# Using Interior Point Methods for Fast Parallel Algorithms for Bipartite Matching and Related Problems

*Andrew V. Goldberg*[*]
Department of Computer Science
Stanford University
Stanford, CA 94305

*Serge A. Plotkin*[†]
Department of Computer Science
Stanford University
Stanford, CA 94305

*David B. Shmoys*[‡]
School of Operations Research and Industrial Engineering
Cornell University
Ithaca, NY 14853

*Éva Tardos*[§]
School of Operations Research and Industrial Engineering
Cornell University
Ithaca, NY 14853

March 1991

[*]Research partially supported by NSF Presidential Young Investigator Grant CCR-8858097 with matching funds from AT&T and DEC, IBM Faculty Development Award, a grant from 3M Corp., and ONR Contract N00014–88–K–0166.

[†]Research supported by NSF Research Initiation Award CCR900-8226 and by ONR Contract N00014–88–K–0166.

[‡]Research partially supported by an NSF PYI award CCR-89-96272 with matching support from UPS, and Sun Microsystems, by Air Force contract AFOSR-86–0078, and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550, as well the NSF PYI award of the first author.

[§]Research supported in part by a Packard Research Fellowship and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550 and by Air Force contract AFOSR-86–0078, as well as by the NSF PYI award of the first author.

**Abstract**

In this paper we use interior-point methods for linear programming, developed in the context of sequential computation, to obtain a parallel algorithm for the bipartite matching problem. Our algorithm finds a maximum cardinality matching in a bipartite graph with $n$ nodes and $m$ edges in $O(\sqrt{m}\log^3 n)$ time on a CRCW PRAM. Our results extend to the weighted bipartite matching problem and to the zero-one minimum-cost flow problem, yielding $O(\sqrt{m}\log^2 n \log nC)$ algorithms, where $C > 1$ is an upper bound on the absolute value of the integral weights or costs in the two problems, respectively. Our results improve previous bounds on these problems and introduce interior-point methods to the context of parallel algorithm design.

# 1 Introduction

In this paper we use interior-point methods for linear programming, developed in the context of sequential computation, to obtain a parallel algorithm for the bipartite matching problem. Although Karp, Upfal, and Wigderson [6] have shown that the bipartite matching problem is in $RNC$ (see also [12]), this problem is not known to be in $NC$. Special cases of the problem are known to be in $NC$. Lev, Pippenger, and Valiant [9] gave an $NC$ algorithm to find a perfect matching in a regular bipartite graph. Miller and Naor [10] gave an $NC$ algorithm to decide whether a planar bipartite graph has a perfect matching.

The best previously known deterministic algorithm for the problem, due to Goldberg, Plotkin, and Vaidya [4], runs in $O^*(n^{2/3})$ time on graphs with $n$ nodes, where an algorithm runs in $O^*(f(n))$ time if it runs in $O(f(n)\log^k(n))$ time for some constant $k$. In this paper we describe an $O^*(\sqrt{m})$ algorithm to find a maximum cardinality matching in a bipartite graph with $m$ edges, which is based on an interior-point algorithm for linear programming and on Gabow's algorithm [3] for edge-coloring bipartite graphs. For graphs of low-to-moderate density, this bound is better than the bound mentioned above.

The results presented in this paper extend to the maximum-weight matching problem and to the zero-one minimum-cost flow problem. The resulting algorithms run in $O^*(\sqrt{m}\log C)$ time, where $C > 1$ is an upper bound on the absolute value of the integral weight and costs in the two problems, respectively. The best previously known algorithm for the zero-one minimum-cost flow problem runs in $O^*((nm)^{2/5}\log C)$ time [4]. The new algorithm is better for both the zero-one maximum flow and the zero-one minimum-cost flow problems for all graph densities.

Interior-point algorithms work as follows. The algorithm starts with a point in the interior of the feasible region of the linear program and its dual that is close to the so-called central path. In its main loop, the algorithm moves from one interior point to another, decreasing the value of the duality gap at each iteration. When this value is small enough, the algorithm terminates with an interior-point solution that has a near-optimal value. The finish-up stage of the algorithm converts this near-optimal solution into an optimal basic solution.

