May 1989

Report No. STAN-CS-89-1259

Interior-Point Methods in Parallel Computation

by

A. V. Goldberg, S. A. Plotkin, D. B. Shmoys, and É. Tardos

Department of Computer Science

Stanford University

Stanford, California 94305

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-O 188

1 a REPORT SECURITY CLASSIFICATION

1 b RESTRICTIVE MARKINGS

2a SECURITY CLASSIFICATION AUTHORITY

3 DISTRIBUTION /AVAILABILITY OF REPORT
Unclassified: Distribution Unlimite

2b DECLASSIFICATION / DOWNGRADING SCHEDULE

4 PERFORMING ORGANIZATION REPORT NUMBER(S)
STAN-CS-89-1259

5 MONITORING ORGANIZATION REPORT NUMBER(S)

6a NAME OF PERFORMING ORGANIZATION
Computer Science Dept.

6b OFFICE SYMBOL (if applicable)

7a NAME OF MONITORING ORGANIZATION

6c. ADDRESS (City, State, and ZIP Code)
Stanford University
Stanford, CA 94305

7b ADDRESS (City, State, and ZIP Code)

8a NAME OF FUNDING /SPONSORING ORGANIZATION
Partial: ONR

8b OFFICE SYMBOL (If applicable)

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
N00014-88-K-0166

8c ADDRESS (City, State, and ZIP Code)

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Interior-Point Methods in Parallel Computation

12 PERSONAL AUTHOR(S)
Goldberg, Plotkin, Shmoys, and Tardos

13a TYPE OF REPORT	13b TIME COVERED FROM ___ TO ___	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
		1989, May	14

16 SUPPLEMENTAPY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

In this paper we use interior-point methods for linear programming, developed in the contest of sequential computation, to obtain a parallel algorithm for the bipartite matching problem. Our algorithm runs in $O^*(\sqrt{m})$ time[1]. Our results extend to the weighted bipartite matching problem and to the zero-one minimum-cost flow problem, yielding $O^*(\sqrt{m}\log C)$ algorithms'. This improvés previous bounds on these problems and illustrates the importance of interior-point methods in the contest of parallel algorithm design.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION

22a NAME OF RESPONSIBLE INDIVIDUAL
Andrew Goldberg

22b TELEPHONE (Include Area Code)
(415)723-2273

22c OFFICE SYMBOL

DD Form 1473, JUN 86 Previous editions are obsolete SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

Interior-Point Methods in Parallel Computation

*Andre w 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‡
Department of Mathematics
M.I.T.
Cambridge, MA 02139

Éva Tard os§
Department of Mathematics
M.I.T.
Cambridge, MA 02139
and
Computer Science Department
Eötvös University
Budapest

May 1989

*Research partially supported by NSF Presidential Young Investigator Grant CCR-885809'7, IBM Faculty Development Award, and ONR Contract N00014-88-K-0166.

† Research partially supported by ONR Contract N00014-88-K-0166.

‡Research partially supported by an NSF Presidential Young Investigator award with matching support, from UPS, IBM, and Sun Microsystems, by Air Force contract AFOSR-86-0078, as well as by the IBM award of the first author.

§Research partially supported by Air Force contract AFOSR-86-0078, as well as by the IBM award of the first

Abstract

In this paper we use interior-point methods for linear programming, developed in the contest of sequential computation, to obtain a parallel algorithm for the bipartite matching problem. Our algorithm runs in $O^*(\sqrt{m})$ time[1]. Our results extend to the weighted bipartite matching problem and to the zero-one minimum-cost flow problem, yielding $O^*(\sqrt{m}\log C)$ algorithms?. This improves previous bounds on these problems and illustrates the importance of interior-point methods in the contest of parallel algorithm design.

1 Introduction

In this paper we use interior-point methods for linear programming, developed in the contest, of sequential computation, to obtain a parallel algorithm for the bipartite matching problem. Although Karp, Upfal, and Wigderson [20] have shown that the bipartite matching problem is in RNC (see also [25]), this problem is not known to be in NC. Special cases of the problem are known to be in NC. Lev, Pippenger, and Valiant [23] gave an NC algorithm to find a perfect matching in a regular bipartite graph. (This algorithm is based on a sequential algorithm of Gabow and Kariv [12]; see also [5] .) Miller and Naor [24] gave an NC algorithm to find a perfect matching in a. planar bipartite graph (if one exists) and other special cases are considered in [16].

