NC Algorithms for Weighted Planar Perfect Matching and Related Problems

Piotr Sankowski
Institute of Informatics, University of Warsaw
sank@mimuw.edu.pl

Abstract

Consider a planar graph $G = (V, E)$ with polynomially bounded edge weight function $w : E \to [0, \mathsf{poly}(n)]$. The main results of this paper are NC algorithms for finding minimum weight perfect matching in G. In order to solve this problems we develop a new relatively simple but versatile framework that is combinatorial in spirit. It handles the combinatorial structure of matchings directly and needs to only know weights of appropriately defined matchings from algebraic subroutines.

Moreover, using novel planarity preserving reductions, we show how to find: maximum weight matching in G when G is bipartite; maximum multiple-source multiple-sink flow in G where $c : E \to [1, \mathsf{poly}(n)]$ is a polynomially bounded edge capacity function; minimum weight f-factor in G where $f : V \to [1, \mathsf{poly}(n)]$; min-cost flow in G where $c : E \to [1, \mathsf{poly}(n)]$ is a polynomially bounded edge capacity function and $b : V \to [1, \mathsf{poly}(n)]$ is a polynomially bounded vertex demand function. There have been no known NC algorithms for these problems previously.

2012 ACM Subject Classification Theory of computation → Network flows, Theory of computation → Shared memory algorithms, Theory of computation → Pseudorandomness and derandomization

Keywords and phrases planar graph, NC algorithms, maximum cardinality matching, maximum weight matching, min-cost flow, maximum multiple-source multiple-sink flow, f-factors

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.97

Related Version A full version of the paper is available at [36], `https://arxiv.org/abs/1709.07869`.

Acknowledgements I would like to thank anonymous reviewers for helpful comments on this and previous version of this paper. This work was partially supported by grant 2014/13/B/ST6/01811 of Polish National Science Center and ERC StG grant TOTAL no. 677651.

1 Introduction

In this paper we study deterministic parallel algorithms for the maximum planar matching problem. In particular, we concentrate our attention on the NC class where we are given polynomially many processors which need to solve the problem in polylogarithmic time. The fundamental algorithmic challenge is to compute a maximum cardinality matching, or a maximum weight matching, or a minimum weight perfect matching.

So far in the case of NC algorithms for planar graphs there seemed to be no tools to attack the most general case and we were only able to find perfect matchings in planar bipartite graphs [30, 29]. This might be bit surprising as in non-planar graph, perfect matchings are essentially as powerful as maximum cardinality matchings, see e.g., [34] for a reduction. Such reductions either make two copies of the graph, where each vertex is connected with its copy, or add a set of vertices that are connected to every vertex in the graph. However, there exists no similar planarity preserving reduction, and so the usefulness of algorithms for planar perfect matchings is limited. Hence, in order to find a maximum cardinality matching we needed to retool to less efficient polynomial time algorithms [14, 18, 19] that are limited to bipartite case only. Or, alternatively, we could find a 2-approximate solution [21]. The lack of such planarity preserving reduction is one of the reasons why computing maximum cardinality matchings in almost linear time was recognized as one of the core problems in planar graph algorithms [4]. This despite the fact that almost linear time algorithms for the perfect matching problem existed before [31, 11].

Formally speaking, our results are as follows. Consider a planar graph $G = (V, E)$ with polynomially bounded edge weight (cost) function $w : E \to [0, \mathsf{poly}(n)]$. A *matching* in G is an independent set of edges, whereas a *perfect matching* is a matching that is incident to every vertex in G. We start by giving a relatively simple NC algorithm for finding a perfect matching in G. Then we extend this algorithm to finding *minimum weight perfect matching*, i.e., a perfect matching M of G with minimum total weight $w(M)$. Our algorithm is combinatorial in spirit as it does not manipulate a fractional matching, but only requires to know weights of appropriately defined minimum weight perfect matchings. These weights can be computed using Kasteleyn's Pfaffian orientation of planar graphs and standard algebraic techniques. The algorithm is based on the fundamental notion of balanced duals that was developed in [7] as well as in [16]. This idea allows to define and construct a dual solution in a unique way. In particular, a very simple NC algorithm for constructing a family of blossoms of this dual follows by a direct application of the algorithm given in [7].

The next problem we consider is the *minimum weight f-factor problem*, where for a given $f : V \to [1, \mathsf{poly}(n)]$ we need to find a set of edges F of minimum total cost $w(F)$ such that $\deg_F(v) = f(v)$ for all $v \in V$. Typically, this problem is reduced to the perfect matching problem via vertex splitting [15, 3]. These reductions, however, do not preserve planarity. Our contribution, is to show a new planarity preserving reduction, that allows to solve the f-factor problem in NC. Due do space limitations of this paper, this reduction and the next ones are given in the full version of this paper available on arXiv [36].

The following implication of this result was rather surprising (at least for the author). We show that the maximum weight bipartite planar matching problem can be efficiently reduced to the minimum weight non-bipartite planar perfect matching problem. Thus implying the first known NC algorithm for finding maximum weight matching in bipartite planar graphs. Moreover, our reduction preserves the size of the graph, thus any further development for weighted perfect matching in planar graphs will imply similar results for maximum weight bipartite matchings. There seem to be no easy way to extend this result to non-bipartite case, and we note that finding NC algorithms for this problem remains open. Still, we report partial progress on this problem by showing the first $o(n)$ time PRAM algorithm for maximum cardinality matching – again available in the full version of this paper.

Finally, we consider directed flow networks where capacity of edges is given by a polynomially bounded capacity function $c : E \to [1, \mathsf{poly}(n)]$ and vertex demand is given by a polynomially bounded demand function $b : V \to [1, \mathsf{poly}(n)]$. Our aim is to compute a min-cost flow that obeys edge capacities and satisfies all vertex demands. We give the first known NC algorithm for this problem resolving the long standing open problem from [31]. This result is further extended to finding maximum multiple-source multiple-sink flow in planar graphs.

