A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs

[Extended Abstract]

Meena Mahajan*
The Institute of Mathematical Sciences
Chennai 600 113, India
meena@imsc.ernet.in

Kasturi R. Varadarajan†
Department of Computer Science
The University of Iowa
Iowa City, IA 52242-1419 USA
kvaradar@cs.uiowa.edu

ABSTRACT

It has been known for a long time now that the problem of counting the number of perfect matchings in a planar graph is in NC. This result is based on the notion of a pfaffian orientation of a graph. (Recently, Galluccio and Loebl [7] gave a P-time algorithm for the case of graphs of small genus.) However, it is not known if the corresponding search problem, that of finding one perfect matching in a planar graph, is in NC. This situation is intriguing as it seems to contradict our intuition that search should be easier than counting.

For the case of planar *bipartite* graphs, Miller and Naor [22] showed that a perfect matching can indeed be found using an NC algorithm. We present a very different NC-algorithm for this problem. Unlike the Miller-Naor algorithm, our approach directly uses the fact that counting is in NC, and it also generalizes to the problem of finding a perfect matching in a bipartite graph of small ($O(\log n)$) genus. It also rekindles the hope for an NC-algorithm to find a perfect matching in a non-bipartite planar graph.

Along the way, we modify the algorithm of Gallucio and Loebl [7] to show that counting the number of perfect matchings in graphs of small genus is in NC.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems
; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms, counting problems*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

General Terms

Algorithms

1. INTRODUCTION

The problem of checking if a given graph has a perfect matching, and the corresponding search problem of finding a perfect matching, have received considerable attention in the field of parallel algorithms. (See for instance [32; 13].) The earliest result was in 1979, when, based on the work of Tutte, Lovasz observed that there is an RNC algorithm for detecting if a given graph has a perfect matching [18]. The search problem of finding a perfect matching in parallel proved to be considerably harder, and was finally shown to be in RNC in 1986 by Karp, Upfal and Wigderson [12]. Subsequently, Mulmuley, Vazirani and Vazirani [25] gave a different algorithm for the search problem.

The outstanding open problem here is to give a deterministic NC algorithm for detecting/searching for a perfect matching in a given graph. Some progress in this direction was achieved with the sublinear time ($O(n^{2/3}\log^3 n)$) parallel algorithm for bipartite matching due to Goldberg, Plotkin and Vaidya [8]. NC algorithms have been obtained for some special cases of graphs; for instance, dense graphs (minimum degree at least $n/2$) [4], regular bipartite graphs [16], strongly chordal graphs [5], graphs with polynomially bounded number of perfect matchings [9], P_4-tidy graphs[26], convex bipartite graphs [6], incomparability graphs [15], claw-free graphs [3], and planar bipartite graphs [22].

The problem of counting the number of perfect matchings in a given graph is #P-hard [30], and a polynomial time algorithm is therefore unlikely for it.

The starting point of the stream of results to which our work belongs is the classic paper of Kastelyn [14] which shows that the number of perfect matchings in a planar graph can be counted in polynomial time. Kastelyn defined the notion of Pfaffian Orientation of a graph. He showed that if a graph G has a pfaffian orientation, then the number of perfect matchings in G can be computed by evaluating a certain determinant. The idea is to associate an orientation with each edge of an undirected graph G, and hence attach signs to each perfect matching. An orientation D of the graph can be

*Part of this work was done when this author was supported by the NSF grant CCR-9734918 on a visit to Rutgers University, USA.

†Parts of this work were done when this author was at DIMACS, Rutgers University, USA and then the Tata Institute of Fundamental Research, Mumbai, India.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
STOC 2000. Portland, Oregon, USA
Copyright 2000 ACM 1-58113-184-4/00/5 ..$5.00

