A New NC-Algorithm for Finding a Perfect Matching in d-Regular Bipartite Graphs When d Is Small

Raghav Kulkarni

The Department of Computer Science, University of Chicago, Chicago, USA
raghav@cs.uchicago.edu

Abstract. The perfect matching problem for general graphs reduces to the same for regular graphs. Even finding an NC algorithm for the perfect matching problem in cubic (3-regular) or 4-regular graphs will suffice to solve the general problem (see [DK 92]). For regular bipartite graphs an NC algorithm is already known [LPV 81], while [SW 96] give an NC algorithm for cubic-bipartite graphs.

We present a new and conceptually simple parallel algorithm for finding a perfect matching in d-regular bipartite graphs. When d is small (polylogarithmic) our algorithm in fact runs in NC. In particular for cubic-bipartite graphs, our algorithm as well as its analysis become much simpler than the previously known algorithms for the same. Our techniques are completely different from theirs.

Interestingly, our algorithm is based on a method used by [MV 00] for finding a perfect matching in planar-bipartite graphs. So, it is remarkable that, circumventing the planarity, we could still make the same approach work for a non-planar subclass of biparitite graphs.

1 Introduction

The *perfect matching problem* is of particular interest to a variety of people including combinatorists, algorithmists and complexity-theorists. In parallel settings, the complexity of the problem is still unresolved. In this paper we propose an approach based on an interior point method for the *perfect matching problem*, especially for bipartite graphs.

Given a graph $G = (V, E)$ with n vertices and m edges, a *perfect matching* in G is a subgraph M of G such that every vertex in G has degree exactly 1 in M. The *decision* problem is to determine whether G has a perfect matching. The *search* problem is to construct a perfect matching, if one exists. Both these problems have a *Randomized* NC algorithm ([KUW 86], [MVV 87]) but are not known to be in deterministic NC even for bipartite graphs.

For special classes of graphs, however, there are deterministic NC algorithms. These classes include bipartite-planar graphs [MN 95], regular bipartite graphs [LPV 81], small-genus bipartite graphs, [MV 00] bipartite graphs having polynomially bounded permanent [GK 87]. In [SW 96], another NC algorithm was presented for cubic bipartite graphs.

T. Calamoneri, I. Finocchi, G.F. Italiano (Eds.): CIAC 2006, LNCS 3998, pp. 308–319, 2006.

This work is motivated by [MV 00], where a simple and elegant NC algorithm was presented for bipartite planar graphs. There, counting was used to get a point inside the perfect matching polytope and then starting from this point, navigating outwards, a vertex of the prefect matching polytope was reached. However, it is not clear how to use this approach for non-planar graphs as the analysis of the algorithm in [MV 00] crucially uses planarity.

In this paper, we use the same approach for regular bipartite graphs. In particular, we show that for d-regular bipartite graphs, when d is polylogarithmic, one can get a point inside the perfect matching polytope and navigate outwards to seek a vertex of the perfect matching polytope in NC. Our first observation is that getting a point inside the perfect matching polytope of a regular bipartite graph is easy. This observation is simple but crucial. Next we crucially exploit the notion of *2-3 graphs* (developed by [KM 04] in the context of planar graphs) coupled with a delicately chosen potential function. The techniques we use are very elementary.

Our main contribution here is making the approach taken by [MV 00] and [KM 04] for planar graphs, work for a non-planar subclass of bipartite graphs, namely d-regular bipartite graphs having polylogarithmic d. These two subclasses of bipartite graphs are totally different in structure. This suggests that the same approach should work for much more general subclass of bipartite graphs. Moreover, our algorithm is conceptually simple, especially for cubic-bipartite graphs, and we hope that the techniques here will generalize.

The organization of this paper is as follows. Section 2 builds preliminaries used in this paper. Section 2.1 defines some matching polytopes. A special class of graphs, namely 2-3 graphs is defined in Section 2.2. Section 3 talks about the cycle space of 2-3 graphs. Section 3.1 describes the proceudure *find-big-even* and section 3.2 has the procedure *manip* which will be used in the algorithms of next sections. Section 4 describes a new NC algorithm to find a perfect matching in bipartite cubic graphs. In Section 5, we generalize the algorithm given in Section 3 for d-regular bipartite graphs. This gives an NC algorithm when d is small (polylogarithmic). We discuss the possibility of generalizing our algorithm for regular non-bipartite gaphs and conclude with some remarks and immediate open questions in Section 6.