The previous best deterministic algorithm for the problem. due to Goldberg, Plotkin, and Vaidya. [13], runs in $O^*(n^{2/3})$ time. This algorithm is based on combinatorial algorithms for the maximum flow and bipartite matching problems [G, 7, 8, 14, 17] and on a parallel connectivity algorithm [26]. In this paper we describe an $0^*(\sqrt{m})$ algorithm for the bipartite matching problem that is based on an interior-point algorithm for linear programming and on Gabow's algorithm [11] for edge-coloring bipartite graphs. For graphs of low-to-moderate density. this bound is better than the best previous bound mentioned above.

The significance of the bipartite matching problem has been well-recognized in the contest of sequential computation, combinatorics, and graph theory. More recently. the importance of the problem for parallel computation has been recognized as well. The efficiency of several parallel algorithms depends on the parallel complexity of the bipartite matching problem. For example, Aggarwal and Anderson [1] and Aggarwal, Anderson, and Kao [2] show, respectively, that an NC algorithm for bipartite matching implies NC algorithms for the problem of constructing a depth-first search tree in undirected and directed graphs.

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. The previous best algorithm for the zero-one minimum-cost flow problem runs in $O^*((nm)^{2/5}\log C)$ time [13]. The new algorithm is better for both the zero-one maximum flow and the zero-one minimum-cost flow problems for all graph densities.

[1]Throughout the paper, n and m denotes the number of nodes and edges of the input graph. An algorithm runs in $O^*(f(n))$ time if it runs in $O(f(n)\log^k(n))$ time for some constant k.

[2]Throughout the paper we assume that all costs and weights are integers in the range $[-C\ldots C]$, where C > 1.

An interior-point algorithm works as follows. The algorithm starts with a point in the interior of the feasible region of the linear program. In its main loop, the algorithm moves from one interior point to another, decreasing the value of a *potential function* 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-zip* stage of the algorithm converts this near-optimal solution into an optimal basic solution.

Karmarkar's revolutionary paper [19] 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.[3] Gonzaga [15] discovered a simple variation of Karmarkar's algorithm that uses an affine transformation instead of the projective transformation used by Karmarkar. Renegar [28] was the first to give an interior-point algorithm that runs in 0($\sqrt{N}L$) iterations. A different $O(\sqrt{N}L)$-iteration algorithm was developed by Ye [33], which is an improvement of Gonzaga's algorithm (a similar algorithm is described in [10]). The matching algorithm discussed in this paper is based on Ye's linear programming algorithm. The fastest linear programming algorithm currently known is due to Vaidya [32]. This algorithm is based on Renegar's method and terminates in the same number of iterations, but reduces the time per iteration using fast matrix multiplication, rank-one updates, and careful balancing. See [30] for a. survey of the interior-point algorithms.

Interior-point algorithms have proved to be an important tool for developing efficient sequential algorithms for linear programming, its special cases, and quadratic programming (see *e.g.* [18]). In this paper we apply these tools in the context 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 takes $O(\log^3 \mathrm{N})$ time on a PRAM using N^3 processors [26].

Roughly speaking, every $\sqrt{N}$ iterations of an 0($\sqrt{N}L$) iteration interior-point algorithm decrease the gap between the current value of the objective function and the optimum value 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 optima.1 value is at most N. In Section 3, we give such a formulation with N = O(m) and L = 0(log II). 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 0*($\sqrt{m}$) time. In this paper we develop an algorithm running in this time bound that finds a maximum matching as well as its value.

For this we need to overcome two difficulties. First, we need to find an initial interior point with small potential function value, so that the number of iterations is 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, which makes L, and therefore the number of iterations of the main loop, superlinear.

[3] N and **L** denote the number of variables and the size of the linear program. See Section 2 for formal definitions.

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 [11].

Our techniques apply to the more general masimum-weight matching problem. The algorithm and its analysis are only slightly more involved in this more general case. and for brevity, we focus on it. The results for bipartite matching are obtained as a simple corollary of the results for weighted bipartite matching. The main loop of our masimum-weight matching algorithm runs in $O^*(\sqrt{m}\log C)$ time, and the finish-up stage runs in O*(logC) time. 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. [4, 20]) gives $O^*(\sqrt{m}\log$ C) algorithms for these problems.