naturally associated with a skew-symmetric adjacency matrix A_D, whose (i,j) entry is 1 if the edge (i,j) is oriented from i to j, and is -1 otherwise. The Pfaffian of the orientation, $Pf(D)$, or equivalently of its skew-symmetric adjacency matrix $Pf(A_D)$, is defined as the sum of the signed weights of all perfect matchings. Clearly, if all edge weights are 1, then $|Pf(D)|$ is bounded by M, the number of perfect matchings. D is said to be a pfaffian orientation if $|Pf(D)|$ actually equals M, i.e. the signs on the matchings do not introduce any cancellations. Linear algebra techniques show that $(Pf(A_D))^2 = Det(A_D)$, so computing the (absolute value of the) Pfaffian of any orientation is computationally easy. Kastelyn's major breakthrough was in proving that every planar graph has a pfaffian orientation, which can be found in polynomial time. His algorithm can in fact be implemented in NC. Combined with any NC algorithm for evaluating the determinant, this yields an NC algorithm for counting the number of perfect matchings in planar graphs. (For a direct NC implementation, see [21].) Little [17] extended Kastelyn's result to show that every $K_{3,3}$-free graph has a pfaffian orientation, and Vazirani [31] obtained an algorithmic version of this result which puts the problem of counting the number of perfect matchings in $K_{3,3}$-free graphs in NC. See [19; 13] for a complete treatment of the topic.

Recently, Gallucio and Loebl [7] extended the results of Kastelyn to the case of graphs of small genus. They proved that the generating function of the perfect matchings of a graph of genus g may be obtained as a linear combination of 4^g pfaffians. This means that one can count the number of perfect matchings by first finding 4^g orientations, and then taking a linear combination of 4^g corresponding determinants. This gives a polynomial time algorithm for any graph with genus upto $O(\log n)$. A modification of their method, along with the algorithm of [21] that computes the pfaffian along with its sign in NC, gives a counting algorithm that can in fact be implemented in NC.

Surprisingly, an NC algorithm for finding a perfect matching in a planar graph has proved quite elusive. This situation is rather intriguing, as it contradicts our intuition that search should be easier than counting. For the case of bipartite planar graphs, Miller and Naor [22] succeeded in giving an NC algorithm to find a perfect matching. Their method reduces this problem to that of finding a circulation in the planar graph, which then corresponds to computing shortest paths in its planar dual.

Our results and method

Our main results are stated in Theorem 1, Theorem 2, and Theorem 3.

We give a conceptually simple and elegant NC algorithm for finding a perfect matching in a planar bipartite graph. Our method is quite different from that of Miller and Naor [22]. Unlike the Miller-Naor algorithm, our method also generalizes to the case of graphs of upto $O(\log n)$ genus.

Our first observation, which is simple but crucial, is that we can use the counting algorithm to find a point inside the matching polytope. In other words, we can find a convex combination of perfect matchings. This part works for the nonbipartite case as well. For the bipartite case, a convex combination of perfect matchings can be viewed as an assignment of a rational "flow" or "weight" $x_e \in [0,1]$ to each edge e of the graph so that the following *conservation constraint* holds at each vertex v: the sum of the weights on edges incident to v equals one.

We begin the second step by first discarding all edges with zero weight, and then repeat the following as long as there is a cycle C in the remaining graph: We adjust the weights on the edges in C, always maintaining the conservation constraint (and so staying within the matching polytope), till the weight of one of the edges in C becomes zero; we discard from the graph all edges of C whose weight is zero.

At the end of the second step, we are left with an acyclic subgraph (forest) and weights satisfying the conservation constraint. Such an acyclic subgraph can only be a perfect matching.

In order to implement the first step for small genus graphs, we modify the algorithm of Gallucio and Loebl so that it can count the number of perfect matchings in such graphs in NC. The second step generalises from the planar case in a fairly straightforward manner.

It has been known for quite some time that computing an $s-t$ max-flow or min-cut in a planar graph is in NC. All algorithms for these problems use the correspondence between cuts in the graph and cycles in the dual graph. This is also true of the Miller-Naor algorithm for planar bipartite matching, where bipartite matching is viewed as the problem of computing a flow with multiple sources and sinks. (Note that the standard reduction of bipartite matching to $s-t$ max-flow does not maintain planarity.)

This correspondence between cuts and cycles in the dual graph breaks down in graphs of higher genus. We are not aware of any NC algorithm for max-flow or min-cut in such graphs. Our algorithm for bipartite matching does not use this correspondence, which also indicates that our algorithm for the planar case is quite different from the algorithm of Miller and Naor.