Karmarkar's revolutionary paper [5] spurred the development of the area of interior-point linear programming algorithms, and many papers have followed his lead. Karmarkar's algorithm runs in

$O(NL)$ iterations, where $N$ and $L$ denote the number of variables and the size of the linear program, respectively. Renegar [15] was the first to give an interior-point algorithm that runs in $O(\sqrt{N}L)$ iterations. Since then, several different $O(\sqrt{N}L)$-iteration algorithms have been developed. For an overview of work on interior-point algorithms, the reader is referred to the survey paper of Todd [17]. The matching algorithm discussed in this paper is based on an algorithm due to Monteiro and Adler [11], though similar algorithms can also be based on other $O(\sqrt{N}L)$ iteration algorithms.

Interior-point algorithms have proved to be an important tool for developing efficient sequential algorithms for linear programming and its special cases. In this paper we apply these tools in the context of parallel computation. For the purpose of parallel computation, an important fact is that the running time of an iteration of an interior-point algorithm is dominated by the time required for matrix multiplication and inversion. Therefore, an iteration of such an algorithm can be done $O(\log^2 N)$ time on a CRCW PRAM using $N^3$ processors [13].

The interior-point method used here follows a central path in the interior of the feasible region. After every $\sqrt{N}$ iterations, this algorithm has decreased the duality gap by a constant factor. The bipartite matching problem can be formulated as a linear program with an integral optimum value. Therefore the size of the maximum matching is known as soon as this gap is below one. Furthermore, the gap between the value of an initial solution and the optimal value is at most $N$. This suggests that an interior-point algorithm can be used to find the value of the maximum matching in a bipartite graph in $O(\sqrt{m}\log n)$ iterations, or $O^*(\sqrt{m})$ time. In this paper we develop an algorithm running in this time bound that finds a maximum matching as well as its value.

To find a maximum matching we need to overcome two difficulties. First, we need to find an initial interior point and dual solution that is close to the central path and has a small duality gap, so that the number of iterations will be small. The second difficulty comes from the fact that standard implementations of the finish-up stage of interior-point algorithms either are inherently sequential or perturb the input problem to simplify the finish-up stage, increasing the number of iterations of the main loop by an $\Omega(n)$ factor. For the special case of the bipartite matching problem, we give a parallel implementation of the finish-up stage that runs in $O(\log^2 n)$ time using $m$ processors. This implementation is based on Gabow's edge-coloring algorithm [3].

Our techniques apply to the more general maximum-weight matching problem. The algorithm and its analysis are only slightly more involved. For brevity we focus on the more general case. The results for the maximum matching problem are obtained as a simple corollary of the results for the maximum-weight matching problem. The main loop of our maximum-weight matching algorithm runs in $O(\sqrt{m}\log^2 n \log nC)$ time and uses $m^3$ processors, and the finish-up stage runs in $O(\log n \log nC)$ time and uses $m$ processors. Therefore, the algorithm runs in $O^*(\sqrt{m}\log C)$ time. A standard reduction between the weighted matching and the zero-one minimum-cost flow problems (see *e.g.*, [1, 6]) gives an $O^*(\sqrt{m}\log C)$ algorithm.

This paper is organized as follows. Section 2 introduces definitions and terminology used throughout the paper and reviews the Monteiro–Adler linear programming algorithm. Section 3

gives a linear programming formulation of the bipartite matching problem that has an initial interior point close to the central path with a small duality gap, and shows how to use the linear programming algorithm to obtain a near-optimal fractional matching. Section 4 describes a parallel procedure that, in $O^*(\log C)$ time, converts the near-optimal fractional matching into an optimal zero-one matching. Section 5 contains concluding remarks.

## 2 Preliminaries

In this section we define the matching problem and the linear programming problem, and review some fundamental facts about them. For a detailed treatment, the reader is referred to the textbooks by Papadimitriou and Steiglitz [14] or Schrijver [16]. We also give an overview of the Monteiro–Adler algorithm.

The *bipartite matching problem* is to find a maximum cardinality matching in a bipartite graph $G = (V, E)$. The *maximum-weight bipartite matching problem* is defined by a bipartite graph $G = (V, E)$ and a weight function on the edges $w : E \longrightarrow \mathbf{R}$. We shall assume that the weights are integral. The *weight* of a matching $M$ is $\sum_{e \in M} w(e)$. The problem is to find a matching with maximum weight.