Related Work

The key result that allowed development of NC algorithms for the perfect matching problems in planar graph is Kasteleyn's idea [25]. He showed that the problem of counting perfect matchings in planar graphs is reducible to determinant computation. Thus [6] implied an NC algorithm checking whether a planar graph contains a perfect matching. In [41] this result was extended to $K_{3,3}$-free graphs. Counting, however, does not allow to construct a perfect matching. The first algorithm for constructing perfect matchings was given in [31] but only in the bipartite case. Another solution that is based on different principles was developed in [29]. A partial solution for the non-bipartite case was given in [27] where an algorithm for computing half integral solution was shown. Finally, an algorithm for non-bipartite planar graphs has been proposed in [2]. The result of [2] extends to finding minimum weight perfect matchings as well. Here, we give an independent proof of this result.[1] In comparison to [2], our paper gives a simpler algorithm for constructing a perfect matchings in NC than the one in [2]. Moreover, we show several nontrivial extension of this results, i.e., to maximum weight bipartite matching, to minimum weight f-factor, or min-cost flow. We note that previously we knew how to solve only (non-weighted) maximum cardinality matching in bipartite graphs in NC [22]. However, this result contains a gap in the argument, as is based on [9] which embeds planar graph into a grid graph what does not preserve the size of maximum matchings. This problem can be solved by using [26] or [8, 40]. Moreover, in [1] it was shown how to construct perfect and weighed perfect matchings in NC when the number of perfect matchings is polynomially bounded.

When allowing randomization (i.e., when considering RNC complexity) the problem can be solved even in general non-planar graphs [24, 17, 33, 35]. All of these Monte Carlo algorithms can be turned into Las Vegas ones using [23]. Finally, we note that we are unaware of any previous parallel deterministic algorithms for weighted non-bipartite problems like minimum weight perfect matching or f-factor problems. Bipartite versions of these problem have some solutions that require at least polynomial $\Omega(n^{2/3})$ time [14, 18, 19]. Finally, we note that very recently it was shown that the general problem is solvable in quasi-NC, i.e., in polylogarithmic time using quasi-polynomially many processors [12, 38]. Moreover, the bipartite problem can be solved in pseudo-NC, i.e., in polylogarithmic time, using polynomially many processors and polylogarithmically many random bits [20].

2 Preliminaries

$G = (V, E)$ denotes an n-vertex, embedded, undirected graph. This embedding partitions the plane into maximal open connected sets and we refer to the closures of these sets as the *faces* of G. The *dual* G^* of G is a multigraph having a vertex for each face of G. For each edge e in G, there is an edge e^* in G^* between the vertices corresponding to the two faces of G adjacent to e. We identify faces of G with vertices of G^* and since there is a one-to-one correspondence between edges of G and edges of G^*, we identify an edge of G with the corresponding edge in G^*.

[1] Our framework was developed independently from [2] and was posted on arXiv on the same day [36]. Nevertheless, the author's initial write-up was very "crude" and contained some gaps. In particular, Algorithm 1 was described in a confusing way without stating which parts should happen in parallel. Moreover, we aimed to give, an alternative to [7], construction of matching duals that was incorrect. This, however, is a known result, so in this paper we just cite [7] instead. Finally, the presentation of this paper was greatly improved with respect to [36], while keeping the original framework. Moreover, consequences for more general problems have been added.

The number of faces is denoted by f. For a subset of vertices $U \subseteq V$, $\delta(U)$ denotes all edges $uv \in E$ having $|\{u,v\} \cap U| = 1$. We write $\delta(u)$ for $\delta(\{u\})$, $x(F)$ for $\sum_{e\in F} x_e$ and $\deg_F(u) = |F \cap \delta(u)|$.

The linear programming formulation of the minimum weight perfect matching problem and its dual is as follows [10]. An *odd set* has odd size; Ω is the collection of odd subsets of V of size ≥ 3.

LP of minimum weight perfect matching

$$\min \sum_{e\in E} w(e)x_e$$
$$\begin{aligned} x(\delta(v)) &= 1, \forall v \in V \\ x(\delta(U)) &\geq 1, \forall U \in \Omega \\ x_e &\geq 0, \forall e \in E \end{aligned}$$

LP of the dual prolem

$$\max \sum_{v\in V} \pi_v + \sum_{U\in\Omega} \pi_U$$
$$\begin{aligned} \pi_u + \pi_v + \sum_{U\in\Omega,\ uv\in\delta(U)} \pi_U &\leq w(uv), \quad \forall uv \in E \quad (*) \\ \pi_U &\geq 0, \qquad \forall U \in \Omega \end{aligned}$$

The variables x_e in the primal indicate when an edge is included in the solution. The dual problem has variables π_v for each vertex v and π_U for each odd set U.

Moreover, a graph G is *factor critical* if for all $v \in V$ after removing v the graph has a perfect matching. A *laminar family* is a collection $\mathcal{B}$ of subsets of V such that for every pair $X, Y \in \mathcal{B}$ either $X \cap Y = \emptyset$, or $X \subseteq Y$, or $Y \subseteq X$. We use the existence of the following dual.

▶ **Lemma 1** (implicitly in [10]). *If the dual LP is feasible, then there exists an optimal dual solution $\pi : V \cup \Omega \to \mathbb{R}$ that:*
1. *the set system $\{U \in \Omega : \pi_U > 0\}$ forms a laminar family,*
2. *for each $U \in \Omega$ with $\pi_U > 0$, the graph denoted by G_U obtained by contracting each set $\{S \in \Omega : S \subset U, \pi_S > 0\}$ to a point is factor critical.*

An optimum dual solution π satisfying the above conditions is a *critical dual solution.* A set $U \in \Omega$ such that $\pi_U > 0$ is a *blossom* w.r.t. π. An important idea that is used in almost all algorithms for weighted matching is that after computing the dual we can work with a non-weighted problem. This non-weighted problem is defined in the following way: leave only *tight* edges in the graph, i.e., there is equality in (*); find a perfect matching *respecting* all blossoms $\mathcal{B}$, i.e., such that for all $B \in \mathcal{B}$ we have $|M \cap \delta(B)| = 1$. By duality slackness any matching obtained this way is a minimum weight perfect matching.

Laminar family of sets is equipped with a natural parent-child relation and can be seen as a forest. We assume this tree is rooted at V and call the resulting tree the *laminar tree*. We note that given a laminar family it is straightforward to deduce the parent-child relation, i.e., parent of a set B is the minimal set containing B, whereas children of B are maximal sets contained in B. Hence, whenever working with a laminar family we assume that the laminar tree is available as well as it can be easily computed in NC. A useful property of this view is that tree T has a *vertex separator*, i.e., there exists a vertex v such that the size of every connected component of $T - x$ is at most $\frac{|T|}{2}$.

