

## CONSTRUCTING A PERFECT MATCHING IS IN RANDOM NC

R. M. KARP<sup>†</sup>, E. UPFAL<sup>‡</sup> and A. WIGDERSON<sup>††</sup>

*Received 15 April 1985*

We show that the problem of constructing a perfect matching in a graph is in the complexity class Random NC; i.e., the problem is solvable in polylog time by a randomized parallel algorithm using a polynomial-bounded number of processors. We also show that several related problems lie in Random NC. These include:

- (i) Constructing a perfect matching of maximum weight in a graph whose edge weights are given in unary notation;
- (ii) Constructing a maximum-cardinality matching;
- (iii) Constructing a matching covering a set of vertices of maximum weight in a graph whose vertex weights are given in binary;
- (iv) Constructing a maximum  $s-t$  flow in a directed graph whose edge weights are given in unary.

### 1. Introduction

In this paper we show that the problem of constructing a perfect matching in a graph is in the complexity class Random NC; i.e., the problem is solvable in polylog time by a randomized parallel algorithm using a polynomial-bounded number of processors. We also show that several related problems lie in Random NC. These include:

- (i) Constructing a perfect matching of maximum weight in a graph whose edge weights are given in unary notation;
- (ii) Constructing a maximum-cardinality matching;
- (iii) Constructing a matching covering a set of vertices of maximum weight in a graph whose vertex weights are given in binary;
- (iv) Constructing a maximum  $s-t$  flow in a directed graph whose edge weights are given in unary.

Our results are based on a theorem of Tutte [15] showing that a graph has a perfect matching if and only if the determinant of a certain skew-symmetric matrix with indeterminates as elements is not identically zero. Around 1979 Lovász [11]

<sup>†</sup> Research supported by NSF Grant # DCR-8411954.

<sup>‡</sup> Research supported by a Weizmann Post-Doctoral fellowship, and by DARPA Grant N00039-83-C-1036.

<sup>††</sup> Research supported in part by DARPA Grant N00039-82-C-0235.

AMS subject classification (1980): 68 E 10.

suggested that Tutte's Theorem, combined with a fundamental randomized technique for testing whether a matrix with polynomial entries has a nonzero determinant [13], provides a simple polynomial-time randomized algorithm for testing whether a graph has a perfect matching. In 1982 Borodin, von zur Gathen and Hopcroft [3] observed that, since the problem of computing the determinant of a numerical matrix lies in NC, a parallel algorithm can be constructed based on Lovász's approach. This algorithm establishes that the problem of *deciding* whether a graph contains a perfect matching lies in the complexity class Random NC, but leaves open the question of whether a parallel algorithm of comparable efficiency exists for *constructing* a perfect matching in a graph that is known to have such a matching. In 1984 Rabin and Vazirani [12], using clever algebraic techniques related to Tutte's Theorem, gave an attractive randomized polynomial-time *sequential* algorithm for constructing a maximum matching. They also observed that if a graph has a unique perfect matching then the problem of finding it lies in NC. The special case of finding a perfect matching in an interval graph was shown to be in Random-NC by Kozen, Vazirani and Vazirani [10].

The additional ingredient that allowed us to obtain a Random-NC *parallel* algorithm for constructing a perfect matching in an arbitrary graph was the introduction of a useful set function called *Rank*. Our result is based on a Random-NC reduction of the matching problem to the problem of computing the *Rank* function, and on a Random-NC algorithm for computing the *Rank* of sets.

## 2. The perfect matching algorithm

We present the perfect matching algorithm in three stages. First we reduce the problem of finding a perfect matching to the problem of identifying a large set of redundant edges in a graph that has a perfect matching and is not very sparse. We say that a set of edges in a graph that has a perfect matching is redundant if the removal of these edges results in a graph that still has a perfect matching. We then show that a random-NC procedure can construct a large set of redundant edges provided that a certain integer-valued function, called the rank, which is defined over all subsets of edges in the graph, is computable in random-NC. Finally we give a random-NC algorithm for computing the rank function.

### 2.1. A high level description of the algorithm

Let  $IG=(IV, IE)$  denote the input graph. The procedure **Find-Perfect-Matching** uses three main variables  $M$ ,  $V$  and  $E$ .

The set variable  $M$  collects the edges that the procedure designates for the output matching. In contrast with the classical matching algorithms, our procedure never retreats from partial solutions. Once an edge is added to the set  $M$  it remains there until the procedure terminates, and it appears in the final output.

Throughout the execution of the procedure the variable  $V$  stores the set of vertices that are not yet covered by the matching  $M$ . The set  $E$  stores a subset of the set of edges connecting vertices in  $V$ ; these are the edges that are still candidates for inclusion in the matching.

We start the procedure with  $M$ . If  $IG$  has a perfect matching then with probability 1 the condition holds throughout the execution of the procedure.

The graph  $G=(V, E)$

If this condition is ever violated the procedure terminates in  $IG$ . On the other hand, if the condition holds throughout the execution of the procedure then the algorithm to consider edges in the set  $E$ , since  $E$  is a perfect matching in  $G$  is clearly a perfect matching in  $IG$ .

```

Procedure Find-Perfect-Matching ( $V, E$ )
  while  $|E| \geq 0$  do
    if  $|E| < (3/4)|V|$  then [ $G$  is sparse]
      find a set  $NM$ ,  $|NM| \geq (1/4)|V|$ 
       $M \leftarrow M \cup NM$ ;
       $V \leftarrow V - \{\text{vertices covered by } M\}$ 
       $E \leftarrow \{(v, u) \mid (v, u) \in E, v, u \in V\}$ 
    else [ $G$  is not sparse]
      Find-Redundant-Set( $RE$ );
      [with probability  $\beta > 0$ ,  $|RE| \geq (1/4)|E|$ ]
       $E \leftarrow E - RE$ ;

```

**end.**

The main **while** loop of the procedure terminates when  $V$  is empty. Assuming that the graph  $(V, E)$  always has a perfect matching, the set  $V$  is also empty, thus, that the variable  $E$  is also empty.

Each iteration of the **while** loop takes a constant amount of time, thus, the execution time is bounded by a constant factor. The execution distinguishes between two cases:

*Case a.*  $|E| < (3/4)|V|$ , (the graph is sparse). In this case, since  $G$  has a perfect matching, the degrees of all its vertices are at least  $(1/2)|V|$ . Thus, at least  $(1/2)|V|$  vertices have degree 1. This edge is included in the matching. We can identify at least  $(1/4)|V|$  edges that are redundant. The graph  $(V', E')$ , defined by the vertices  $V'$  and edges  $E'$ , has a perfect matching. Further, the computation can be done in  $O(1)$  time.

*Case b.*  $|E| \geq (3/4)|V|$ . This case is solved by the probabilistic procedure. With probability  $1 - o(1/|E|^2)$  the procedure produces a set  $RE$  such that  $|RE| \geq (1/4)|E|$ . Thus, with probability  $\beta > 0$  the procedure removes a constant fraction of the edges.

The analysis of the number of iterations required to find a perfect matching requires a fact from probability theory.

**Fact 1.** [1] Let  $X$  be a random variable with independent Bernoulli trials with probability  $a$ . Then  $\Pr[X \leq (1-a)np] < \exp(-1/2 a^2 np)$ . ■

with a fundamental randomized technique. The polynomial-time randomized algorithm for testing in 1982 Borodin, von zur Gathen and others. The problem of computing the determinant of a matrix can be constructed based on Lovász's problem of deciding whether a graph is in the complexity class Random NC, but leaves open the question of whether a polynomial-time algorithm of comparable efficiency exists that is known to have such a matching. Algebraic techniques related to Tutte's polynomial-time sequential algorithm for finding a perfect matching in a graph were observed that if a graph has a unique perfect matching it lies in NC. The special case of finding a perfect matching was shown to be in Random-NC by

and us to obtain a Random-NC parallel algorithm for finding a perfect matching in an arbitrary graph was the introduction of the rank function. Our result is based on a Random-NC algorithm for computing the Rank function, which is the Rank of sets.