Organization of the paper

In Section 3, we present our algorithm to find a perfect matching in a planar graph. In Section 4, we describe its generalization to graphs of fixed genus. We offer our conclusions in Section 5, where we address the possibility of extending our scheme for nonbipartite planar graphs.

2. PRELIMINARIES

A planar embedded graph G partitions the plane into connected regions called *faces*. Let F denote the set of faces of G. Let $G^* = (F, E^*)$ be the *dual graph* of G. Its vertex set is the set F of faces of G. There is a one-to-one correspondence between its edge set E^* and E as follows: for each edge $e \in E$, there is a corresponding edge $e^* \in E^*$ connecting the two faces in G that border e. Note that G^* may have loops and multi-edges even when G does not. However, G^* is always a connected graph, irrespective of whether or not G is. We sometimes refer to G as the *primal graph*. We further denote by G^{**} the graph obtained from G^* by deleting self-loops and replacing multiple edges by a single edge. Let n, m and f (respectively n^*, m^*, f^*, and n^{**}, m^{**}, f^{**}) denote the number of vertices, edges and faces in G (resp. G^* and G^{**}), and let c be the number of components in G. Then $e^{**} \leq e^* = e$, $f^{**} \leq f* = n - c + 1$, and $n^{**} = n^* = f$. Also, $f \in O(e)$.

Euler's formula states that a planar graph with c components satisfies $e + c + 1 = n + f$. Furthermore, if the graph has no self-loops or multiple edges, it also satisfies $e \leq 3n - 6c$.

These definitions generalize naturally if the graph G is embedded on an orientable surface of higher genus. Throughout this paper, surface will mean orientable surface. The genus of a graph G, denoted $\gamma(G)$, is the minimum g for which G can be embedded on a surface of genus g. If the graph is connected and has no self-loops or multiple edges, it satisfies $e \leq 3n + 6(\gamma(G) - 1)$.
A combinatorial embedding is a scheme that specifies, for each vertex v, a cyclic permutation of the neighbours of v. For connected graphs, combinatorial embeddings are in one-to-one correspondence with embeddings on surfaces of appropriately calculated genus.
For a full treatment of this topic, see for instance [29; 33].

3. SEARCH IN PLANAR BIPARTITE GRAPHS

We are given an embedding of a planar bipartite graph $G = (V, E)$, where V is the vertex set and E is the edge set. We assume that G has no loops or multi-edges. In this section, we describe our NC algorithm for finding one perfect matching in G.
We assume that the graph G is given to us as an embedding in some standard form.

3.1 The Algorithm

The algorithm is summarised in the box. We first compute a rational "flow" or "weight" $x_e \in [0, 1]$ on each edge $e \in E$ satisfying the following *conservation constraint*: for each vertex $v \in V$, the *demand* at v, defined to be the sum of the weights of edges incident to v, equals one. To do this, we simply set x_e to be the ratio of the number of perfect matchings in which e participates to the total number of perfect matchings. It is clear that this assignment satisfies the conservation constraint. As described in the summary, we can compute this assignment by using the counting algorithm as a subroutine.
Next, we manipulate the weight on the edges, always maintaining the conservation constraint, till we obtain an integral assignment of weights which corresponds to a perfect matching. In what follows, G will always denote the subgraph consisting of the edges with non-zero weights. When the weight on some edge becomes zero, it is deleted from G. The manipulation of the weights is accomplished by the while loop in the summary. Using the algorithm in Lemma 3, we pick a subset of faces in G such that no two faces share an edge. The number of faces in this subset is a constant fraction of the total number of faces in G. For each face in parallel, we extract a simple cycle C from the edges bounding the face. (Note that such a simple cycle always exists.) Since G is bipartite, C has an even number of edges. Let e denote the edge on C with the smallest weight, and let x denote this weight. We decrease the weights of all edges of C at an even distance from e by x, and increase the weights of all edges of C at an odd distance from e by x. (Since C is an even cycle, "odd distance" (resp. "even distance") from e is well-defined.) We delete from G the edge e and any other edges of C whose weight becomes zero.
In the above step, we can operate on each cycle in parallel because the cycles are edge-disjoint. It is straightforward to check that at the end of the step the weight of any edge is in the range $(0, 1]$, and that the conservation constraint continues to hold. This completes the description of an iteration of the while loop, and also the algorithm.