This paper is organized as follows. Section 2 introduces definitions and terminology and reviews Ye's linear programming algorithm. Section 3 gives a linear programming formulation of the bipartite matching problem that has an initial interior-point with a small potential function value, and shows how to use the linear programming algorithm to obtain a near-optima.1 fractiona. matching. Section 4 describes a parallel procedure that, in O*(logC) 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, see [27, 29]. We also give an overview of Ye's 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 $w : E \longrightarrow$ R. 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. G = *(V, E)* denotes the (bipartite) input graph, *n* denotes the number of nodes in G, *m* denotes the number of edges in G, and C denotes the maximum absolute value of the weights of edges in G, which we assume to be integral. 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 ith coordinate of x. We use a CRCW PRAM [9] as our model of parallel computation.

It is well known that the node-edge incidence matrix of a bipartite graph is totally unimodular. Therefore, any optimal solution of the following linear program is the convex combination of maximum-weight matchings, and hence the optimal value of this linear program is equal to the

maximum weight of a matching.

$$\textit{Matching-l:}\quad \left.\begin{array}{lll} \text{maximize} & w^t f & \\ \text{subject to:} & \sum_{e\in\delta(v)} f(e) \le 1, & \text{for each v} \in V, \\ & f \ge 0. & \end{array}\right\}$$

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

Ye's algorithm handles linear programs in the following form:

$$\textit{Primal LP:}\quad \left.\begin{array}{lll} \text{minimize} & c^t x & \\ \text{subject to:} & Ax = b, & \\ & x \ge 0, & \end{array}\right\}$$

where A is a matrix, and b, c and x are vectors of 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 linear programs we consider. A vector x is a *feasible solution* if it satisfies the constraints $Ax = b$ and $x \ge 0$. A feasible solution x is *optimal* if it minimizes the objective function value $c^t x$, and is an *interior point* if it is in the interior of the feasible region, i.e., if coordinates of x are positive.

The linear programming duality theorem states that the minimum value of the Prima.1 LP is equal to the maximum value of the following *Dual LP*:

$$\textit{Dual LP:}\quad \left.\begin{array}{lll} \text{maximize} & b^t\pi & \\ \text{subject to:} & A^t\pi + s = c, & \\ & s \ge 0, & \end{array}\right\}$$

where π and s are the variables of the Dual LP, the dimension of π 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.

Let x be a feasible solution to the Primal LP, and let (π, 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.*

Ye's algorithm is based on algorithms of Gonzaga [15] and Todd and Ye [31]. Freund [10] describes a very similar algorithm, and gives a detailed discussion of a good choice of q (defined below). The algorithm is applied to a pair of primal and dual linear programs in the above form. It starts with a vector (x_0, π_0, s_0), where x_0 and (π_0, s_0) are interior points of the primal and dual linear problems, respectively. At each iteration of the main loop, the algorithm moves either from the current interior point of the primal problem to another interior point of the problem, or from the current interior point of the dual problem to another interior point of the problem. Progress is measured by a potential function

$$\Phi(x, s) = q\log(x^t s) - \sum_{i=1}^{N} \log(x(i)s(i)) - \text{N log } N.$$

where $q = N + \sqrt{N}$. Each iteration reduces this potential function by a constant.

The number of iterations of interior-point algorithms depends on a parameter L that is related to the size of the input numbers. This parameter is often defined to be the total number of bits in the binary description of all coefficients in *A*, *b* and c. We use a different definition [19, 32], which leads to a much smaller value of L in the case of the bipartite matching problem. Let D(A) denote the maximum absolute value of a subdeterminant of *A*, and let *B* denote the maximum absolute value of the coefficients of *b* and c. Then L is defined by

$$\mathrm{L} = \log D(A) + \log \mathrm{N} + \log B.$$

With this definition, $L = O(\log(nC))$ for linear program *Matching-l.*

When the value of the current feasible solution x is less then 2^{-L} away from the optima.1 value, a standard (sequential) rounding procedure yields an optimal solution.

The following lemma is the basis for the analysis of Ye's algorithm.

Lemma 2.1 [33] If we have an initial solution (x_0, π_0, s_0) such that $\Phi(x_0, s_0) \leq O(\sqrt{N}L)$, then after 0($\sqrt{N}L$) iterations the duality gap $x^t s < 2^{-L}$.