### Ranking algorithm

The algorithm is in three stages. First we reduce the problem of identifying a large set of edges that has a perfect matching and is not very sparse. Then we identify a large set of edges that still has a perfect matching. We then construct a large set of redundant edges, called the rank, which is computable in random-NC. Finally we identify the rank function.

### Outline of the algorithm

The procedure **Find-Perfect-Matching** is based on the procedure **Find-Perfect-Matching**. The procedure designates for the set of edges that are still candidates for a perfect matching. The set  $E$  stores a subset of the edges that are still candidates

We start the procedure with  $M = \emptyset$ ,  $V = IV$ , and  $E = IE$ . If the input graph  $IG$  has a perfect matching then with probability  $1 - o(1)$  the following property holds throughout the execution of the procedure:

*The graph  $G = (V, E)$  has a perfect matching.*

If this condition is ever violated the procedure will fail to produce a perfect matching in  $IG$ . On the other hand, if the condition remains true then it suffices for the algorithm to consider edges in the set  $E$ , since the set  $M$  together with a perfect matching in  $G$  is clearly a perfect matching in the input graph  $IG$ .

**Procedure Find-Perfect-Matching** ( $V, E, M$ );

**while**  $|E| \geq 0$  **do**

**if**  $|E| < (3/4)|V|$  **then** [ $G$  is sparse]

    find a set  $NM$ ,  $|NM| \cong (1/4)|V|$ , of edges that lie in every perfect matching;

$M \leftarrow M \cup NM$ ;

$V \leftarrow V - \{\text{vertices covered by } NM\}$ ;

$E \leftarrow \{(v, u) \mid (v, u) \in E, v, u \in V\}$ ;

**else** [ $G$  is not sparse]

**Find-Redundant-Set** ( $RE$ );

    [with probability  $\beta > 0$ ,  $|RE| \cong \alpha|E|$ ]

$E \leftarrow E - RE$ ;

**end.**

The main **while** loop of the procedure is executed until the set  $E$  is empty. Assuming that the graph  $(V, E)$  always has perfect matching, this implies that the set  $V$  is also empty, thus, that the variable  $M$  stores a perfect matching in  $IG$ .

Each iteration of the **while** loop tries to decrease the size of the set  $E$  by a constant factor. The execution distinguishes between two cases:

*Case a.*  $|E| < (3/4)|V|$ , (the graph  $G = (V, E)$  is sparse). Since  $G$  has a perfect matching, the degrees of all its vertices are at least 1. However,  $|E| < (3/4)|V|$ , thus at least  $(1/2)|V|$  vertices have degree exactly 1. If an edge is incident to a vertex with degree 1 this edge is included in any perfect matching in this graph. Thus we can identify at least  $(1/4)|V|$  edges that can be added to the set  $M$  and the remaining graph  $(V', E')$ , defined by the vertices in  $V$  that are not covered by the new matching edges, has a perfect matching. Furthermore,  $|E'| \leq |E| - (1/4)|V| \leq (2/3)|E|$ , and the computation can be done in  $O(1)$  steps using  $O(|E|)$  processors.

*Case b.*  $|E| \geq (3/4)|V|$ . This case, which is the novel part of the algorithm, is solved by the probabilistic procedure **Find-Redundant-Set**. With probability  $1 - o(1/|E|^2)$  the procedure produces a set  $RE$  such that the graph  $(V, E - RE)$  has a perfect matching. With probability  $\beta > 0$ ,  $|RE| \cong \alpha|E|$  for some  $\alpha > 0$ . Thus with probability  $\beta > 0$  the procedure reduces the size of our problem by a constant factor.

The analysis of the number of iterations of the **while** loop required to create a perfect matching requires a fact from probability theory.

**Fact 1.** [1] *Let  $X$  be a random variable distributed as the number of successes in  $n$  independent Bernoulli trials with probability of success  $p$ . If  $0 < a < 1$  then  $P[X \leq (1-a)np] < \exp(-1/2 a^2 np)$ . ■*

**Lemma 2.1.** *If there are constants  $\alpha, \beta > 0$  such that with probability at least  $\beta$  a call to the procedure **Find-Redundant-Set** identifies at least a fraction  $\alpha$  of the edges in  $E$  as redundant, then, uniformly for all problem instances, the expected number of iterations of the while loop within this procedure is  $O(\log E)$ , and, moreover, for some  $c > 0$  and  $d > 0$  the number of iterations of the while loop is bounded above by  $d \log E$  with probability  $1 - O(E^{-c})$ .*

**Proof.** We may assume without loss of generality that  $\alpha \leq 1/3$ . Call an iteration of the while loop a success if it disposes of at least  $\alpha|E|$  edges, either by determining that they lie in every perfect matching (Case a) or determining that they are redundant (Case b). Then every iteration in which Case a occurs is necessarily a success, and every iteration in which Case b occurs is a success with probability at least  $\beta$ . Let  $\eta = (1 - \alpha)^{-1}$ . The number of successes required to reduce the number of edges to 1, starting from an initial edge set  $E$ , is at most  $\log_{\eta} |E|$ , and hence the number of successes required to complete the entire computation is at most  $1 + \log_{\eta} |E|$ . Since each iteration has probability of success at least  $\beta$ , independently of the outcomes of all previous iterations, the expected number of iterations is at most  $\beta^{-1}(1 + \log_{\eta} |E|)$ . Moreover, the number of successes in the first  $2\beta^{-1} \log_{\eta} |E|$  iterations is a random variable that stochastically dominates the number of successes in  $2\beta^{-1} \log_{\eta} |E|$  independent Bernoulli trials with probability of success  $\beta$ . Applying Fact 1, we find that the probability of having  $\log_{\eta} |E|$  or fewer successes in the first  $2\beta^{-1} \log_{\eta} |E|$  iterations is at most  $\exp(-(1/8) \log_{\eta} |E|) = |E|^{-c}$ , where  $c = (8 \ln \eta)^{-1} > 0$ . Hence, with probability  $1 - |E|^{-c}$  procedure **Find-Perfect-Matching** terminates within  $2\beta^{-1} \log_{\eta} |E|$  iterations.

As we have mentioned before, the probability that a call to the procedure **Find-Perfect-Matching** fails to find a perfect matching is  $o(1)$ . Since this bound on the failure probability is independent of the input, and since we can check at the end of the procedure whether  $M$  is a perfect matching, we can simply repeat the procedure until a perfect matching is found. This leads to the following randomized algorithm, which finds a perfect matching in polylog expected time in any graph in which such a matching exists.

### Perfect Matching Algorithm

```

M ← ∅;
while |M| < |V|/2
  V ← IV;
  E ← IE;
  M ← ∅
  Find-Perfect-Matching (V, E, M);
end.

```

### 2.2. Identifying a large set of redundant edges

Given a graph  $G = (V, E)$  such that

1.  $G$  has a perfect matching,

and

2.  $E \geq 3/4|V|$ ,

we need to find a set  $RE \subseteq E$  such that

1.  $|RE| \geq \alpha|E|$  for some  $\alpha > 0$

and

2.  $(V, E - RE)$  has a perfect matching.

Our algorithm uses a powerful set of edges in  $G$ .

**Definition 2.1.** Let  $PM$  denote the set of perfect matchings. For a set  $S \subseteq E$  define

$$\text{Rank}(S) = \text{number of perfect matchings in } PM \text{ that contain } S.$$

In words  $\text{Rank}(S)$  is an integer representing the number of perfect matchings in  $PM$  that occur together in a perfect matching.

We use the Rank function in the following way.

**Lemma 2.2.** For a fixed  $S \subseteq E$ , let  $G = (V, E)$  has a perfect matching then

**Proof.** If  $\text{Rank}(S) = k$  then there is a perfect matching  $M$  that contains  $S$ . If  $\text{Rank}(S \cup \{e\}) = \text{Rank}(S)$  then  $(S, e) \in M$ . The argument holds simultaneously for all  $e \in E - S$ . Hence,  $RE$  leaves the graph with at least one perfect matching.