We use the following notation and assumptions. Let $G = (V, E)$ denote the (bipartite) input graph, let $n$ denote the number of nodes in $G$, let $m$ denote the number of edges in $G$, and let $C$ denote the maximum absolute value of the weights of edges in $G$. To simplify the running time bounds, we assume, without loss of generality, that $m \geq n - 1 > 1$, and $C > 1$. We denote the degree of a node $v$ by $d(v)$, and the set of edges incident to node $v$ by $\delta(v)$. For a vector $x$, we let $x(i)$ denote the $i$th coordinate of $x$. We use a PRAM [2] as our model of parallel computation.

Consider the following standard linear programming formulation of the bipartite matching problem.

$$
\begin{array}{lllrcl}
\textit{Matching-1:} & \text{maximize} & & w^t f & & \\
 & \text{subject to:} & \sum_{e \in \delta(v)} f(e) & \leq & 1, & \text{for } v = 1, \ldots, n, \\
 & & f & \geq & 0. &
\end{array}
$$

A feasible solution to the system of the linear inequalities above is called a *fractional matching*. We denote an optimal solution of the linear program by $f^*$.

The constraint matrix of *Matching-1* is the node-edge incidence matrix of the bipartite graph $G$. A matrix is *totally unimodular* if all of its submatrices have determinants +1, -1 or zero. It is well known that the node-edge incidence matrix of a bipartite graph is totally unimodular [14]. This implies the following theorem.

**Theorem 2.1** [14] Any optimal solution of the linear program *Matching-1* is the convex combination of maximum-weight matchings. An optimal value of this linear program is equal to the maximum weight of a matching.

The Monteiro–Adler algorithm handles linear programs in the following form:

$$\begin{array}{lllll} \textit{Primal LP:} & \text{minimize} & c^t x & & \\ & \text{subject to:} & Ax & = & b \\ & & x & \geq & 0 \end{array}$$

where $A$ is a matrix, and $b$, $c$, and $x$ are vectors of the appropriate dimensions. We assume that the matrix $A$ and the vectors $b$ and $c$ are integral. We use $N$ to denote the number of variables in the (primal) linear programs we consider. A vector $x$ is a *feasible solution* if it satisfies the constraints $Ax = b$ and $x \geq 0$. A feasible solution $x$ is *optimal* if it minimizes the objective function value $c^t x$, and *interior point* if it is in the interior of the nonnegative orthant.

The linear programming duality theorem states that the minimum value of the Primal LP is equal to the maximum value of the following *Dual LP*:

$$\begin{array}{lllll} \textit{Dual LP:} & \text{maximize} & b^t \pi & & \\ & \text{subject to:} & A^t \pi + s & = & c \\ & & s & \geq & 0 \end{array}$$

where $\pi$ and $s$ are the variables of the Dual LP, the dimension of $\pi$ is equal to the dimension of $b$, and the dimension of $s$ is equal to the dimension of $x$. Feasible and optimal solutions and interior points for the dual problem are defined in the same way as for the primal problem.

Let $x$ be a feasible solution to the Primal LP, and let $(\pi, s)$ be a feasible solution to the Dual LP. The value $c^t x$ is an upper bound and $b^t \pi$ is a lower bound on the common optimal value of the two problems. Hence the difference $c^t x - b^t \pi = s^t x$ measures how far the current solutions are from being optimal. This quantity is called the *duality gap*. The *central path* of this pair of linear programs is defined as the set of points with $s(i)x(i)$ identical for every $i$, $i = 1, \ldots, N$. Notice that the complementary slackness conditions state that the pair of primal and dual solutions is optimal if these products are all zero.

The Monteiro–Adler algorithm is applied to a pair of primal and dual linear programs in the above form. The algorithm starts with a vector $(x_0, \pi_0, s_0)$, where $x_0$ and $(\pi_0, s_0)$ are interior points of the primal and dual linear problems, that are in some sense close to the central path. At each iteration of the main loop, the algorithm moves from the current interior point to another interior point, so that the duality gap is decreased by a factor of $(1 - \Omega(1/\sqrt{N}))$ every iteration.

The measure of closeness to the central path required by the algorithm is defined as follows. Consider a primal–dual solution pair $(x, \pi, s)$. Define $\mu = s^t x / N$, and define the vector $\sigma$ such that $\sigma(i) = s(i)x(i)$ for $i = 1, \ldots, N$. The solution pair is *close to the central path* if $||\sigma - \mu \mathbf{1}|| \leq \theta \mu$, where $\mathbf{1}$ denotes the vector with all coordinates 1, $||\cdot||$ denotes the Euclidean norm, and $\theta$ is 0.35, as suggested by Monteiro and Adler.

