COMBINATORICA
Akadémiai Kiadó — Springer-Verlag

COMBINATORICA 8 (2) (1988) 189—200

IMPROVED PROCESSOR BOUNDS FOR COMBINATORIAL PROBLEMS IN RNC*

Z. GALIL** and V. PAN***

Received 30 March 1986

Revised 20 December 1986

Our main result improves the known processor bound by a factor of n^4 (maintaining the expected parallel running time, $O(\log^3 n)$) for the following important problem: *find a perfect matching in a general or in a bipartite graph with n vertices.* A solution to that problem is used in parallel algorithms for several combinatorial problems, in particular for the problems of finding i) a (perfect) matching of maximum weight, ii) a maximum cardinality matching, iii) a matching of maximum vertex weight, iv) a maximum $s-t$ flow in a digraph with unit edge capacities. Consequently the known algorithms for those problems are substantially improved.

1. Introduction

Given a computational problem, the usual initial goal of the design and analysis of parallel algorithms is to establish that the problem belongs to NC (or RNC), that is, to solve it with a parallel algorithm (or with a randomized parallel algorithm, respectively) in time $t=O(\log^k n)$ with $P=O(n^q)$ processors. Once such a goal is achieved, we try to improve the solution (to reduce the constants k or q or both). In particular, the initial solutions are usually not processor efficient (q is very large).

It is well known that every parallel algorithm A that uses P processors and time t gives rise to a family of parallel algorithms with P' processors $(1\leqq P'\leqq P)$ and time $t'=t\lceil P/P'\rceil$. In particular this yields a sequential algorithm $(P'=1)$ with $t'=tP$. In case P is very large, this sequential algorithm may have time complexity much inferior to the best sequential algorithm known. Thus A is useful only if the number of available processors is close to the best sequential time (within, say a polylogarithmic factor).

The best one can hope for (without improving the best sequential algorithm) is to design a parallel algorithm A with time t and P processors, such that $Pt=t'$, where t' is the time complexity of the best sequential algorithm.

---

* The results of this paper have been presented at the 26-th Annual IEEE Symp. FOCS, Portland, Oregon (October 1985).

** Partially supported by NSF Grants MCS 8303139 and DCR 8511713.

*** Supporeted by NSF Grants MCS 8203232 and DCR 8507573.

AMS subject classification (1980): 05 C 68,

In this paper we will substantially (by a factor of n^4) improve the previous best processor complexity bound of [7], for the important problem of computing a perfect matching in a general undirected graph $G=(V, E)$, $|V|=2n$. This processor improvement does not require to increase the time bound of [7]. As a result, similar (and sometimes even greater) improvements are derived for several other computational problems.

We will state the processor bounds in the two machine models of parallel computation, [3], that is, in the arithmetic model and in the (more realistic) Boolean model where each processor may perform an arithmetic (or, respectively, Boolean) operation in one unit of time. We denote the processor bounds in these two models $O_A(f(n))$ and $O_B(g(n))$, respectively. (For the problems that we consider, our asymptotic *time* bounds are the same in both models.) We estimate the number of processors in terms of n and $M(n)$ where $M(n)$ is the best sequential arithmetic time for $n\times n$ matrix multiplication. In practice (for moderate n) $n\times n$ matrix multiplication takes $2n^3-n^2$ arithmetic operations but asymptotically $M(n)=O(n^\beta)$ where $\beta<2.38$ [5]. (For simplicity we will present our processor bounds ignoring polylogarithmic factors; in fact the estimate for the exponent β seems to be a subject for a further improvement.)

Next we will state our main problem and our main result.

Given an undirected graph $G=(V, E)$, a *perfect matching* is a subset M of the edge set E such that every vertex appears in exactly one edge of M. We take $|V|=2n$, so that $|M|=n$. Testing whether a graph has a perfect matching has been known to be in RNC for some time [3], but it was not known until very recently whether the following problem is in RNC.

Problem 1. Given a graph G that has a perfect matching, find a perfect matching in G.

In a recent elegant paper, Karp, Upfal, and Wigderson [7] showed that Problem 1 is in RNC by designing a randomized parallel algorithm of time complexity $O(\log^3 n)$ that uses $O(n^4M(n))$ processors. They do not specify the computational model, but their argument only supports such a bound in the arithmetic model. More precisely, their processor bound is $O_A(n^4M(n))$ or $O_B(n^5M(n))$, see our more specific comments in Section 2. We improve that processor bound by a factor of n^4.

Theorem 1.1. *Problem 1 can be solved by a probabilistic algorithm in expected time* $O(\log^3 n)$ *using* $O_A(M(n))$ *or* $O_B(nM(n))$ *processors.*

As does the algorithm in [7], our improved version has an expected number $O(\log n)$ of iterations. Our improvement reduces each iteration to the following problem (where $k=8$).

Problem 2. Compute the inverse A^{-1} and the determinant $\det A$ of an $n\times n$ integer matrix A with entries bounded by $O(n^k)$ for a constant k.

Then Theorem 1.1 follows from the next result.