2 Preliminaries

In this paper, we allow graphs to have self loops and multiple edges between two vertices, i.e., the term multigraph is abused as graphs unless specified otherwise.

The *weighted graph* $\mathcal{G} = (G, g)$ is a graph $G = (V, E)$, together with the weight function $g : E \to \mathbb{Q}$. We denote by g_e the weight of the edge e, i.e., $(g_e := g(e))(\forall e \in E)$. When we talk about the weighted graph $\mathcal{G}$ then it is assumed that G is the underlying unweighted graph and g is the weight function on the edges of G.

2.1 Some Polytopes Related to Matching

Given a graph G on n vertices and m edges, following are the definitions of the some of the polytopes in $\mathbb{R}^m$ associated with it. Every matching M in G

corresponds to a $0-1$ vector v_M in $\mathbb{R}^m$ namely $v_M(e) = 1$ if $e \in M = 0$ otherwise. We will use this corresponds throughout this paper.

Matching Polytope. The matching polytope, $\mathcal{M}(G)$, is the convex hull of all the matchings in G.

If G is a bipartite graph then it turns out that $\mathcal{M}(G)$ is given by the following contraints: $(g_e \geq 0)(\forall e \in E(G))$ and $(\sum_{e \perp v} g_e \leq 1)(\forall v \in V)$ ($e \perp v$ means e incident on v). However, for general graphs we need additional constraints.

Perfect Matching Polytope. The perfect matching polytope $\mathcal{PM}(G)$, is the convex hull of all the perfect matchings in G. For bipartite graphs, $\mathcal{PM}(G)$ is described by the following inequalities: $(g_e \geq 0)(\forall e \in E(G))$ and $(\sum_{e \perp v} g_e = 1)(\forall v \in V(G))$. However, this is not the case for general graphs. We need some additional contraints called "odd-cut" constraints (see [LP 86]). For a weighted graph, the *minimum weight perfect matching polytope* is the convex hull of all the perfect matchings of the minimum weight.

Fractional Perfect Matching Polytope (or 2-matching Polytope). The fractional matching polytope, $\mathcal{FPM}(G)$ is defined by the following inequalities: $(g_e \geq 0)(\forall e \in E(G))$ and $(\sum_{e \perp v} g_e = 1)(\forall v \in V(G))$.

It turns out that for bipartite graphs $\mathcal{FPM} = \mathcal{PM}$ but this is not true in general. Note that any interior point (of any matching polytope) corresponds to a weighted graph . We will treat an interior point as a weighted graph.

2.2 The Notion of 2-3 Graphs

A graph G is said to be a *2-3 graph* if
(a) the degree of every vertex of G is either 2 or 3
(b) both the neighbours of a degree 2 vertex in G have degree 3 in G.

A *3-bounded path* in a 2-3 graph G is a path in which both the end points have degree 3 in G but rest of the vertices on the path have degree 2 in G.

The definition of 2-3 graphs is motivated by the following theorem.

Theorem 1 (KM 04). *Given (i) graphs $\overline{G}$, G and interior points $\overline{\mathcal{G}} = (\overline{G}, \overline{g})$ and $\mathcal{G} = (G, g)$ of $\mathcal{FPM}(\overline{G})$ and $\mathcal{FPM}(G)$ respectively and (ii) a "backtracker" $\pi : E(\overline{G}) \rightarrow \{0,1\} \times E(G) \bigcup \{\mathbf{0}, \mathbf{1}\}$ there is an NC procedure which outputs*

(i) a 2-3 graph G' and an interior point $\mathcal{G}' = (G', g')$ of $\mathcal{FPM}(G')$ such that $(g'_{e'} > 0)(\forall e' \in E(G'))$ and
(ii) a "backtracker" $\pi' : E(\overline{G}) \rightarrow \{0,1\} \times E(G') \bigcup \{\mathbf{0}, \mathbf{1}\}$, where a "backtracker" $\pi : E(\overline{G}) \rightarrow \{0,1\} \times E(G) \bigcup \{\mathbf{0}, \mathbf{1}\}$ is a function such that
if $\pi : \overline{e} \mapsto \mathbf{0}$ then $\overline{g}_{\overline{e}} = 0$,
if $\pi : \overline{e} \mapsto \mathbf{1}$ then $\overline{g}_{\overline{e}} = 1$,
if $\pi : \overline{e} \mapsto (i, e)$ then $\overline{g}_{\overline{e}} = g_e$ if $i = 0$ and $\overline{g}_{\overline{e}} = 1 - g_e$ if $i = 1$.