Monteiro and Adler prove the following theorem.

**Theorem 2.2** [11] If we have an initial solution $(x_0, \pi_0, s_0)$ that is close to the central path, then for any constant $\delta > 0$, after $O(\sqrt{N}\log(s_0^t x_0))$ iterations the duality gap $s^t x$ of the current solution $(x, \pi, s)$ is at most $\delta$.

To get the algorithm started, one has to provide an initial solution $(x_0, \pi_0, s_0)$ that is close to the central path. Monteiro and Adler present a way to obtain an equivalent linear programming formulation with such an initial solution. In the next section we give a slightly simplified version of this construction for the bipartite matching problem, for which the initial solution also has a sufficiently small duality gap.

# 3 Finding a Near-Optimal Solution

In this section we show how to convert the *Matching-1* linear program into a linear program that is in the form required by the Monteiro–Adler algorithm and has an initial solution close to the central path with a small duality gap. Then we show how to compute a near-optimal fractional matching from the initial solution to this linear program.

We restate the matching problem as follows:

$$
\begin{array}{lll}
\textit{Matching-2:} & \text{minimize} & -w^t f + \frac{N^2 C}{n-1} z \\
& \text{subject to:} & \sum_{e \in \delta(v)} f(e) + (n - d(v))g(v) - z = 1, \quad \text{for } v = 1, \ldots, n, \\
& & \mathbf{1}^t f + \mathbf{1}^t g + y = n + m + 1, \qquad (*) \\
& & f, g, z, y \geq 0.
\end{array}
$$

We denote the objective function of this linear program by the vector $c$, and coefficients of the left-hand-side of the constraint $(*)$ by the vector $a$. The number of variables in this linear program is $m+n+2 = N$. We denote a feasible solution to *Matching-2* by $x = (f, g, y, z)$, and a feasible solution of the corresponding dual problem by $\pi$ and $s$, where $\pi(i)$, for $i = 1, \ldots, n$, is the dual variable corresponding to the primal constraint for node $i$, and $\pi(n+1)$ is the dual variable corresponding to the constraint $(*)$.

Intuitively, the transformation works as follows. Variables $g(v)$ are the slack variables introduced to replace inequality constraints by equality constraints. The positive multipliers $(n - d(v))$ scale the slack variables so that there is a feasible solution with all original and slack variables equal. The coefficient of $z$ in the objective function is large enough to guarantee that $z = 0$ in an optimal solution. The variable $z$ is introduced to make it possible to have a starting primal solution with coordinates of $f$, $g$ and $y$ equal (for example, to 1). The constraint $(*)$ does not affect the primal problem since $y$ is not in the objective function and, as we have just mentioned, in an optimal solution $z = 0$ and therefore $g^t\mathbf{1} + f^t\mathbf{1} \leq n$ is automatically satisfied. This constraint, however, allows us to obtain an initial solution for the dual problem such that the dual slack variables corresponding to the primal variables $f$, $g$ and $y$ roughly equal. This will imply that the starting solution is close to the central path.

**Lemma 3.1** If $(f, g, y, z)$ is an optimal solution of *Matching-2*, then $f$ is an optimal solution to *Matching-1*.

*Proof*: Every solution to *Matching-1* can be extended to a solution to *Matching-2* with $z = 0$; this follows from the fact that both $f$ and the slacks in *Matching-1* are at most 1.

Next we have to show that every optimal solution to *Matching-2* has $z = 0$. Consider a feasible point $x_1 = (f_1, g_1, z_1, y_1)$ with $z_1 \neq 0$. Since $f_1$ satisfies $\sum_{e \in \delta(v)} f_1(e) \leq 1 + z_1$ for every node $v$, decreasing $f_1$ on some edges, by a total of at most $z_1 n$, converts $f_1$ into a vector $f_2$ that is a fractional matching. Above we observed that any fractional matching can be extended to a feasible solution of *Matching-2*. Let $x_2$ denote a feasible solution extending $f_2$. If we replace $x_1$ by $x_2$, the decrease in the objective function value caused by the reduction in $z$ is $z_1 \frac{N^2 C}{n-1} > z_1 NC$. The increase due to the change in $f$ is bounded by $z_1 nC < z_1 NC$. Therefore, the value $c^t x_2$ is smaller than $c^t x_1$, which implies that any optimal solution must have $z = 0$. ∎