**Example.** Suppose the graph  $G$  is a cycle with 6 vertices. Then  $PM$  consists of the two perfect matchings  $\{1, 3, 5\}$  and  $\{2, 4, 6\}$ . Let  $S = \{1, 3, 5\}$ . Then  $\text{Rank}(S) = 1$  and  $RE = \{4, 6\}$ . Note that  $RE$  destroys the perfect matching  $\{1, 3, 5\}$ .

To guarantee a uniform lower bound on the probability of success, for  $n$  large enough, we use the following procedure.

**Procedure Find-Redundant-Set;**

**begin**

choose a random number  $i$  from  $\{1, \dots, 5/6|E|\}$ ;

choose a  $S \subseteq E$  from the uniform distribution of  $i$ -element subsets of  $E$ ;

$RE \leftarrow \{e \in E - S \mid \text{Rank}(S \cup \{e\}) = \text{Rank}(S) + 1\}$ ;

**end.**

**Lemma 2.3.** There are constants  $\alpha, \beta > 0$  such that

**Proof.** Let  $|V| = n$ ,  $|E| = m$ , and let  $p_i$  be the probability that a random  $i$ -element subset  $S$  of  $E$  has  $\text{Rank}(S \cup \{e\}) = \text{Rank}(S) + 1$ .

We can think of a random set  $S$  as a sequence of  $i$  independent random choices (without repetitions) from  $E$ .

h that with probability at least  $\beta$  a call  
es at least a fraction  $\alpha$  of the edges  
problem instances, the expected number  
edure is  $O(\log E)$ , and, moreover, for  
s of the while loop is bounded above by

erality that  $\alpha \leq 1/3$ . Call an iteration  
east  $\alpha|E|$  edges, either by determining  
a) or determining that they are redun-  
Case a occurs is necessarily a success,  
a success with probability at least  $\beta$ .  
quired to reduce the number of edges  
most  $\log_\eta |E|$ , and hence the number  
computation is at most  $1 + \log_\eta |E|$ .  
s at least  $\beta$ , independently of the out-  
ted number of iterations is at most  
f successes in the first  $2\beta^{-1} \log_\eta |E|$   
lly dominates the number of successes  
with probability of success  $\beta$ . Applying  
g  $\log_\eta |E|$  or fewer successes in the  
p  $(-(1/8) \log_\eta |E|) = |E|^{-c}$ , where  $c =$   
 $1 - |E|^{-c}$  procedure **Find-Perfect-**  
erations.

probability that a call to the procedure  
t matching is  $o(1)$ . Since this bound  
e input, and since we can check at the  
t matching, we can simply repeat the  
his leads to the following randomized  
polylog expected time in any graph

of redundant edges

we need to find a set  $RE \subseteq E$  such that

1.  $|RE| \geq \alpha|E|$  for some  $\alpha > 0$
- and
2.  $(V, E - RE)$  has a perfect matching.

Our algorithm uses a powerful integer-valued function defined over all subsets of edges in  $G$ .

**Definition 2.1.** Let  $PM$  denote the set of perfect matchings in the graph  $G$ . For any set  $S \subseteq E$  define

$$\text{Rank}(S) = \text{Max}_{A \in PM} |S \cap A|.$$

In words  $\text{Rank}(S)$  is an integer giving the maximum number of edges from  $S$  that occur together in a perfect matching of  $G$ .

We use the Rank function in the following way.

**Lemma 2.2.** For a fixed  $S \subseteq E$ , let  $RE = \{e \in E - S \mid \text{Rank}(S \cup \{e\}) = \text{Rank}(S)\}$ . If  $G = (V, E)$  has a perfect matching then so does  $G' = (V, E - RE)$ .

**Proof.** If  $\text{Rank}(S) = k$  then there is a perfect matching  $A$  in  $G$  such that  $|S \cap A| = k$ . If  $\text{Rank}(S \cup \{e\}) = \text{Rank}(S)$  then  $(S \cup \{e\}) \cap A = S \cap A$  and  $e$  is not in  $A$ . This argument holds simultaneously for all the edges in  $RE$ ; therefore removing the set  $RE$  leaves the graph with at least one perfect matching, the set  $A$ . ■

**Example.** Suppose the graph  $G$  is a cycle consisting of the successive edges 1, 2, 3, 4, 5, 6. Then  $PM$  consists of the two sets  $\{1, 3, 5\}$  and  $\{2, 4, 6\}$ . Hence, for any set  $S$ ,  $\text{Rank}(S) = \text{Max}(|S \cap \{1, 3, 5\}|, |S \cap \{2, 4, 6\}|)$ . Consider the set  $S = \{1, 2, 3\}$ . Then  $\text{Rank}(S) = 2$  and  $RE = \{4, 6\}$ . Note that the deletion of edges 4 and 6 does not destroy the perfect matching  $\{1, 3, 5\}$ .

To guarantee a uniform lower bound on the probability that the set  $RE$  is large enough, we use the following procedure:

**Procedure Find-Redundant-Set;**

**begin**

choose a random number  $i$  from the uniform distribution over the range  $\{1, \dots, 5/6|E|\}$ ;  
choose a  $S \subseteq E$  from the uniform distribution over the  $i$ -element subsets of  $E$ ;  
 $RE \leftarrow \{e \in E - S \mid \text{Rank}(S \cup \{e\}) = \text{Rank}(S)\}$

**end.**

**Lemma 2.3.** There are constants  $\alpha, \beta > 0$  such that  $\text{Prob}\{|RE| \geq \alpha|E|\} \geq \beta$ .

**Proof.** Let  $|V| = n$ ,  $|E| = m$ , and let  $p_i$  denote the probability that for a set  $S$  drawn at random from the  $i$ -element subsets of  $E$ , and for an edge drawn at random from  $E - S$ ,  $\text{Rank}(S \cup \{e\}) = \text{Rank}(S) + 1$ .

We can think of a random set  $S$  as being constructed by a sequence of  $i$  random choices (without repetitions) from the set  $E$ .  $\text{Rank}(S)$  is then equal to the num-

new random edge was added to the set

$$= \sum_{j=0}^{i-1} p_j.$$

graph has a perfect matching, so we can

$$k(E) = \frac{n}{2} \cong \frac{4}{6} m.$$

compute a lower bound for the average. The size  $i$  of the set  $S$  is chosen when a random set  $S$  of size  $i$  is chosen, is added to the set  $RE$  is  $1 - p_i$ . There fore

$$\frac{1}{5} \left( \frac{5}{6} m - \sum_{i=1}^{(5/6)m} p_i \right) \cong \frac{1}{5} \frac{1}{6} m.$$

$$\text{Prob}\{|RE| = j\} \cong$$

$$n(\text{Prob}\{|RE| \cong \mu m\}),$$

$$\{|RE| \cong \mu m\},$$

$$\cong \frac{1}{60} \blacksquare$$

$(|E|)$  steps using  $h_1(|E|)$  processors' g functions, then **Find-Redundant-Set**  $h_1(|E|)$  processors.

Rank( $S$ ) and of Rank( $S \cup \{e\}$ ) for all  $(|E|)$  processors. The determination been done requires  $O(1)$  steps using

**Rank function**

give a solution for the case where the the general case is handled.

graph with edge set  $E$  and  $n$  vertices ex set of one part and  $V = \{v_1, v_2, \dots$

$\dots, v_n\}$ , the vertex set of the other part. Associate with each edge  $\{u_i, v_j\}$  an indeterminate  $x_{ij}$ , and let  $B = (b_{ij})$  be an  $n \times n$  matrix of indeterminates defined by the following rule:

$$b_{ij} = \begin{cases} x_{ij} & \text{if } \{u_i, v_j\} \in E \\ 0 & \text{if } \{u_i, v_j\} \notin E. \end{cases}$$

Edmonds [5] has observed that  $G$  has a perfect matching if and only if  $\det(B) \neq 0$ . This is true because each of the  $n!$  terms in the sum

$$\sum_{\sigma \in S_n} \text{sign}(\sigma) \prod b_{i, \sigma(i)}$$

is a product of  $n$  entries in  $B$  that correspond to the edges of one of the  $n!$  perfect matchings in the complete bipartite graph. If all the edges of a perfect matching exist in  $G$  then the corresponding term in  $\det(B)$  is not identically zero; otherwise one of the entries is zero and  $\prod b_{i, \sigma(i)} = 0$ . It is easy to verify that monomials can not cancel each other in this summation, and therefore  $\det(B) \neq 0$  if and only if  $G$  has a perfect matching.