Theorem 1.2. *Problem 2 can be solved deterministically in time* $O(\log^2 n)$ *using* $O_A(M(n))$ *or* $O_B(n^2M(n))$ *processors or probabilistically in expected time* $O(\log^2 n)$ *using* $O_B(nM(n))$ *processors.*

Remark 1.1. Note that Theorem 1.2 is about exact integer matrix inversion over rationals. In case of inverting a matrix over $\mathbf{Z}_p$ for a constant p, Theorem 1.2 does not apply and it is an open problem how to compute the inverse in time $O(\log^2 n)$ with $O_B(M(n))$ processors. The known methods ([11]) require larger numbers of processors.

Remark 1.2. Recall that $A^{-1}=\text{adj } A/\det A$ where adj A is the matrix of cofactors of A. In our case adj A and det A are integer. (The matrix A^{-1} is output as the pair of adj A and det A.)

In [7] the authors use their solution to Problem 1.1 to demonstrate that the following problems are in RNC: (i) finding a (perfect) matching of maximum weight, when edge weights are given in unary, (ii) finding a maximum cardinality matching, (iii) finding a matching of maximum total vertex weight when vertex weights are given in binary, (iv) finding a maximum $s-t$ flow in a directed graph with unit edge capacities. The following table shows the upper bounds on the processor complexity for Problems 1 and (i)—(iv) in the Boolean circuit model. Those bounds support the time bounds $O(\log n \log^2 (nw^*))$ in (i), where w^* is the maximum weight of all edges, and $O(\log^3 n)$ in (ii)—(iv) (both in [7] and in our algorithm), compare also Remark 3.5 in Section 3.

Problem	Previous best bounds	New algorithms
1.	$O_A(n^4M(n))$, $O_B(n^5M(n))$	$O_A(M(n))$, $O_B(nM(n))$
i)	$O_B(w^*n^6M(n))$	$O_B(w^*n^2M(n))$
ii)	$O_B(n^6M(n))$	$O_B(n^2M(n))$
iii)	$O_B(n^7(M(n)))$	$O_B(n^3M(n))$
iv)	$O_B(n^{12}M(n))$	$O_B(n^2M(n))$

The new processor bounds follow from Theorem 1.1 and reductions of Problems (i)—(iv) to Problem 1 similar to those in [7]. In the case of bipartite graphs the new algorithms for problem i)—iii) can be further improved, so that the processor bounds displayed in the summary can be further decreased by a factor of n, see Lemma 3.2 in Section 3. H. Gabow [6], announced that this improvement (of the above processor bounds by a factor of n in cases i)—iii)) can be extended to general graphs also. Recently K. Mulmuley, U. Vazirani and V. Vazirani [9], improved the expected time bound for Problem 1 to $O(\log^2 n)$, but this required to increase the processor bound n^β times, $\beta>2$.

In Section 2 we present our algorithm for Problem 1, in Section 3 we extend it to solving Problems i)—iv), in Section 4 we state the related open problems.

Acknowledgement. The authors thank the referee for helpful comments, in particular, for the idea of the improvement of the proof of Lemma 2.5, which initially was more complicated.

2. Computing a perfect matching in a graph

With high probability the Krap—Upfal—Wigderson (KUW)-algorithm has $O(\log n)$ iterations. In each iteration the algorithm identifies a *redundant* set of edges RE. A set $A \subseteq E$ of edges in a graph $G=(V, E)$ is called redundant if $G'=(V, E-A)$ contains a perfect matching. The bound $O(\log n)$ on the expected number of iterations follows from the fact that in each iteration we have that (for some positive constants α and β)

$$|RE| \geq \alpha|E| \quad \text{with probability} \quad >\beta, \tag{2.1}$$

until less than $0.75\,|V|$ nonredundant edges are left. The latter case is easily treated, because at that point at least $0.25\,|V|$ edges have at least one endpoint of degree 1; all such edges can be included into a perfect matching, M.

To identify quickly a set RE, the algorithm makes use of the following crucial definition. For $S \subseteq E$, $\operatorname{rank}(S)=\max|S \cap M|$, where the maximum is taken over all the perfect matchings M in G. An immediate observation is that for any set S, the set $RE=\{e \in E-S \mid \operatorname{rank}(S)=\operatorname{rank}(S \cup e)\}$ is redundant. If i is chosen uniformly in the set $\{1, \ldots, (5/6)|E|\}$, $|E| \geq 0.75|V|$, and S is then chosen uniformly among the subsets of E of size i, it is shown that the derived RE satisfies (2.1).

The computational task in each iteration is thus:

Problem 3. For input S, $S \subset E$, compute $\operatorname{rank}(S \cup e)-\operatorname{rank}(S)$ for all $e \in E-S$.

The KUW-algorithm solves Problem 3 as follows.