3.2 Correctness

LEMMA 1. *When the algorithm terminates, the edges of G form a perfect matching.*

PROOF. It is clear that the algorithm terminates with an acyclic graph (i.e. forest, since the graph is undirected) G and a weight $x_e \in (0, 1]$ on edge e of G satisfying the conservation constraint. Let G' be any connected component of G. Due to the conservation constraint, G' is not an isolated vertex. Since G is acyclic, G' is a tree. Let v be any pendant vertex of G', and let (u, v) be the unique edge incident on v. Because the demand at v equals one, the weight of (u, v) must equal one. But since the demand at u does not exceed one, (u, v) is the only edge incident on u as well. Thus, G' consists of just the edge (u, v). It follows that the edge set of G forms a perfect matching. □

3.3 Complexity

LEMMA 2. *The number of perfect matchings in a planar graph can be computed in NC.*

PROOF. Given an embedding of G, [31] shows how to obtain a pfaffian orientation, and [21] shows how to compute the pfaffian. The absolute value of this pfaffian is the number of perfect matchings in G. Further, it follows from [11; 23] (see also [27; 1]) that given the adjacency matrix of G, an embedding can be constructed in NC, allowing the above two steps to be carried out. □

LEMMA 3. *In a planar graph G with f faces, a set of $\Omega(f)$ edge-disjoint faces and $\Omega(f)$ edge-disjoint simple cycles can be found in NC.*

PROOF. A planar graph G as well as its dual G^* and pruned dual G^{**} are sparse graphs. So they do not have too many high degree vertices. In G^{**}, call a vertex high-degree if it has 12 or more neighbours. Since $e^{**} \leq 3n^{**} - 6 = 3f - 6$, there are less than $f/2$ high-degree vertices; the rest are all of low degree. Now consider the subgraph of G^{**} induced by these low degree vertices, and construct any maximal independent set S. For each vertex that we put in S, at most 11 others stay out. So 1 out of every 12 low-degree vertices must be in S; i.e. $S \geq f/24$. And each independent set in G^{**} gives a set of edge-disjoint of faces in G.
To find the set of faces, use any NC algorithm for finding maximal independent sets in G^{**} (see [20] for general graphs, or [10] for planar graphs). If G^{**} has fewer than 24 faces, use a singleton set corresponding to any face as the independent set.
For each face F in the above set, consider the graph with only those edges that bound F. A spanning tree in this graph and a fundamental cycle with respect to it can be found in NC. And since the faces are edge-disjoint, so are these cycles. (If the edges bounding a face do not contain a cycle (i.e. they form a forest), then it must be the case that the entire graph has only one face.) □

LEMMA 4. *In $O(\log n)$ iterations of the while loop, G becomes acyclic.*

PROOF. We argue that the number of iterations of the while loop is $\log |F|$, where F is the initial set of faces. It follows that the number of iterations is $O(\log n)$.

```
Compute M = the number of perfect matchings in G.
If M = 0, exit.
For each e ∈ E, pardo:
        Compute M_e = the number of perfect matchings in G − {e}.
        If M_e = M Then delete edge e
                    Else Assign weight 1 − (M_e/M) to (a, b).
        Endif
while G has a cycle do
        Find a set S of edge-disjoint cycles in G.
        For each cycle C ∈ S, pardo:
                Find the smallest edge weight, say x on edge e.
                Decrease weight of all edges at even distance from e on C by x.
                Increase weight of all edges at odd distance from e on C by x.
                Delete all edges on C whose weight is zero.
endwhile
```

Since any simple cycle divides the plane into two faces, a one-face planar graph is acyclic. Thus it suffices to prove that after each iteration of the while loop, the number of faces in G falls by a constant fraction. This follows from the following two observations. At the beginning of the loop we pick a subset of edge-disjoint faces which constitute a constant fraction of the set of all faces. And at the end of the loop, we delete one edge from each picked face, thus destroying the face. □

THEOREM 1. *Given a planar bipartite graph G on n vertices, there is an algorithm $\mathcal{A}$ which either produces a perfect matching M in G, or reports that G has no perfect matching. $\mathcal{A}$ runs in time $(\log n)^{O(1)}$ using $n^{O(1)}$ processors.*

PROOF. The initial assignment of weights satisfying the conservation constraint can be found in NC (use Lemma 2). Each iteration of the while loop can be implemented in NC. (Use Lemma 3 to find the set of faces. Killing each face is straightforward.) By Lemma 4, the number of iterations of the while loop is $O(\log n)$. □

4. THE EXTENSION TO SMALL GENUS GRAPHS

Let G be a bipartite graph embedded on an orientable surface S_g of genus g. In this section, we describe how the algorithm of the previous section can be extended to find a perfect matching in G. The overall scheme is quite similar to the algorithm of Section 3.