Basic NC Algorithms

Our algorithm builds upon the following NC algorithms for computing:
- components and a spanning forest of an undirected graph [37, 5],
- paths in a directed graph – this can be done by repetitive matrix squaring,
- a maximal independent set in a graph [28],
- a vertex separator of a tree – by computing the numbers of vertices in each subtree using any of the standard techniques [32, 39].

Consider a graph G with an edge weight function $w : E \to [0, \mathsf{poly}(n)]$. For a vertex v we denote by G_v the graph $G - v$. If G has an even number of vertices then M_v denotes some *minimum weight almost perfect matching* in G_v, i.e., a minimum weight matching that misses exactly one vertex. If G has an odd number of vertices then M_v denotes some minimum weight perfect matching in G_v. M_v is not defined in a unique way, but its weight $w(M_v)$ is. In our algorithms we will only use these weights that can be computed using standard techniques (see full version for the proof).

▶ **Corollary 2.** *For a graph $G = (V, E)$ with edge weights $w : E \to [0, \mathsf{poly}(n)]$ we can in NC:*

- *for a given vertex $v \in V$, compute the weight $w(M_v)$,*
- *for a given edge $e \in E$, check whether e is* allowed*, i.e., belongs to some minimum weight perfect matching.*

Observe that the set of allowed edge is a subset of tight edges. Hence, when we remove all not allowed edges only tight edges are left in the graph.

Let us now state the following implication of the results in [7]. Basically, assuming that all $w(M_v)$ are given, Algorithm 5 from [7] gives an NC procedure for computing the blossoms of the critical dual solution.

▶ **Lemma 3** (based on Lemma 6.19 [7])**.** *Let $G = (V, E)$ be undirected connected graph where edge weights are given by $w : E \to \mathbb{Z}$ and where every edge is allowed. Given all values $w(M_v)$ for $v \in V$, the blossoms of the critical dual solution can be computed in NC.*

Algorithm 5 from [7] actually constructs a critical dual with an additional property which is called balanced. In a *balanced dual* the root of the laminar tree is required to be the central vertex of this tree as well. This makes balanced duals unique. Thus when one constructs them in parallel, all processors construct the same solution, and it can be implemented in NC.

3 The High Level Idea: Walks and Blossoms

This section aims to introduce two core ideas of our algorithm that allow us to reduce the size of the graph for the recursion, as well as a high level idea that reveals around them. We will first give an algorithm for finding a perfect matching in a graph. However, we will view the problem as weighted and seek a minimum weight perfect matching. The weighed view is useful as we will find even length walks in the graph and introduce weights on them. These weights will either cause some edges to become not allowed, or induce a blossom as shown by the following lemma. Hence, till Section 7, where we show how to handle weighted case, $w : E \to [0, \mathsf{poly}(n)]$ denotes weights defined by the algorithm as the graph considered is unweighted.

To make it more precise we say that a closed walk C in graph G is *semi-simple* if it contains an edge that appears on C only once. By e_C we denote arbitrary such edge on C.

▶ **Lemma 4.** *Consider $G = (V, E)$ with edge weight function $w : E \to [0, \mathsf{poly}(n)]$. Let C be an even semi-simple closed walk in G. Let $w(e_C) = 1$ and let $w(e) = 0$ for all $e \in C - e_C$. Then, either some edge of C is not allowed or some edge of C is in $\delta(B)$ for some blossom B.*

Proof. Assume by a contradiction that all edges of C are allowed and there is no blossom intersecting C. Hence, by complementary slackness conditions we have that $\pi_x + \pi_z = w(xz)$ for all edges $xz \in C$. In particular, for the edge $u_C v_C = e_C$ we need to have $\pi_{u_C} + \pi_{v_C} = 1$, whereas for all other edges $uv \in C - e_C$ we have $\pi_u + \pi_v = 0$. Now consider edge $zy \in C$ which is next to xz. By subtracting equalities for edge xz and zy we obtain $\pi_x - \pi_y = w(xz) - w(zy)$.

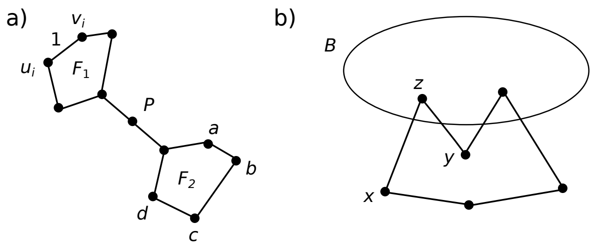

Figure 1 Figure a) shows a non-simple even length closed walk composed out of two faces F_1, F_2 and a path P. We have $\pi_a + \pi_b = 0$, $\pi_b + \pi_c = 0$ and $\pi_c + \pi_d = 0$, what gives $\pi_a - \pi_c = 0$ and $\pi_a + \pi_d = 0$. In Figure b) even length closed cycle does not contain an edge inside a blossom. In such a case $\pi_z + \pi_x + \pi_B = 0$ and $\pi_z + \pi_y + \pi_B = 0$, so $\pi_x - \pi_y = 0$.

If yz and xz are not equal to e_C we have $\pi_x - \pi_y = 0$. In general, we have $\pi_x - \pi_y = 0$ for any two vertices at even distance along path $C - e_C$. And $\pi_x + \pi_y = 0$ for any two vertices at odd distance along path $C - e_C$. Note that the distance from u_C to v_C along $C - e_C$ is odd, so we obtain $\pi_{u_C} + \pi_{v_C} = 0$, what leads to contradiction. See Figure 1 a) for an illustration. ◀

Blossoms are natural objects to recurse on, as by duality slackness there must be exactly one edge of any perfect matching M that belongs to $\delta(B)$ for any blossom B. Thus, in the recursion, we can find an almost perfect matching outside of B, an almost perfect matching inside of B and then combine them by matching one edge in $\delta(B)$ – see Section 6. However, having an edge in $\delta(B)$ does not directly guarantee that the size of the graph reduces in the recursion, as the same blossom can intersect many closed walks. We need the following stronger observation for this.