At first we consider the simpler case of bipartite graphs, $G=(V, E)$, with the two sets of vertices $\{u_1, u_2, \ldots, u_n\}$ and $\{v_1, v_2, \ldots, v_n\}$. For a subset $T \subseteq E$ a matrix, $B(T, y)=(b_{ij}(T, y))$, is defined by

$$b_{ij}(T, y) = \begin{cases} yx_{ij}, & \text{if } \{u_i, v_j\} \in T; \\ x_{ij}, & \text{if } \{u_i, v_j\} \in E-T; \\ 0, & \text{if } \{u_i, v_j\} \notin E. \end{cases} \tag{2.2}$$

Here y, x_{ij} are indeterminates. Then it is observed that

$$\operatorname{rank}(T) = \deg_y \det B(T, y), \tag{2.3}$$

where $\deg_y q(y)$ denotes the degree in y of a polynomial $q(y)$ (q may also depend on other variables). The algorithm computes $\operatorname{rank}(T)$ by using random substitution for the variables x_{ij} and by computing the determinant of a matrix of polynomials (in one variable y). The degree of the resulting polynomial is equal to the rank of T, with high probability. Problem 3 is solved by computing $\operatorname{rank}(T)$ for $T \in \{S\} \cup \cup \{S \cup e \mid e \in E-S\}$. For computing the determinant of a matrix of polynomials, the very general algorithm of [2], [1], is used. To compute the $|E-S|$ ranks, [7] compute $|E-S|=\Theta(n^2)$ determinants; the algorithm of [2], [1] computes each determinant using n $M(n)$ arithmetic operations (performed with polynomials of degrees up to n in y), so a total of $O_A(n^4 M(n))$ processors is required (even provided that the fast and efficient algorithms based on the FFT are applied in order to perform the n-th degree polynomial multiplications using $O(\log n)$ time, n processors). The coefficients of the computed polynomials are generally large; such a coefficient

may require more than n binary digits to represent it, so the Boolean processor bound grows to $n^5M(n)$, even if we apply the known advanced parallel algorithms [13], for arithmetic operations with numbers.

We substantially improve this solution of Problem 3 using the following changes:

(a) Operate with integers rather than with polynomials.
(b) Make use of Theorem 1.2.
(c) Use the possibility to compute all (up to n^2) ranks for the price of one.
([12], implement an observation similar to c).)

We let $|q|_p$ denote the *p-adic order of an integer* q (where $p>1$ is a fixed integer, usually a prime), that is, the largest integer k such that p^k divides q, compare [15]. Surely $O(\log(\log q/\log p))$ *multiplications and divisions suffice to compute the p-adic order* $|q|_p$ *for given p and q.* The *p-adic order of a nonzero polynomial* $q(x)$ is defined as the least p-adic order of the coefficients of $q(x)$ and is denoted $|q(x)|_p$. Here and hereafter x denotes a set of variables $\{x_{ij}\}$.

The following lemma is obvious.

Lemma 2.1. *Let* $q_0(x)$ *denote the sum of all monomials of a polynomial* $q(x)$ *having the p-adic order* $s=|q(x)|_p$. *Let* $x^0=\{x_{ij}^0\}$ *denote a set of integer values of* x_{ij}. *Then*

$$|q(x^0)|_p = |q(x)|_p \quad \textit{unless } p^{s+1} \textit{ divides } q_0(x^0).$$

Lemma 2.1 immediately suggests a randomized algorithm for the evaluation of the p-adic order of a polynomial $q(x)$, by choosing random integers x_{ij}^0 and a random prime p in sufficiently large intervals and computing $|q(x^0)|_p$. For any matrix of polynomials $B(x)$ below we use the notation $\tilde{B}$ for $B(x^0)$.

To solve Problem 3, we "tag" the edges differently from (2.2). For $T\subseteq E$ and a prime p, we define the matrix $B(p, T)=(b_{ij}(p, T))=pB(T, 1/p)$, such that

$$b_{ij}(p, T) = \begin{cases} x_{ij}, & \text{if } \{u_i, v_j\}\in T; \\ px_{ij}, & \text{if } \{u_i, v_j\}\in E-T; \\ 0, & \text{if } \{u_i, v_j\}\notin E. \end{cases}$$

Lemma 2.2. $\deg_y \det B(T, y)=n-|\det B(p, T)|_p$.

Proof. Expand $\det B(T, y)$ and $\det B(p, T)$ as the two sums, each of $n!$ monomials $m_k(T, y)$ and $m_k(p, T)$, respectively, $k=1, 2, \ldots, n!$, and note that $\deg_y m_k(T, y)+|m_k(p, T)|_p=n$ for all k. ∎

It follows from (2.3) and Lemma 2.2 that $\operatorname{rank}(T)=n-|\det B(p, T)|_p$.

We are able to use one solution to Problem 2 for computing all the ranks in Problem 3 by observing that $B(p, S\cup e)$ differs from $B(p, S)$ in exactly one entry and thus

$$\det B(p, S\cup e) = \det B(p, S)+(1-p)x_{ij}(\operatorname{adj} B(p, S))_{ij}, \tag{2.4}$$

where $e=\{u_i, v_j\}$. So we solve Problem 3 in case of bipartite graphs as follows:
— Choose a random prime p, $n\leqq p\leqq n^r$, $r>4$ fixed.
— Choose a random assignment $x^0=\{x_{ij}^0\}$, $x_{ij}^0\in[1, \ldots, n^r]$.
— For $\tilde{B}=\tilde{B}(p, S)$ compute $\det \tilde{B}$ and $\operatorname{adj} \tilde{B}$ as in [10].