1. Using the counting algorithm as a subroutine, we find a convex combination of perfect matchings of G, i.e. a weight on each edge so that the conservation constraint is satisfied at each vertex of G.

2. As long as the graph G has more than one face remaining, we repeat the following. We find a subset of faces in G such that no two faces share an edge. For each picked face f in parallel, we find a simple cycle C from the edges bounding the face. We manipulate the weight on the edges in C as before, and delete from G the edges with zero weight.

3. The graph G now has only one face, but since it is an embedding on a surface of higher genus, it might still have cycles. We repeat the following until the graph becomes acyclic. We pick any cycle in G, and manipulate the weights on the edges of the cycle until some edge-weight becoms zero. We delete the edges of C with zero weight from the graph.

We are left with an acyclic graph with weights satisfying the conservation constraint. By the argument in Lemma 1, this must be a perfect matching. In the rest of this section, we briefly describe the NC implementations of each of the above steps when the graph genus is logarithmically bounded.

4.1 Input and basic operations.

We now describe the *polygonal representation* of an oriented surface S_g of genus g. Informally, S_g is a sphere with g handles, and its polygonal representation is obtained by cutting the g handles of its space model. The polygonal representation consists of a convex $4g$-gon B_0 whose vertices $a_1, \ldots, a_{4g}$ are numbered clockwise. For $0 \leq i \leq g-1$, the edge $[a_{4i+1}, a_{4i+2}]$ is identified with the edge $[a_{4i+4}, a_{4i+3}]$ via a bridge B_1^i, and the edge $[a_{4i+2}, a_{4i+3}]$ is identified with the edge $[a_{4i+5}, a_{4i+4}]$ via a bridge B_2^i. (The vertex a_{4g+1} is, by convention, the vertex a_1.) The bridges B_1^i and B_2^i together constitute the ith handle of the sphere. We will use $[a, b]$ to denote an edge of the convex polygon B_0 as opposed to an edge of the graph. We will refer to two edges of B_0 that are identified as *partners*.

Thus a graph G embedded on S_g, when translated to the polygonal representation, has its vertices inside B_0. An edge of G might "cross over" from an edge of B_0 to its partner using a bridge. Our algorithm assumes some combinatorial representation of the embedding of G within B_0 (any scheme for representing planar graphs suffices), plus some additional information that identifies points on an edge of G when it crosses over from an edge of B_0 to its partner. Given such an embedding, we can easily modify existing NC-algorithms for planar graphs to perform standard operations like extracting the edges bounding each face of G, or constructing the graph G^* dual to G.

For more details concerning embeddings onto non-planar surfaces, see for instance [29; 33; 7].

4.2 Implementing the algorithm.

In order to implement step 1 of the algorithm, i.e. find a weight function satisfying the conservation constraints, we need an NC-algorithm to count the number of perfect matchings given a polygonal embedding of the graph G. We obtain such an algorithm by appropriately modifying the polynomial time algorithm of Gallucio and Loebl [7] for counting the number of perfect matchings in G.

THEOREM 2. *Let G be a graph embedded on S_g given by its polygonal embedding as above. We can count the number of perfect matchings of G in parallel time $O(g)+(\log n)^{O(1)}$, using $4^g n^{O(1)}$ processors. In particular, when $g \in O(\log n)$, the procedure is in NC.*

PROOF. Gallucio and Loebl actually consider the following generalization of the counting problem. Let w_e be a non-negative integral weight associated with each edge e of the graph G. The weight of a perfect matching M is defined to be $w(M) = \prod_{e \in M} w(e)$, the product of the weights of the edges in M. Define $\mathcal{C}(G)$ to be the sum of the weights of all perfect matchings in the graph; let us call this the *generating function of perfect matchings* of G. When all the weights are equal to one, $\mathcal{C}(G)$ is the number of perfect matchings in G.