▶ **Lemma 5.** *Consider $G = (V, E)$ with edge weight function $w : E \to [0, \mathsf{poly}(n)]$. Let C be an even semi-simple closed walk in G and assume that all edges on C are allowed. Let $w(e_C) = 1$ and let $w(e) = 0$ for all $e \in C - e_C$. Then there exist edges $e_1, e_2 \in C$, such that $e_1 \in E(G \setminus B)$ and $e_2 \in E(B)$ for some blossom B.*

Proof. For contradiction assume that no edge of C is in $E(B)$ – the other case is symmetric. Hence, there can only be vertices z in $E(B)$ such that their both incident edges xz and zy on C are in $\delta(B)$. In such a case we have $\pi_x + \pi_z + \pi_B = w(xz)$ and $\pi_z + \pi_y + \pi_B = w(zy)$. Thus $\pi_x - \pi_y = w(xz) - w(zy)$, i.e., the contribution of π_B when subtracting these equalities cancels. See Figure 1 b) for illustration. Thus B does not contribute anything to the telescoping sum along $C - e_C$ and we reach similar contradiction as in the previous lemma. ◀

Hence, when recursing on the inside of the blossom or on the outside of the blossom we reduce the graph size as well. However, to obtain an NC algorithm we need to reduce the size of the graph by a constant factor, so we need to have $\Omega(n)$ edge disjoint even closed walks. The following lemma, that was implicitly proven in [27], becomes handy now.

▶ **Lemma 6.** *In a 2-connected planar graph G with f faces we can find $\Omega(f)$ edge disjoint even semi-simple closed walks in NC.*

Intuitively, the proof of this lemma puts faces into pairs that are incident. Now, either a pair contains an even face and thus this face is semi-simple, or both faces are odd. In the later case we can build an even semi-simple closed walk by walking around both faces.

The above lemma shows that in order to have many even semi-simple closed walks we just need to guarantee that the graph has many faces. Let us say that a planar graph G is *simplified*, if there are no degree 1 vertices and no two vertices of degree 2 are incident. The next lemma shows that such graphs have many faces.

▶ **Lemma 7.** *Let $G = (V, E)$ be a simplified planar graph with a perfect matching, then G has at least $\frac{n}{4} + 2$ faces.*

Proof. By Euler's formula $n - m + f = 2$, where m is the number of edges, and f is the number of faces. Let V_2 be the set of degree 2 vertices in G. Consequently, the vertices in $V \setminus V_2$ have degree at least 3. As G has a perfect matching and no two degree 2 vertices are incident, all vertices in V_2 need to be matched to vertices in $V \setminus V_2$, i.e., $|V_2| \leq |V \setminus V_2|$. By using this inequality we have.

$$2m = \sum_{v \in V} \deg(v) = \sum_{v \in V_2} \deg(v) + \sum_{v \in V \setminus V_2} \deg(v) \geq 2|V_2| + 3|V \setminus V_2| \geq \frac{5}{2}|V_2| + \frac{5}{2}|V \setminus V_2| = \frac{5n}{2}.$$

By plugging this inequality into Euler's formula we obtain $f = 2 + m - n \geq 2 + \frac{5n}{4} - n = 2 + \frac{n}{4}$. ◀

A simplified graph has many faces and thus many edge disjoint even length closed walks. We can assign weights to each of these closed walks separately. This way either many edges become not allowed, or there exists a family of blossoms containing many edges inside. The main technical part of the algorithm is to handle this family of blossoms. The algorithm has the following steps.

1. Simplify the graph as shown in Section 4. First, we take care of degree 1 vertices, by removing not allowed edges and matching edges incident to degree 1 vertices. Next, we contract paths composed of degree 2 vertices. Finally, we find perfect matchings in each connected component separately. Due to removal of not allowed edges each connected component is 2-connected as well.
2. Using Lemma 6 we find many edge disjoint even length closed walks. We assign weights to even closed walks using Lemma 4 and we remove not allowed edges. Next, we find blossoms of the critical dual using Lemma 3. These steps are described in Section 5.
3. Finally, we recurse on a critical dual as explained in Section 6, where we show how to construct a perfect matching that respects all blossoms in the dual. This construction is recursive and calls back step 1 for subgraphs of G that do not have any blossoms.

The recursion depth in this procedure is $O(\log n)$ as either $\Omega(n)$ edges become not allowed, or there are many edges separated by blossoms. In the second case for each blossom B_e there is an edge e_1 inside and an edge e_2 outside of it – see Figure 2 a). We observe that blossom divide the graph into regions and each lowest level recursive call goes to one of these regions, i.e., we recourse on inside and outside of some blossom as long as there are some blossoms intersecting the current subgraph. As visualized on Figure 2 b) there are many edges that are not incident to each region, as one of the edges from each pair (e_1, e_2) needs be outside of this region. Hence, the size of each of these regions decreases by a constant factor with respect to the original graph. This will be proven more formally in Section 5 where we analyze the running time of the algorithm.

4 Simplifying the Graph

The first ingredient of our algorithm is the following entry procedure – Algorithm 1 that simplifies the graph and assures that we are working with 2-connected graphs.

The following lemma proves the correctness of this algorithm.

▶ **Lemma 8.** *Algorithm 1 executes Algorithm 2 on a simplified 2-connected graph.*

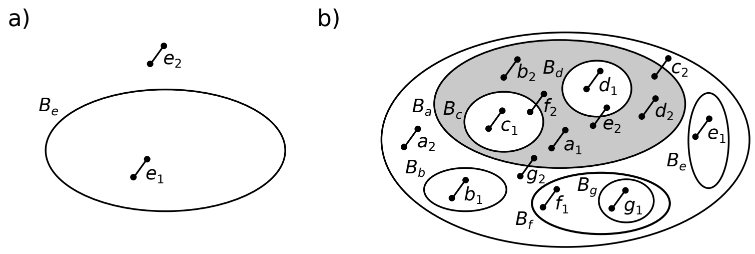

Figure 2 For each blossom B_e there needs to be a pair of edges: e_1 inside and edge e_2 outside of B_e – Figure a). Figure b): when we recurse from Algorithm 4 to Algorithm 1 the blossom tree is empty, and we are recursing on one of the regions into which blossoms divide the plane – marked with gray. Observe that at least one edge from each pair is not incident to this region.

