A Pseudo-Deterministic RNC Algorithm for General Graph Perfect Matching

Nima Anari[1] and Vijay V. Vazirani[2]

[1]Computer Science Department, Stanford University, anari@cs.stanford.edu
[2]Computer Science Department, University of California, Irvine, vazirani@ics.uci.edu

January 29, 2019

Abstract

The difficulty of obtaining an NC perfect matching algorithm has led researchers to study matching vis-a-vis clever relaxations of the class NC. In this vein, recently Goldwasser and Grossman [GG15] gave a pseudo-deterministic RNC algorithm for finding a perfect matching in a bipartite graph, i.e., an RNC algorithm with the additional requirement that on the same graph, it should return the same unique perfect matching for almost all choices of random bits. In this paper, we give an analogous algorithm for general, not necessarily bipartite, graphs. More generally, we give a pseudo-deterministic RNC algorithm for finding a minimum weight perfect matching when the edge weights are polynomially bounded.

Our algorithm builds on Anari and Vazirani [AV18], whose result used planarity of input graphs critically; in fact, in three different ways. The main challenge was to adapt these steps to general graphs by exploiting the leeway that seeking a pseudo-deterministic RNC algorithm and not an NC algorithm gives us.

1 Introduction

Finding an NC algorithm[1] for perfect matching is currently among the outstanding open problems of theoretical computer science. This problem has been open for over three decades, ever since the discovery of RNC matching algorithms [KUW86; MVV87], and its difficulty has led researchers to study matching vis-a-vis certain clever relaxations of the class NC. One such relaxation is quasi-NC, under which the algorithm must run in polylogarithmic time, though it can use $O(n^{\log^{O(1)} n})$ processors; see section 1.1 for results obtained for this model. Several nice algorithmic ideas were discovered in these works which eventually led to the solution of the long-standing open problem of obtaining an NC algorithm for finding a perfect matching in planar graphs [AV18] and restoring order on the following issue: On the one hand, counting the number of perfect matchings is far harder than finding one (the former is #P-complete and the latter is in P), and on the other, for planar graphs, counting has long been known to be in NC whereas finding one had resisted a

[1]That is, a deterministic parallel algorithm that runs in polylogarithmic time using polynomially many processors.

solution! We note that subsequently, Sankowski [San18] obtained the same result via different techniques.

A second relaxation of NC is pseudo-deterministic RNC [GG11; GG15], which is an RNC algorithm with the additional requirement that on the same graph, it should return the same (i.e., unique) perfect matching for almost all choices of random bits. Goldwasser and Grossman [GG15] recently gave such an algorithm for perfect matching in bipartite graphs.

In this paper, we give a pseudo-deterministic RNC algorithm for perfect matching in general graphs. More generally, if edge weights are polynomially bounded, we give an NC reduction from search to decision for the minimum weight perfect matching problem, where the latter problem is: Given an edge weighted graph G and an integral weight W, is there a perfect matching of weight at most W in G? Since the weights are polynomially bounded, this decision problem is NC equivalent to: Given an edge weighted graph G, find the weight of a minimum weight perfect matching in G. This question is easy to answer in RNC with inverse-polynomial probability of error using the algorithm of Mulmuley, Vazirani, and Vazirani [MVV87]. This together with the reduction stated above yields the pseudo-deterministic RNC algorithm for general graphs.

Our main result is:

Theorem 1.1. *There is a pseudo-deterministic* RNC *algorithm for finding a minimum weight perfect matching in a general graph with polynomially bounded edge weights.*

1.1 Related work and a brief history of parallel matching algorithms

The notion of a pseudo-deterministic algorithm with polynomial expected running time was given by Gat and Goldwasser [GG11]. Such an algorithm runs in expected polynomial time and is required to output the same (i.e., unique) solution on a given instance on each run with high probability. Hence, in this sense, it resembles a deterministic algorithm. One motivation given for this notion was its use in a distributed setting for verifying delegated computation and for generating cryptographic keys. Gat and Goldwasser [GG11] gave pseudo-deterministic polynomial expected running time algorithms for several number theoretic and cryptographic problems. The extension to pseudo-deterministic RNC algorithms was defined by Goldwasser and Grossman [GG15].

An RNC algorithm for the decision problem, of determining if a graph has a perfect matching, was obtained by Lovász [Lov79], using the Tutte matrix of the graph. The first RNC algorithm for the search problem, of actually finding a perfect matching, was obtained by Karp, Upfal, and Wigderson [KUW86]. This was followed by a somewhat simpler algorithm due to Mulmuley, Vazirani, and Vazirani [MVV87].

The matching problem occupies an especially distinguished position in the theory of algorithms: Some of the most central notions and powerful tools within this theory were discovered in the context of an algorithmic study of this problem, including the notion of polynomial time solvability [Edm65a] and the counting class #P [Val79]. The parallel algorithms perspective has also led to such gains: The first RNC matching algorithm led to a fundamental understanding of the computational relationship between search and decision problems [KUW85] and the second algorithm yielded the Isolation Lemma [MVV87], which has found several applications in complexity theory and algorithms. Considering the fundamental insights gained by an algorithmic study of perfect matchings, the problem of obtaining an NC algorithm for it has remained a premier open question ever since the 1980s.

The first substantial progress on this question was made for the case of planar bipartite graphs by Miller and Naor [MN89] via a flow-based approach, followed by Mahajan and Varadarajan [MV00] using the fact that there is an NC algorithm for counting perfect matchings in planar graphs. The NC algorithm of Anari and Vazirani [AV18] for non-bipartite planar graphs also uses this approach, though it requires a number of new ideas to deal with odd set constraints appearing in the LP-relaxation of perfect matching in general graphs. They also extended their algorithm to constant genus graphs. Subsequently, Eppstein and Vazirani [EV18] gave an NC algorithm for perfect matching in one-crossing-minor-free graphs, which include K_5-free graphs and $K_{3,3}$-free graphs; the resolution of the latter class settles a thirty year old open problem asked in [Vaz89].

The quasi-NC algorithms for matching and its generalizations, mentioned above, work by achieving a partial derandomization of the Isolation Lemma. First, Fenner, Gurjar, and Thierauf [FGT16] gave a quasi-NC algorithm for perfect matching in bipartite graphs, which was followed by the algorithm of Svensson and Tarnawski [ST17] for general graphs. Algorithms were also obtained for the generalization of bipartite matching to the linear matroid intersection problem by Gurjar and Thierauf [GT17], and to a further generalization of finding a vertex of a polytope with faces given by totally unimodular constraints, by Gurjar, Thierauf, and Vishnoi [GTV17].

1.2 Bipartite vs non-bipartite matching: An intriguing phenomenon

Decades of algorithmic work on the matching problem, from numerous perspectives, exhibits the following intriguing phenomenon: The bipartite case gets solved first. Then, using much more elaborate machinery, the general graph case also follows and yields the exact same result! This phenomenon is made all the more fascinating by the fact that the "elaborate machinery" consists not of one fact but numerous different structural properties and mathematical facts which happen to be just right for the problem at hand! We give a number of examples below.

The duality between maximum matching and minimum vertex cover for bipartite graphs extends to general graphs via the notion of an odd set cover, see Lovász and Plummer [LP09]. The formulation of the perfect matching polytope for bipartite graphs extends by introducing constraints corresponding to odd sets [Edm65b]. Polynomial time algorithms for maximum matching and maximum weight matching in bipartite graphs generalize via the notion of blossoms [LP09]. The most efficient known algorithm for maximum matching in bipartite graphs [HK73; Kar73] obtained via an alternating breadth first search, extends via a much more elaborate algorithm with the same running time via the graph search procedure of double depth first search [MV80] and blossoms defined from the perspective of minimum length alternating paths [Vaz94]. The RNC matching algorithms [KUW86; MVV87] use Tutte's theorem to extend to general graphs. The randomized matching algorithm of Rabin and Vazirani [RV89] uses Tutte's theorem and a theorem of Frobenius on ranks of sub-matrices of skew-symmetric matrices.

More recent work exhibits this phenomenon as well. The quasi-NC algorithm of Fenner, Gurjar, and Thierauf [FGT16] for bipartite graphs extends by handling tight odd cuts appropriately [ST17].

The NC algorithm of [MV00] for planar bipartite graphs was extended to non-bipartite graphs via Edmonds' formulation of the perfect matching polytope [Edm65b], an NC algorithm for max-flow in planar graphs [Joh87], and a result of Padberg and Rao [PR82] for finding tight odd cuts and an elaborate NC algorithm for uncrossing tight odd cuts [AV18]. In the same vein, the current paper is extending the bipartite algorithm of [GG15] using Edmonds' formulation of the perfect matching polytope [Edm65b] and an NC procedure for finding a maximal laminar family of tight odd cuts [CGS12; San18].