We define initial primal and dual solutions as suggested by the above discussion. The initial primal solution $x_0$ is defined by

$$f = \mathbf{1}, g = \mathbf{1}, y = 1, z = n - 1.$$

The initial dual solution $(\pi_0, s_0)$ is defined by

$$\begin{array}{ll} \pi(i) = 0, & \text{for } 1 \leq i \leq n, \\ \pi(n+1) = -N^2 C, & \\ s(i) = c(i) + N^2 C a(i) & \text{for } 1 \leq i \leq N. \end{array}$$

**Lemma 3.2** The vectors $x_0$ and $(\pi_0, s_0)$ are interior-point solutions of the primal and the dual problems that are close to the central path, and the value of the duality gap is $O(N^3 C)$.

*Proof*: It is easy to verify that $x_0$ is a primal solution, and $(\pi_0, s_0)$ is a dual solution. Recall that $N = n + m + 2$. The duality gap is

$$s_0{}^t x_0 = nN^2 C + mN^2 C - w^t \mathbf{1} + 2N^2 C = N^3 C - w^t \mathbf{1} = O(N^3 C)$$

as required.

Next we have to verify that the initial solution is close to the central path. Let $\mu = s_0^t x_0 / N = N^2 C - (w^t \mathbf{1}/N)$, and define the vector $\sigma$ with coordinates $\sigma(i) = s(i)x(i)$. Consider $\sigma(i) - \mu$ for each type of variable separately. For variables $s(i)$ and $x(i)$ corresponding to $z, y$, and $g$, we get

$$|\sigma(i) - \mu| = |s_0(i) x_0(i) - \mu| = |w^t \mathbf{1}/N|.$$

For variables $s(i)$ and $x(i)$ corresponding to $f$, we get

$$|\sigma(i) - \mu| = |s_0(i) x_0(i) - \mu| = |w^t \mathbf{1}/N - w(i)|.$$

Using these values we get that

$$||\sigma(i) - \mu||^2 \leq N(w^t\mathbf{1}/N)^2 + \sum_i w^2(i) \leq 2NC^2.$$

Since $N \geq 3$, we have that $2NC^2 \leq \theta^2(N^4C^2 - 2NCw^t\mathbf{1}) \leq (\theta\mu)^2$. This proves the lemma. ∎

Now we are ready to give the $O^*(\sqrt{m}\log C)$-time algorithm to compute the weight of an optimal matching and to find a near-optimal fractional matching. In the next section we show how to convert such a near optimal fractional matching into an optimal matching.

**Lemma 3.3** A fractional bipartite matching with weight at most $1/2$ less than the weight an optimal matching can be computed in $O^*(\sqrt{m}\log C)$ time on a PRAM with $m^3$ processors.

*Proof*: By applying Lemma 3.2 and Theorem 2.2 (with $\delta = 1/4$), we see that after $O(\sqrt{N}\log(NC)) = O(\sqrt{m}\log(nC))$ iterations of the LP algorithm, we have obtained a point $(x, \pi, s)$ with a duality gap $x^ts \leq 1/4$. Hence we have

$$-w^tf + \frac{N^2C}{n-1}z + w^tf^* \leq \frac{1}{4}, \tag{1}$$

where $f^*$ is an optimal solution to *Matching-1*. Since $z \geq 0$, this implies that $w^tf^* - w^tf \leq 1/4$. As in Lemma 3.1, we can argue that $f$ can be converted to a feasible solution of the *Matching-1* problem by decreasing its value on some of the edges by a total of at most $zn$. Therefore, $w^tf^* \geq w^tf - znC$. From (1), this implies that $z\frac{N^2C}{n-1} \leq 1/4 + znC$. Thus,

$$z \leq \frac{n-1}{4C(N^2 - n^2 + n)} < \frac{1}{4mC}.$$

Now round all values of $f$ and $g$ down to have a common denominator $4mC$, and denote the rounded solution by $f_1, g_1$. Clearly, $w^tf^* - w^tf_1 \leq 1/4 + (mC)/(4mC) \leq 1/2$. After the rounding, we have:

$$\sum_{e\in\delta(v)} f_1(e) + (n - d(v))g_1(v) \quad \leq \quad 1 + z$$

The left-hand side is an integer multiple of $(4mC)^{-1}$ and $z < (4mC)^{-1}$. This implies that

$$\sum_{e\in\delta(v)} f_1(e) + (n - d(v))g_1(v) \quad \leq \quad 1$$

Hence, the resulting vector $f_1$ is a fractional matching whose weight is within $1/2$ of the optimum. ∎

**Corollary 3.4** The cardinality of the maximum matching in a bipartite graph can be computed in $O^*(\sqrt{m})$ time using $m^3$ processors.