Algorithm 1 Simplifies graph G and seeks perfect matchings in its 2-connected components.

```
 1: Remove all not-allowed edges from $G$.
 2: Add edges incident to degree 1 vertices to $M$ and remove them from $G$.
 3: for all maximal paths $p$ composed out of degree 2 vertices do        ▷ In parallel
 4:     if $p$ has odd length then
 5:         remove $p$ and connect vertices incident to $p$ with an edge $e_p$.
 6:     else
 7:         remove $p$ and add a single vertex $v_p$ connected to vertices incident to $p$.
 8: for all connected components $C \in \mathcal{C}$ of $G$ do             ▷ In parallel
 9:     find perfect matching $M_C$ using Algorithm 2.
10: Extend matching $M$ on paths of degree 2 vertices.
11: return $M \cup \bigcup_{C \in \mathcal{C}} M_C$.
```

Proof. We first observe that after removing not allowed edges, an edge e incident to a vertex of with degree 1 needs to belong to all perfect matchings. Hence, it's both endpoints need to be of degree 1. After matching such edges all vertices have degree 2 or higher. Now note that the manipulation of the path p does not change degrees of other vertices. In the algorithm such path is either replaced by an edge, or single vertex. Hence, afterwards degree 2 vertices cannot be incident. Thus G is simplified and its all connected components are simplified.

Now, for contradiction assume that a connected component C is not 2-connected, i.e., there exists an articulation point v. Consider the connected components obtained from C after removal of v. Only one of them can be matched in the perfect matching to v. Thus only this one has odd number of vertices. The remaining components must have even number of vertices and no perfect matching can match them to v. Hence, their edges incident to v are not allowed. However, all not allowed edges were removed by the algorithm, so we reach a contradiction. ◀

Finally, we note that the above algorithm can be implemented in NC. First of all, in order to compute maximal paths composed out of degree 2 vertices, we just need to compute connected components of the subgraph induced by degree 2 vertices. In the final step of the algorithm we need to expand all paths that were replaced during the execution of Algorithm 1. Observe that the matching M in the simplified graph can be extended to the matching in the original graph in a straight-forward way. If p was odd, then depending on whether e_p is matched we either match even or odd edges of p. If p was even, then v_p has degree 2 and it can be matched in one of two possible ways. In these two cases the matching can be extended to the whole path.

Algorithm 2 Finds a perfect matching M in a connected graph G.

```
1: Find a set $\mathcal{F}$ of $\Omega(n)$ edge disjoint even semi-simple closed walks using Lemma 6.
2: Set $w(e) = 0$ for all $e \in E$.
3: for all $C \in \mathcal{F}$ do                                    ▷ In parallel
4:     Set $w(e_C) = 1$.
5: Remove all not-allowed edges from $G$.
6: Compute blossoms of a critical dual $\mathcal{B}$ with respect to $w$ using Lemma 3.
7: Compute a matching $M$ that respects $\mathcal{B}$ using Algorithm 4.
8: Return $M$.
```

5 The Main Routine

Algorithm 2 implements our main procedure. First, we find $\Omega(n)$ even semi-simple closed walks and introduce weights on them. Next, in order to reduce the size of the graph we remove not allowed edges. We then find blossoms of the critical dual solution with respect to these weights and call Algorithm 4 to find a perfect matching that respects all blossoms.

6 Finding a Perfect Matching that Respects a Family of Blossoms

Let $G = (V, E)$ be a graph and let M be a matching in G. An *alternating path* p is a path in G such that edges on p alternate between matched and unmatched. Assume that G is factor critical (e.g., inside of a blossom) and that M_s is an almost perfect matching that misses vertex s. We start by showing how using M_s we construct an almost perfect matching M_v for any v. To this end, we need to find a simple alternating path starting in s and ending in v. Denote by w_M a weight function assigning 0 to edges in M_s and 1 to edges not in M_s.

▶ **Lemma 9.** *Let M_v be the minimum weight almost perfect matching in G_v with respect to w_M, then $2w_M(M_v)$ is the length of the shortest alternating path with respect to M_s from s to v.*

Proof. Observe that the symmetric difference $M_s \oplus M_v$ contains an alternating path with respect to M_s that needs to start at s and end at v. The weight of this path is equal to the number of edges from M_s on it. As this path is alternating, the number of edges of M_v is the same. Thus minimizing $w_M(M_v)$ we minimize the length of an alternating path with respect to M_s from s to v. ◀

Now, we want to construct a graph G_L that will represent all shortest alternating paths from s with respect to M. G_L will be a layered graph, where layer l contains vertices at distance l from s — the distance is measured along alternating paths. For each $v \in V$, let G_L contain two copies v^o and v^e of v. We define $l(v^e) = 2w_M(M_v)$ for all $v \in V$, and $l(v^o) = l(u^e) - 1$ for all $uv \in M$. We add edges of G to G_L only if they connect two consecutive layers given by l – see Figure 3. Every shortest alternating path from s is contained in G_L by Lemma 9. Alteratively, if a path in G_L represents a simple path in G then it is a shortest alternating path in G. However, there are paths in G_L that do not correspond to simple paths in G, i.e., they contain both v^o and v^e for some $v \in V$ – see Figure 3 b). Nevertheless, as every vertex in G is reachable via alternating path, we can modify G_L in such a way that only paths corresponding to simple paths in G remain. This is done using Algorithm 3.

The correctness of this algorithm is established by the next lemma.

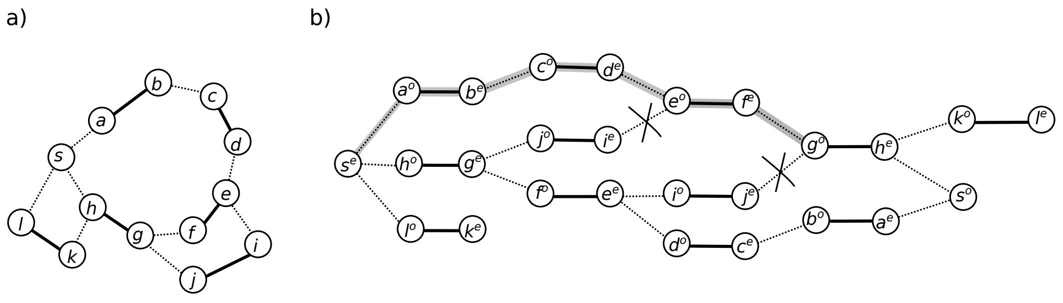

Figure 3 Figure a) shows a factor-critical graph, matched edges are marked with solid lines and s is the free vertex. Figure b) presents layered graph G_L from Algorithm 3. There are two non-simple paths represented by this graph that go from g^e to g^o. These paths are destroyed by the algorithm as g^o is reachable via a simple path p marked with grey.