We call such a procedure as *make2-3* and we assume that it takes as input weighted graphs $\overline{\mathcal{G}}$, $\mathcal{G}$ and a "backtracker" $\pi : E(\overline{G}) \to \{0,1\} \times E(G) \bigcup \{\mathbf{0},\mathbf{1}\}$ and outputs a weighted 2-3 graph $\mathcal{G}'$ and a "backtracker" which is given by the function $\pi' : E(\overline{G}) \to \{0,1\} \times E(G') \bigcup \{\mathbf{0},\mathbf{1}\}$. In our algorithm, we will write this shortly as the following: $(\mathcal{G}', \pi') = make2-3(\overline{\mathcal{G}}, \pi, \mathcal{G})$. The procedure is quite simple and straightforward (see [KM 04]). Now we briefly describe why these "backtrackers" come into the picture.

The "backtracker" $\pi : E(\overline{G}) \to \{0,1\} \times E(G) \bigcup \{\mathbf{0},\mathbf{1}\}$, is useful in the following way. If we could find a perfect matching in G then we can translate this into a perfect matching of $\overline{G}$ by using the "backtracker." The translation is obvious: Let M be the perfect matchingin G then $M(e) = 1$ if e is in M and 0 otherwise. The perfect matching $\overline{M}$ is given as follows: if $\pi : \overline{e} \mapsto \mathbf{0}$ then $\overline{M}(\overline{e}) = 0$,
if $\pi : \overline{e} \mapsto \mathbf{1}$ then $\overline{M}(\overline{e}) = 1$,
if $\pi : \overline{e} \mapsto (i, e)$ then $\overline{M}(\overline{e}) = M(e)$ if $i = 0$ and $\overline{M}(\overline{e}) = 1 - M(e)$ if $i = 1$. $\overline{M}$ is just the set of edges in $\overline{G}$ with weight 1.

Thus "backtrackers" are useful and the 2-3 graphs are quite general with respect to these "backtrackers" and we will use the *make2-3* procedure at every step of our algorithm to convert the graph into a 2-3 graph and will continue searching for a perfect matching in this new graph as using the "backtracker" one can backtrack a perfect matching of the original graph in NC.

3 The Cycle Space of a 2-3 Graph

In this section we will prove some results which have significance in the further sections. We also describe a procedure *find-big-even* which will be used as a subroutine in our algorithm.

Given a graph $G = (V, E)$, consider the vector space $\mathbb{F}_2^{|E|}$. Any subgraph H of G corresponds to a vector v_H in $\mathbb{F}_2^{|E|}$ and vice versa. The correspondence can be given as $v_H(e) = 1$ iff $e \in E(H)$ and $v_H(e) = 0$ otherwise. The *cycle space* of G is the subspace spanned by the vectors corresponding to cycles in G.

An element of the cycle space of G is called a *cycle vector* in G. Every cycle vector corresponds to a disjoint union of cycles. A cycle vector is called an *even cycle vector* if all the cycles in the cycle vector are of even length.

Given a spanning tree T of a graph G, every non-tree edge is in a unique cycle in G, called a *fundamental cycle* in G with respect to T. The non-tree edge is called a *fundamental edge* in G with respect to T. The set of all such fundamental cycles corresponding to the nontree edges, is called the set of *fundamental cycles* in G with respect to T. The set of fundamental cycles in G with respect to a spanning tree T of G forms a basis for the cycle space of G.

We call a vertex a 3-vertex if its degree is exactly 3. We denote by $V^{(3)}(G)$ the set of all 3-vertices in G.

Let $k := |V^{(3)}(G)|$ be the number of 3-vertices in G. We say that an even cycle vector in G is a *big even cycle vector* if it contains $\Omega(k)$ 3-vertices of G.