To compute the Rank of a given set  $S \in E$  we refine this argument. Define the matrix  $B[S] = (b_{ij}^S)$  as follows:

$$b_{ij}^S = \begin{cases} yx_{ij} & \text{if } \{u_i, v_j\} \in S \\ x_{ij} & \text{if } \{u_i, v_j\} \in E - S \\ 0 & \text{if } \{u_i, v_j\} \notin E. \end{cases}$$

i.e.  $B[S]$  is derived from the matrix  $B$  by tagging all entries that correspond to edges in  $S$  with a new variable  $y$ .

Each non-zero monomial in  $\det(B[S])$  corresponds to a perfect matching in  $G$ . The degree in  $y$  of a monomial is equal to the number of elements from  $S$  that participate in the corresponding perfect matching. Again monomials can not cancel each other, hence the degree in  $y$  of  $\det(B[S])$  is equal to that of the monomial with the maximum degree in  $y$ , which in turn is equal to the maximum number of elements from  $S$  that participate in a perfect matching in  $G$ , i.e., to Rank( $S$ ).

So in order to compute Rank( $S$ ) we have to compute the degree in  $y$  of the multivariate polynomial  $\det(B[S])$ . Rewriting  $\det(B[S])$  as a polynomial in  $y$ , we get

$$\det(B[S]) = \sum_{t=0}^n Q_t [x_{ij}] y^t,$$

and Rank( $S$ ) = Max $\{t | Q_t \neq 0\}$ . Unfortunately, we can not test directly whether  $Q_t$ , which is a polynomial in up to  $|E|$  indeterminates, is identically zero. Instead, we use a well-known probabilistic method.

**Theorem 2.1.** (Schwartz) [13] *Let  $\tilde{Q}_t$  denote the value of the polynomial  $Q_t$  when each indeterminate in  $Q_t$  is replaced by a random integer in the range  $1, \dots, J$ . If  $Q_t \neq 0$  then  $\text{Prob}\{\tilde{Q}_t = 0\} \leq |E|/J$ .  $\blacksquare$*

Let  $\tilde{B}[S]$  denote the matrix  $B[S]$  after all its indeterminates, except  $y$ , have been replaced by random integers. To compute the  $\tilde{Q}_t$ 's we have to compute  $\det(\tilde{B}[S])$  as a polynomial in one variable  $y$ .

**Theorem 2.2.** (Borodin, Cook, Pippenger) [2]. *The determinant of an  $n \times n$  matrix of polynomials with a constant number of variables and the degree of each matrix element bounded by  $n$  can be computed in  $O(\log^2 n)$  steps using  $O(n^{4.5})$  processors.*

We can now summarize the algorithm for computing the Rank function in the bipartite case.

**Procedure Rank( $S$ );**

**begin**

construct the matrix  $B[S] = [b_{ij}^S]$ ;  
 replace each indeterminate  $x_{ij}$  by a random integer in the range  $1, \dots, |E|^4$ ;  
 compute  $\det(\tilde{B}[S]) = \sum_i \tilde{Q}_i y^i$ ;

$\text{Rank}(S) = \text{Max} \{i | \tilde{Q}_i \neq 0\}$ ;

**end.**

In the case of a general graph our method of computing the rank function relies on the following theorem of Tutte.

**Theorem 2.3.** (Tutte) [15]. *Let  $G=(V, E)$  denote a general graph with vertex set  $V=\{1, 2, \dots, n\}$  and define the skew-symmetric matrix  $B=(b_{ij})$  as follows:*

$$b_{ij} = \begin{cases} x_{ij} & \text{if } \{i, j\} \in E \text{ and } i < j \\ -x_{ij} & \text{if } \{i, j\} \in E \text{ and } i > j \\ 0 & \text{if } \{i, j\} \notin E. \end{cases}$$

*Then  $G$  has a perfect if and only if  $\det(B) \neq 0$ .* ■

Tutte's theorem can be extended to yield an algorithm for computing the Rank function in general graphs.

**Theorem 2.4.** *Let  $G=(V, E), S \subseteq E$ , and define the matrix  $B[S] = [b_{ij}^S]$  as follows:*

$$b_{ij}^S = \begin{cases} yx_{ij} & \text{if } \{i, j\} \in S \text{ and } i < j \\ -yx_{ij} & \text{if } \{i, j\} \in S \text{ and } i > j \\ x_{ij} & \text{if } \{i, j\} \in E - S \text{ and } i < j \\ -x_{ij} & \text{if } \{i, j\} \in E - S \text{ and } i > j \\ 0 & \text{if } \{i, j\} \notin E. \end{cases}$$

*Then  $\text{Rank}(S)$  is equal to half of the degree in  $y$  of  $\det(B[S])$ .*

**Proof.** Let  $P$  be the set of all permutations  $\sigma$  of  $\{1, 2, \dots, n\}$  such that, for all  $i$ ,  $b_{i, \sigma(i)} \neq 0$ . We classify these permutations according to their cycle structures. Since all elements on the main diagonal are 0, no permutation in  $P$  contains a cycle of length 1. Let  $OP$  be the set of all permutations in  $P$  which contain at least one odd cycle and let  $EP$  be the set of all permutations in  $S$  in which all cycles are of even length. Then  $P$  is the disjoint union of  $OP$  and  $EP$ . Let  $M$  be the set of all permutations in  $P$  such that every cycle is of length 2. Then  $M \subseteq EP$ , and the permutations in  $M$  are in one-to-one correspondence with the perfect matchings in  $G$ .

We have

$$\det(B[S]) = \sum_{\sigma \in P} \text{sign}(\sigma) \prod_i b_{i, \sigma(i)}.$$

We shall show that the contribution of  $\det(B[S])$  is equal to zero; i.e., that

$$\sum_{\sigma \in OP} \text{sign}(\sigma)$$

We shall partition the permutations into two permutations in each pair make a net contribution. Let  $\sigma$  be a permutation containing at least one cycle of  $\{1, 2, \dots, n\}$  occurring in an odd cycle. Then  $\sigma$  is paired with a permutation  $\bar{\sigma}$  in  $EP$ .  $C$  is reversed. Thus  $\bar{\sigma}$  is defined as follows:  $\bar{\sigma}(j) = \sigma^{-1}(j)$ . It is easy to verify that each pair makes a net contribution of zero.

Call an ordered pair  $\langle i, j \rangle$  an  $S$ -pair if  $\{i, j\} \in S$ .  $S$ -pairs correspond to the entries in  $B[S]$ . Associated with each permutation  $\sigma$  in  $M$  is a nonzero monomial in the variables  $x_{ij}$  of degree  $|E|$ . Then the  $y$ -degree of  $\tau$  is the number of  $S$ -pairs  $\langle i, \sigma(i) \rangle$  in  $\sigma$ , i.e. the number of  $S$ -pairs  $\langle i, \sigma(i) \rangle$ .

We shall show that, for every permutation  $\tau \in M$  such that the  $y$ -degree of  $\tau$  is odd, there exists a permutation  $\sigma \in M$  such that the  $y$ -degree of  $\sigma$  is even and  $\langle \sigma(i), \sigma(\sigma(i)) \rangle$  is an even pair. In other words, in the traversal of any cycle of  $\sigma$ , odd pairs are possible because all the cycles in  $\sigma$  are of even length.

Each permutation in  $M$  corresponds to a perfect matching in  $G$ . The  $y$ -degree of this permutation is twice the rank of  $S$ . Hence, using the result proven in [2], the  $y$ -degree of any permutation is just twice the rank of  $S$ . Thus  $\det(B[S])$  is twice the rank of  $S$ , we need to show that the  $y$ -degree of  $\sigma$  is even. If the  $y$ -degree of  $\sigma$  is odd, then  $\sigma$  contains an odd number of  $S$ -pairs. Now define  $\tau$  by the following:  $\tau(i) = \sigma(i)$  and  $\tau(\sigma(i)) = i$ . Then  $\tau$  lies in  $M$  and has an even  $y$ -degree.