Algorithm 3 Finds an alternating path with respect to M_s in G from a vertex s to vertex t.

```
 1: For all $v \in V$ compute $w_M(M_v)$.
 2: Let $G_L$ be a graph where $v$ has two copies $v^o$ and $v^e$.
 3: For all $v \in V$ set $l(v^e) = 2w_M(M_v)$.
 4: For all $vu \in M$ set $l(v^o) = l(u^e) - 1$.
 5: Add edges of $G$ to $G_L$ only if they connect vertices in consecutive layers $l$.
 6: for all $v \in V$ do                                                  ▷ In parallel
 7:     for all $u^z$ on some $v^x$-$v^y$ path in $G_L$, where $x, y, z \in \{e, o\}$, $x \neq y$ do   ▷ In parallel
 8:         if there exists $s^e$-$u^z$ path $p$ avoiding $v^x$ in $G_L$ then
 9:             remove all edges entering $u^z$ but the edge on $p$.
10: Return any path from $s^e$ to $t^e$ in $G_L$.
```

▶ **Lemma 10.** *Let G be a factor critical graph and let M_s be an almost perfect matching missing vertex s. An almost perfect matching M_t missing vertex t can be found in NC using Algorithm 3.*

Proof. We first observe that the removal of edges entering u^z from G_L does not affect reachability from s, as the path p from s^e to u^z is left in the graph. Now, by contradiction, assume that at the end of the algorithm there is a path q in G_L that contains both v^e and v^o for some $v \in V$. Without loss of generality assume v^e precedes v^o. As G_L contains all simple alternating paths, there exists an s^e-v^o path p avoiding v^e. Hence, the edge of q entering the first shared vertex with p was removed by the algorithm – see Figure 3 b).[2] ◀

Algorithm 4 constructs a perfect matching M that respects a family of blossoms. Here, we explicitly consider $\mathcal{B}$ as a tree $T_{\mathcal{B}}$, i.e., the vertices of $T_{\mathcal{B}}$ are sets in $\mathcal{B}$ whereas edges in $T_{\mathcal{B}}$ represent child parent relationship. See Figure 4 for an example and an illustration of the recursion. It calls Algorithm 1 that handles the case without blossoms. The next theorem argues about the correctness of this algorithm.

▶ **Lemma 11.** *Algorithm 4 finds a perfect matching M respecting $\mathcal{B}$. The recursion depth of the internal calls of the algorithm to itself is $O(\log n)$.*

[2] The graph constructed in this algorithm can be seen as an extended version of generalized shortest path tree [13]. Alternatively, such tree could be constructed using Algorithm 5 from [7]. This, however, would result in a slightly more complicated solution.

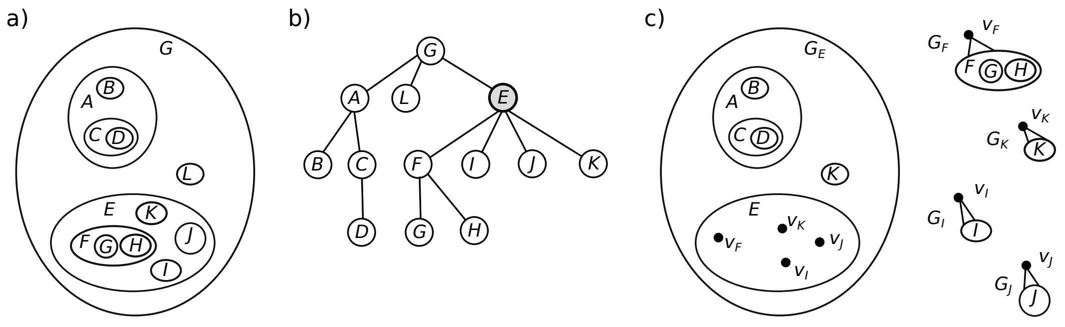

Figure 4 Figure a) shows a laminar family of sets. Figure b) gives the same family using laminar tree. Vertex separator – node E is marked with grey. Figure c) shows the graphs on which we recurse in Algorithm 4 using the marked separator E.

Algorithm 4 Computes a perfect matching M of G respecting family of blossoms $\mathcal{B}$.

```
 1: if $\mathcal{B} = \emptyset$ then
 2:     Return the matching $M$ computed by Algorithm 1 on $G$.
 3: Let $B$ be a vertex separator of $T_{\mathcal{B}}$.
 4: Begin                                              ▷ In parallel with the next loop
 5:     Let $G_B$ be the graph with all children of $B$ contracted.
 6:     Let $T_B$ be $T_{\mathcal{B}}$ with children of $B$ removed.
 7:     Recurse on $G_B, T_B$ to obtain matching $M_B$.
 8: End
 9: for all children $C$ of $B$ in $T_{\mathcal{B}}$ do                        ▷ In parallel
10:     Let $G_C$ be $G$ with all vertices not in $C$ contracted to a vertex denoted by $v_C$.
11:     Let $T_C$ be subtree of $T_{\mathcal{B}}$ rooted at $C$.
12:     Recurse on $G_C, T_C$ to obtain matching $M_C$.
13: for all children $C$ of $B$ in $T_{\mathcal{B}}$ do                        ▷ In parallel
14:     Remove from $G_C$ vertex $v_C$ and let $u_C$ be the resulting free vertex in $G_C$.
15:     Let $e_B$ be the edge of $M_B$ incident to $G_C$.
16:     Let $v_B$ be the endpoint of $e_B$ in $G_C$ after expanding $G_C$.
17:     Apply to $M_C$ an alternating path from $u_C$ to $v_B$ found using Algorithm 3
18: Return $M_B \cup \bigcup_{C \text{ child of } B} M_C$.
```

Proof. We need to argue that M_B can be extended to a perfect matching in the whole graph. Consider a child blossom C of B. By Lemma 3 we know that C is factor-critical, so there exists an almost perfect matching M'_C in C that together with M_B forms a perfect matching. This matching differs from M_C by a single alternating path. Hence, for each child blossom the algorithm is able to combine M_B with M_C by finding a single alternating path.