3.1 Finding a *Big Even Cycle Vector*

We shall describe the procedure *find-big-even* which will find a *big even cycle vector* in a bipartite 2-3 graph.

find-big-even(G)
1. find a spanning tree T of G
2. find S = the set of fundamental cycles in G with respect to T
3. return $\mathcal{C} = \oplus_{C \in S} C$.

Lemma 1. *The procedure* find-big-even *runs in NC.*

Proof. Easy to check. □

To show that it outputs a *big even cycle vector*, we prove the following lemma.

Lemma 2. *If G is a bipartite 2-3 (multi)graph having k 3-vertices then the dimension of its cycle space is $\Omega(k)$. (at least $k/2$)*

Proof. Consider any spanning tree T of G. We claim that the number of nontree edges is $\Omega(k)$. Let l be the number of vertices in G having degree 2. Hence, total number of edges in G is $l + \frac{3}{2}k$. The number of tree edges is $l + k - 1$. Hence, the number of nontree edges is greater than $\frac{k}{2}$. □

Lemma 3. find-big-even (G) *finds a* big even cycle vector *in G.*

Proof. Consider a spanning tree T of G. S is the set of fundamental cycles in G with respect to T. $\mathcal{C} = \oplus_{C \in S} C$. $|\mathcal{C}| = \Omega(k)$. Now, when we add all the fundamental cycles in G with respect to T, all the fundamental edges are still preserved because each fundamental edge is in a unique fundamental cycle. In a 2-3 graph, every edge has at least one endpoint of degree 3. Hence we still have $\Omega(k)$ 3-vertices of G in $\mathcal{C}$. This proves that $\mathcal{C}$ is the required *big even cycle vector.* □

3.2 Manipulating the *Big Even Cycle Vector* in *Right* Direction

Here we describe procedures *simple-manip* and *manip* to manipulate even cycles so that some of the edges get destroyed. It will be easy to check that both these procedure run in NC.

Given an interior point $\mathcal{G}$ of $\mathcal{FPM}(G)$ and an even cycle C in G one can move to another interior point $\overline{\mathcal{G}}$ of $\mathcal{FPM}(G)$ using the procedures *simple-manip* or *manip.*

simple-manip ($\mathcal{C}$, $\mathcal{G}$)
1. do parallely for every cycle $C \in \mathcal{C}$
2. pick a minimum weight edge (say e) in C of weight w (say)
3. add w to the weights of edges in C at odd distance from e
4. subtract w from the weights of edges in C at even distance from e.

Lemma 4 (MV 00). *The procedure* simple-manip *runs in NC, preserves bipartiteness, and we are still inside $\mathcal{FPM}$ at the end of* simple-manip.

Proof. Note that *simple-manip* preserves the contraint at every vertex that the sum of the weights of edges incident on that vertex is exactly 1. So, we are still inside the perfect matching polytope. □

This procedure will be used as a subroutine for the algorithm in the next section for finding a perfect matching in cubic bipartite graphs. Somewhat more sophisticated version of *simple-manip* is required for d-regular bipartite graphs (in Section 6). That procedure is described here as *manip.*

The even cycle manipulation with C. Suppose we have an interior point of $\mathcal{FPM}(G)$ and C is an even cycle in G.

1. Fix an edge f in C.
2. Let C_{odd} be the edges in C at odd distance from f.
3. Similarly, let C_{even} be the edges in C at even distance from f.
4. Choose one of the sets C_{odd} or C_{even}.
5. Let e be the minimum weight edge in that set and w be the weight of e.
6. Add w to the weight of all edges in C at odd distance from e.
7. Subtract w from the weight of all edges at even distance from e.
8. We get a new interior point of $\mathcal{FPM}(G)$ in which weight of e is 0.

This procedure is called as *even cycle manipulation.* This can be done for any closed walk of even length. If we have an even cycle vector then we can manipulate each of its even cycle.

Lemma 5. *The* even cycle manipulation *can be done in NC and it doesn't leave the $\mathcal{FPM}$ of the graph.*

Proof. Trivial to check. □

Note that when we manipulate an even cycle C in G, we had choice between C_{odd} and C_{even} in Step 4 of the *even cycle manipulation.* We now define what is the *right direction* which we have to choose during the algorithm.