Proof': Recall that the algorithm decreases the potential function by a constant per iteration. Thus after $O(\sqrt{N}L)$ iterations $\Phi(x, s) < -\sqrt{N}L$. The potential function can be rewritten as follows:

$$\Phi(x, \mathrm{s}) = \sqrt{N}\log(s^t x) + \sum_{i=1}^{N} \log \frac{s^t x}{s(i)x(i)} - \mathrm{N}\log \mathrm{N}. \qquad (1)$$

Note that the second term is minimized when the values of $s(i)x(i)$ are the same for all i, and therefore this term is at least N log N. Therefore, if the potential function value is at most $-\sqrt{N}L$, then $\sqrt{N}\log(s^t x) < -\sqrt{N}L$. Hence we have $x^t s < 2^{-L}$. ∎

To obtain an 0($\sqrt{N}L$) bound on the number of iterations, one has to provide an initial solution (x_0, π_0, s_0) with $\Phi(x_0, s_0) \leq \sqrt{N}L$. Consider the potential function Φ written as in (1). It is easy to find an initial solution for which the first term is bounded by $O(\sqrt{N}L)$. The difficulty is to guarantee that the second term is fairly close to the N log N lower bound. A good initial solution is one where the terms $s(i)x(i)$ are almost equal. As mentioned in [3], Ye proposed a way to obtain an equivalent formulation with such an initial solution. This uses the usual definition of *L,* but can also be shown to work for the definition of *L* that we use in this paper. In the nest section we provide a slightly simplified construction for the bipartite matching problem.

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 Ye's algorithm and has an initial solution with small potential function

value. Then we show how to compute a near-optimal fractional matching from this initial solution.

We restate the matching problem as follows:

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

where 1 denotes the vector all of whose coordinates are 1. We denote the objective function of this linear program by c. 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 π and s, where $\pi(i)$ for $i = 1, \ldots, n$ is the dual variable corresponding to the primal constraint for node v_i, and $\pi(n + 1)$ is the dual variable corresponding to the constraint $(*)$. Note that for this linear program, $L = O(\log(nC))$.

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 optima.1 solution. 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 $g^t 1 + f^t 1 \leq n$ is automatically satisfied. This constraint, however, allows us to obtain an initial solution for the dual problem such that the dual slack vaaiables corresponding to the primal variables f, g and y are roughly equal. A natural dual solution is to set the vector π to 0, and set the dual slack variables equal to the primal objective coefficients. Even if this were feasible, the coefficients are very different (the coefficients of g are zero, the others are not), and so this is not a good starting solution. However, by setting the coordinate of π corresponding to the additional constraint to be a large negative number (while keeping the others equal to 0) the slacks (except for the one corresponding to z) are made feasible, and roughly equal to the dual variable for $(*)$. 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). We choose the dual variable corresponding to $(*)$ so that for the initial primal solution mentioned above, the $x(i)s(i)$ terms are all roughly equal. As we shall see, this results in a small initial value of the second term of Φ (written as in (1)).

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

$$f = 1, g = 1, y = 1, z = n - 1.$$

The initial dual solution (π_0, s_0) is defined by

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

where $a(i)$ is the i-th coefficient of the equation $(*)$ in the definition of the Matching-2 LP.

The following two lemmas formalize the above intuition.

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': It suffices 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. Note that any fractional matching f 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 $z_1 \frac{N^2C}{n-1} > z_1 NC$. 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, which implies that any optimal solution must have $z = 0$. ∎

Lemma 3.2 The vectors x_0 and (π_0, s_0) are interior-point solutions of the primal and the dual problems, respectively. The value of the potential function $\Phi(x_0, s_0)$ is at most $O(\sqrt{N} \log(nC))$.

Proof': The first claim of the lemma is easy to verify. To verify the second claim, consider the potential function written as in **(1).** The first term is at most $O(\sqrt{N} \log(nC))$. We show that the second term is at most N log N + $O(1)$. First we show that for every i, $s^t x/(s(i)x(i)) \leq$ N + $O(1)$. Recall that N = $n + m + 2$ and note that

$$s_0{}^t x_0 = nN^2C + mN^2C - w^t \mathbf{1} + 2N^2C.$$

consider each type of variable separately.