# 4 The Finish-Up Stage

In the previous section we have shown how to compute, in $O^*(\sqrt{m}\log C)$ time, a fractional bipartite matching with weight at most 1/2 less than the optimum. In this section we give an $O^*(\log C)$ algorithm for converting any such fractional matching into a maximum-weight matching. Note that for the unweighted bipartite matching, this algorithm runs in polylogarithmic time.

Let $f$ be a fractional bipartite matching that has weight at most 1/2 less than the maximum weight, and let $f^*$ denote a maximum weight-matching. First we construct a fractional matching $f'$, such that the values of $f'$ have a relatively small common denominator that is a power of two and the weight of $f'$ differs from the maximum weight by less than 1. Define $\Delta$ by

$$\Delta = 2^{\lceil \log mC \rceil + 1}.$$

By definition, $\Delta$ is a power of 2 and $\Delta = O(mC)$. Let $f'$ be the fractional matching obtained by rounding $f$ down to the nearest multiple of $1/\Delta$. Note that

$$|w^t f - w^t f'| < \frac{mC}{\Delta} = \frac{mC}{2^{\lceil \log mC \rceil + 1}} < \frac{1}{2}.$$

Therefore $w^t f^* - w^t f' < 1$.

Consider a multigraph $G' = (V, E')$ with the edge set containing $\Delta \cdot f'(e)$ copies of $e$ for each $e \in E$, and no other edges.

**Lemma 4.1** For any coloring of the edges of $G'$ with $\Delta$ colors, there exists a color class which is a maximum-weight matching of $G$.

*Proof*: The proof is by a simple counting argument. The sum of the weights of the color classes is equal to $\Delta w^t f' > \Delta(w^t f^* - 1)$. Since there are $\Delta$ color classes, at least one of them has weight above $w^t f^* - 1$. The claim follows from the integrality of $w$. ∎

The above lemma implies that, in order to find a maximum-weight matching, it is sufficient to edge-color $G'$ using $\Delta$ colors. Since $G'$ is bipartite graph and its maximum degree is bounded by $\Delta$, which is a power of 2, we can use a parallel implementation of Gabow's algorithm [3] to edge-color $G'$ using $\Delta$ colors. However, $G'$ has $O(mC)$ edges and therefore the algorithm uses $\Omega(mC)$ processors. In order to reduce the processor requirement, we use a somewhat different algorithm. The algorithm does not use an explicit representation of the multigraph, but rather uses a weighted representation of a simple graph. A divide-and-conquer approach is then used to split the (implicit) multigraph so that the bound on the maximum degree of a note is halved, and then recurses on the part with greater weight. A subroutine for finding such a partitioning is also the basis of Gabow's edge-coloring algorithm.

Figure 1 describes the algorithm to find a maximum-weight matching given a near-optimal fractional matching. The algorithm starts by rounding the fractional matching to a small common

```
procedure Round(E, f);
    Δ ← 2^⌈log mC⌉+1 ;
    f' ← f rounded down to a common denominator of Δ;
    d' ← Δ;
    while d' > 1 do begin
        E_0 ← {e | e ∈ E, d' · f'(e) is odd};
        (E_1, E_2) ← Degree-Split(V, E_0);
        W_1 ← w(E_1);
        W_2 ← w(E_2);
        if W_1 ≥ W_2
          then begin
            for e ∈ E_1 do f'(e) ← f'(e) + 1/d';
            for e ∈ E_2 do f'(e) ← f'(e) − 1/d';
            end;
          else begin
            for e ∈ E_2 do f'(e) ← f'(e) + 1/d';
            for e ∈ E_1 do f'(e) ← f'(e) − 1/d';
            end;
        d' ← d'/2;
    end;
    return ({e | f'(e) = 1})
end.
```