Making the right choice. Suppose C contains the following 3-vertices in G: $v_1, v_2, \ldots, v_\ell$. Let ${e_i}^{odd}$ be the edge in C incident on v_i and belonging to C_{odd}. Let ${e_i}^{even}$ be the edge in C incident on v_i and belonging to C_{even}. Let $x_i = w_{{e_i}^{odd}}$. Let $y_i = w_{{e_i}^{even}}$. If $\sum_{i=1}^{\ell}(x_i - y_i) \leq 0$ then the *right direction* is choosing C_{odd} otherwise the *right direction* is choosing C_{even}. In our algorithm, we always choose the *right direction.* So, the procedure manip is as described below.

manip ($\mathcal{C}$, $\mathcal{G}$)
1. for every even cycle C in $\mathcal{C}$
2. (* choose the right direction *)
if $\sum_{i=1}^{\ell}(x_i - y_i) \leq 0$ then choose C_{odd}
else choose C_{even}
3. manipulate C as described above in the chosen direction.

Lemma 6. *The procedure* manip *runs in NC, preserves bipartiteness and we are still inside $\mathcal{FPM}$ at the end of* manip.

Proof. Easy to check. □

We will use the procedures *make2-3*, *find-big-even*, *simple-manip* and *manip* as subroutines in the algorithms in next sections.

4 Finding a Perfect Matching in Bipartite Cubic Graphs in NC

Now we describe an NC algorithm to find a perfect matching in bipartite cubic graphs. [SW 96] have already given an NC algorithm for the same but the approach here is totally different as compared to theirs. They maintain a subgraph called pseudo perfect matching (degree of every vertex is odd) at every step and try decreasing the number of 3-vertices in the pseudo perfect matching by a constant fraction. Here, we start from an interior point of the perfect matching polytope and move towards a vertex withoght leaving the polytope.

4.1 Algorithm

Given a cubic bipartite graph $\overline{G}$
1. (a) get an interior point $\mathcal{G}$ of $\mathcal{PM}(\overline{G})$ by assigning $(\overline{g}_{\overline{e}} = \frac{1}{3})(\forall \overline{e} \in E(\overline{G}))$.
(b) let $\mathcal{G} = \overline{\mathcal{G}}$ and let $\pi : E(\overline{G}) \rightarrow \{0,1\} \times E(G) \bigcup \{\mathbf{0}, \mathbf{1}\}$ be a "backtracker" such that $(\pi : \overline{e} \mapsto (0, e))(\forall \overline{e} \in E(\overline{G}))$. (* Initialize the "backtracker" π. *)
2. while (G is nonempty)
{
(a) $\mathcal{C}$ = *find-big-even* ($\mathcal{G}$)
(* Take $\mathcal{C}$ to be the XOR of all fundamental cycles of a spanning tree of G. *)
(b) $\mathcal{G}$ = *simple-manip* ($\mathcal{C}$, $\mathcal{G}$)
(* Delete alternate edges of each even cycle in $\mathcal{C}$. *)
(c) $(\mathcal{G}, \pi)$ = make2-3 $(\overline{\mathcal{G}}, \pi, \mathcal{G})$
(* Replace each 3-bounded path (section 2.2) in G by single edge, update π. *)
}
3. backtrack a perfect matching in $\overline{G}$ using π.
(* When G is empty then π maps each edge of $\overline{G}$ to **0** or **1**, the edges mapped to **1** form a perfect matching in $\overline{G}$. *)

4.2 Analysis

The following sequence of lemmas will show that the above algorithm runs correctly in NC.

Lemma 7. *During the course of the algorithm, the weights of the edges of G will be* 0, 1/3 *or* 2/3.

Proof. The procedures *simple-manip* as well as *make2-3* don't change the denominators of the weights. □

Lemma 8. *At the beginning of each iteration of the* while *loop, G is cubic-bipartite graph with weight* 1/3 *on each edge.*

Proof. Note that G is a 2-3 graph at the beginning of each iteration of *while* loop. So, any edge of G has one end point of degree 3. By the previous lemma, the weights of the edges of G are multiples of $1/3$ and since G is a 2-3 graph output by *make2-3* there are no zero weight edges. So, the weight of any edge is at least $1/3$. Now for any vertex of degree 3, the sum of the weights of edges incident on it should add up to 1. Hence, the weight of each of the edges incident on it should be $1/3$. But every edge in a 2-3 graph is incident on a 3-vertex. So, the graph is again cubic bipartite with weight $1/3$ on each edge. □

Lemma 9. *At the end of each iteration of the* while *loop, the size (number of edges) of G decreases by a contant fraction.*

Proof. Note that during an iteration, *simple-manip* will destroy half the edges of C by making their weight 0 because the weight of any edge is $1/3$. Due to *simple-manip*, the size of the graph has reduced by a constant fraction.