As we do not recompute the family of blossoms while recursing, we need to argue that the blossoms remain intact. Observe that after contraction of a given blossom all nonintersecting blossoms remain blossoms and the graph remains planar, so we can continue recursing on subtrees of $T_{\mathcal{B}}$.

Finally, the recursion depth of the algorithm to itself is $\Omega(\log n)$, because we recurse on a vertex separator of a tree $T_{\mathcal{B}}$, so the size of the subtrees decreases by a constant factor. ◀

Lemma 11 leads to the correctness of Algorithm 2 and Algorithm 1 as well.

▶ **Theorem 12.** *Algorithm 1 finds a perfect matching in G.*

Algorithm 5 Finds a minimum weight perfect matching in $G = (V, E)$ with respect to the edge weight function $w : E \to [0, \mathsf{poly}(n)]$.

```
Remove all not allowed edges from $G$.
Compute blossoms of a critical dual $\mathcal{B}$ with respect to $w$ using Lemma 3.
Find a perfect matching $M$ respecting $\mathcal{B}$ using Algorithm 4.
Return $M$.
```

We now can turn our attention to arguing about the running time of our algorithm. In this section we are going to quantify the progress related to perturbing weights on each semi-simple closed walks. The entry point to our procedures is Algorithm 1. This algorithm calls Algorithm 2, which constructs a family of blossoms. Now, Algorithm 4 is used to recurse on this family till subgraphs with no blossoms are obtained. At this moment Algorithm 1 is called back. We will argue that each time Algorithm 1 is called in this recursion, the size of the graph decreases by a constant factor.

▶ **Lemma 13.** *The number of edges in G decreases by a constant factor in each recursive call to Algorithm 1.*

Proof. By Lemma 6 and Lemma 7, after graph simplification we have $\Omega(n)$ even semi-simple closed walks in the graph G. Now, by Lemma 4, for each semi-simple closed walks, either one edge becomes not-allowed, or there exists a blossom $B \in \mathcal{B}$ that intersects this closed walk. Hence, either $\Omega(n)$ edges become not-allowed, or we have $\Omega(n)$ closed walks intersected by some blossom. Lemma 5 implies that for each such intersection there exists an edge e_1 inside B and an edge e_2 outside B – see Figure 2 a). Consider a plane embedding of G and of dual graph G^*. Mark boundaries of each blossom $\delta(B)$ in G^* on the plane, thus dividing the plane into maximal open connected sets. We call these sets regions $\mathcal{R}$.

The outside of each region $R \in \mathcal{R}$ contains at least one edge from e_1 and e_2, i.e., there are $\Omega(n)$ edges not incident to a region – see Figure 2 b). We note that when we call Algorithm 1 from Algorithm 4, we recurse onto some region $R \in \mathcal{R}$ with parts of G not in R contracted to vertices. This graph contains only edges incident to a region R, so it does not contain $\Omega(n)$ edges. By Euler's formula the total numer of edges is $\leq 3n - 6$, so when recursing on each region it decreases by a constant factor. ◀

By Lemmas 11 and 13 the recursion depth in Algorithm 2 is $O(\log^2 n)$, thus:

▶ **Corollary 14.** *A perfect matching in a planar graph can be computed in NC.*

7 Minimum Weight Perfect Matching

So far we have assumed that the graph is unweighed and we have coped with the problem of constructing any perfect matching. However, our approach is versatile enough to handle weighted case in a rather straightforward way using Algorithm 5.

Hence, we obtain the following.

▶ **Lemma 15.** *A minimum weight perfect matching in a planar graph $G = (V, E)$ with edge weight function $w : E \to [0, \mathsf{poly}(n)]$ can be computed in NC.*

Proof. Observe that all allowed edges in G need to be tight. By complementary slackness conditions, a perfect matching that is composed out of allowed edges and that respects all blossoms is a minimum weight perfect matching. ◀

We note that maximum weight perfect matching can be computed by using redefined weight function $w'(e) = -w(e) + \max_{g \in E} w(g)$.

8 Planarity Preserving Reductions

Let $G = (V, E)$ be a multi-graph, i.e., we allow parallel edges as well as self-loops. For a function $f : V \to [1, \mathsf{poly}(n)]$, an *$f$-factor* is a set of edges $F \subseteq E$ such that $\deg_F(v) = f(v)$ for every $v \in V$. Without loss of generality we assume that any edge uv has multiplicity at most $\min\{f(u), f(v)\}$. A *minimum f-factor* is an f-factor F with minimum weight $w(F)$. The f-factor problem can be reduced via planarity preserving reduction to minimum weight perfect matching problem.

▶ **Lemma 16.** *Let $G = (V, E)$ be a planar multigraph with edge weight function $w : E \to [0, \mathsf{poly}(n)]$. For a function $f : V \to [1, \mathsf{poly}(n)]$, minimum f-factor can be computed in NC.*

Now consider, a graph $G = (V, E)$ be a bipartite planar graph with edge weight function $w : E \to [0, \mathsf{poly}(n)]$, and a problem of computing a maximum matching in graph G, i.e., a matching of maximum total weight. This problem in turn can be reduced to f-factor problem.

▶ **Lemma 17.** *A maximum matching in a planar bipartite graph $G = (V, E)$ with edge weight function $w : E \to [0, \mathsf{poly}(n)]$ can be computed in NC.*

In the min-cost planar flow problem, we are given a directed planar network $N = (V, E)$. The edges have integral capacities given by $c : E \to [1, \mathsf{poly}(n)]$ and integral costs $a : E \to [0, \mathsf{poly}(n)]$. Moreover, each vertex has integral demand $b : V \to [-\mathsf{poly}(n), \mathsf{poly}(n)]$. Again there exists planarity preserving reduction to f-factor problem.