2 Overview and technical ideas

For most of the paper, we will concentrate on the problem of finding a perfect matching. Finally, in section 5.1 we show how our ideas extend to finding a minimum weight perfect matching if the edge weights are polynomially bounded.

2.1 The bipartite case

The techniques that went into designing our algorithm can be more clearly described after giving an overview of the pseudo-deterministic RNC algorithm of Goldwasser and Grossman [GG15], which in turn is easiest seen as following from a suitable modification of the quasi-NC algorithm of Fenner, Gurjar, and Thierauf [FGT16] for bipartite graphs.

The latter first finds a point in the interior of the perfect matching polytope and then iteratively moves to lower dimensional faces of this polytope, terminating when a vertex of the polytope is reached; this will be a perfect matching. For an edge weight vector w, let $\mathrm{PM}[w]$ denote the face of the polytope containing all fractional and integral minimum weight perfect matchings w.r.t. w. Since we are in the bipartite case, $\mathrm{PM}[w]$ has a simple description: It is defined by the set of edges that are set to zero, or equivalently its complement, i.e., the set of edges that participate in minimum weight perfect matchings. Let us denote the latter by $E[w]$.

A key notion, introduced by Datta, Kulkarni, and Roy [DKR10], is that of *circulation* of a cycle C in the graph on edges $E[w]$. This is the absolute value of the difference in the sum of weights of alternate edges. Observe that if the minimum weight perfect matching w.r.t. weight vector w is not unique, then any cycle that is in the symmetric difference of two such matchings, and therefore is also in the graph on $E[w]$, must have zero circulation. Hence, if we find a weight vector w such that each cycle in $E[w]$ has nonzero circulation, then the minimum weight perfect matching must be unique and we can easily find it in NC.

Assume that a cycle $C \subseteq E[w]$ has zero circulation. Let w' be a weight vector such that $E[w'] \subseteq E[w]$ and the circulation of C w.r.t. w' is nonzero. Then it is easy to show that C will not be present in $E[w']$, i.e., at least one of its edges will be dropped in going from $E[w]$ to $E[w']$. If so, we will say that C got *destroyed*. Thus our goal is to find, in NC, a weight vector that destroys all cycles of G. However, G may have exponentially many cycles.

One of the key ideas of [FGT16] is a systematic way of destroying cycles: They iteratively destroy cycles of length $4, 8, 16, \ldots, n$; clearly, the number of iterations needed is $O(\log n)$. In the first round, G has at most $O(n^4)$ cycles of length 4. Fenner, Gurjar, and Thierauf [FGT16] show that if all cycles of length at most 2^i have already been destroyed, then there are at most $O(n^4)$ cycles of length at most 2^{i+1} left. Hence, in each iteration only $O(n^4)$ cycles need to be destroyed.

Suppose a given iteration starts with weight vector w and edge set $E[w]$. The two tasks to be accomplished in this iteration are: find an appropriate weight vector, w', which destroys the $O(n^4)$ relevant cycles, and compute $E[w']$. For the first task, Fenner, Gurjar, and Thierauf [FGT16] prove that in order to destroy any set of s cycles, it suffices to try certain well-chosen $O(n^2 s)$ integer weight vectors each of which uses numbers at most $O(n^2 s)$ large; one of these vectors is sure to work. Since $s = O(n^4)$, $O(n^6)$ vectors suffice. To address the second task, i.e., computing $E[w']$, we will not describe the solution of Fenner, Gurjar, and Thierauf [FGT16] which requires quasi-NC. Instead, we pick up ideas from Goldwasser and Grossman [GG15] to get a pseudo-deterministic RNC algorithm.

First consider the following sub-task: Given weight vector w and an edge $e \in E$, determine if $e \in E[w]$. This can easily be done in RNC with an inverse polynomial probability of error, and hence $E[w]$ can be computed in RNC with very high probability. Now, in parallel, for each of the $O(n^6)$ weight vectors, w, we compute $E[w]$ and find the girth of the resulting graph. We pick the lexicographically first weight vector, say w', such that $E[w']$ has the desired lower bound on girth, i.e, it destroys all the relevant length cycles. Weight vector w' is the output of this iteration.

2.2 Extension to general graphs

New complication arise in general graphs because of the more complex formulation of the perfect matching polytope, and consequently of the face $\mathrm{PM}[w]$: to specify $\mathrm{PM}[w]$ we need not only the edges $E[w]$, which participate in minimum weight perfect matchings, but also a maximal laminar family of tight odd sets, say $\mathcal{L}$.

2.2.1 An approach based on Svensson and Tarnawski [ST17]

For bipartite graphs, the pseudo-deterministic RNC algorithm of Goldwasser and Grossman [GG15] followed from the quasi-NC algorithm of Fenner, Gurjar, and Thierauf [FGT16]. Hence for general graphs it is natural to build on the quasi-NC algorithm of Svensson and Tarnawski [ST17]. However, our attempts at making this approach work have failed. We will first pick up some useful concepts from [ST17] and outline why this approach failed before describing the approach that works.

We start with some definitions from [ST17]. The perfect matching polytope lies in $\mathbb{R}^E$. Let $\mathbb{1}_{E'}$ denote the indicator vector of a subset of edges $E' \subseteq E$. For $S \subset V$, $\delta(S)$ denotes the edges that cross S. For an even cycle C, let $\mathrm{sign}(C)$ denote the vector in $\mathbb{R}^E$ that is $+1$ and -1 on alternate edges of C, starting with an arbitrary edge. For $S \in \mathcal{L}$, C *respects set* S if $\langle \mathbb{1}_{\delta(S)}, \mathrm{sign}(C) \rangle = 0$. Furthermore, C *respects face* $\mathrm{PM}[w]$ if:

1. $C \subseteq E[w]$ and
2. C respects every $S \in \mathcal{L}$.

If the minimum weight perfect matching is not unique, then it is easy to see that each cycle C in the symmetric difference of two such matchings respects face $\mathrm{PM}[w]$. Therefore, if we find weights w so that none of the even cycles in the graph on edges $E[w]$ respects face $\mathrm{PM}[w]$, then the minimum weight perfect matching will be unique.

Once again, the notion of circulation, as defined above, plays a crucial role. Suppose w.r.t. weight vector w, the circulation of cycle C is zero. Let w' be a weight vector such that $E[w'] \subseteq E[w]$ and the circulation of C w.r.t. w' is nonzero. If so, Svensson and Tarnawski [ST17] show that either C must lose an edge in going from $E[w]$ to $E[w']$ or a new odd set S goes tight w.r.t. w' such that C does not respect S. In either case, C does not respect the new face $\mathrm{PM}[w']$ and we will say that C is *destroyed*.

As in the bipartite case, Svensson and Tarnawski [ST17] go about iteratively destroying cycles of length $4, 8, 16, \ldots n$. Once again, they would like to show that if all cycles of length at most λ have already been destroyed, then there are only polynomially many cycles of length at most 2λ. However, to derive the latter condition, they not only require that all cycles of length λ are destroyed, but also a condition on the tight odd sets of size at most λ in the current laminar family $\mathcal{L}$ of tight sets. A weight vector w satisfying both these conditions is called *λ-good*. In the next iterative step they would like to obtain a weight vector that is 2λ-good, and so on.

At this point, a new difficulty shows up: Svensson and Tarnawski [ST17] remark that they do not know how to test in NC if w is λ-good (recall that in the bipartite case, the corresponding step required simply finding the girth of the graph, which is easily done in NC). We managed to clear this hurdle by giving an RNC algorithm, having very low probability of error, for this task. In particular, it involved finding in NC a maximal laminar family of tight odd sets for face $\mathrm{PM}[w']$ using ideas from Cygan, Gabow, and Sankowski [CGS12] and Sankowski [San18]. However, even this does not suffice since the following subtile issue arises.

Assume that at the end of a certain iteration, the weight function is w, which is λ-good and $\mathcal{L}$ is the laminar family of tight odd sets found at this stage by the algorithm of Svensson and Tarnawski [ST17]. Further, let w' be the weight function at the end of the next iteration, which is 2λ-good. Then, for all their inductive assumptions to hold, they require that the next laminar family, $\mathcal{L}'$, is obtained from $\mathcal{L}$ by adding non-crossing sets to $\mathcal{L}$, i.e., $\mathcal{L}' \supseteq \mathcal{L}$. On the other hand, Cygan, Gabow, and Sankowski [CGS12] identify a dual solution to minimum weight perfect matching problem, called a *balanced critical dual*, which is unique and can be found in NC for planar graphs, as observed by Sankowski [San18]; moreover, it can be found in pseudo-deterministic RNC for general graphs as well. It turns out that the families of tight sets thus found in the faces $\mathrm{PM}[w]$ and $\mathrm{PM}[w']$ are not related by inclusion. Indeed, at present we do not know if there is any RNC way of ensuring this.