Theorem 2. *The above algorithm finds a perfect matching in cubic-bipartite graphs in NC.*

Proof. The previous lemma implies that, the while loop terminates in $O(logn)$, proving that the algorithm runs in NC. □

We need more sophisticated version of *simple-manip* (the way it is described in Section 3.2) for extending the same approach for d-regular bipartite graphs. The next section describes the extension of the result in this section. As it was the case with cubic-bipartite graphs here, the notion of 2-3 graphs will play crucial role in the analysis in the next section.

5 Finding a Perfect Matching in d-Regular Bipartite Graphs for Polylogarithmic d in NC

There is already an NC algorithm for d-regular bipartite graphs by [LPV 81]. Our algorithm is totally different from theirs and conceptually simple though it works in NC only when d is polylogarithmic.

5.1 Algorithm

Given a d-regular bipartite graph $\overline{G}$,
1. (a) get an interior point of $\mathcal{PM}(\overline{G})$ by assigning $(\overline{g}_{\overline{e}} = 1/d)(\forall \overline{e} \in E(\overline{G}))$
(b) let $\overline{\pi} : E(\overline{G}) \rightarrow \{0,1\} \times E(\overline{G}) \bigcup \{\mathbf{0},\mathbf{1}\}$ be a "backtracker" such that $(\overline{\pi} : \overline{e} \mapsto (0,\overline{e}))(\forall \overline{e} \in E(\overline{G}))$.
2. $(\mathcal{G},\pi)$ = *make2-3* $(\overline{\mathcal{G}},\overline{\pi},\overline{\mathcal{G}})$
3. while (G has no vertex of degree 3)
{
(a) $\mathcal{C}$ = *find-big-even* $(\mathcal{G})$
(b) $\mathcal{G}$ = *manip* $(\mathcal{C}, \mathcal{G})$
(c) $(\mathcal{G},\pi)$ = *make2-3* $(\overline{\mathcal{G}},\pi,\mathcal{G})$
}

4. (a) Now all the vertices in G have degree 2. So, a perfect matching in G can be found very easily by taking alternate edges of each even cycle in G.
(b) Using the "backtracker" π one can get back a perfect matching in $\overline{G}$ from the perfect matching in G.

5.2 Analysis

Now we will show that the above algorithm runs in NC. To show this we consider certain integer potential function for weighted graphs. A similar potential function is used in [S 98]. We show that the potential of $\mathcal{G}$ decreases by large amount after each iteration of the while loop. Finally, when the potential becomes zero, we get a perfect matching easily.

Lemma 10. *During the course of the algorithm, the weights of the edges of G are multiples of $1/d$.*

Proof. Both *manip* and *make2-3* don't change the denominators of the weights. □

The integer potential function Φ
By the previous lemma we can assume that $g_e = \frac{w_e}{d}$ where w_e is an integer.
The potential of an edge is defined as $\Phi(e) := w_e(d - w_e)$.
The potential of a vertex is $\Phi(v) := \sum_{e \perp v} \Phi(e)$.
The potential of the graph G is $\Phi(G) := \sum_{v \in V^{(3)}(G)} \Phi(v)$. Recall that $V^{(3)}(G)$ is the set of vertices of degree 3 in G.

Lemma 11. *If even cycle C contains ℓ 3-vertices of G then after manipulating C in the* right direction*, the potential of G decreases at least by 2ℓ.*

Proof. Without loss of generality, say the right direction was to choose C_{odd}. Say the min weight edge in C_{odd} has weight w. Let $x_i = w_{e_i{}^{odd}}$. Let $y_i = w_{e_i{}^{even}}$. (As in Section 3.2.) Let $\Delta(\Phi)$ denote the change in potential due to manipulating C. Let G' be the graph obtained by manipulating C in G.

Then, $\Delta(\Phi) = \Phi(G') - \Phi(G)$.