▶ **Theorem 18.** *Let $N = (V, E)$ be a planar flow network with integral capacities $c : E \to [1, \mathsf{poly}(n)]$, integral costs $a : E \to [0, \mathsf{poly}(n)]$ and integral demands $b : V \to [-\mathsf{poly}(n), \mathsf{poly}(n)]$. The minimum cost flow in N can be computed in NC.*

We can modify the above reduction to handle the case when source and sink demands are not fixed but need to be maximized. We are given a directed planar network $N = (V, E)$ with integral edge capacities given by $c : E \to [1, \mathsf{poly}(n)]$. Moreover, each vertex has integral demand $b : V \to [-\mathsf{poly}(n), \mathsf{poly}(n)]$. Vertices $v \in V$ such that $b(v) \geq 0$ are called *sources* and we require for them $0 \leq f(v) \leq b(v)$. Vertices $v \in V$ such that $b(v) \leq 0$ are called *sinks* and we require $b(v) \leq f(v) \leq 0$. For remaining vertices v, i.e., when $b(V) = 0$ we need to have $f(v) = 0$. The *maximum multiple-source multiple-sink flow* in N is the flow that maximizes value $f(S)$, where S is the set of all sources.

▶ **Theorem 19.** *Let $N = (V, E)$ be a planar flow network with integral capacities $c : E \to [1, \mathsf{poly}(n)]$, integral costs $a : E \to [0, \mathsf{poly}(n)]$ and integral demands $b : V \to [-\mathsf{poly}(n), \mathsf{poly}(n)]$. The minimum cost flow in N can be computed in NC.*

References

1 Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in NC2. In *Proceedings of the 24th Annual Conference on Theoretical Aspects of Computer Science*, STACS'07, pages 489–499, Berlin, Heidelberg, 2007. Springer-Verlag.

2 Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in NC, 2017. `arXiv:arXiv:1709.07822`.

3 R.P. Anstee. A polynomial algorithm for b-matching: An alternative approach. *IPL*, 24:153–157, 1987.

4 G. Borradaile, P. N. Klein, S. Mozes, Y. Nussbaum, and C. Wulff-Nilsen. Multiple-Source Multiple-Sink Maximum Flow in Directed Planar Graphs in Near-Linear Time. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, FOCS '11, pages 170–179, Oct 2011.

5 K.W. Chong and T.W. Lam. Finding Connected Components in $O(\log n \log\log n)$ Time on the EREW PRAM. *Journal of Algorithms*, 18(3):378–402, 1995.

6 L. Csanky. Fast Parallel Matrix Inversion Algorithms. *SIAM Journal on Computing*, 5(4):618–623, 1976.

7 Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic Applications of Baur-Strassen's Theorem: Shortest Cycles, Diameter, and Matchings. *J. ACM*, 62(4):28:1–28:30, 2015.

8 Samir Datta, Arjun Gopalan, Raghav Kulkarni, and Raghunath Tewari. Improved Bounds for Bipartite Matching on Surfaces. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *LIPIcs*, pages 254–265, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

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

10 Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research National Bureau of Standards-B.,*, 69B:125–130, 1965.

11 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. Special Issue on FOCS 2001.

12 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*, STOC '16, pages 754–763, New York, NY, USA, 2016. ACM.

13 H. N. Gabow and P. Sankowski. Algebraic algorithms for b-matching, shortest undirected paths, and f-factors. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '13, pages 137–146, Oct 2013.

14 Harold Gabow and Robert Tarjan. Almost-optimum speed-ups of algorithms for bipartite matching and related problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 514–527, New York, NY, USA, 1988. ACM.

15 Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 448–456, New York, NY, USA, 1983. ACM.

16 Harold N. Gabow. A combinatoric interpretation of dual variables for weighted matching and f-factors. *Theor. Comput. Sci.*, 454:136–163, 2012.

17 Zvi Galil and Victor Y. Pan. Improved processor bounds for combinatorial problems in RNC. *Combinatorica*, 8(2):189–200, 1988.

18 A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya. Sublinear-time parallel algorithms for matching and related problems. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '88, pages 174–185, Oct 1988.

19 A. V. Goldberg, D. B. Shmoys, S. A. Plotkin, and E. Tardos. Interior-point methods in parallel computation. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, FOCS '89, pages 350–355, Washington, DC, USA, 1989. IEEE Computer Society.

20 Shafi Goldwasser and Ofer Grossman. Bipartite Perfect Matching in Pseudo-Deterministic NC. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*,

volume 80 of *LIPIcs*, pages 87:1–87:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

21 Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 219–225, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

22 T. M. Hoang. On the matching problem for special graph classes. In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity*, CCC '10, pages 139–150, June 2010.

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

24 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

25 P. W. Kasteleyn. Dimer statistics and phase transitions. *Journal of Mathematical Physics*, 4(2):287–293, 1963.

26 Arpita Korwar. Finding an NC algorithm for perfect matching in planar graphs. Master's thesis, IIT Kanpur, 2009.

27 Raghav Kulkarni and Meena Mahajan. Seeking a vertex of the planar matching polytope in nc. In Susanne Albers and Tomasz Radzik, editors, *In Proceedings of the 12th Annual European Symposium on Algorithms*, pages 472–483, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

28 M Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 1–10, New York, NY, USA, 1985. ACM.

29 Meena Mahajan and Kasturi R. Varadarajan. A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs (extended abstract). In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 351–357, New York, NY, USA, 2000. ACM.

30 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986.

31 Gary L. Miller and Joseph (Seffi) Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24(5):1002–1017, 1995.

32 Gary L. Miller and John H. Reif. Parallel tree contraction part 1: Fundamentals. In Silvio Micali, editor, *Randomness and Computation*, pages 47–72. JAI Press, Greenwich, Connecticut, 1989. Vol. 5.

33 K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, STOC '87, pages 345–354. ACM Press, 1987.

34 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 118–126, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

35 Piotr Sankowski. Processor Efficient Parallel Matching. *Theory of Computing Systems*, 42(1):73–90, Jan 2008.

36 Piotr Sankowski. Planar perfect matching is in NC, 2017. `arXiv:arXiv:1709.07869`.

37 Y. Shiloach and Uzi Vishkin. An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms*, 3:57–67, 1982.

38 Ola Svensson and Jakub Tarnawski. The Matching Problem in General Graphs Is in Quasi-NC. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '17, pages 696–707. IEEE Computer Society, 2017.

39 R. E. Tarjan and U. Vishkin. Finding biconnected componemts and computing tree functions in logarithmic parallel time. In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '84, pages 12–20, Oct 1984.

40 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012.

41 Vijay V. Vazirani. Nc algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. *Information and Computation*, 80(2):152–164, 1989.