Our current algorithm builds on Anari and Vazirani [AV18], whose result uses planarity critically; in fact, in three different ways. The main challenge was to adapt these steps for general graph by exploiting the leeway given by the fact that we want a pseudo-deterministic RNC algorithm and not an NC algorithm. This is described next at a high level. We note that our current approach avoids several complications that were inherent in the previous approach for planar graphs, thereby leading to a simpler algorithm.

2.2.2 An approach based on Anari and Vazirani [AV18]

We start by describing at a high level the steps executed by the algorithm of Anari and Vazirani [AV18] for a planar graph, G:

Step (1). Finding $\Omega(n)$ edge-disjoint even walks: An *even walk* is either an even simple cycle or two odd cycles with a path joining them, traversed so that the cycles are traversed once and the path twice, once in each direction. As shown by Svensson and Tarnawski [ST17], all statements made above about destroying even cycles also hold about destroying even walks. Anari and Vazirani [AV18] critically use Euler's formula and the planar dual of G for executing this step in NC.

Step (2). Finding a point in PM and moving it to a face where all walks are blocked: The first part is based on the fact that counting the number of perfect matchings in planar graphs is in NC. For the second part, Anari and Vazirani [AV18] show how to find in NC a weight function for edges, w, such that each walk has a non-zero circulation. As a result, if x is a point in the face $\mathrm{PM}[w]$, then each walk C either has an edge e such that $e \notin E[w]$ or there is a tight odd set S such that C does not respect S. i.e., each walk is destroyed. Such a weight function is easily obtained; however, finding $x \in \mathrm{PM}[w]$ again resorts to the fact that counting the number of perfect matchings in planar graphs is in NC; more precisely, that a Pfaffian orientation for planar graphs exists and can be computed in NC.

Step (3). If there is a tight odd set S such that walk C does not respect S, finding one such set: An old result of Padberg and Rao [PR82] states that a Gomory-Hu tree for G under edge weights given by w will have a minimum tight odd cut in it. Anari and Vazirani [AV18] give an NC algorithm for finding a Gomory-Hu tree for a planar graph using the fact that a maximum s-t flow in planar graphs can be computed in NC [Joh87]. Anari and Vazirani [AV18] further show that a small perturbation of w ensures that one of the tight odd set S that is not respected by C is the minimum tight odd cut in G and hence will appear in the Gomory-Hu tree. Repeating this process in parallel for each such walk C, yields a set of tight odd sets.

Step (4). Partial uncrossing of tight odd sets found: The tight odd sets found cannot be shrunk simultaneously because they may be crossing. In the absence of an NC uncrossing algorithm, Anari and Vazirani [AV18] give an NC algorithm for finding the *outermost sets* of some laminarization of the tight odd sets found. These will be disjoint and can all be shrunk simultaneously.

Step (5). Finding a balanced viable cut: Anari and Vazirani [AV18] show that executing the previous four steps $O(\log n)$ times, together with the shrinking of sets, will either lead to removal of a constant fraction of the edges of the original graph or to a constant sized graph which will reveal a *balanced viable cut* in G, i.e, a cut $(S, \overline{S})$ such that both S and $\overline{S}$ have a constant fraction of the vertices and there is a perfect matching in G which contains exactly one edge from this cut. In the first case, we recurse on the smaller graph and in the second, we pick one feasible edge from the cut to be in the perfect matching and recursively find perfect matchings in parallel in the two smaller graphs. We observe that this step works for non-planar graphs as well and we will use a slight generalization of this fact below.

We next outline how our algorithm for general graphs avoids the use of planarity. As stated previously, we will assume that we are given an oracle $\mathcal{O}$ for the decision problem, of determining if a given graph, whose edge weights are polynomially bounded, has a perfect matching having weight at most W. Observe that the decision oracle enables us to ensure that throughout the algorithm, each edge in the current graph is *allowed*, i.e., participates in a minimum weight perfect matching.

We finesse the use of planarity in steps (1) and (2) as follows. Using ideas from Caprara, Panconesi, and Rizzi [CPR03] we show that if the graph $G = (V, E)$ is not too sparse, then it contains $\Omega\left(\frac{|E|}{\log^2|V|}\right)$ edge-disjoint even walks, see lemmas 6.5 and 6.7. However, we do not know how to find a set of such walks fast in parallel. At this point we observe that finding walks is not essential; it suffices to find a weight function, w, so that each of these walks has non-zero circulation and is therefore destroyed in the face $\mathrm{PM}[w]$. To accomplish this, we will work obliviously, as was done in [FGT16; ST17]: we will show how to construct $O(n^4)$ weight functions, each assigning a weight of at most $O(n^4)$ to each edge, such that under one of them each walk has non-zero circulation.

Next, in parallel, for each of the $O(n^4)$ weight function w, we compute $E[w]$. Also, using the NC algorithms of Cygan, Gabow, and Sankowski [CGS12] and Sankowski [San18] mentioned above, we find a maximal laminar family of tight odd sets, $\mathcal{L}$, in the face $\mathrm{PM}[w]$. Observe that since $\mathcal{L}$ is maximal, for each walk C that does not lose an edge, $\mathcal{L}$ must contain a set S that C does not respect. Furthermore, since $\mathcal{L}$ is already laminar, we directly shrink its maximal, and hence disjoint, sets. Observe that this has enabled us to accomplish steps (3) and (4) in one shot. Let H_w be the resulting graph. For the weight function that worked, each walk will lose at least one edge hence leading to a loss of at least $\Omega(|E|/\log^2|V|)$ edges.

In cases where the graph G is very sparse, i.e., when $|E| \simeq (1+\epsilon)|V|$, we will prove that we can find $\Omega(|E|)$ disjoint tight sets of size 3. These are induced paths of length 2 on the graph. Shrinking all of them also yields a sizable reduction in the number of edges in the graph.

Finally, step (5) follows as in [AV18], with one difference, namely we require $O(\log^3|V|)$ iterations to reach a graph where there is a unique perfect matching. We use this as a partial matching in the original graph, and recursively extend it to a perfect matching by solving the search problem on graphs that are smaller by a constant factor.

3 Preliminaries

We represent undirected graphs by $G = (V, E)$, where V is the set of vertices and E is the set of edges. Unless otherwise specified, we only work with graphs that have no loops, i.e., an edge from a vertex to itself. An edge between vertices u and v is represented as $\{u, v\}$. For a set $S \subseteq V$, we use $\delta(S)$ to denote the cut between S and its complement, i.e., $\delta(S) = \{\{u, v\} \in E \mid u \in S, v \notin S\}$. When S is a singleton, i.e., $\{v\}$ for some $v \in V$, we use the shorthand $\delta(v) = \delta(\{v\})$. A perfect matching is a subset of edges $M \subseteq E$ such that for all $v \in V$ we have $|M \cap \delta(v)| = 1$.

Definition 3.1. We call an edge $e = \{u, v\}$ isolated if $\deg(u) = \deg(v) = 1$.

By this definition a graph is a perfect matching if it has no isolated vertices and all of its edges are isolated.

For a set $S \subseteq E$ of edges we use $\mathbb{1}_S \in \mathbb{R}^E$ to denote the indicator of S. We use the shorthand $\mathbb{1}_e$ to denote the e-th element of the standard basis for $\mathbb{R}^E$, where $e \in E$. We denote the standard inner product between vectors $w, x \in \mathbb{R}^E$ by $\langle w, x \rangle$.

Given a convex polytope $P \subseteq \mathbb{R}^E$, and a weight vector $w \in \mathbb{R}^E$, we use $P[w]$ to denote the set of points minimizing the weight function $x \mapsto \langle w, x \rangle$:

$$P[w] = \{x \in P \mid \forall y \in P : \langle w, x \rangle \leq \langle w, y \rangle\}.$$

Note that $P[w]$ is a face of P; all faces of P can be obtained as $P[w]$ for appropriately chosen w.

3.1 Maximal independent sets

Given a graph $G = (V, E)$, we call a subset $S \subseteq V$ independent if no edge $e \in E$ has both endpoints in S. We call an independent set maximal if no strict superset $T \supsetneq S$ is independent. We will crucially use the fact that maximal independent sets can be found in NC.

Theorem 3.2 ([Lub86])**.** *There is a deterministic* NC *algorithm that on input graph* $G = (V, E)$ *returns a maximal independent set* $S \subseteq V$.

We usually want a large, rather than a maximal, independent set. We will use the fact that in bounded degree graphs, any maximal independent set is automatically large.

Fact 3.3. *If* $G = (V, E)$ *is a graph with* $\deg(v) \leq \Delta$ *for all* $v \in V$*, then any maximal independent set* $S \subseteq V$ *satisfies*

$$|S| \geq \frac{|V|}{\Delta + 1}.$$

3.2 Perfect matching polytope

Given a graph $G = (V, E)$, we call a subset of edges $M \subseteq E$ a perfect matching if it contains exactly one edge in every degree cut, i.e., $|M \cap \delta(v)| = 1$ for all v. We call a graph matching-covered if any of its edges can be extended to a perfect matching.