— Use (2.4) to compute $\det \tilde{B}(p, S \cup e)$ for all $e \in E-S$.
— Compute the p-adic orders of $\det \tilde{B}$ and $\det \tilde{B}(p, S \cup e)$ for all $e \in E-S$ and use them as the corresponding ranks for computing the output.

Lemma 2.3. *The probability of an error in any of the outputs is* $O(1/n^{r-4})$.

Proof. Let $q(x)$ denote $\det B(p, T)$ where $T=S$ or $T=S \cup e$. Note that $q(x) \neq 0$. It follows from [14], that

$$\text{Probability}\,(q_0(x^0)=0) \leqq n^{2-r} = 1/n^{r-2}. \tag{2.5}$$

Next assume that $q_0(x^0) \neq 0$. Hadamard's inequality, [8], implies that $|q_0(x^0)| \leqq n^{(r+0,5)n}$, so $q_0(x^0)/p^s$ for $s \leqq n$ has less than $(r+0.5)n-s$ distinct prime factors greater than n. On the other hand, we choose p at random among at least $n^r/(cr \log n)$ distinct prime factors greater than n for a constant c, $0<c<\infty$. Therefore

$$\text{Probability}\,(p^{s+1} \text{ divides } q_0(x^0)) \leqq cr((r+0,5)n-s)\log n/n^r \tag{2.6}$$

unless $q_0(x^0)=0$.

(2.5) and (2.6) imply that the probability of an error in computing $|q(x)|_p$ is $O(1/n^{r-2})$. Computing the p-adic orders of $\det B(p, T)$ for all the $1+|E-S|$ edge sets T may increase that error probability at most $1+|E-S| \leqq n^2$ times. ∎

For the general graph the KUW-algorithm defines a $2n \times 2n$ matrix, $B^*(T, y) = (b^*_{ij}(T, y))$, by

$$b^*_{ij}(T, y) = \begin{cases} yx_{ij}, & \text{if } \{v_i, v_j\} \in T \text{ and } i<j; \\ -yx_{ji}, & \text{if } \{v_i, v_j\} \in T \text{ and } i>j; \\ x_{ij}, & \text{if } \{v_i, v_j\} \in E-T \text{ and } i<j; \\ -x_{ji}, & \text{if } \{v_i, v_j\} \in E-T \text{ and } i>j; \\ 0, & \text{if } \{v_i, v_j\} \notin E, \end{cases} \tag{2.7}$$

while we use the matrix $B^*(p, T)=(b^*_{ij}(p, T))=pB(T, 1/p)$, such that

$$b^*_{ij}(p, T) = \begin{cases} x_{ij}, & \text{if } \{v_i, v_j\} \in T \text{ and } i<j; \\ -x_{ji}, & \text{if } \{v_i, v_j\} \in T \text{ and } i>j; \\ px_{ij}, & \text{if } \{v_i, v_j\} \in E-T \text{ and } i<j; \\ -px_{ji}, & \text{if } \{v_i, v_j\} \in E-T \text{ and } i>j; \\ 0, & \text{if } \{v_i, v_j\} \notin E, \end{cases} \tag{2.8}$$

compare (2.2).

The KUW-algorithm immediately follows from the next important lemma.

Lemma 2.4, [7]. $2 \operatorname{rank}(T) = \deg_y \det B^*(T, y)$.

We immediately extend Lemma 2.2 and obtain that

$$\deg_y \det B^*(T, y) = 2n - |\det B^*(p, T)|_p. \tag{2.9}$$

We encounter, however, an obstacle when we vary T: we cannot apply (2.4) because $B^*(p, S)$ differs from $B^*|(p, S \cup e)$ in *two* entries, so we need to extend Lemma 2.4.

For $e=\{v_i, v_j\}$ we define two matrices, $B^+(S, e, y)$, $B^-(S, e, y)$, with the same entries as $B^*(S, y)$ (and $B^*(S\cup e, y)$), except that for the entries corresponding to the edge $e=\{v_i, v_j\}$, $i<j$, we set

$$b^+_{ij}(S, e, y) = yx_{ij},\ b^+_{ji}(S, e, y) = -x_{ij},$$
$$b^-_{ij}(S, e, y) = x_{ij},\ b^-_{ji}(S, e, y) = -yx_{ij}, \tag{2.10}$$

that is, in each of the two matrices only one of the two entries corresponding to e is tagged differently from $B^*(S, y)$. Consequently, we can use an analog of (2.4) with $B^+(S, e, y)$ or $B^-(S, e, y)$ instead of $B^*(S\cup e, y)$. Hereafter we will designate $D(e, y)=\det B^+(S\cup e, y)+\det B^-(S\cup e, y)$. Now we are ready to state the desired extension of Lemma 2.4, which we will combine with the equation (2.9) and with its following simple extension,

$$\deg_y D(e, y) = 2n-|D(p, e)|_p,\ D(p, e) = pD(S\cup e, 1/p).$$

Lemma 2.5. *For each* $e\in E-S$, $\operatorname{rank}(S\cup e)-\operatorname{rank}(S)=\deg_y D(e, y)-\deg_y \det B^*(S, y)$.