Each permutation in  $M$  corresponds to a perfect matching in  $G$ . The  $y$ -degree of this permutation is twice the rank of  $S$ . Hence, using the result proven in [2], the  $y$ -degree of any permutation is just twice the rank of  $S$ . Thus  $\det(B[S])$  is twice the rank of  $S$ , we need to show that the  $y$ -degree of  $\sigma$  is even. If the  $y$ -degree of  $\sigma$  is odd, then  $\sigma$  contains an odd number of  $S$ -pairs. Now define  $\tau$  by the following:  $\tau(i) = \sigma(i)$  and  $\tau(\sigma(i)) = i$ . Then  $\tau$  lies in  $M$  and has an even  $y$ -degree.

Thus, using the probabilistic method of Borodin, Cook and Pippenger, we have a Rank function for a set of vertices in a general graph.

**Lemma 2.5. 1.** *The procedure Rank is efficient and uses  $O(n^{4.5})$  processors.*

**2.** *The probability that the procedure Rank fails is bounded by  $O(|E|^{-3})$ , and this event is rare.*

The determinant of an  $n \times n$  matrix in variables and the degree of each matrix entry is  $O(\log^2 n)$  steps using  $O(n^{4.5})$  processors.

for computing the Rank function in

an integer in the range  $1, \dots, |E|^4$ ;

method of computing the rank function

denote a general graph with vertex set  $V$  and edge set  $E$ . Let  $B = (b_{ij})$  be the matrix as follows:

$b_{ij} = y$  if  $(i, j) \in E$  and  $i < j$

$b_{ij} = y^{-1}$  if  $(i, j) \in E$  and  $i > j$

$b_{ij} = 0$  if  $(i, j) \notin E$ .

0. ■

yield an algorithm for computing the rank function

the matrix  $B[S] = [b_{ij}^S]$  as follows:

$b_{ij}^S = y$  if  $(i, j) \in S$  and  $i < j$

$b_{ij}^S = y^{-1}$  if  $(i, j) \in S$  and  $i > j$

$b_{ij}^S = 0$  if  $(i, j) \in E - S$  and  $i < j$

$b_{ij}^S = 0$  if  $(i, j) \in E - S$  and  $i > j$

$b_{ij}^S = 0$  if  $(i, j) \notin E$ .

in  $y$  of  $\det(B[S])$ .

Let  $P$  be the set of all permutations  $\sigma$  of  $\{1, 2, \dots, n\}$  such that, for all  $i \in V$ , the cycle structure of  $\sigma$  is consistent with their cycle structures. Since  $P$  is a subset of all permutations, it contains a cycle of length  $n$  in  $P$  which contain at least one odd cycle. Let  $S$  be the set of all permutations in  $P$  in which all cycles are of even length. Let  $EP$  be the set of all permutations in  $P$ . Then  $M \subseteq EP$ , and the perfect matchings in  $G$ .

$$\sum_{\sigma \in EP} \text{sign}(\sigma) \prod_i b_{i, \sigma(i)}$$

We shall show that the contribution of the permutations containing odd cycles to  $\det(B[S])$  is equal to zero; i.e., that

$$\sum_{\sigma \in OP} \text{sign}(\sigma) \prod_i b_{i, \sigma(i)} = 0.$$

We shall partition the permutations containing odd cycles into pairs, such that the two permutations in each pair make a net contribution of zero to the determinant. Let  $\sigma$  be a permutation containing at least one odd cycle. Let  $i$  be the least element of  $\{1, 2, \dots, n\}$  occurring in an odd cycle of  $\sigma$ , and let  $C$  be the odd cycle containing  $i$ . Then  $\sigma$  is paired with a permutation  $\bar{\sigma}$  which is the same as  $\sigma$  except that the cycle  $C$  is reversed. Thus  $\bar{\sigma}$  is defined as follows: if  $j \notin C$  then  $\bar{\sigma}(j) = \sigma(j)$ ; if  $j \in C$  then  $\bar{\sigma}(j) = \sigma^{-1}(j)$ . It is easy to verify that this rule partitions  $OP$  into pairs, and that each pair makes a net contribution of zero to the determinant.

Call an ordered pair  $\langle i, j \rangle$  an  $S$ -pair if  $\{i, j\}$  is an edge in  $S$ . Then the  $S$ -pairs correspond to the entries in the matrix  $B[S]$  which involve variable  $y$ . Associated with each permutation  $\sigma$  in  $EP$  is the term  $\text{sign}(\sigma) \prod_i b_{i, \sigma(i)}$ . This term is a nonzero monomial in the variables  $\{x_{ij}\}$  and  $y$ . Define the  $y$ -degree of  $\sigma$  as the degree of  $y$  in this monomial. Then the  $y$ -degree of  $\sigma$  is just the number of  $S$ -pairs in  $\sigma$ , i.e. the number of  $S$ -pairs  $\langle i, \sigma(i) \rangle$ .

We shall show that, for every permutation  $\sigma \in EP$  there exists a permutation  $\tau \in M$  such that the  $y$ -degree of  $\tau$  is greater than or equal to the  $y$ -degree of  $\sigma$ . To construct  $\tau$ , partition the pairs  $\langle i, \sigma(i) \rangle$ ,  $i = 1, 2, \dots, n$  into two sets, called the odd pairs and the even pairs, in such a way that,  $\langle i, \sigma(i) \rangle$  is an odd pair if and only if  $\langle \sigma(i), \sigma(\sigma(i)) \rangle$  is an even pair. In other words, the partition is chosen so that, in the traversal of any cycle of  $\sigma$ , odd pairs and even pairs alternate. Such a partition is possible because all the cycles in  $\sigma$  are even. Assume without loss of generality that the set of odd pairs in  $\sigma$  contains at least as many  $S$ -pairs as does the set of even pairs in  $\sigma$ . If the  $y$ -degree of  $\sigma$  is  $d$ , then the set of odd pairs contains at least  $d/2$   $S$ -pairs. Now define  $\tau$  by the following rule: if  $\langle i, \sigma(i) \rangle$  is an odd pair then  $\tau(i) = \sigma(i)$  and  $\tau(\sigma(i)) = i$ . Then  $\tau$  lies in  $M$  and the  $y$ -degree of  $\tau$  is at least  $d$ .

Each permutation in  $M$  corresponds to a perfect matching in  $G$ , and the  $y$ -degree of this permutation is twice the number of edges from  $S$  in this perfect matching. Hence, using the result proven in the last paragraph, the maximum  $y$ -degree of any permutation is just twice the rank of  $S$ . To show that the degree of  $y$  in  $\det(B[S])$  is twice the rank of  $S$ , we need to show that the permutations of maximum  $y$ -degree make a nonzero net contribution to  $\det(B[S])$ . But this is clear, since there exists a permutation in  $M$  among those of maximum  $y$ -degree, and its monomial is not cancelled by the monomial of any other permutation. ■

Thus, using the probabilistic method of Schwartz and the algorithm of Borodin, Cook and Pippenger, we have a Random-NC algorithm for computing the rank of a set of vertices in a general graph.

**Lemma 2.5.** 1. The procedure Rank is executed in  $O(\log^2 |V|)$  steps using  $O(|V|^{4.5})$  processors.

2. The probability that the procedure Rank fails to compute the correct value of Rank( $S$ ) is bounded by  $O(|E|^{-3})$ , and this event does not depend on the input. ■



Combining now the results of Lemmas 2.1—2.5 we have

**Theorem 2.5.** For any input graph  $IG=(IV, IE)$ :

1. The procedure **Find-Perfect-Matching** uses  $O(|IV|^{6.5})$  processors and terminates within  $O(\log^3 |IE|)$  steps with probability  $1 - |IE|^{-c}$  for some  $c > 0$ .
2. The probability that the procedure fails to produce a perfect matching when applied to a graph that possesses one is bounded by  $|IE|^{-1}$ . ■

**3. Further results**

In this section we derive Random-NC algorithms for several further problems related to matching and network flows. We begin by giving such an algorithm for finding a perfect matching of maximum weight in an edge-weighted graph  $G=(IV, IE, \omega)$ , when the edge-weights  $\omega(\{i, j\})$  are given in unary.