- For variables $s(i)$ and $x(i)$ corresponding to z, y, and g, we get

$$s_0^t x_0/(s_0(i)x_0(i)) = n + m + 2 - \frac{w^t 1}{N^2C} \leq N + \mathrm{O}(1).$$

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

$$s_0^t x_0/(s_0(i)x_0(i)) = n + m + 2 - \frac{w^t 1 + nw(i) + mw(i) + 2w(i)}{N^2C - w(i)} \leq \mathrm{N} + 0\ (1).$$

Since $\log(1 + h) \leq h$ for $h >$ -1, the above calculations imply that

$$\sum_{i=1}^{N} \log \frac{s_0^t x_0}{s_0(i)x_0(i)} \leq N \log(N + \mathrm{O}(1)) \leq N \log N + \mathrm{O}(1).$$

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 nest section we show how to find an optimal matching.

The following lemma is based on the fact that the objective coefficient of z has been chosen large enough to ensure that any near-optimal solution to *Matching-2* can be rounded to a nearby feasible solution to *Matching-l.*

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

Proof': Lemmas 2.1 and 3.2 imply that, after $0(\sqrt{N}\log(nC)) = 0(\sqrt{m}\log(nC))$ iterations of the LP algorithm, we obtain a point (x, π, s) with duality gap $x^t s \leq 1/4$. Hence we have

$$-w^t f + \frac{N^2 C}{n-1} z + w^t f^* \leq \frac{1}{4}, \tag{3}$$

where f^* is an optimal solution to *Matching-1*. Since $z \geq 0$, this implies that $w^t f^* - w^t f \leq 1/4$. 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, $w^t f^* \geq w^t f - znC$. From (3), this implies that $z\frac{CN^2}{n-1} \leq 1/4 + znC$. 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, $w^t f^* - w^t f_1 \leq 1/4 + (mC)/(4mC) \leq 1/2$. 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 $(4Cm)^{-1}$ and $z < (4Cm)^{-1}$. This implies that

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

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

Corollary 3.4 A fractional bipartite matching with cardinality at most $1/2$ less than that of the maximum cardinality matching can be computed in $0^*(\sqrt{m}\log C)$ time on a PRAM with m^3 processors. The cardinality of the maximum matching can be computed within the same bounds.

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 0*(log C) algorithm for converting any such fractional matching into a masimum-weight matching. Note that for the unweightecl bipartite matching, this algorithm runs in polylogarithmic time.

Let f be a fractional bipartite matching which has weight at most $1/2$ less than the maximum weight. 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 4 by

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

By definition, 4 is an integer power of 2 and 4 = $O(mC)$. 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 $w^t f^* - w^t f' < 1$.

Nest we show how to construct from f' a multi-graph that will allow us to find f^*. Consider a multi-graph $G' = (V', E')$ with the edge set containing $\Delta \cdot f'(e)$ copies of e for every $e \in E$, and no other edges. The following lemma shows a relationship betweeng this multigraph and maximum-weight matchings of G.

Lemma 4.1 For any coloring of the edges of G' with Δ 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 4 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 4 colors. Since G' is a bipartite graph and its maximum degree is bounded by 4, which is a power of 2, we can use a parallel implementation of Gabow's algorithm [11] to edge-color G' using 4 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 weight of an edge is halved, and then recurses on the part with greater weight. A subroutine to find 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 A;
    d' ← A;
    while d' > 1 do begin
        E0 ← {e | e ∈ E, d' . f'(e) is odd};
        (E1, E2) ← Degree-Split(V, E0);
        W1 ← w(El) ;
        W2 ← w(E2);
        if W1 ≥ W2
          then begin
            for e ∈ E1 do f'(e) ← f(e) + l/d';
            for e ∈ E2 do f'(e) ← f'(e) – l/d';
            end;
          else begin
            for e ∈ E2 do f'(e) ← f'(e) + l/d';
            for e ∈ E1 do f'(e) ← f'(e) – l/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. Then it computes from the fractional matching f' with common denominator Δ, two fractional matchings f_1 and f_2 such that $f' = \frac{1}{2}(f_1 + f_2)$ and both f_1 and f_2 have common denominator $\Delta/2$. This is accomplished with the help of the procedure *Degree-split* that 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. Then f' is replaced by f_1 or f_2 depending on which one has larger weight. This process is iterated $O(\log(nC))$ times, until the current fractional matching is integral. This matching has an integral weight that is more than $w^t f^* - 1$, and therefore the matching is optimal.

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' = A. 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'.

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

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 $1/d_1$ 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 therefore at most one.

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

The *Degree-Split* procedure is described in Figure 2. The following two lemmas imply the desired time bound.