Proof. The proof will follow the line of the proof of Lemma 2.4 in [7]. Let P denote the set of all the permutations σ of $\{1, 2, \ldots, n\}$. The set P is partitioned into two subsets, EP and OP. EP is the set where all cycles of each permutation are even; OP is the set where each permutation includes an odd cycle. Let M denote the set of all permutations where every cycle has length 2. Each permutation of P defines the unique monomial in $\{x_{ij}\}y$ in each of $\det B^*(S, y)$, $\det B^+(S\cup e, y)$, $\det B^-(S\cup e, y)$ (the monomial may equal 0). Let a permutation σ define monomials, $m^*(\sigma)$, $m^+(\sigma)$, $m^-(\sigma)$, of $\det B^*(S, y)$, $\det B^+(S, y)$, $\det B^-(S, y)$, respectively. The argument of the proof of Lemma 2.4 in [7] and its immediate extension lead to the following results.

Proposition 2.1. *Either* $m^*(\sigma)=0$ *or* $\deg_y m^*(\sigma)$ *equals the number of edges* $\langle h, \sigma(h)\rangle$ *in* S. *Either* $m^+(\sigma)=m^-(\sigma)=0$ *or* $i=\sigma(j)$, $j=\sigma(i)$, *and* $\deg_y(m^+(\sigma)+m^-(\sigma))-1$ *equals the number of edges* $\langle h, \sigma(h)\rangle$ *in* $S\cup e$ *or* $\deg_y(m^+(\sigma)+m^-(\sigma))$ *equals the number of edges* $\langle h, \sigma(h)\rangle$ *in* $S\cup e$.

Proposition 2.2.

$$\sum_{\sigma\in OP} m^*(\sigma) = \sum_{\sigma\in OP} m^+(\sigma) = \sum_{\sigma\in OP} m^-(\sigma) = 0.$$

Proposition 2.3. *If* $\sigma\in EP$, *then there exists permutations* τ, τ^+, τ^- *in* M, *such that* $\deg_y m^*(\tau)\geqq\deg_y m^*(\sigma)$, $\deg_y m^+(\tau^+)\geqq\deg_y m^+(\sigma)$, $\deg_y m^-(\tau^-)\geqq\deg_y m^-(\sigma)$.

Since the monomials do not cancel each other, Propositions 2.1—2.3 combined immediately imply Lemma 2.5. ∎

The algorithm that solves Problem 3 for general graphs follows. (Lemma 2.3 still holds.)

— Choose a random prime p, $2n\leqq p\leqq(2n)^r$, $r>4$ fixed.
— Choose a random assignment $x^0=\{x^0_{ij}\}$, $x^0_{ij}\in[1, \ldots, (2n)^r]$.
— For $\tilde{B}^*\equiv\tilde{B}^*(p, S)$ compute $\det\tilde{B}^*$ and $\operatorname{adj}\tilde{B}^*$ as in [10].
— Use the analogs of (2.4) for computing $\det\tilde{B}^+(p, S, e)+\det\tilde{B}^-(p, S, e)$ and output $\operatorname{rank}(S)$ and $\operatorname{rank}(S\cup e)$ for all $e\in E-S$.

3. Extension to other combinatorial problems

The extension of Theorem 1.1 to problems i)—iv) listed in the Introduction follows the pattern of [7]. In particular problem i) of finding a perfect matching of maximum weight in a graph $G=(V, E, w)$ is reduced in [7] to computing $\mathrm{Rank}_w(T)$ for $T=S$, $T=S\cup e$, $T\subseteq E$; $\mathrm{Rank}_w(T)$ is defined to be the maximum number of edges from T that participate in a perfect matching of maximum weight in G. Surely we may reduce finding a maximum weight matching in G to finding a *perfect* matching of maximum weight in the complete graph with the same vertices and with zero weights for all edges not in E.

To compute $\mathrm{Rank}_w(T)$, [7] define the following matrix, $B(T, w, y, z)=(b_{ij}(T, w, y, z))$,

$$b_{ij}(T, w, y, z)=\begin{cases} yz^{w(i,j)}x_{ij} & \text{if } \{i,j\}\in T,\ i<j,\\ -yz^{w(i,j)}x_{ji} & \text{if } \{i,j\}\in T,\ i>j,\\ z^{w(i,j)}x_{ij} & \text{if } \{i,j\}\in E-T,\ i<j,\\ -z^{w(i,j)}x_{ji} & \text{if } \{i,j\}\in E-T,\ i>j,\\ 0 & \text{if } \{i,j\}\notin E. \end{cases} \tag{3.1}$$

Here y, z, x_{ij} are indeterminates, $w(i,j)=w(\{i,j\})$ is the weight of the edge $\{i,j\}$. (We may add the same constant to all $w(i,j)$, so we assume that $\min_{i,j} w_{ij}=0$.)

Theorem 3.1, [7]. *Let*

$$T\subseteq E, \det B(T, w, y, z)=\sum_{h=0}^{L} Q_h(y)z^h \quad \textit{and} \quad L=\max_h\{h\,|\,Q_h(y)\not\equiv 0\}.$$

Then 1) *the maximum weight of a perfect matching in* G *is* $L/2$; 2) $\mathrm{Rank}_w(T)=\deg_y(Q_L(y))$.

