Bipartite Perfect Matching in Pseudo-Deterministic NC[*]

Shafi Goldwasser[1] **and Ofer Grossman**[2]

1 **Massachusetts Institute of Technology, Cambridge, MA, USA**
`shafi@theory.csail.mit.edu`

2 **Massachusetts Institute of Technology, Cambridge, MA, USA**
`ofer@mit.edu`

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 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, 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.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Parallel Algorithms, Pseudo-determinism, RNC, Perfect Matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.87

1 Introduction

Perfect Matching: 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.

[*] Full version available on ECCC as TR15-208, `https://eccc.weizmann.ac.il/report/2015/208/`.

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 [15] showed that using randomization, determining the decision problem is reducible to testing that certain integer matrices are non-singular[1]. Since 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 [14] via a Monte-Carlo algorithm and by Karloff [13] via a Las-Vegas algorithm.

The next breakthrough was the RNC algorithm of Mulmuley, Vazirani, and Vazirani [16]. 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 $\text{poly}(\log n)$ time algorithm which uses quasi-polynomially many processors. 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 [16] 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 8 of its relation to Lemma 2.3 of [7].

Pseudo-determinism: In a different line of work, initiated by Goldwasser and Gat, [8, 5, 11, 12] 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, 8, 12, 11] for several search problems for which no efficient deterministic algorithms are known. These problems include number theoretic search problems [12, 8, 5], multi-variate polynomial non-zero findings [8],

[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.

and several sub-linear algorithms [11]. The latter work of Goldwasser, Goldreich and Ron [11] 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 [10] and in [8, 11] 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 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 [9].

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 [16]) would output different matchings on different executions.

▶ **Theorem 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 the full version of the paper, we show a general method for using pseudo-determinism to reducing the number of random bits used by general randomized algorithms. The main

[2] This follows from the characterization of Gat and Goldwasser [8] 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.

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 possible 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 $NC = RNC$. 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. More generally, even if $NC = RNC$, it does not generally imply that every *search* problem that is solvable by an RNC algorithm has a deterministic NC solution.

▶ **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, 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 High Level Ideas of the Algorithm

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

▶ **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 9. 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 [16], 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 $\text{poly}(n)$ such that every cycle of length up to $4 \log n$ has nonzero circulation for at least one of the weight assignments.*

This Lemma appears in Section 3 as Lemma 8. We remark that the weight assignments used by [7] are different from ours. We point the reader to a discussion following Lemma 8 for its relation to Lemma 2.3 of [7].

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. 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 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

[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.

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 [10]). 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 [8], 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 pseudo-random 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 the full version of the paper 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 [8] 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.

1.3.1 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 the full version of the paper, 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. Furthermore, in the full version we show that if $NC = RNC$, then all pseudo-deterministic NC algorithms can be fully derandomized. In the full version of the paper, 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** (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 3** (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 4** (Theorem 2 in [16]). *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 5** (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 6** (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 a single copy of each odd edge of C from G', and add a single copy of each even edge of C to G'. 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 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

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

In [16], 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 8** (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.*

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 $A = \{a_1, a_2, \dots, a_k\}$ and $B = \{b_1, b_2, \dots, b_k\}$ 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 9** (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 4 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 4) 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 [16].
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 weight assignments from Lemma 8 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 9).
9   Contract vertices of degree up to 2 in $G_t$ to create $G'$
    (see full version of paper for details).
10  Let $M' =$ PERFECT-MATCHING($G'$).
11  Extend the matching $M'$ in $G'$ to a matching $M$ in $G_t$
    (see full version of paper for details).
12  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 the full version of the paper.

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 8. The number of iterations of the loop in step 7 is of length $O(\log(n)^2)$, by Lemma 8, and taking the union of min-weight perfect matchings within the loop in step 8 is in RNC by Lemma 9. 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 8 and Lemma 6, we see that after completing the loop, G_t has no cycles of length up to $4 \log n$. By Lemma 7, 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 10.** *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 Theorem 10 and a result appearing in the full version of the paper (the result that if $NC = RNC$, then every pseudo-deterministic NC algorithm can be fully derandomized), if $NC = RNC$ then the bipartite perfect matching search problem can be solved in NC. In the full version, we improve upon Theorem 10 by showing a pseudo-deterministic NC algorithm for bipartite perfect matching which uses only $O(\log^4 n)$ random bits. In the full version of the paper, we further improve upon that by showing a pseudo-deterministic NC algorithm using only $O(\log^2 n)$ random bits.

5 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 [17] 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 [14])

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 6 (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. In *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. `doi:10.1145/2897518.2897564`.

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 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.

10 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.

11 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.

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

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

14 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.

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

16 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. `doi:10.1007/BF02579206`.

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