Definition 3.4. A graph $G = (V, E)$ is matching-covered if for every edge $e \in E$, there exists a perfect matching M such that $e \in M$.

The perfect matching polytope for $G = (V, E)$ is defined to have perfect matchings as its vertices

$$\mathrm{PM}_G = \mathrm{conv}\{\mathbb{1}_M \mid M \subseteq E \text{ is a perfect matching of } G\}.$$

When G is clear from context, we simply use PM to refer to this polytope. PM is alternatively described by the following set of linear equalities and inequalities [Edm65a]:

$$\mathrm{PM} = \left\{ x \in \mathbb{R}^E \;\middle|\; \begin{array}{ll} \langle \mathbb{1}_{\delta(v)}, x \rangle = 1 & \forall v \in V, \\ \langle \mathbb{1}_{\delta(S)}, x \rangle \geq 1 & \forall S \subseteq V, \text{ with } |S| \text{ odd}, \\ \langle \mathbb{1}_e, x \rangle \geq 0 & \forall e \in E. \end{array} \right\}. \tag{1}$$

Any face F of PM can be either described by a weight vector w, i.e., $F = \mathrm{PM}[w]$, or it can be alternatively described by the set of inequalities turned into equalities in eq. (1). These correspond to odd sets S and edges e. When face F is clear from context, we call odd sets whose inequalities have been turned into equalities, *tight* odd sets. We call an edge e *allowed* if $x_e > 0$ for some $x \in F$, i.e., if the inequality corresponding to e in eq. (1) has not been turned into equality. We use $E[w]$ or $E[F]$ to denote the set of allowed edges in the face $F = \mathrm{PM}[w]$. Putting it all together, to describe a face F it is enough to describe the set of allowed edges as well as tight odd sets.

Two tight odd sets $S_1, S_2 \subseteq V$ are said to *cross* if they are not disjoint and neither is a subset of the other. A family of these sets $\mathcal{L} \subseteq 2^V$ is said to be *laminar* if no pair of sets in it cross. It is well-known that each face F of the perfect matching polytope can be described by the set of allowed edges and a laminar family of tight odd sets $\mathcal{L}$:

$$F = \left\{ x \in \mathrm{PM} \;\middle|\; \begin{array}{ll} \langle \mathbb{1}_{\delta(S)}, x \rangle = 1 & \forall S \in \mathcal{L}, \\ \langle 1_e, x \rangle = 0 & \forall e \notin E[F]. \end{array} \right\}.$$

In fact, $\mathcal{L}$ can be taken to be any *maximal* laminar family of tight odd sets for the given face F [see, e.g., ST17, Lemma 2.2]. We will always include all singletons $\{v\}$ in the laminar family $\mathcal{L}$ as the equalities $\langle \mathbb{1}_{\delta(v)}, x \rangle = 1$ are automatically satisfied over all of PM.

Note that for a face $F = \mathrm{PM}[w]$, there are potentially many choices for the laminar family $\mathcal{L}$ describing F. Cygan, Gabow, and Sankowski [CGS12] studied the notion of *balanaced critical dual solutions* to isolate a unique choice for $\mathcal{L}$, and they showed how this unique $\mathcal{L}$ can be found by computing *primal* solutions to the minimum weight perfect matching problem. Sankowski [San18] used this procedure to design an alternative NC algorithm for planar graph perfect matching. We describe this procedure below. For more details see the work of Cygan, Gabow, and Sankowski [CGS12].

Definition 3.5. Suppose we are given a face $F = \mathrm{PM}[w]$. A *laminar optimal dual solution* is a laminar family $\mathcal{L}$ of tight odd sets, including all singletons, together with a function $\pi : \mathcal{L} \to \mathbb{R}$ such that for $S \in \mathcal{L}$, $\pi(S) > 0$ whenever $|S| > 1$ and for all edges e

$$w_e \geq \sum_{S \in \mathcal{L} : e \in \delta(S)} \pi(S),$$

with equality for allowed edges.

This definition precisely gives dual solutions for the linear program $\min\{\langle w, x \rangle \mid x \in \mathrm{PM}\}$ that satisfy complimentary slackness and are in *laminar* form. By complimentary slackness, for any such solution, $\sum_{S \in \mathcal{L}} \pi(S)$ is equal to the weight of a minimum weight perfect matching.

Laminar optimal dual solutions exist and are not unique. But Cygan, Gabow, and Sankowski [CGS12] showed that extra conditions can be imposed to make them unique. They dub pairs

$(\mathcal{L}, \pi)$ that satisfy these extra conditions *balanced critical duals* [CGS12, Definition 24]. We will not use these extra conditions; hence, we refrain from phrasing them here.

The following was shown by Cygan, Gabow, and Sankowski [CGS12, Lemma 28].

Lemma 3.6 ([CGS12]). *If $E[w]$ is connected, then a balanced critical dual is unique and algorithm 1 finds its support, the laminar family $\mathcal{L}$.*

Algorithm 1: Finding a balanced critical dual.

$\mathcal{L} \leftarrow \{\{v\} \mid v \in V\}$.
for $v \in V$ **in parallel do**
  $\mu(v) \leftarrow \min\{\langle w, \mathbb{1}_M \rangle \mid M \subseteq E[w]$ is a perfect matching on $V \setminus \{u, v\}$ for some vertex $u\}$.
end
Let $w'_e \leftarrow w_e + \mu(u) + \mu(v)$ for each $e \in E[w]$.
for $t \in \{w'_e \mid e \in E[w]\}$ **in parallel do**
  Find the connected components of the graph $(V, \{e \in E[w] \mid w'_e \le t\})$.
  Add each nontrivial connected component to $\mathcal{L}$.
end
return $\mathcal{L}$.

It was observed by Sankowski [San18] that all steps of algorithm 1 can be performed in NC except for finding allowed edges $E[w]$ and the computation of $\mu(v)$'s. We use the same insight to design our pseudo-deterministic RNC algorithms.

Remark 3.7. When $E[w]$ is not connected, algorithm 1 still works but should be run in parallel for *each connected component* of $E[w]$.

3.3 Contraction of tight odd sets and matching minors

It was observed by Edmonds [Edm65b] that if a collection of tight odd sets are disjoint, one can shrink each one to a single node and obtain a smaller graph whose perfect matchings can be extended to perfect matchings in the original graph. For the sake of completeness we state and prove this fact here.

Fact 3.8. *Suppose that $F = \mathrm{PM}[w]$ is a face of the matching polytope for $G = (V, E)$ and $S_1, \ldots, S_k$ are tight odd sets w.r.t. F. Let H be obtained from G by removing disallowed edges and contracting each S_i to a single node. Then any perfect matching in H can be extended to a perfect matching in G.*

Proof. Suppose that M is a perfect matching in H. We can think of edges in M as edges in E as well; in fact $M \subseteq E[w]$, because we remove disallowed edges to obtain H. Because M is a perfect matching in H, for each S_i, there is a unique $e_i \in M \cap \delta_G(S_i)$. Now since e_i is an allowed edge, there must be some perfect matching M_i of G such that $e_i \in M_i$ and $\mathbb{1}_{M_i} \in F$. Since S_i is a tight odd set, M_i cannot have any other edge in $\delta(S_i)$, except for e_i. So if we look at $\{\{u, v\} \in M_i \mid u, v \in S_i\}$, we must have a matching covering all vertices of S_i except for the endpoint of e_i. Combining all of these matchings for $i = 1, \ldots, k$ together with M will give us a perfect matching in G as desired. ☐

Note that the graph H obtained above is a minor of the graph G. But it is not an arbitrary minor. It has the additional property that every perfect matching of it can be extended back to a perfect matching of the original graph. For convenience we name these minors, matching minors.

Definition 3.9. A matching minor H of a graph G, is a graph that can be obtained by a sequence of the following operations: Pick a face of the matching polytope and a collection of disjoint tight odd sets. Remove disallowed edges, and contract each tight odd set into a single node.

The following statement follows directly from fact 3.8.

Lemma 3.10. *If H is a matching minor of the graph G, then every perfect matching in H can be extended to a perfect matching in G.*

In our algorithms, we use the simple observation that a path of length 2 on vertices of degree 2 yields a tight odd set for the entire matching polytope. We call these paths worms.

Definition 3.11. A worm in graph $G = (V, E)$ is a set of three vertices $\{a, b, c\}$ such that $\deg(a) = \deg(b) = \deg(c) = 2$, and $\{a, b\}, \{b, c\} \in E$.