Given a polygonal representation of G, they first transform G into another graph G', satisfying certain additional properties, such that $\mathcal{C}(G') = \mathcal{C}(G)$. The transformation as they describe it cannot be implemented in NC. Using a somewhat different transformation which we sketch below and which is in NC, we obtain a graph G' satisfying the required properties.

Next, for each of the $2g$ subgraphs obtained by considering edges in B_0 and in one fixed bridge, Galluccio and Loebl construct pfaffian orientations which all agree on the edges of B_0. By using such an orientation or its reversal on all edges of each bridge, they obtain 4^g different orientations of the edge set of G'. For each orientation, they evaluate the pfaffian of the corresponding skew-symmetric matrix, and then take an appropriate linear combination of the pfaffians to obtain $\mathcal{C}(G')$. It is straightforward to implement this part of their algorithm within the stated resource bounds.

The counting procedure of [7] requires G to be given in its polygonal representation satisfying the following properties:

1. All vertices are embedded in B_0.
2. Vertices on the boundary of B_0 form a cycle embedded on the boundary.
3. Each edge uses at most one of the $2g$ bridges.
4. Endpoints of an edge that uses a bridge are embedded on the boundary of B_0, one vertex on each partner.
5. Each vertex has at most one incident edge not in G_0.
6. G_0 has a perfect matching M_0.

Given any polygonal representation, it is converted to one satisfying these properties by adding dummy vertices and edges and setting the dummy edge weights to 0 or 1, so chosen that the generating function of perfect matchings remains the same.

The method from [7] of satisfying all but the last property is easily seen to be implementable in NC. To ensure the last property, their method requires the computation of a perfect matching in the original graph G, which can be done in P. To satisfy the property in NC, we construct a perfect matching M of the vertices in B_0 using "new" edges that don't belong to G_0. These edges in M are embedded so that they do not cross each other and are assigned weight 0. However the edges in M may cross some edges of G_0; additional local gadgets at each such crossing and an appropriate assignment of weights ensure that the generating function is preserved. □

In each iteration in Step 2 of the algorithm, we find a subset of edge-disjoint faces of G. A subset whose cardinality is a constant fraction of the total number of faces of G can be found using the algorithm of the following lemma.

LEMMA 5. *Let G be a graph embedded on S_g given by its polygonal embedding. We can find a subset of $\Omega(f)$ edge-disjoint faces in G in NC, where f is the number of faces in G, provided $f \geq 4g$.*

PROOF. We construct the dual G^* of G, and delete from G^* any self-loops and multiple copies of any edge to get G^{**}. Let $n^{**} = f$, e^{**}, and f^{**} denote, respectively, the number of vertices, edges, and faces in G^{**}. Recall that G^{**} is a connected graph of genus at most g, and $e^{**} \leq 3n^{**} + 6(g-1)$.

Now, as in Lemma 3, we can argue that at least $n^{**}/2 - g$ vertices of G^{**} have small (less than 12) degree, and so a maximal independent set has size at least $n^{**}/24 - g/12$. Since $n^{**} = f \geq 4g$, this set has size at least $f/48$. And this independent set corresponds to a subset of edge-disjoint faces in G. □

For each face in the subset, we find an appropriate cycle bounding part of the face using the following algorithm.

LEMMA 6. *Let graph G be embedded on S_g, with more than one face, and let f be a face of the embedding. There exists a simple cycle C using edges incident to f with the following property: if we delete any edge in C from G, the face f merges with some adjacent face different from f. Further, such a cycle can be found in NC.*

PROOF. We look at the edges incident to f and throw away any edge that has f on both sides of it. It is easy to see that the remaining set of edges forms a union of simple cycles, and deleting any of these edges causes f to merge with some adjacent face. We pick any one of these cycles. □

LEMMA 7. *In $O(g + \log n)$ iterations of the while loop, G is reduced to a single face graph.*

PROOF. The while loop has two phases: first, when there are at least $4g$ faces in the graph, and second, when the reduced graph has fewer than $4g$ faces.