**Definition 3.1.** Let  $MW$  denote the set of perfect matchings of maximum weight in the weighted graph  $G=(V, E, \omega)$ . For any set  $S \subseteq E$  define

$$\text{Rank}_w(S) = \text{Max}_{A \in MW} |S \cap A|.$$

In words  $\text{Rank}_w(S)$  is an integer giving the maximum number of edges from  $S$  that participate in a perfect matching of maximum weight in  $G$ .

It is easy to verify that running the procedure **Find-Perfect-Matching** with the new rank function computes the desired perfect matching in an expected number of iterations of order  $\log |E|$ . The only difficulty is to show that the new  $\text{Rank}_w$  function is computable in Random NC. The following theorem, in combination with Theorem 2.2, establishes this fact.

**Theorem 3.1.** Let  $G=(V, E, \omega)$ ,  $S \subseteq E$ , and define the matrix  $B[S]$  as follows:

$$b_{ij}^S = \begin{cases} yz^w x_{ij} & \text{if } \{i, j\} \in S, \quad i < j, \quad \text{and } \omega(\{i, j\}) = w \\ -yz^w x_{ij} & \text{if } \{i, j\} \in S, \quad i > j, \quad \text{and } \omega(\{i, j\}) = w \\ z^w x_{ij} & \text{if } \{i, j\} \in E - S, \quad i < j, \quad \text{and } \omega(\{i, j\}) = w \\ -z^w x_{ij} & \text{if } \{i, j\} \in E - S, \quad i > j, \quad \text{and } \omega(\{i, j\}) = w \\ 0 & \text{if } \{i, j\} \notin E. \end{cases}$$

Let  $\det(B[S]) = \sum_i Q_i z^i$  and let  $L = \text{Max}\{i | Q_i \neq 0\}$ .

1. The maximum weight of a perfect matching in  $G$  is  $L/2$ .
2.  $\text{Rank}_w(S)$  is equal to half the degree in  $y$  of  $Q_L$ , the coefficient of  $z^L$ . ■

The proof of Theorem 3.1 is quite similar to that of Theorem 2.4. First it is shown that the permutations containing odd cycles make a net contribution of zero to  $\det(B[S])$ . Then attention is restricted to  $EP$ , the set of permutations with all cycles even that make a nonzero contribution to  $\det(B[S])$ . Let  $L$  be the highest

degree to which  $z$  occurs in the monomial  $Q_L$ . Let  $d$  be the highest degree of  $y$  that occurs in  $Q_L$ . It is shown that, among the permutations of degree  $d$  in  $y$ , there is at least one permutation that corresponds to a perfect matching from  $S$ ; moreover, it is shown that this perfect matching has maximum weight. It is shown that the monomials of degree  $L$  in  $z$  in  $EP$  make a nonzero net contribution to  $\det(B[S])$  is of degree  $L$  in  $z$ , and the coefficient of  $z^L$  is  $Q_L$ .

In the following paragraphs we show how further matching and flow problems like these below can be performed in logspace.

**1. Maximum cardinality matching.** The problem of finding a maximum cardinality matching in a graph  $G$  with  $|V|$  vertices is reduced to the problem of constructing a perfect matching in a graph  $G'$  with vertex set  $V$  in which each edge  $\{i, j\}$  has weight  $w_{ij}$  if  $\{i, j\} \in E$  and 0 otherwise. In this reduction  $G'$  has  $|V|$  vertices and  $|E|$  edges. This reduction shows that maximum cardinality matching lies in Random-NC. (A different reduction is given in [16].)

**2. Vertex weighted matching.** The problem of finding a maximum weight matching in a graph  $G=(V, E, \omega)$  with  $|V|$  vertices and  $|E|$  edges and a positive weight  $w(v)$  for each vertex  $v$  is reduced to the problem of finding a maximum weight matching in a graph  $G'$  with  $|V|$  vertices and  $|E|$  edges and a positive weight  $w(v)$  for each vertex  $v$ . We seek a matching of maximum total weight.

This problem can be approached from matroid theory that we require [16]. Call a set of vertices  $S$  independent if  $|S| \leq L/2$ . Let  $I$  be the family of all independent sets. Then  $(I, \subseteq)$  is a matroid, the rank of a set of vertices from  $S$  that are covered by a matching of maximum weight is  $|S|$ . A matroid can be constructed by the following greedy algorithm: sort the edges in order of decreasing weight; then, for  $i=1, 2, \dots, |E|$ , add edge  $e_i$  to the matching if  $|S \cup \{e_i\}| \leq L/2$ . A matching is constructed by finding a perfect matching in this matroid by  $T$ , using the main algorithm of this paper.

Thus, a Random-NC algorithm for maximum weight matching is at hand provided we can give a Random-NC algorithm for finding a maximum weight matching in this matroid. But this is reduced to that of determining a maximum weight matching in a graph with vertex set  $V$  whose edges are of weight  $w_{ij}$  if  $\{i, j\} \in E$  and 0 otherwise. In other words, of generality that the given graph  $G=(V, E, \omega)$  is a subgraph of the complete graph  $K$  on vertex set  $V$ .

s 2.1—2.5 we have

$E$ ):  
uses  $O(|V|^{6.5})$  processors and terminates in time  $1 - |E|^{-c}$  for some  $c > 0$ .  
produce a perfect matching when applied to  $E^{-1}$ . ■

results

algorithms for several further problems. We begin by giving such an algorithm for finding a maximum weight perfect matching in an edge-weighted graph  $G = (V, E)$  are given in unary.

perfect matchings of maximum weight in  $G$ . Let  $S \subseteq E$  define

$$\max_{M \subseteq E} |S \cap M|.$$

Let  $L$  be the maximum number of edges from  $S$  in any maximum weight matching in  $G$ .

The procedure **Find-Perfect-Matching** finds a maximum weight perfect matching in an expected time  $O(L^2 |E|)$ . The only difficulty is to show that the new algorithm is in Random NC. The following theorem, in combinatorial form, defines the matrix  $B[S]$  as follows:

$$\begin{aligned} i < j, \text{ and } \omega(\{i, j\}) &= w \\ i > j, \text{ and } \omega(\{i, j\}) &= w \\ i < j, \text{ and } \omega(\{i, j\}) &= w \\ i > j, \text{ and } \omega(\{i, j\}) &= w \end{aligned}$$

$$Q_L \neq 0.$$

Rank of  $Q_L$  in  $G$  is  $L/2$ .

$Q_L$  is the coefficient of  $z^L$ . ■

similar to that of Theorem 2.4. First it is shown that cycles make a net contribution of zero to  $\det(B[S])$ . Let  $L$  be the highest

degree to which  $z$  occurs in the monomial associated with such a permutation, and let  $d$  be the highest degree of  $y$  that occurs in a monomial that is of degree  $L$  in  $z$ . It is shown that, among the permutations whose monomials are of degree  $L$  in  $z$  and of degree  $d$  in  $y$ , there is at least one whose cycles are all of length 2. Such a permutation is shown to correspond to a matching of weight  $L/2$  containing  $d/2$  edges from  $S$ ; moreover, it is shown that this matching is of maximum weight and, among matchings of maximum weight, has a maximum number of edges from  $S$ . Finally, it is shown that the monomials of degree  $L$  in  $z$  and  $d$  in  $y$  associated with permutations in  $EP$  make a nonzero net contribution to  $\det(B[S])$ , so that the polynomial  $\det(B[S])$  is of degree  $L$  in  $z$ , and the coefficient of  $z$  in this polynomial is of degree  $d$  in  $y$ .

In the following paragraphs we use the technique of reducibility to show that further matching and flow problems lie in Random NC. All the reductions mentioned below can be performed in logspace.

**1. Maximum cardinality matching.** The problem of constructing a maximum cardinality matching in a graph  $G$  with vertex set  $V$  and edge set  $E$  is easily reduced to the problem of constructing a perfect matching of maximum weight in a graph  $G'$  with vertex set  $V$  in which each edge has weight zero or one. We can assume that  $|V|$  is even. In this reduction  $G'$  is the complete graph on vertex set  $V$ ; edge  $\{i, j\}$  of the complete graph receives weight one if  $\{i, j\}$  lies in  $E$ , and weight zero otherwise. This reduction shows that the maximum cardinality matching problem lies in Random-NC. (A different reduction for this problem is given in [12]).