Lemma 3.12. *A worm $\{a, b, c\}$ is a tight odd set for the matching polytope and all of its faces.*

Proof. The only two neighbors of b are a, c. So in every perfect matching, b must be matched to one of them. The other vertex must have an edge to an outside vertex, and in fact that is the only possible edge in $\delta(\{a, b, c\})$. □

Remark 3.13. Note that the proof of lemma 3.12 does not use the assumptions $\deg(a) = \deg(c) = 2$ and only uses $\deg(b) = 2$. We will use these extra assumptions elsewhere, to prove that in certain situations, we can find many worms in our graph.

3.4 Even walks and weight vectors

In this section, we present some definitions about even walks which we will need; some are borrowed from Svensson and Tarnawski [ST17]. We call an ordered list of evenly many edges $C = (e_1, \ldots, e_{2k})$, not necessarily distinct, that start and end at the same vertex, an *even walk*. An even walk is allowed to visit a vertex any number of times; if it visits each vertex at most once, it is a simple cycle. The *signature* of walk C is defined to be the vector:

$$\operatorname{sign}(C) = \sum_{i=1}^{2k} (-1)^i \mathbb{1}_{e_i}.$$

Even if C is nonempty, it may be that $\operatorname{sign}(C)$ is the zero vector. If so, we will say that C is a *null walk*. Unless otherwise specified, every walk we refer to will be assumed to be not null.

In fact in our analysis, we will only consider even walks C which are either a simple cycle, or two edge-disjoint cycles joined by a path whose edges are also disjoint from the cycles; here the path is traversed twice, once in each direction. It is easy to see that in either of these cases the even walks are not null.

Consider a weight vector w and an even walk C, such that

$$\langle w, \operatorname{sign}(C)\rangle \neq 0.$$

This means that there cannot be any two distinct points $x, y \in \mathrm{PM}[w]$ whose difference $x - y$ is a multiple of $\operatorname{sign}(C)$. Otherwise we would have $\langle w, x\rangle \neq \langle w, y\rangle$. Another way of stating this is that if $x \in \mathrm{PM}[w]$, then $x + \epsilon \operatorname{sign}(C) \notin \mathrm{PM}[w]$ for any $\epsilon \neq 0$. So, some inequality or equality describing $\mathrm{PM}[w]$ must be violated for this point.

If we pick x to be in the relative interior of the face $\mathrm{PM}[w]$ we will have some slack for non-tight inequalities describing $\mathrm{PM}[w]$. So the violated constraint for $x + \epsilon \operatorname{sign}(C)$ must be a constraint that is tight for the entire face $\mathrm{PM}[w]$. This implies that

Lemma 3.14. *Suppose that C is an even walk such that $\langle w, \operatorname{sign}(C)\rangle \neq 0$. Then either there is an edge $e \in C$ that is disallowed, i.e., $e \notin E[w]$, or for any laminar dual $(\mathcal{L}, \pi)$ describing* $\mathrm{PM}[w]$, *there is some set S with $\langle \mathbb{1}_{\delta(S)}, \operatorname{sign}(C)\rangle \neq 0$.*

For more detailed proof of this, see [AV18]. Note that when $\langle \mathbb{1}_{\delta(S)}, \operatorname{sign}(C)\rangle \neq 0$, it also means that C must have an edge with both endpoints inside S.

Lemma 3.15. *If C is an even walk and $S \subseteq V$ is such that $\langle \mathbb{1}_{\delta(S)}, \operatorname{sign}(C)\rangle \neq 0$, then there is an edge $e = \{u, v\} \in C$ such that $u, v \in S$.*

Proof. If this is not true, then every time C enters S it must immediately exit. So if we compute $\langle \mathbb{1}_{\delta(S)}, \operatorname{sign}(C)\rangle$ by looking at edges that cross S, we always get a $+1$ followed by a -1, and a -1 followed by a $+1$. So the entire sum would be 0 which is a contradiction. □

We also borrow from Fenner, Gurjar, and Thierauf [FGT16] the following important result, which is also stated in Svensson and Tarnawski [ST17].

Lemma 3.16 ([FGT16])**.** *There exists a polynomial sized family of polynomially bounded weight vectors $\mathcal{W}$, such that for any set of edge disjoint even walks $C_1, \ldots, C_k$, there is some $w \in \mathcal{W}$ which ensures*

$$\forall i : \langle w, \operatorname{sign}(C_i)\rangle \neq 0.$$

Proof. This lemma is actually proved in [FGT16; ST17] for any collection of nonzero vectors, not just $\operatorname{sign}(C_i)$'s, as long as there is both a polynomial bound on the number of vectors and the absolute value of their coordinates. Edge-disjointness of even walks automatically puts a bound of $|E|$ on their number, and the coordinates of our even walks are always bounded in absolute value by 2. □

4 Decision oracle

Throughout the paper we assume that there is an oracle $\mathcal{O}$ which answers the following type of queries: Given a graph $G = (V, E)$ and a polynomially bounded weight vector $w \in \mathbb{Z}^E$, what is the weight of the minimum weight perfect matching in G? We denote the answer by

$$\mathcal{O}(G, w) = \min\{\langle w, x\rangle \mid x \in \mathrm{PM}_G\}.$$

We now list several deterministic NC primitives based on $\mathcal{O}$. Versions of these two lemmas appear implicitly, stated for planar graphs, in Sankowski [San18], but we prove them for the sake of completeness.

Lemma 4.1. *Given access to $\mathcal{O}$, for polynomially bounded $w \in \mathbb{Z}^E$, one can find $E[w]$ in* NC.

Proof. An edge $e = \{u, v\}$ can be in a minimum weight perfect matching if and only if $\mathcal{O}(G, w) = w_e + \mathcal{O}(G - \{u\} - setv, w)$, where $G - \{u\} - \{v\}$ is obtained from G by removing vertices u, v. This can be checked in parallel for all edges e. □

Lemma 4.2. *Given access to $\mathcal{O}$, for polynomially bounded $w \in \mathbb{Z}^E$, one can run algorithm 1 in* NC.

Proof. As was observed by Sankowski [San18], all steps of algorithm 1 can be run in NC except for finding $E[w]$ and computing $\mu(v)$. Given access to $\mathcal{O}$, we can find $E[w]$ in NC by lemma 4.1. Furthermore observe that for any $v \in V$

$$\mu(v) = \min\{\mathcal{O}(G - \{u\} - \{v\}, w) \mid u \in V - \{v\}\},$$

which can be computed by making all queries $\mathcal{O}(G - \{u\} - \{v\}, w)$ in parallel and then taking the minimum. □

An implementation for the oracle, in RNC with arbitrarily small inverse polynomial probability of error for general graphs, follows from Mulmuley, Vazirani, and Vazirani [MVV87], since they give an RNC algorithm for finding a minimum weight perfect matching if the edge weights are given in unary. Note that the oracle $\mathcal{O}$ is promised to be called at most polynomially many times. So even a randomized Monte Carlo oracle, as in the case of RNC implementation for general graphs, is good enough, since we can boost its success probability to make sure the total error probability over all calls remains bounded.

5 Main algorithm

In this section we describe an algorithm for finding one perfect matching. Then in section 5.1, we will show how to extend this to finding a *minimum weight* perfect matching for polynomially bounded weights.

On input $G = (V, E)$, our algorithm proceeds by finding smaller and smaller matching minors H of G, until H has a unique perfect matching, or in other words is a perfect matching. Then we pick the edges in H as a partial matching in G and extend this partial matching to a perfect matching independently and in parallel for the preimage of each node in H. That is for each node s in H, we take the set $S \subseteq V$ that got shrunk to s, remove the single endpoint of the partial matching from S, and recursively find a perfect matching in S. In the end we return the results of all these recursive calls along with the edges of H as the final answer.

We crucially make sure that the preimage of nodes in H never contain more than a constant fraction of $|V|$. This makes sure that our recursive calls end in $O(\log|V|)$ many steps.

In all of our algorithms, when we construct matching minors, we implicitly maintain the mapping from the resulting edges to the original edges, and the mapping from original vertices to the minor's vertices. These are trivial to maintain in NC, but for clarity we avoid explicitly mentioning them. We also keep node weights for matching minors, where the weight of a node is simply the number of original vertices that got shrunk to it.

Algorithm 2: Divide-and-conquer algorithm for finding a perfect matching.

PerfectMatching($G = (V, E)$)
if $V = \emptyset$ **then**
  return $\emptyset$.
else
  Call PartialMatching(G), and let H be the matching minor returned.
  Let $M \subseteq E$ be the edges of H.
  for *each node s of H* **in parallel do**
    Let $S \subseteq V$ be the nodes of G that are shrunk to s.
    Let v be the unique endpoint of the unique edge of M in $\delta(S)$.
    Let G_s be the induced graph on $S - \{v\}$.
    $M \leftarrow M \cup$ PerfectMatching(G_s).
  end
  return M.
end

The pseudocode for the main algorithm PerfectMatching can be seen in algorithm 2. On input G, the algorithm calls PartialMatching to find a matching minor H of G, that itself is a perfect matching. Then the edges of H, which form a partial matching in G, are extended to a perfect matching independently and in parallel in the preimage of each node from H. Since H is a matching minor, this extension can always be performed by lemma 3.10.