In the first phase, we select, at the beginning of each iteration, a subset of edge-disjoint faces which constitute a constant fraction of the set of all faces. In the second phase, we pick any one face at the beginning of each iteration.

In either phase, at the end of the loop, we delete one edge from each picked face, such that the picked face then merges with an adjacent face (that was not picked).

We now argue that the number of iterations of the while loop is $O(\log |F| + g)$, where F is the initial set of faces.

($|F| \in O(n^2)$.) After each iteration of the while loop in the first phase, the number of faces in G falls by a constant fraction; clearly, there are at most $O(\log |F|)$ such iterations. And the second phase has no more than $4g$ iterations, since a face is destroyed in each iteration. □

LEMMA 8. *The number of iterations in Step 3 of the algorithm is at most* $2g$.

PROOF. The analogue of Jordan's Curve Theorem on the plane states that if a surface has genus g, then any set of 2g+1 closed curves will make the surface disconnected. (See for instance [2].)
At the beginning of step 3, the graph G is embedded onto S_g and the embedding has a single face. Since any cycle in G will embed on to a closed curve on S_g, it follows that G has at most $2g$ cycles. And each iteration of step 3 destroys at least one cycle. □

THEOREM 3. *Given a bipartite graph G on n vertices along with its embedding on a polygonal representation of an oriented surface of genus g, there is an algorithm $\mathcal{B}$ which either produces a perfect matching M in G, or reports that G has no perfect matching.*
$\mathcal{B}$ runs in time $(\log n)^{O(1)}g$ using $4^g n^{O(1)}$ processors.
In particular, if $g \in O(\log n)$, then $\mathcal{B}$ is an NC algorithm.

PROOF. The bounds for step 1 are stated in Theorem 2. Lemma 5 and Lemma 6 show that each iteration of step 2 and of step 3 is in NC. Lemma 7 and Lemma 8 bound the number of iterations for steps 2 and 3 respectively. □

5. CONCLUSIONS

The obvious question is how to extend this counting-based approach to non-bipartite graphs. As we mentioned before, the first step of our algorithm does work for planar nonbipartite graphs. That is, we can compute a point inside the matching polytope using the counting algorithm. However, we encounter some difficulties in rounding this fractional solution to an integer solution. The difficulties stem from the fact that it is not enough to just maintain the conservation constraint at each vertex. The matching polytope in the nonbipartite case also needs inequalities which say that the weight of any cut isolating an odd subset of vertices is at least one. We have so far not been able to devise an efficient rounding scheme which can also enforce this constraint. However, we think that there is hope for designing a search algorithm that is guided by the fact that we can efficiently compute a point inside the matching polytope.
Note that for planar non-bipartite graphs, a sublinear time ($O(n^{1/2+\epsilon})$) algorithm can be easily devised using separators: find a separator S, match vertices of S sequentially to some unmatched neighbour, then recursively process the unmatched components.
Another open problem is to find embeddings of graphs onto surfaces of small genus in NC, or even to test whether such embeddings exist. No NC algorithms are known for testing or embedding onto surfaces of genus greater than 0. While testing whether a graph has genus at most k is an NP-complete problem [28], for a fixed k the problem is known to be in P and even in linear time [24].

6. ACKNOWLEDGEMENTS

We thank Jan Vondrák for pointing us to the work of Galluccio and Loebl [7]. We thank Jaikumar Radhakrishnan for helping us with the transformation in the proof of Theorem 2. We thank Srikanth Iyengar and S Venkatesh for helpful discussions.

7. REFERENCES

[1] E. Allender and M. Mahajan. The complexity of planarity testing. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science STACS, LNCS Volume 1770*. Springer-Verlag, 2000.

[2] Bonnington and Little. *The foundations of topological graph theory*. Springer Verlag.

[3] M. Chrobak, J. Naor, and M. Novick. Using bounded degree spanning trees in the design of efficient algorithms on claw free graphs. In *Proceedings of the Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science Vol 382*, pages 147–162, Ottawa, Canada, 1989. Springer-Verlag.

[4] E. Dahlhaus, P. Hajnal, and M. Karpinski. On the parallel complexity of Hamiltonian cycles and matching problem in dense graphs. *Journal of Algorithms*, 15:367–384, 1993.