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 runs 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 $O(\log\ n)$ time using m processors [2 1, 22]. 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 $O(\log \text{nlog } nC)$ 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. ∎

Corollary *4.5* On unweighted bipartite matching problem, the algorithm *Round* runs in $O(\log^2 n)$ time using m processors.

Theorem *4.6* A maximum-weight bipartite matching can be computed in $O^*(\sqrt{m}\log C)$ time using m^3 processors.

Proof: Immediate from Lemmas 3.3 and 4.4. ∎

Corollary 4.7 A maximum cardinality bipartite matching can be computed in $O^*(\sqrt{m})$ time using m^3 processors.

5 Conclusions

Interior-point methods have proved to be very powerful in the context of sequential computation. In this paper we show how to apply these methods to the design of parallel algorithms. We believe that these methods will find more applications in the context of parallel computation, and 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 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-optima.1 fractional solution in less that $O^*(\sqrt{m})$ time; an O*(l) bound would be especially interesting, since in combination with results of Section *4* it would imply that bipartite matching is in NC.

References

[1] A. Aggarwal and R. J. Anderson. A Random NC Algorithm for Depth First Search. In *Proc. 19th ACM Symposium on Theory of Computing,* pages 325-334, 1987.

[2] A. Aggarwal, R. J. Anderson, and M. Y. Kao. Parallel Depth-First Search in General Directed Graphs. In *Proc. 21st ACM Symposium on Theory of Computing*, page (to appear), 1989.

[3] K. M. Anstriecher and R. A. Bosch. Long Steps in an $O(n^3L)$ Algorithm for linear programming. Unpublished manuscript, Yale School of Management, Yale University, 1989.

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

[5] R. Cole and J. Hopcroft. On Edge Coloring Bipartite Graphs. *SIAM J. Comput.*, 11:540–546, 1992.

[6] E. A. Dinic. Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.

[7] S. Even and R. E. Tarjan. Network Flow and Testing Graph Connectivity. *SIAM J. Comput.*, 4:507–518, 1975.

[8] L. R. Ford? Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, Princeton, NJ., 1962.

[9] S.' Fortune and J. Wyllie. Parallelism in Random Access Machines. In *Proc. 10th A CM Symp. on Theory of Computing,* pages 114-118, 1978.

[10] R. M. Freund. Polynomial-Time Algorithms for Linear Programming Based only on Primal Scaling and Projective Gradients of a Potential Function. Technical Report OR- 182-88, Operations Research Center, M.I.T., 1988.

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

[12] H. N. Gabow and 0. Kariv. Algorithms for Edge-Coloring Bipartite Graphs and Multigraphs. *SIAM J. Comput.*, 11:117–129, 1982.

[13] 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.

[14] i-1. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum Flow Problem. *J. Assoc. Comput. Mach.*, 35:921–940, 1988.

[15] C. C. Gonzaga. An Algorithm for Solving Linear Programming in $O(n^3L)$ Operations. In N. Megiddo, editor, *Progress in Mathematical Programming,* pages t-28. Springer Verlag, Berlin, 1989.

[16] D. Y. Grigoriev and M. Karpinski. The Matching Problem for Bipartite Graphs with Polynomially Bounded Permanents is in *NC.* In *Proc. 28th IEEE Synap. on Foundations of Comp. Sci.*, pages 166-172, 1987.

[17] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[18] S. Kapoor and P. M. Vaidya. Fast Algorithms for Convex Quadratic Programming and Multicommodity Flows. In *Proc. 18th ACM Symp. on Theory of Computing*, pages 147–159, 1986.

[19] N. Karmarksr. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorirn.* 4:373–395, 1984.

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

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

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

[23] 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.

[24] 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.

[25] Ii. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, pages 105-131, 1987.

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

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

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

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

[30] M. J. Todd. Recent Developments and New Directions in Linear Programming. Technical Report 829, School of Operations Research and Industrial Engineering, Cornell University. 1988.

[31] M. J. Todd and Y. Ye. A Centered Projective Algorithm for Linear Programming. Technical Report 763, School of Operations Research and industrial Engineering, Cornell University, 1989.

[32] P. M. Vaidya. Speeding up Linear Programming Using Fast Matrix Multiplication. Technical Memorandum, AT&T Bell Laboratories, Murray Hill, NJ, 1989.

[33] Y. Ye. An $O(n^3L)$ Potential Reduction Algorithm for Linear Programming. Unpublished manuscript, The University of Iowa, 1989.