Remark 3.1. Part 1 of Theorem 3.1 does not depend on y; it remains true even if $y=1$, which turns $B(T, w, y, z)$ into the matrix $B(E, w, z)=B(T, w, 1, z)$, whose entries do not depend on T,

$$b_{ij}(E, w, z)=b_{ij}(T, w, 1, z)=\begin{cases} z^{w(i,j)}x_{ij} & \text{if } \{i,j\}\in E,\ i<j,\\ -z^{w(i,j)}x_{ji} & \text{if } \{i,j\}\in E,\ i>j,\\ 0 & \text{if } \{i,j\}\notin E. \end{cases}$$

To improve the processor bounds, we replace the matrix $B(T, w, y, z)$ by the matrix $B(p, T, w)=(b_{ij}(p, T, w))=p^{1+2nw^*}B(T, w, 1/p, 1/p^{2n})$, such that

$$b_{ij}(p, T, w)=\begin{cases} p^{2n(w^*-w(i,j))}x_{ij} & \text{if } \{i,j\}\in T,\ i<j,\\ -p^{2n(w^*-w(i,j))}x_{ji} & \text{if } \{i,j\}\in T,\ i>j,\\ p^{1+2n(w^*-w(i,j))}x_{ij} & \text{if } \{i,j\}\in E-T,\ i<j,\\ -p^{1+2n(w^*-w(i,j))}x_{ji} & \text{if } \{i,j\}\in E-T,\ i>j,\\ 0 & \text{if } \{i,j\}\notin E, \end{cases} \tag{3.2}$$

$$w^*=\max_{i,j} w(i,j).$$

Remark 3.2. To obtain (3.2) from (3.1), substitute $y=1/p$, $z=1/p^n$ in (3.1) and multiply the resulting matrix by p^{1+2nw^*}. In the case where $w(i,j)=1$ for all i, j, (3.2) turns into (2.8).

Let us verify that

$$\text{Rank}_w(T) = \deg_y Q_L(y) = 2n - |\det B(p, T, w)|_p \,(\text{mod}\,(2n)), \tag{3.3}$$

compare (2.9), Lemma 2.2 and part 2 of Theorem 3.1. Indeed,

$$|\det B(p, T, w)|_p = \left|\det\left(p^{1+2nw^*} B(T, w, 1/p, 1/p^{2n})\right)\right|_p =$$

$$= |p^{2n(1+2nw^*)} \det B(T, w, 1/p, 1/p^{2n})|_p =$$

$$= |p^{2n(1+2nw^*)} \sum_{h=0}^{L} Q_h(1/p)/p^{2nh}|_p = |p^{2n(1+2nw^*-L)} Q_L(1/p)|_p.$$

The latter equation follows since $\deg_y Q_h(y) \leqq 2n$, so $Q_h(1/p)=k_h/p^{2n}$ for some integers k_h for all h. This immediately implies (3.3). ∎

It remains to compute the p-adic ranks of $\det B(p, T, w)$ for $T=S$ and for $T=S\cup e$ for all $e\in E-S$. We apply our algorithm for Problem 3 for general graphs, given at the end of the previous section, with the following changes:

— Choose r such that

$$(2n)^{r-5} > w^*. \tag{3.4}$$

— Replace the matrices $B^*(p, S)$ by $B(p, S, w)$, $B^+(p, S, e)$ by $B^+(p, S, w, e)$, and $B^-(p, S, e)$ by $B^-(p, S, w, e)$.

— Replace rank (S) by $\text{rank}_w(S)$ and rank $(S\cup e)$ by $\text{rank}_w(S\cup e)$.

Here the auxiliary matrices $B^+(p, S, w, e)$ and $B^-(p, S, w, e)$ have the same entries as $B(p, S, w)$, except that the entries corresponding to the fixed edge $e=\{v_i, v_j\}$, $i<j$, are defined as follows (compare (2.10)),

$$\begin{aligned} &b^+_{ij}(p, S, w, e) = p^{2n(w^*-w(i,j))}x_{ij},\ b^+_{ji}(p, S, w, e) = -p^{1+2n(w^*-w(i,j))}x_{ij},\\ &b^-_{ij}(p, S, w, e) = p^{1+2n(w^*-w(i,j))}x_{ij},\ b^-_{ji}(p, S, w, e) = -p^{2n(w^*-w(i,j))}x_{ij}. \end{aligned} \tag{3.5}$$

Lemmas 2.3 and 2.5 are easily extended to this case. (Note that we change our choice of r while extending Lemma 2.3 and we use (3.3) and (3.5), instead of Lemma 2.2 and (2.9), while extending Lemma 2.5.) The resulting processor bound in the Boolean circuit model is $O_B(w^*n^2M(n))$, which is the desired bound for Problem i) in case of general graphs; this is the nw^* times increase of our processor bound for computing a perfect matching in a graph, see Section 2 (such an increase occurs because the entries of the matrices $B(p, S, w)$, $B^+(p, S, w^*, e)$, $B^-(p, S, w, e)$ can be $(cnw^* \log n)$—bit integers for a positive constant c, that is, generally they are nw^* times "longer" than the entries of the matrices $B(p,S)$, $B^+(p, S, e)$, $B^-(p, S, e)$).