[5] E. Dahlhaus and M. Karpinski. The matching problem for strongly chordal graphs is in NC. Technical Report 855-CS, University of Bonn, 1986.

[6] E. Dekel and S. Sahni. Parallel matching algorithm for convex bipartite graphs and applications to scheduling. *Journal of Parallel and Distributed Computing*, 1(2):185–205, 1984.

[7] A. Galluccio and M. Loebl. On the theory of Pfaffian orientations I: Perfect matchings and permanents. *The Electronic Journal of Combinatorics*, R6, 1999.

[8] M. Goldberg, S. Plotkin, and M. Vaidya. Sublinear time parallel algorithms for matching and related problems. *Journal of Algorithms*, 14:180–213, 1993.

[9] D. Grigoriev and M. Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *Proceedings of 28th IEEE Conference on Foundations of Computer Science*, pages 166–172. IEEE Computer Society Press, 1987.

[10] X. He. A nearly optimal parallel algo for constructing maximal independent set in planar graphs. *Theoretical Computer Science*, 61:33–47, 1988.

[11] J. Ja'Ja' and J. Simon. Space efficient algorithms for some graph-theoretic problems. *Acta Informatica*, 17:411–423, 1982.

[12] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6:35–48, 1986.

[13] M. Karpinski and W. Rytter. *Fast parallel algorithms for graph matching problems*. Oxford University Press, Oxford, 1998. Oxford Lecture Series in Mathematics and its Applications 9.

[14] P. W. Kastelyn. Graph theory and crystal physics. In F. Harary, editor, *Graph Theory and Theoretical Physics*, pages 43–110. Academic Press, 1967.

[15] D. Kozen, U. Vazirani, and V. Vazirani. NC algorithms for comparability graphs, interval graphs, and testing for unique perfect matching. In *Proceedings of FST&TCS Conference, LNCS Volume 206*, pages 496–503. Springer-Verlag, 1985.

[16] G. Lev, M. Pippenger, and L. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Transactions on Computers*, C-30:93–100, 1981.

[17] C. H. C. Little. An extension of Kasteleyn's method of enumerating the 1-factors of planar graphs. In *Combinatorial Mathematics, Proc. 2^{nd} Australian Conference*, pages 63–72. D.Holton, *Lecture Notes in Mathematics*, 403, 1974.

[18] L. Lovasz. On determinants, matchings and random algorithms. In L. Budach, editor, *Proceedings of Conference on Fundamentals of Computing Theory*, pages 565–574. Akademia-Verlag, 1979.

[19] L. Lovasz and M. Plummer. *Matching Theory*, volume 29 of *Annals of Discrete Mathematics*. North-Holland, 1986.

[20] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[21] M. Mahajan, P. R. Subramanya, and V. Vinay. A combinatorial algorithm for Pfaffians. In *Proceedings of the Fifth Annual International Computing and Combinatorics Conference COCOON, LNCS Volume 1627*, pages 134–143. Springer-Verlag, 1999. DIMACS Technical Report 99-39.

[22] G. Miller and J. Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24:1002–1017, 1995.

[23] G. L. Miller and J. H. Reif. Parallel tree contraction, part ii: Further applications. *SIAM Journal on Computing*, 20:1128–1147, 1991.

[24] B. Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, 12(1):6–26, Feb 1999.

[25] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–131, 1987.

[26] I. Parfenoff. An efficient parallel algorithm for maximum matching for some classes of graphs. *Journal of Parallel and Distributed Computing*, 52(1):96–108, 1998.

[27] V. Ramachandran and J. Reif. Planarity testing in parallel. *Journal of Computer and System Sciences*, 49:517–561, 1994.

[28] C. Thomassen. The graph genus problem is NP-complete. *Journal of Algorithms*, 10(4):568–576, Dec 1989.

[29] C. Thomassen. Embeddings and minors. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics*, pages 301–349. Elsevier Science, 1995.

[30] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[31] 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.

[32] V. Vazirani. Parallel graph matching. In J. Reif, editor, *Synthesis of Parallel Algorithms*. Kaufman-Morgan, 1991. chapter 18.

[33] A. T. White. *Graphs, Groups and Surfaces*. North-Holland, Amsterdam, 1973.