Now, $\Phi(G') = \sum_{v \in C \cap V^{(3)}(G')} \Phi'(v)$, where Φ' is the new potential of a vertex after the manipulation.

Similarly, $\Phi(G) = \sum_{v \in C \cap V^{(3)}(G)} \Phi(v)$.

But, $C \cap V^{(3)}(G') \subseteq C \cap V^{(3)}(G)$.

Therefore, $\Phi(G') \leq \sum_{v \in C \cap V^{(3)}(G)} \Phi'(v)$.

Now, $\Phi'(v_i) - \Phi(v_i) = (x_i - w)(d - x_i + w) + (y_i + w)(d - y_i - w) - (x_i)(d - x_i) - (y_i)(d - y_i)$
$= x_i w - wd + wx_i - w^2 - y_i w + wd - wy_i - w^2$
$= 2(x_i - y_i)w - 2w^2$.

Therefore, $\Delta(\Phi) \leq \sum_{i=1}^{\ell} 2(x_i - y_i)w - 2w^2$.

Since we chose the *right direction* to manipulate C, $\sum_{i=1}^{\ell} (x_i - y_i) \leq 0$. Hence, the change in potential of G, $\Delta(\Phi) \leq -2w^2\ell$. Hence, the potential of G decreases at least by 2ℓ as w^2 is at least 1. □

Lemma 12. *If the number of* 3*-vertices in G is k at the beginning of a* while *loop, then at the end of the* while *loop, the potential of G decrease by at least $\frac{k}{2}$.*

Proof. The *big even cycle vector* contains at least $\frac{k}{4}$ 3-vertices in G. For each cycle C in the *big even cycle vector*, after manipulating, the potential decreases at least by 2 times $\#3-$*vertices in* C. Hence, when we manipulate a *big even cycle vector* in G containing at least $\frac{k}{4}$ 3-vertices in G, the potential of G decreases at least by $\frac{k}{2}$. □

Lemma 13. *In* $O(d^2)$ *iterations of while loop, the potential of* G *becomes* 0.

Proof. Initially, the potential of G at the start of the first iteration of *while* loop is $O(d^2k)$. By previous lemma, the potential decrease by $\Omega(k)$ in each iteration of while loop. Hence, in $O(d^2)$ steps the potential becomes 0 and *while* loop terminates.

Lemma 14. *When the* while *loop ends, we get a perfect matching in the original graph.*

Proof. When the *while* loop terminates, G is a disjoint union of even cycles and a perfect matching in G can be found easily and could be backtracked to the perfect matching in $\overline{G}$ using π. □

Theorem 3. *For polylogarithmic* d*, the above algorithm runs correctly in NC.*

Proof. The number of iterations of while loop is $O(d^2)$. In particular, for polylogarithmic d , the algorithm runs in NC. □

Note that the algorithm presented above works in NC if you start with any small *magnitude* interior point of a bipartite graph, i.e., if the least common multiple of the denominators of the weights is polylogarithmic.

6 Discussion

The starting point of our algorithm - to get an interior point of the perfect matching polytope - was simple but crucial. First of all, a non-bipartite regular graph need not have a perfect matching and even if it has a perfect matching, it is not clear how to find an interior point of the $\mathcal{PM}$ of such a graph to provide the start for the algorithm. In certain cases, however we can get a start.

6.1 Getting an Interior Point for Regular Expander Graphs

A graph G is said to be an α-expander if for every $S \subset V$ such that $|S| \leq \frac{|V|}{2}$, the number of edges (size of the cut $(S, \overline{S})$) from S to its complement ($\overline{S} = V \backslash S$) is at least α times the cardinality of S. The α is called the expansion factor of G.