Figure 1: Rounding an approximate fractional matching to an optimal integral one

denominator as described above. A fractional matching $f'$ with common denominator $\Delta$, can be written as $f' = \frac{1}{2}(f_1 + f_2)$ such that both $f_1$ and $f_2$ are fractional matchings with common denominator $\Delta/2$. On edges with $\Delta f'(e)$ even we can set $f_1(e) = f_2(e) = f'(e)$. Otherwise we set $f_1(e) = f'(e) + 1/\Delta$ and $f_2(e) = f'(e) - 1/\Delta$ or the other way around. Whether to add or to subtract $1/\Delta$ on these edges is decided with the help of the procedure *Degree-split*. This procedure partitions the edges of a bipartite graph $G_0 = (V, E_0)$ into two classes $E_1$ and $E_2$, so that for every node $v$, the degree of $v$ in the two induced subgraphs differs by at most one. The procedure is used for the graph on $V$ with edges $E_0 = \{e \in E : \Delta f'(e) \text{ is odd.}\}$. To obtain $f_1$ we increase $f'$ on one color class and decrease it on the other one. Both $f_1$ and $f_2$ are fractional matchings with common denominator $\Delta/2$. Now $f'$ is replaced by $f_1$ or $f_2$ depending on which one has larger weight. This process is iterated $O(\log(mC))$ times, until the current fractional matching is integral. The resulting matching has an integral weight that is more than $w^t f^* - 1$, and therefore the matching is optimal.

The heart of the algorithm is the procedure *Degree-Split* described in Figure 2. This procedure decomposes the graph into cycles and paths, such that at most one path ends at each node. This can be accomplished by pairing up the edges incident to each node separately. Then we two-color the paths and cycles separately. This gives a two-coloring of the graph where the difference in the degree of a node in the two subgraphs is at most 1.

**procedure** *Degree-Split*$(V, E)$;
    Construct a new node set $V'$ by replacing each node $v \in V$ by an independent set of size $\lceil d(v)/2 \rceil$;
    For each node in $V$, assign its incident edges to nodes in $V'$, so that each node $v$ in $V'$ has $d(v) \leq 2$;
    Edge-color the resulting graph using two colors;
    Return the edges of each color class;
**end.**

Figure 2: Splitting the maximum degree of the graph

**Lemma 4.2** The algorithm *Round* produces a maximum-weight matching.

*Proof*: Consider the parameter $d'$ used in the algorithm in Figure 1. Initially $d' = \Delta$. Note that after iteration $i$ we have $d' = \Delta/2^i$. We show by induction that after iteration $i$:

- $f'$ is a fractional matching,
- $w^t f' > w^t f^* - 1$,
- coordinates of $f'$ have common denominator $d'$.

Initially all three conditions are satisfied. Assuming that all three conditions are satisfied after iteration $i-1$, we prove that they remain satisfied after iteration $i$. Let $d_1$ and $f_1$ denote $d'$ and $f'$ before iteration $i$ and let $d_2$ and $f_2$ denote $d'$ and $f'$ after iteration $i$.

The last claim follows from the fact that the coordinates of $f_1$ that are odd multiples of $1/d_1$ are adjusted by plus or minus $1/d_1$ in this iteration, and so all coordinates of $f_2$ are even multiples of $1/d_1$, and hence multiples of $1/d_2$.

The second claim follows from the fact that the components of $f_2$ that have been increased correspond to edges of greater total weight than those that have been decreased.

Now consider the first claim. By the inductive assumption, $\sum_{e \in \delta(v)} f_1(e) \leq 1$. By the definition of Procedure *Degree-split*, $\sum_{e \in \delta(v)} f_2(e) \leq \sum_{e \in \delta(v)} f_1(e) + 1/d_1 \leq 1 + 1/d_1$. However, we have seen already that $f_2$ has a common denominator of $d_2$. Hence, $\sum_{e \in \delta(v)} f_2(e)$ is an integer multiple of $1/d_2 = 2/d_1$ and is therefore at most one.

After $\log \Delta$ iterations we construct an $f'$ that is integral and whose weight is above $w^t f^* - 1$. By integrality of $w$, the set of edges where this $f'$ is 1 is the desired maximum-weight matching of the input graph. ∎

**Lemma 4.3** The procedure *Degree-Split* partitions the input graph into two graphs with disjoint edge-sets, such that the degrees of any node $v$ in the two graphs differ by at most one. The procedure can be implemented in $O(\log n)$ time.