The pseudocode for PartialMatching can be seen in algorithm 3. This algorithm keeps a node-weighted matching minor of the input graph G. It tries several ways of obtaining a smaller matching minor, where size is measured in terms of the number of non-isolated edges. One way of obtaining a smaller matching minor is by picking a maximal node-disjoint set of worms and shrinking them simultaneously. By lemma 3.12, this produces a matching minor. Also note that the maximal set of node-disjoint worms can be found in NC by enumerating all worms and using theorem 3.2.

Another way of obtaining smaller matching minors is by trying weights from the set of weight vectors $\mathcal{W}$ and calling Reduce to remove disallowed edges $e \notin E[w]$ and to shrink top level sets of a laminar family of tight odd sets w.r.t. w.

Finally the pseudocode for Reduce can be seen in algorithm 4. This algorithm is simply fed a graph $G = (V, E)$ and a weight vector w. It removes disallowed edges $e \notin E[w]$ and shrinks the maximal sets of a laminar family of tight odd set. The laminar family is found using algorithm 1, but is modified to make sure that no shrunk set becomes too large; to be more precise no shrunk vertex in the end will have node weight more than half of the total node weight.

Algorithm 3: Find a matching minor of the input graph that is itself a perfect matching.

```
PartialMatching(G = (V, E))
Assign node weight 1 to each node v ∈ V.
while G is not a perfect matching do
    if any node v of G has at least 1/6 of the total node weight then
        Remove disallowed edges e ∉ E[0] from G.
        Contract the complement of {v} to a single node. If there are parallel edges, remove all
          except for an arbitrary one.
        return G.
    end
    Find a maximal set of node-disjoint worms in G.
    Let H be obtained from G by removing disallowed edges and contracting each worm into a
      single node.
    U ← {H}.
    for w ∈ 𝒲 in parallel do
        Call Reduce(G, w) and let the result be H.
        U ← U ∪ {H}.
    end
    Find the graph H ∈ U with the minimum number of non-isolated edges.
    G ← H.
end
return G.
```

Algorithm 4: Remove disallowed edges and contract some tight odd sets w.r.t. a weight vector.

```
Reduce(G = (V, E), w) ;                              // The graph G has node weights.
Remove disallowed edges e ∉ E[w] from G.
Find all connected components of G.
for each connected component C of G in parallel do
    Run algorithm 1 on C to find a laminar family of tight odd sets ℒ.
    for S ∈ ℒ in parallel do
        if node weight of S is more than half of the node weight of C then
            Replace S in ℒ with C − S.
        end
    end
    Find the inclusion-wise maximal sets in ℒ and shrink each one to a single node.
end
return G.
```

5.1 Finding a minimum weight perfect matching

Here we describe how we can construct an algorithm that returns not just any perfect matching, but rather a minimum weight perfect matching, for polynomially bounded weights.

Given an input graph $G = (V, E)$ and a weight vector w, we can remove disallowed edges $e \notin E[w]$, and find a laminar family of tight odd sets $\mathcal{L}$ w.r.t. w, by calling algorithm 1 on each connected component of G. By complementary slackness, any perfect matching that has only one edge in $\delta(S)$ for each $S \in \mathcal{L}$ will automatically be of minimum weight, see definition 3.5. We can simply contract the top level sets in $\mathcal{L}$, use algorithm 2 to find a perfect matching in the shrunk graph, and recursively extend this to a minimum weight perfect matching in each shrunk piece. Following an almost identical argument as in the proof of fact 3.8, the perfect matching in the shrunk graph can be extended to a minimum weight perfect matching.

The only problem with this method is that the recursion depth is not guaranteed to be polylogarithmic. However we can fix that by making sure that tight odd sets $S \in \mathcal{L}$ do not have more than half of the vertices in the graph; if they do, we replace them by their complements and we will see in lemma 6.1 why this operation preserves laminarity.

6 Analysis of the algorithm

In this section we analyze our algorithm. First we prove that the algorithm returns a correct answer, assuming all calls to the decision oracle $\mathcal{O}$ had no error. Then we will bound the running time and prove that our algorithm is in NC modulo the calls to $\mathcal{O}$; this constitutes the most challenging part of the analysis.

6.1 Correctness

To prove the correctness, we use the following lemma.

Lemma 6.1. *Suppose that $\mathcal{L}$ is a laminar family of sets in a node-weighted graph $G = (V, E)$, and we replace every $S \in \mathcal{L}$ whose node weight is larger than half of the total node weight by the complement $V - S$. The resulting family of sets $\mathcal{L}'$ is still laminar.*

Proof. Let S, S' be two sets in $\mathcal{L}$. They are either disjoint or one is contained in the other.

If $S \cap S' = \emptyset$: They cannot both have node weight more than $1/2$. So at most one of them gets replaced by its complement. Then it is easy to see that the resulting sets do not cross.

If $S \subseteq S'$: There are three possibilities. If none of them gets replaced by their complements, or both of them get replaced by their complements, they remain nested and therefore do not cross. If one of them gets replaced by its complement, it has to be the larger set S'. In that case the resulting sets become disjoint, and still do not cross. $\square$

Now using lemma 6.1 and lemma 3.10, we deduce that Reduce always returns a matching minor of its input graph. Almost by definition, PartialMatching also returns a matching minor of its graph when it finishes (for the analysis of running time see section 6.2).

This proves the correctness of the algorithm, since we always find a matching minor that has a unique perfect matching (itself), and by lemma 3.10, we can extend it to a perfect matching, independently in the preimage of each node.

6.2 Running time

First we analyze PerfectMatching (algorithm 2) assuming the calls to PartialMatching (algorithm 3) are in NC.

Lemma 6.2. *Assuming the calls to* PartialMatching() *are in* NC*,* PerfectMatching() *is in* NC*.*

Proof. We simply need to bound the number of levels in the recursion. We will prove that when PartialMatching returns a matching minor H, the node weight of every node is at most $5/6$ the total node weight. This proves that in each recursive call to PerfectMatching, the number of vertices gets reduced by a factor of $5/6$.

Note that the first time in algorithm 3 that a node's weight goes above $1/6$ the total weight, the algorithm stops and returns a two-node minor. So we just need to prove that the weight of the node that just went above $1/6$ is not more than $5/6$. The current minor was obtained from the previous minor by either Reduce, or by shrinking worms. But Reduce automatically never creates nodes with weight more than half the total weight. The weight of each node in a worm is also by assumption at most $1/6$ the total weight, so after shrinking the worm, the new weight can be at most $1/6 + 1/6 + 1/6 = 1/2$ the total weight. This finishes the proof. □

All that remains to prove our algorithm is in NC is to show that PartialMatching finishes in polylogarithmically many steps. To show this, we will prove the following lemma.

Lemma 6.3. *In each iteration of algorithm 3, the number of non-isolated edges gets reduced by a factor of* $1 - \Omega(1/\log^2|V|)$.

Before proving lemma 6.3, note that it implies the number of iterations in algorithm 3 is at most polylogarithmic. Since if we track the number of non-isolated edges, after every $\Theta(\log^2|V|)$ steps we get a constant factor reduction, and therefore it only takes at most $O(\log|E| \cdot \log^2|V|)$ iterations for it to reach 0.

In the rest of this section we prove lemma 6.3. The high-level overview of the proof is as follows: Assume that we simply ignore isolated edges. Then, either the graph G is very sparse, roughly speaking $|E| = (1+\epsilon)|V|$ for a tiny constant ϵ, in which case we will prove the number of node-disjoint worms found is $\Omega(|E|)$. On the other hand if the graph is not very sparse, we will prove that there exist $\Omega(|E|/\log^2|V|)$ edge-disjoint even walks; one of the weight functions $w \in \mathcal{W}$ will give a nonzero circulation to all of these even walks. With this w, every such even walk will lose at least one edge in Reduce(G, w); so the number of edges in Reduce(G, w) would be smaller by a factor of $1 - \Omega(1/\log^2|V|)$.

Formally we prove the following two lemmata.

Lemma 6.4. *If $G = (V, E)$ is a matching-covered connected graph with $(1-\epsilon)|E| < |V|$ for some constant $\epsilon < 1/9$, then the number of worms in any maximal set of node-disjoint worms in G is at least $c_1|E|$ for some constant $c_1(\epsilon) > 0$.*

Lemma 6.5. *If $G = (V, E)$ is a matching-covered connected graph on $|V| > 2$ vertices with $(1-\epsilon)|E| \geq |V|$ for some constant $\epsilon > 0$, then there exist $c_2|E|/\log^2|V|$ many edge-disjoint even walks in G for some constant $c_2(\epsilon) > 0$.*

Let us prove lemma 6.3 using lemmas 6.4 and 6.5.