Next we will refine the latter processor bound to $O_B(w^*nM(n))$ in case of bipartite graphs. We will use the following simple extension of the first statement of Theorem 3.1.

Lemma 3.1. *The maximum weight of a perfect matching in G equals $W=nw^*-(|\det B(p, E, w)|_p)/2$.*

Proof. Recall Remark 3.1, substitute $z=1/p$ into the matrix $B(E, w, z)=B(T, w, 1, z)$, multiply the resulting matrix by p^{w^*} and arrive at the matrix $B(p, E, w)=$ $=p^{w^*}B(T, w, 1, 1/p)$, whose entries do not depend on T. Similarly to the proof of Lemma 2.2, we now verify that $L+|\det B(p, E, w)|_p=2nw^*$ where $2L$ equals the maximum weight of a matching in G (see Theorem 3.1). ∎

Remark 3.3. It follows that the maximum weight of a (perfect) matching in G can be computed by a probabilistic algorithm in expected time $O(\log^2(w^*n))$ using $O_B(w^*n\,M(n))$ processors. Furthermore the same estimates follow for the problem of computing (perfect) matching of maximum weight in G if such a matching is unique. Indeed, we may apply Lemma 3.1 replacing E by $E-e$ for all $e\in E$. It remains to test for each $e\in E$ if $|\det B(p, E, w)|_p=|\det B(p, E-e, w)|_p$ in order to identify the edges of the unique perfect matching of maximum weight in G. Extending Lemma 2.5 we reduce computing the p-adic ranks of $\det B(p, E, w)$ and $\det B(p, E-e, w)$ for all $e\in E$ to computing $\det B(p, E, w)$ and $\operatorname{adj} B(p, E, w)$.

This approach can be extended to the following interesting result.

Lemma 3.2. (A. Broder [4], and H. Gabow [6]). *Let $G=(V, E, w)$ be a bipartite graph; let an edge of G be called good if and only if it belongs to a perfect matching of maximum weight in G. Then every perfect matching in G consisting of only good edges has maximum weight.*

Remark 3.4. Lemma 3.2 does not hold for general graphs G.

Lemmas 3.1 and 3.2 immediately imply the desired processor bound $O_B(nw^*M(n))$ for problem i) for bipartite graphs. Indeed we identify the set S of all the good edges $e\in E$ as the edges such that $\operatorname{Rank}_w(\{e\})\neq\operatorname{Rank}_w(\emptyset)$ where $\emptyset$ denotes the empty set and $\{e\}$ denotes the singleton consisting of the edge e. (3.3) implies that the latter inequality can be equivalently rewritten as follows, $|\det B(p, \{e\}, w)|_p\neq|\det B(p, \emptyset, w)|_p \bmod (2n)$. Testing the latter inequality for all $e\in E$ can be reduced to the evaluation of $\det B(p, \emptyset, w)$ and $\operatorname{adj} B(p, \emptyset, w)$, compare (2.4). Furthermore we may replace n by 1 in the exponents of (3.2) and (3.5) because we only seek the ranks of singletons, so the processor bound decreases by a factor of n, as we desired. Then it remains to find a perfect matching in the unweighted subgraph $G^*=(V, S)$ of G. ∎

The arguments of [7] can be used to deduce the required estimates for Problems ii)—iv) from our estimates for Problems 1 and i). (Note that Problem iv) is reduced to solving Problem i) for a *bipartite* graph with $|E|<|V|^2$ vertices.)

Remark 3.5. Let us show that Problem i) can be solved in expected parallel time $O(\log^3 n)$ using $(W+n(1+\log\tilde{w}))nM(n))$ processors where $W\leqq nw^*$ is the maximum weight of a perfect matching in G, $\tilde{w}\leqq w^*$ is the maximum weight of an edge in perfect matchings in G. We may assume that W is known (see Remark 3.3). Next we recall that our definition of the p-adic rank of an integer q can be extended to the rationals of the form q/p^h. Then (3.3) immediately reduces the evaluation of $\operatorname{Rank}_w(T)$ to computing $|\det \tilde{B}_T|_p$ where $\tilde{B}_T=\tilde{B}(T, w, 1/p, 1/p^{2n})$, compare (3.1), (3.2). The expansion of $\det B_T$ as a sum of monomials consists of at most $(2n)!$ monomials, each associated with a perfect matching in G (other monomials vanish, see (3.2)). Every nonzero monomial has its absolute value lying between p^{-W} and