**2. Vertex weighted matching.** The vertex-weighted matching problem is in Random NC even when the weights of the vertices are given in binary notation. In this problem we are given a graph  $G$  with vertex set  $V$  in which each vertex  $v$  has a positive weight  $w(v)$ . We seek a matching that covers a set of vertices of maximum total weight.

This problem can be approached with the help of matroid theory. The results from matroid theory that we require can be found in the comprehensive reference [16]. Call a set of vertices  $S$  independent if there is a matching that covers all the vertices in  $S$ ; then our goal is to construct an independent set of maximum weight. Let  $I$  be the family of all independent sets. The structure  $(V, I)$  is a matroid. In this matroid, the rank of a set of vertices  $S$  is just the maximum number of vertices from  $S$  that are covered by a matching. A maximum-weight independent set  $T$  in a matroid can be constructed by the following rule: let the elements be  $v_1, v_2, \dots, v_{|V|}$  in order of decreasing weight; then, for  $i=1, 2, \dots, |V|$ ,  $v_i$  lies in  $T$  if and only if  $\text{Rank}(\{v_1, v_2, \dots, v_i\}) > \text{Rank}(\{v_1, v_2, \dots, v_{i-1}\})$ . Once  $T$  is known, the desired matching is constructed by finding a perfect matching in the subgraph of  $G$  induced by  $T$ , using the main algorithm of this paper.

Thus, a Random-NC algorithm for the vertex-weighted matching problem is at hand provided we can give a Random-NC algorithm for computing the rank of a set of vertices in this matroid. But the problem of computing  $\text{Rank}(S)$  is easily reduced to that of determining a maximum-weight perfect matching in a graph  $K$  with vertex set  $V$  whose edges are of weight 0, 1 and 2. We can assume without loss of generality that the given graph  $G = (V, E)$  has an even number of vertices. The graph  $K$  is the complete graph on vertex set  $V$ . If edge  $\{i, j\}$  does not lie in  $E$  then

$\{i, j\}$  is given weight 0; if  $\{i, j\}$  lies in  $E$ , then the weight of  $\{i, j\}$  is  $|S \cap \{i, j\}|$ . Clearly, the maximum weight of a perfect matching in  $K$  is the rank of  $S$ . Thus we have shown that the vertex-weighted matching problem lies in Random NC.

**3. Network flow.** First, consider the problem of constructing a maximum  $s-t$  flow in a directed graph in which each edge has capacity 1. There is a classical reduction of this problem to the problem of constructing a maximum matching in a bipartite graph, as follows. Let the flow network be  $G=(V, E)$ , with source  $s$  and sink  $t$ . We may assume that  $s$  has in-degree 0 and  $t$  has out-degree 0. The reduction constructs a bipartite graph  $H$  with bipartition  $(V_1, V_2)$ . Each part of the bipartition is a copy of the edge set of  $G$ . Thus,  $V_1 = \{(e, 1) | e \in E\}$  and  $V_2 = \{(e, 2) | e \in E\}$ . If the head of edge  $e$  is also the tail of edge  $f$  (i.e.,  $e = (i, j)$  and  $f = (j, k)$  for some  $i, j$  and  $k$ ) then  $H$  contains and edge between  $(e, 1)$  and  $(f, 2)$ . If an edge  $e$  in  $G$  is incident with neither  $s$  nor  $t$ , then  $H$  has an edge from  $(e, 1)$  to  $(e, 2)$ . Then a maximum matching in  $H$  yields a maximum flow in  $G$  according to the following rule: edge  $e$  carries a flow of 1 if and only if  $(e, 1)$  is matched with some vertex  $(f, 2)$ , where  $e \neq f$ , or  $(e, 2)$  is matched with some vertex  $(f, 1)$ , where  $e \neq f$ . The reduction just given extends easily to the case in which the flow network has edges with integer capacities, provided these capacities are given in unary. The idea is to replace each edge  $\{i, j\}$ , of capacity  $c$ , with  $c$  parallel edges from  $i$  to  $j$ , each of capacity 1. All capacities in the resulting network are 1, and thus the reduction to bipartite matching applies. Thus, we have shown that the following problem is in Random NC: construct a maximum  $s-t$  flow in a directed flow network whose edge capacities are given in unary.

We summarize the constructions and reductions given in this section by a theorem.

**Theorem 3.2.** *The following problems lie in Random-NC:*

- (i) *Constructing a perfect matching of maximum weight in a graph whose edge weights are given in unary;*
- (ii) *Constructing a maximum matching;*
- (iii) *Constructing a matching covering a set of vertices of maximum weight in a graph whose vertex weights are given in binary;*
- (iv) *Constructing a maximum  $s-t$  flow in a directed graph whose edge weights are given in unary.* ■

Our result about network flows stands in interesting contrast to the following result due to Goldschlager, Shaw and Staples [7]: the problem of constructing a maximum  $s-t$  flow in a directed flow network with edge capacities given in binary is complete in P with respect to logspace reductions. Since it is generally believed that such complete problems do not lie in Random NC, it appears that the parallel complexity of the max-flow problem depends critically on whether the capacities are given in unary or in binary. Nevertheless, the following result can be given:

**Theorem 3.3.** *There is a randomized parallel algorithm to construct a maximum  $s-t$  flow in a directed network whose edge weights are given in binary, such that the number of processors used is bounded by a polynomial in the number of vertices, and the time used is  $O((\log V)^k \log C)$ , where  $C$  is the largest capacity of any edge and  $k$  is a constant.* ■

This result is proved by combining the Edmonds—Karp scaling technique with the problem in Random NC, since  $\log C$  is the number of bits needed to express

#### 4. Dis

Each of the randomized algorithms of giving an erroneous result; for example, may fail to produce a perfect matching in its failure does not indicate with certainty the algorithm many times in parallel can be exponentially low level, but can never err given a Random NC algorithm for the solution of the matching problem. As Karloff pointed out, our algorithms to yield algorithms for the maximum matching problem which run in polynomial time; i.e., Las Vegas algorithm.

It remains an open question whether the problem of constructing a perfect matching is entirely in these problems. It would be very interesting to know whether a graph has a perfect matching is a problem of constructing a perfect matching.

Finally, the investigations reported in this study are reported in the companion paper.

#### Refer

- [1] D. ANGLUIN and L. G. VALIANT, Fast algorithms for finding maximum matchings. *J. of Comp. Syst. Sci.* **18** (1979), 135–147.
- [2] A. BORODIN, S. A. COOK and N. PIPPENGER, Space bounded probabilistic machines. *Inf. Proc. Letters* **13** (1982), 65–71.
- [3] A. BORODIN, J. VON ZUR GATHEN and J. HOPFINGER, On the complexity of matrix multiplication. *Proc. STOC* (1982), 65–71.
- [4] S. A. COOK, An overview of computation complexity. *SIAM Review* **23** (1981), 219–238.
- [5] J. EDMONDS, Systems of distinct representatives and linear inequalities. *J. of Standards*, **71A** (1967), 241–245.
- [6] J. EDMONDS and R. M. KARP, Theoretical complexity of bipartite matching and flow problems. *J. of ACM* **19** (1972), 248–256.
- [7] L. M. GOLDSCHLAGER, R. A. SHAW and J. STAPLES, The complexity of the maximum flow problem. *Theoretical Computer Science* **10** (1980), 1–10.
- [8] H. J. KARLOFF, A Las Vegas RNC algorithm for bipartite matching. *SIAM J. Comput.* **13** (1984), 1–10.
- [9] R. M. KARP, E. UPFAL and A. WIGDERSON, The complexity of bipartite matching: an equivalent? *STOC* 1985.
- [10] D. KOZEN, U. V. VAZIRANI and V. V. VAZIRANI, The complexity of bipartite matching in R-NC. *STOC* 1985.
- [11] L. LOVÁSZ, Determinants, matchings and matroids. *Combinatorial Theory, FCT'79*, (ed. L. Budach), 1–10.
- [12] M. O. RABIN and V. V. VAZIRANI, Maximum bipartite matching in R-NC. *Harvard University Center for Research in Computing Science*, TR-15-84.