Proof of lemma 6.3. First consider the case where G is a connected graph. Then we can directly apply lemmas 6.4 and 6.5 for some fixed $\epsilon < 1/9$ to see that we either find $c_1|E|$ worms, or there exist $c_2|E|/\log^2|V|$ many edge-disjoint even walks. In the former case, after contracting the worms, the number of edges gets reduced by a factor of $1 - c_1$. In the latter case, let $C_1, C_2, \ldots, C_k$ be the edge-disjoint even walks, and let $w \in \mathcal{W}$ be the weight vector such that $\langle w, \mathrm{sign}(C_i)\rangle \neq 0$. Note that w is guaranteed to exist by lemma 3.16. Now in the call to $\textsc{Reduce}(G, w)$, every C_i loses an edge by lemmas 3.14 and 3.15; either an edge becomes disallowed, or it gets shrunk when contracting tight odd sets (we might contract a superset of the conflicting tight odd set, but that still shrinks every edge inside).

So in either case, one of the candidate graphs in U in algorithm 3 will have a factor of $1 - c_3/\log^2|V|$ fewer edges compared to $|E|$ for some constant $c_3 > 0$.

Now consider the case where G is not connected. Then we can apply the argument to each connected component that is not an isolated edge. We can further assume the same weight vector w works for all connected components. Now if H_1 is the graph obtained from shrinking worms, and H_2 is the result of $\textsc{Reduce}(G, w)$, then we know that the average number of edges in H_1 and H_2 for each connected component is at most $1 - c_3/2\log^2|V|$ times the number of edges in the connected component. So one of H_1, H_2 must have at most $(1 - c_3/2\log^2|V|)$ times as many non-isolated edges as G. $\square$

In the rest of this section we prove lemmas 6.4 and 6.5.

First we prove lemma 6.4. To do this, we prove that the total number of worms is large, and then prove that a maximal node-disjoint set of them must also be large.

Lemma 6.6. *Suppose that $G = (V, E)$ is a graph with no vertices of degree 0 or 1. Then the number of worms in G is at least $9|V| - 8|E|$.*

Proof. Consider a charging scheme, where we allocate a budget of 1 to each edge, and the edge distributes its budget between its two endpoints. We then sum up the charge on all vertices and use the fact that this sum is exactly $|E|$.

Let $e = \{u, v\}$ be an edge. If neither u nor v is of degree 2, let the edge give $1/2$ to u, and $1/2$ to v. If both u and v are of degree 2, we allocate the budget the same way by splitting it equally between u and v. The only remaining case is when one of u and v has degree 2 and the other has degree at least 3; by symmetry let us assume that $\deg(u) = 2$ and $\deg(v) \geq 3$. Then we allocate $5/8$ to u and $3/8$ to v.

Now let us lower bound the charge that each vertex v receives. Note that the minimum amount v receives from any of its adjacent edges is $3/8$, so an obvious lower bound is $3\deg(v)/8$. If $\deg(v) \geq 3$, this is at least $9/8$. Now consider the case when $\deg(v) = 2$. Then v receives at least $1/2$ from each of its adjacent edges. If one of the neighbors of v is not of degree 2, then the charge that v receives will be at least $1/2 + 5/8 = 9/8$. The only possible case where v does not receive at least $9/8$ is when it is of degree 2, and both of its neighbors are also of degree 2 (the center of a worm), in which case it receives 1.

Now let k be the number of worms. Then, by the above argument the total charge on all the vertices is at least

$$\frac{9}{8}(|V| - k) + k \leq |E|.$$

Rearranging yields $k \geq 9|V| - 8|E|$. □

Proof of lemma 6.4. We know that the number of worms is at least $9|V| - 8|E| = (1 - 9\epsilon)|E|$. Now consider the conflict graph of worms, where nodes represent worms, and edges represent having an intersection. It is easy to see that any worm can only intersect at most 4 other worms. So the degrees in this conflict graph are bounded by 4. By fact 3.3, any maximal node-disjoint set of worms will contain at least $(1 - 9\epsilon)|E|/5$ many worms. So we can take $c_1(\epsilon) = (1 - 9\epsilon)/5$ which is positive for $\epsilon < 1/9$. □

Now it only remains to prove lemma 6.5. We will first prove that there are many edge-disjoint cycles in a non-sparse graph. Then either half of the these cycles are even, in which case we are done, or half of them are odd. We then show how to pair the odd cycles and connect them with paths to get many edge-disjoint even walks.

We first prove the following lemma. A proof of this lemma can be found in [CPR03], but for the sake of completeness we provide it here.

Lemma 6.7. *In a graph $G = (V, E)$ there exists a collection of edge-disjoint cycles with at least the following number of cycles:*

$$\frac{|E| - |V|}{2\log_2 |V|}.$$

Proof. We prove this by induction on $|V| + |E|$. We have several cases:

i) If there are any loops in the graph, we extract that as one of our cycles, and remove the edge from the graph. The promised quantity goes down by $1/(2\log_2|V|)$ which is $\leq 1/2$. So from now on we assume that G has no loops.

ii) If there are any two parallel edges e, e', we extract those as a cycle of length 2, and remove both from the graph. The promised number of edge-disjoint cycles goes down by $2/(2\log_2|V|) \leq 1$. So adding the cycle we extracted fulfills the promise. From now on we assume that G is simple.

iii) If G has any vertices of degree 0: We can simply remove it and the promised quantity grows.

iv) If G has a vertex of degree 1: We can also remove this vertex. This operation does not change the numerator but shrinks the denominator, which results in a larger promised quantity.

v) If G has a vertex v of degree 2: Let e, e' be the two adjacent edges to v. Remove v, e, e' from the graph, and place a new edge e'' between the two former neighbors of v. By doing this, both $|V|$ and $|E|$ go down by 1. So now the promised number of edge-disjoint cycles becomes larger. By induction we find them, and now we replace the edge e'' if it is used at all in a cycle, by the path of length two consisting of e, e'. Since e'' appears in at most one cycle, this operation preserves edge-disjointness.

vi) Finally if G is a simple graph with no vertices of degree ≤ 2, it must have a cycle of length at most $2\log_2|V|$. If we prove this, we are done by induction, because we can remove the edges of this cycle and the promised quantity goes down by at most 1. Now to prove the existence of this cycle, assume the contrary, that the length of the minimum cycle of the graph is at least $2\log_2|V| + 1$. Pick a vertex v and look at all simple paths of length at most $\log_2|V|$ going out of v. The number of paths of length i is at least twice the number of paths of length $i-1$. This is because every path of length $i-1$ ending at a vertex u can be extended in at least $\deg(u) - 1 \geq 1$ ways, and none of these extensions will intersect themselves, otherwise we would get a cycle of length $\log_2|V| + 1$. So in the end, the total number of such paths will be $> 2^{\log_2|V|} = |V|$, which means that two of the paths must share an endpoint. But now from the union of these two paths, we can extract a cycle of length at most $\log_2|V| + \log_2|V| = 2\log_2|V|$.

□

If some of these cycles from lemma 6.7 are odd, we need to pair them up and connect them with paths. We use a spanning tree to do this.

Fact 6.8 ([For proof see Lemma 20 in AV18]). *Consider a tree T with an even number of tokens placed on its vertices, with possibly multiple tokens on each vertex. There is a pairing, i.e., a partitioning of tokens into partitions of size two, such that the unique tree paths connecting each pair are all edge-disjoint.*

Lemma 6.9. *Suppose that there are 2ℓ edge-disjoint cycles of odd length in a matching-covered connected graph $G = (V, E)$. Then G contains at least $\Omega(\ell^2/|E|)$ many edge-disjoint even walks.*

Proof. We will pair up the odd cycles by paths connecting each pair. This will create ℓ even walks, but they might not be edge-disjoint. We will then show how to extract $\Omega(\ell^2/|E|)$ *edge-disjoint* even walks out of them.

Consider a spanning tree T of G. For each of the 2ℓ odd cycles, pick an arbitrary vertex, and put a token on that vertex. Now we have an even number of tokens on the vertices. We can pair up these tokens, so that the unique tree paths (of possibly length 0) connecting each pair are edge-disjoint, see lemma 6.9.

Now for each pair of odd cycles C_1, C_2 whose tokens got paired up, we create an even walk. Let P be the tree path connecting tokens from C_1 and C_2. If P has no common edges with C_1, C_2 we can simply create our even walk, but this is not guaranteed to happen. So instead, traverse P from C_1's token to C_2's token and look at the last exit from C_1; afterwards look for the first time any vertex of C_2 is visited. This portion of P is a subpath connecting C_1 and C_2 having no common edged with either. We use C_1, C_2 and this subpath of P to create our even walk.