$(x^*)^n$ where $x^*=\max_{i,j} x_{ij}\leqq(2n)^r$. We choose r such that $x^*\leqq(2n)^r<\tilde{w}n^6$ (note that we may replace w^* by $\tilde{w}$ in (3.4) if we have already identified the set of good edges) and assume that $\det \tilde{B}_T\neq 0$, compare [14]. Then $p^{-W}\leqq|\det \tilde{B}_T|\leqq(2n)!(\tilde{w}n^6)^n$. Next we choose q such that

$$q > 2p^W(2n)!(\tilde{w}n^6)^n \geqq 2p^W|\det \tilde{B}_T|, \tag{3.6}$$

compute $\det \tilde{B}_T \bmod q$, then $p^W \det \tilde{B}_T \bmod q$, then recover $p^W \det \tilde{B}_T$, see (3.6), and finally compute $|p^W \det \tilde{B}_T|_p$ and $|\det \tilde{B}_T|_p=|p^W \det \tilde{B}_T|_p-W$. We apply the algorithm of [10] with appropriate modifications in order to compute $\det \tilde{B}_T \bmod q$ (as well as $\operatorname{adj} \tilde{B}_T \bmod q$) in $O(\log^2(nq))$ steps using $O_B(\log qM(n))$ processors. This implies the desired bounds on the complexity of solving problem i). (At first the algorithm of [10] is applied to computing $\det \tilde{B}_T \bmod g$, $g\leqq n^{O(1)}$, and then that value is lifted to computing $\det \tilde{B}_T \bmod q$, q being a power of g.) Similar saving can be achieved in bipartite case.

4. Open problems

1. Find an algorithm to invert a general matrix $A=(a_{ij})$ and to compute its determinant in time $O(\log^2 n)$ with $O_A(M(n))$ processors, see also Remark 1.1. (In [10] the latter bounds were proven assuming that all the entries a_{ij} are integers and that $\log \max |a_{ij}|=n^{O(1)}$.)
2. Improve the time complexity of Problem 1 to $O(\log^2 n)$ preserving our processor bounds.
3. Improve the processor complexity of Problems (i)—(iv). Note that for Problem (iv) we achieved the largest improvement ($O(n^2M^2(n))$ versus $O(n^{12}M^2(n))$), but a different reduction might yield an even better processor bound.
4. Find efficient parallel algorithms for Problems 1 and/or (i)—(iv) that do not reduce them to Problem 2.
5. Is the problem of finding a maximum weighted matching in general (or even bipartite) graphs in RNC (in the arithmetic model or in the Boolean model) when weights are given in binary (or even as infinite precision real numbers)?
6. Find parallel *deterministic* algorithms for Problems 1, 2, i)—iv) leading to the same (or even to slightly larger) complexity bounds as our algorithms.

References

[1] S. Berkowitz, On Computing the Determinant in Small Parallel Time Using a Small Number of Processors, *Information Processing Letters*, **18** (1984), 147—150.
[2] A. Borodin, S. Cook and N. Pippenger, Parallel Computation for Well-Endowed Rings and and Space-Bounded Probabilistic Machines, *Inform. and Control*, **58** (1983), 113—136.
[3] A. Borodin, J. von zur Gathen and J. Hopcroft, Fast Parallel Matrix and GCD Computations, *Inform. and Control*, **53** (1982), 241—256.
[4] A. Broder, *private communication*, 1985.
[5] D. Coppersmith and S. Winograd, Matrix Multiplication via Arithmetic Progressions, *Proc. 19th Ann. ACM Symp. on Theory of Computing*, 1987, 1—6.
[6] H. Gabow, *private communication*, 1985.

[7] R. M. KARP, E. UPFAL and A. WIGDERSON, Constructing a Perfect Matching Is in Random NC, *Proc. 17-th Ann. ACM Symp. on Theory of Computing*, (1985), 22—32.
[8] M. MARCUS and H. MINC, *A Survey of Matrix Theory and Matrix Inequalities*, Allyn and Bacon, Boston, 1964.
[9] K. MULMULEY, U. VAZIRANI and V. VAZIRANI, Matching Is As Easy As Matrix Inversion, *Combinatorica*, **7** (1987), 105—114.
[10] V. PAN, Fast and Efficient Parallel Algorithms for Exact Inversion of Integer Matrices, *Proc. Fifth Conf. on Software Technology and Theoretical Computer Science, Computer Science*, **206** (1985), 504—521; Complexity of Parallel Matrix Computations, *Theoretical Computer Science*, **54** (1987) (*to appear*).
[11] F. P. PREPARATA and D. V. SARWATE, An Improved Parallel Processor Bound in Fast Matrix Inversion, *Inform. Proc. Letters*, **7** (1978), 148—149.
[12] M. O. RABIN and V. VAZIRANI, Maximum Matchings in Graphs through Randomization *Tech. Report TR-15-84, Center for Research in Computer Technology, Aiken Computation Laboratory*, Harvard University, 1984.
[13] J. E. SAVAGE, *The Complexity of Computing*, John Wiley and Sons, N.Y., 1976.
[14] J. T. SCHWARTZ, Fast Probabilistic Algorithms for Verification of Polynomial Identities, *J. of ACM*, **27** (1980), 701—717.
[15] D. Y. Y. YUN, Algebraic Algorithms Using *p*-adic Constructions, *Proc. 1976 ACM Symp. Symbolic and Algebraic Computation*, (1976), 248—259.

Zvi Galil

Computer Science Department
Columbia University and
Tel-Aviv University

Victor Pan

Computer Science Department
State University of New York
at Albany