Lemma 15. *If* G *is a simple (no multiple edges)* d*-regular expander graph on even number of vertices with the expansion factor* $\alpha > \frac{d-1}{d}$*, then* $(g_e = 1/d)(\forall e \in E(G))$ *gives an interior point of the perfect matching polytope of* G*.*

Proof. It suffices to check "odd-cut" contraints (sum of the weights of the edges in a cut $(S, \overline{S})$ at least one whenever S is odd) [LP 86]. For S of size at least

d, expanstion property guarantees that there are at least d edges from S to $\overline{S}$. Hence, the size of cut $(S, \overline{S})$ is at least 1. If size of S is less than d, assuming that G is simple, one can show that at least d edges should go out of S by counting the total of degrees. □

Corollary 1. *Such an expander graph always has a perfect matching. In fact it has at least d perfect matchings.*

Corollary 2. *One can check in NC whether* $(g_e = 1/d)(\forall e \in E)$ *gives an interior point of* $\mathcal{PM}(G)$ *for a d-regular expander graph with expansion* α*, if d and* α *are constants.*

Another main difficulty is in maintaining the "odd-cut" constraints in the perfect matching polytope. The procedure *manip* just maintains the constraints at a vertex but the odd-cut contraints might get violated. By just maintaining the constraints at each node of a regular graph, we can get a perfect 2-matching: a vertex of the 2-matching polytope (Section 2.1).

Lemma 16. *A perfect 2-matching in non-bipartite regular graphs can be found in NC.*

Proof. The problem of finding a perfect 2-matching in regular graphs reduces to the problem of finding a perfect matching in regular bipartite graphs (see for instance [KR 98]). □

6.2 Conclusion

We have presented a different and conceptually simple parallel algorithm for finding a perfect matching in d-regular bipartite graphs. In particular, when d is small (polylog), our algorithm runs in NC. The connection between our algorithm and the algorithm of [MV 00] for bipartite-planar graphs is notable. [MV 00] uses planarity crucially and we could still use their approach to get an NC algorithm in a non-planar subclass of bipartite graphs. It is also remarkable that the notion of 2-3 graphs, developed in the context of planar graphs in [KM 04] plays a crucial role in our algorithm. This suggests that these techniqes seem more general. As in case of [MV 00], the "odd-cut" contraints are difficult to maintain. The problem here is more basic, even it is difficult to get a starting point inside $\mathcal{PM}$. We have observed that in certain cases it is possible to get a starting point and without maintaining the "odd-cut" contraints, one can still get a perfect 2-matching in regular graphs, yet, as of now, the quest for the perfect matching continues.

Acknowledgements

I sincerely thank Prof. Janos Simon for helpful discussions and Prof. Meena Mahajan for introducing the problem and for numerous useful suggestions.

References

[DK 92] E. Dahlhouse and M. Karpinski. Perfect matching for regular graphs is AC^0-hard for the general matching problem. *J. Comput. Syst. Sci., 44, p. 94-102*, 1992.

[GK 87] D. Grigoriev and M. Karpinski. The matching problem for bipartite graphs with polynomially bounded permanent is in NC. In *Proceedings of 28th IEEE Conference on Foundations of Computer Science, pages 166-172.* IEEE Computer Society Press, 1987.

[KM 04] Raghav Kulkarni, Meena Mahajan. Seeking a vertex of the planar matching polytope in NC. *In the Proceedings of the 12th European Symposium on Algorithms ESA, LNCS vol. 3221, pages 472-483.* Springer, 2004.

[KR 98] M. Karpinski and W. Rytter. Fast parallel algorithms for graph matching problems. *Oxford Science Publications*, 1998.

[KUW 86] R Karp, E Upful, A Wigderson. Constructing a perfect matching is in random NC. *Combinatorica, 6:35-48*, 1986.

[LP 86] Lovasz and Plummer. Matching theory. *Mathematical Studies, Annals of Discrete Maths*,Vol. 25, North-Holland, Amsterdam, 1986.

[LPV 81] G. Lev, M. Pippenger, L. Valiant. A fast parallel algorithm for routing in permutation networks, *IEEE Transactions on Computers, C-30:93-100*, 1981.

[MN 95] G Millller and J Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal of Computing*, 24:1002-1017, 1995.

[MV 00] M Mahajan, K Varadarajan. A new NC algorithm to find a perfect matching in planar and bounded genus graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (STOC)*, pages 351-357, 2000.

[MVV 87] K Mulmuley, U Vazirani, V Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1): 105-131, 1987.

[S 98] Alexander Schrijver: Bipartite Edge Coloring in O(Delta m) Time. SIAM J. Comput. 28(3): 841-846 (1998)

[SW 96] R Sharan, A Wigderson. A new NC algorithm for perfect matching in cubic bipartite graphs. *Proc. of ISTCS 96*, pp. 56-65, 1996.