So far we have created ℓ even walks, but they might not be edge-disjoint. The odd cycles are edge-disjoint, as are the paths connecting them, but one of the paths might share an edge with an unrelated odd cycle. This also means that no edge e can be shared between more than two even walks; e can be used once as part of an odd cycle, and once as part of a path.

Now consider the number of edges in each even walk. If we sum this over all even walks, we get at most $2|E|$, since each edge can appear in at most two even walks. So the average number of edges in an even walk is $\leq 2|E|/\ell$. By Markov's inequality at least half of the even walks, $\ell/2$ of them, will have at most twice this average number of edges, $4|E|/\ell$. Now create a conflict graph where nodes represent these $\ell/2$ even walks, and an edge is placed when the two even walks share an edge. The degree of each node is at most $4|E|/\ell$. So if pick a maximal independent set in this conflict graph, it will consist of at least $\Omega(\ell^2/|E|)$ many even walks. □

We are finally ready to prove lemma 6.5.

Proof of lemma 6.5. First note that if our graph is not an isolated edge and is matching-covered it must contain at least one even cycle. This is because there must be at least two perfect matchings in the graph, and in the symmetric difference of them, we can find one such cycle.

Because we are guaranteed to have at least 1 cycle, we can simply show that asymptotically we can extract $\Omega(|E|/\log^2|V|)$ many edge-disjoint even walks. Then the asymptotic statement translates to the more concrete bound of $c_2|E|/\log^2|V|$.

When $(1-\epsilon)|E| \geq |V|$, by lemma 6.7, we have at least $\epsilon|E|/2\log_2|V| = \Omega(|E|/\log|V|)$ cycles. If at least half of them are of even length, we are done. Otherwise we get $\Omega(|E|/\log|V|)$ odd cycles. Perhaps by throwing away one of them, we can assume the number of odd cycles we have is even. Then we can apply lemma 6.9 to obtain $\Omega(|E|/\log^2|V|)$ many edge-disjoint walks. This finishes the proof. □

7 Discussion

An obvious open question is to build on the quasi-NC algorithms of Gurjar and Thierauf [GT17] and Gurjar, Thierauf, and Vishnoi [GTV17] to obtain pseudo-deterministic RNC algorithms for linear matroid intersection and for finding a vertex of a polytope with faces given by totally unimodular constraints.

The phenomenon identified in section 1.2 clearly deserves to be studied in depth. To the best of our knowledge, there are only two algorithmic results for bipartite matching that have not been extended to general graphs. The first is obtaining a fully polynomial randomized approximation scheme for counting the number of perfect matchings [JSV04]; this is also among the outstanding open problems of theoretical computer science today. The second is obtaining an $O(m^{10/7})$ algorithm for maximum matching [Mad13], which beats the earlier algorithms for sparse graphs.

References

[AV18] Nima Anari and Vijay V. Vazirani. "Planar Graph Perfect Matching is in NC". In: *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Oct. 2018. DOI: 10.1109/focs.2018.00068.

[CGS12] Marek Cygan, Harold N Gabow, and Piotr Sankowski. "Algorithmic Applications of Baur-Strassen's Theorem: Shortest Cycles, Diameter and Matchings". In: *arXiv preprint arXiv:1204.1616* (2012).

[CPR03] Alberto Caprara, Alessandro Panconesi, and Romeo Rizzi. "Packing cycles in undirected graphs". In: *Journal of Algorithms* 48.1 (2003), pp. 239–256.

[DKR10] Samir Datta, Raghav Kulkarni, and Sambuddha Roy. "Deterministically isolating a perfect matching in bipartite planar graphs". In: *Theory of Computing Systems* 47.3 (2010), pp. 737–757.

[Edm65a] Jack Edmonds. "Maximum matching and a polyhedron with 0, 1-vertices". In: *Journal of research of the National Bureau of Standards B* 69.125-130 (1965), pp. 55–56.

[Edm65b] Jack Edmonds. "Maximum matching and a polyhedron with 0, 1-vertices". In: *Journal of research of the National Bureau of Standards B* 69.125-130 (1965), pp. 55–56.

[EV18] David Eppstein and Vijay V Vazirani. "NC Algorithms for Perfect Matching and Maximum Flow in One-Crossing-Minor-Free Graphs". In: *arXiv preprint arXiv:1802.00084* (2018).

[FGT16] Stephen Fenner, Rohit Gurjar, and Thomas Thierauf. "Bipartite perfect matching is in quasi-NC". In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM. 2016, pp. 754–763.

[GG11] Eran Gat and Shafi Goldwasser. "Probabilistic Search Algorithms with Unique Answers and Their Cryptographic Applications." In: *Electronic Colloquium on Computational Complexity (ECCC)*. Vol. 18. 2011, p. 136.

[GG15] Shafi Goldwasser and Ofer Grossman. "Perfect Bipartite Matching in Pseudo-Deterministic RNC." In: *Electronic Colloquium on Computational Complexity (ECCC)*. Vol. 22. 2015, p. 208.

[GT17] Rohit Gurjar and Thomas Thierauf. "Linear matroid intersection is in quasi-NC". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. ACM. 2017, pp. 821–830.

[GTV17] Rohit Gurjar, Thomas Thierauf, and Nisheeth K Vishnoi. "Isolating a vertex via lattices: Polytopes with totally unimodular faces". In: *arXiv preprint arXiv:1708.02222* (2017).

[HK73] John E Hopcroft and Richard M Karp. "An nˆ5/2 algorithm for maximum matchings in bipartite graphs". In: *SIAM Journal on computing* 2.4 (1973), pp. 225–231.

[Joh87] Donald B Johnson. "Parallel algorithms for minimum cuts and maximum flows in planar networks". In: *Journal of the ACM (JACM)* 34.4 (1987), pp. 950–967.

[JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. "A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries". In: *Journal of the ACM (JACM)* 51.4 (2004), pp. 671–697.

[Kar73] Alexander V Karzanov. "O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh; title translation: On finding maximum flows in networks with special structure and some applications". In: *Mathematicheskie Voprosy Upravleniya Proizvodstvom* (1973).

[KUW85] Richard M Karp, Eli Upfal, and Avi Wigderson. “Are search and decision programs computationally equivalent?” In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. ACM. 1985, pp. 464–475.

[KUW86] Richard M Karp, Eli Upfal, and Avi Wigderson. “Constructing a perfect matching is in random NC”. In: *Combinatorica* 6.1 (1986), pp. 35–48.

[Lov79] László Lovász. “On determinants, matchings, and random algorithms.” In: *FCT*. Vol. 79. 1979, pp. 565–574.

[LP09] László Lovász and Michael D Plummer. *Matching theory*. Vol. 367. American Mathematical Soc., 2009.

[Lub86] Michael Luby. “A simple parallel algorithm for the maximal independent set problem”. In: *SIAM journal on computing* 15.4 (1986), pp. 1036–1053.

[Mad13] Aleksander Madry. “Navigating central path with electrical flows: From flows to matchings, and back”. In: *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*. IEEE. 2013, pp. 253–262.

[MN89] Gary L Miller and Joseph Naor. “Flow in planar graphs with multiple sources and sinks”. In: *Foundations of Computer Science, 1989., 30th Annual Symposium on*. IEEE. 1989, pp. 112–117.

[MV00] Meena Mahajan and Kasturi R Varadarajan. “A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs”. In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. ACM. 2000, pp. 351–357.

[MV80] Silvio Micali and Vijay V Vazirani. “An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs”. In: *Foundations of Computer Science, 1980., 21st Annual Symposium on*. 1980, pp. 17–27.

[MVV87] Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. “Matching is as easy as matrix inversion”. In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM. 1987, pp. 345–354.

[PR82] Manfred W Padberg and M Ram Rao. “Odd minimum cut-sets and b-matchings”. In: *Mathematics of Operations Research* 7.1 (1982), pp. 67–80.

[RV89] Michael O Rabin and Vijay V Vazirani. “Maximum matchings in general graphs through randomization”. In: *Journal of Algorithms* 10.4 (1989), pp. 557–567.

[San18] Piotr Sankowski. “NC Algorithms for Weighted Planar Perfect Matching and Related Problems”. In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[ST17] Ola Svensson and Jakub Tarnawski. “The matching problem in general graphs is in quasi-NC”. In: *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*. Ieee. 2017, pp. 696–707.

[Val79] Leslie G Valiant. “The complexity of computing the permanent”. In: *Theoretical computer science* 8.2 (1979), pp. 189–201.

[Vaz89] Vijay V Vazirani. “NC algorithms for computing the number of perfect matchings in K3, 3-free graphs and related problems”. In: *Information and computation* 80.2 (1989), pp. 152–164.

[Vaz94] Vijay V Vazirani. “A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{|V|}|E|)$ general graph maximum matching algorithm”. In: *Combinatorica* 14.1 (1994), pp. 71–109.