*Proof*: Observe that the graph constructed on $V'$ is bipartite, and the degree of a node is at most two. Therefore the graph consists of paths and even cycles. Hence it can be two edge-colored in $O(\log m)$ time using $m$ processors [7, 8]. The claim of the lemma follows from the fact that each node $v \in V$ is an end point of at most one path. ∎

**Lemma 4.4** The algorithm *Round* runs in $O(\log n \log nC)$ time using $m$ processors.

*Proof*: The number of iterations of the loop of the algorithm is $O(\log \Delta) = O(\log nC)$, because $d$ is halved at each iteration. The running time of each iteration is dominated by *Degree-Split*, which takes $O(\log n)$ time by Lemma 4.3. ∎

**Theorem 4.5** A maximum-weight bipartite matching can be computed in $O^*(\sqrt{m} \log C)$ time using $m^3$ processors.

Note that the exact running time of our algorithm on a CRCW PRAM is $O(\sqrt{m} \log^2 n \log nC)$, which is the time required to approximately solve the linear program.

**Corollary 4.6** A maximum cardinality bipartite matching can be computed in $O^*(\sqrt{m})$ time using $m^3$ processors.

# 5 Concluding Remarks

Interior-point methods have proved to be very powerful in the context of sequential computation. In this paper we have shown an application of these methods to the design of parallel algorithms. We believe that these methods will find more applications in the context of parallel computation. We would like to mention the following two research directions.

One direction is to attempt to generalize our result to general linear programming, showing that any linear programming problem can be solved in $O^*(\sqrt{N}L)$ time. This would require a parallel implementation of the finish-up stage of the algorithm that runs in $O^*(\sqrt{N}L)$ time. A related question is whether the problem of finding a vertex of a polytope with the objective function value smaller than that of a given interior point of the polytope is $P$-complete.

The other direction of research is to attempt to use the special structure of the bipartite matching problem to obtain an interior-point algorithm for this problem that finds an almost-optimal fractional solution in less that $O^*(\sqrt{m})$ time; an $O^*(1)$ bound would be especially interesting, since in combination with results of Section 4 it would imply that bipartite matching is in $NC$.

## Acknowledgments

We would like to thank an anonymous referee for very useful comments on an earlier version of this paper.

## References

[1] A. K. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM J. Comput.*, 13(2):423–439, May 1984.

[2] S. Fortune and J. Wyllie. Parallelism in Random Access Machines. In *Proc. 10th ACM Symp. on Theory of Computing*, pages 114–118, 1978.

[3] H. N. Gabow. Using Euler Partitions to Edge-Color Bipartite Multi-graphs. *Int. J. Comput. Inform. Sci.*, 5:345–355, 1976.

[4] A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya. Sublinear-Time Parallel Algorithms for Matching and Related Problems. In *Proc. 29th IEEE Symp. on Found. of Comp. Sci.*, pages 174–185, 1988.

[5] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, 4:373–395, 1984.

[6] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a Maximum Matching is in Random NC. *Combinatorica*, 6:35–48, 1986.

[7] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *J. Assoc. Comp. Mach.*, 27:831–838, 1980.

[8] C. Leiserson and B. Maggs. Communication-efficient parallel graph algorithms. In *Proc. of International Conference on Parallel Processing*, pages 861–868, 1986.

[9] G. F. Lev, N. Pippenger, and L. G. Valiant. A Fast Parallel Algorithm for Routing in Permutation Networks. *IEEE Trans. on Comput.*, C-30:93–100, 1981.

[10] G. L. Miller and J. Naor. Flow in Planar Graphs with Multiple Sources and Sinks. Unpublished Manuscript, Computer Science Department, Stanford University, Stanford, CA, 1989.

[11] R. D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. Part I: Linear programming. *Mathematical Programming*, 44:27–41, 1989.

[12] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, pages 105–131, 1987.

[13] V. Pan and J. Reif. Efficient Parallel Solution of Linear Systems. In *Proc. 17th ACM Symposium on Theory of Computing*, pages 143–152, 1985.

[14] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[15] J. Renegar. A Polynomial Time Algorithm, Based on Newton's Method, for Linear Programming. *Mathematical Programming*, 40:59–94, 1988.

[16] A. Schrijver. *Theory of Linear and Integer Programming*. J. Wiley & Sons, 1986.

[17] M. J. Todd. Recent developments and new directions in linear programming. In *Mathematical Programming: Recent Developments and Applications*, pages 109–159. Kluwer Academic Publishers, 1989.