the weight of  $\{i, j\}$  is  $|S \cap \{i, j\}|$ . Clearly,  $K$  is the rank of  $S$ . Thus we have the problem lies in Random NC.

problem of constructing a maximum  $s-t$  flow network with capacity 1. There is a classical reduction to constructing a maximum matching in a bipartite graph. Let  $G=(V, E)$ , with source  $s$  and sink  $t$  and  $t$  has out-degree 0. The reduction is to a bipartite graph  $(V_1, V_2)$ . Each part of the bipartition is  $V_1 = \{(e, 1) | e \in E\}$  and  $V_2 = \{(e, 2) | e \in E\}$ . If  $e = (i, j)$  and  $f = (j, k)$  for some  $i, j, k$ , then there is an edge from  $(e, 1)$  to  $(f, 2)$ . If an edge  $e$  in  $G$  is a directed edge from  $(e, 1)$  to  $(e, 2)$ . Then a maximum matching in  $G$  according to the following rule: each vertex  $(e, 1)$  is matched with some vertex  $(f, 2)$ , where  $e \neq f$ . The reduction is to a flow network whose edges have integer capacities in unary. The idea is to replace each edge from  $i$  to  $j$ , each of capacity 1. All edges from  $i$  to  $j$  are replaced by a single edge with capacity 1. Thus the reduction to bipartite matching problem is in Random NC: construct a bipartite graph whose edge capacities are given

reductions given in this section by a

Random-NC:

Maximum weight in a graph whose edge

vertices of maximum weight in a graph

directed graph whose edge weights are

An interesting contrast to the following results [7]: the problem of constructing a maximum flow network with edge capacities given in binary is in Random NC, it appears that the parallel complexity is critically on whether the capacities are given in binary, the following result can be given:

Algorithm to construct a maximum flow network whose edge weights are given in binary, such that the complexity is polynomial in the number of vertices, and the complexity is the largest capacity of any edge and

This result is proved by combining the methods of the present paper with the Edmonds—Karp scaling technique [6]. Of course the result does not place the problem in Random NC, since  $\log C$  is a linear, rather than polylogarithmic, function of the number of bits needed to express  $C$  in binary.

#### 4. Discussion

Each of the randomized algorithms given in this paper has a small probability of giving an erroneous result; for example, procedure **Find-Perfect-Matching** may fail to produce a perfect matching in a graph that possesses one, and therefore its failure does not indicate with certainty that no perfect matching exists. Running the algorithm many times in parallel can reduce the probability of error to an exponentially low level, but can never eradicate it entirely. Howard Karloff [8] has given a Random NC algorithm for the odd-set cover problem, which is the “dual” of the matching problem. As Karloff points out, this result can be combined with our algorithms to yield algorithms for the perfect matching problem and the maximum matching problem which run in polylog expected time and always give the correct result; i.e., Las Vegas algorithms rather than Monte Carlo algorithms.

It remains an open question whether randomization can be dispensed with entirely in these problems. It would be very nice to show that the problem of deciding whether a graph has a perfect matching lies in NC, and even nicer to show that the problem of constructing a perfect matching lies in NC.

Finally, the investigations reported here have led us into a broader study of the relation between decision problems and search problems; the results of that study are reported in the companion paper [9].

#### References

- [1] D. ANGLUIN and L. G. VALIANT, Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. of Comp. Syst. Sci.* **18** (1979), 155—193.
- [2] A. BORODIN, S. A. COOK and N. PIPPENGER, Parallel computation for well-endowed rings and space bounded probabilistic machines. *Information and Control* **58** (1983), 113—136.
- [3] A. BORODIN, J. VON ZUR GATHEN and J. HOPCROFT, Fast parallel matrix and GCD computations. *Proc. 23 STOC* (1982), 65—71.
- [4] S. A. COOK, An overview of computation complexity. *CACM* **26** (1983), 400—408.
- [5] J. EDMONDS, Systems of distinct representatives and linear algebra. *J. of Res. Nat. Bureau of Standards*, **71A** (1967), 241—245.
- [6] J. EDMONDS and R. M. KARP, Theoretical improvements in algorithmic efficiency for network flow problems. *J. of ACM* **19** (1972), 248—264.
- [7] L. M. GOLDSCHLAGER, R. A. SHAW and J. STAPLES, The maximum flow problem is logspace complete for P. *Theoretical Computer Science* **21** (1982), 105—111.
- [8] H. J. KARLOFF, A Las Vegas RNC algorithm for maximum matching *Combinatorica*, to appear.
- [9] R. M. KARP, E. UPFAL and A. WIGDERSON, Are search and decision problems computationally equivalent? *STOC* 1985.
- [10] D. KOZEN, U. V. VAZIRANI and V. V. VAZIRANI, The two-processors scheduling problem is in R-NC. *STOC* 1985.
- [11] L. LOVÁSZ, Determinants, matchings and random algorithms, in: *Fundamentals of Computation Theory, FCT'79*. (ed. L. Budach), Akademie-Verlag Berlin 1979. pp 565—574.
- [12] M. O. RABIN and V. V. VAZIRANI, Maximum matchings in general graphs through randomization, TR-15-84, *Harvard University Center for Research in Computing Technology*, 1984.

- [13] J. T. SCHWARTZ, Fast probabilistic algorithms for verification of polynomial identities. *J. of ACM*, 27 (1980), 701—717.
- [14] E. SHAMIR and E. UPFAL,  $N$ -processors graphs distributively achieve perfect matching in  $O(\log^2 N)$  beats. *Proceeding of the First ACM SIGACT-SIGMOD Symp. on Principles of Distributed Computing*. Ottawa, 1982, 238—241.
- [15] W. T. TUTTE, The factors of graphs. *Canad. J. Math.* 4 (1952), 314—328.
- [16] D. J. A. WELSH, *Matroid Theory*. Academic Press (1976).

Richard M. Karp

*Comp. Sci. Division  
University of California  
Berkeley, CA 94720, U.S.A.*

Avi Wigderson

*IBM San Jose Res. Lab.  
San Jose, CA 95193, U.S.A.*

Eli Upfal

*Comp. Sci. Department  
Stanford University  
Stanford, CA 94305, U.S.A.*

LEGAL COLORING

N.

Received

The following computational problem (presented at the Conference, 1981): Given a graph  $G=(V, E)$  and a vertex coloring  $f$ . Decide whether  $f$  is a proper vertex coloring. Linear comparisons.

Lower bounds on the complexity of this problem are known. It is shown how geometric parameters of  $G$  can be used to establish lower bounds on the complexity of this problem.

Existing methods for analyzing such problems are based on establishing lower bounds for a variety of computational problems.

### 1. Deciding the legality of a vertex coloring

Let  $G=(V, E)$  be a graph and  $f$  a vertex coloring of  $G$ . We are to decide whether  $f$  is a proper vertex coloring, namely, whether or not for every two adjacent vertices  $u, v \in V$ ,  $f(u) \neq f(v)$ . The elementary step is one linear comparison.

The *element uniqueness problem* with  $n$  elements [1] is the following: Given  $n$  reals  $f_1, \dots, f_n$ , are they all distinct. The elementary steps are linear comparisons. The legal coloring problem where the graph  $G$  is given in article [1] serves as the starting point for this paper. We consider the legal coloring problem. For the legal coloring problem into an equivalent geometric problem. Geometric objects involved can be used to decide the legal coloring problem. Letting  $V = V(G)$  and  $f = f(G)$  with  $G$  a subset of  $\mathbb{R}^n$  as follows:

$$S(G) = \{(x_1, \dots, x_n) \in \mathbb{R}^n \mid x_i \neq x_j \text{ for } (i, j) \in E(G)\}$$

The geometric equivalent of the legal coloring problem is  $\vec{f} = (f(a_1), \dots, f(a_n))$  belongs to  $S(G)$  or not.

AMS subject classification (1980): 68C01