difference will contain a cycle of non-zero circulation).

In section 4 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 the set of edges in a max-weight matching with w_e.)

Because the construction uses $\log^4(n)$ random bits and can achieve $\frac{1}{n^3}$ probability of failure, we can use 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 $O(\log^4 n)$ bits, completing the proof. □

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il