Maximum Matchings via Gaussian Elimination

[Extended Abstract]

Marcin Mucha*
Institute of Informatics
Warsaw University
Warsaw, Poland
mucha@mimuw.edu.pl

Piotr Sankowski*
Institute of Informatics
Warsaw University
Warsaw, Poland
sank@mimuw.edu.pl

ABSTRACT

We present randomized algorithms for finding maximum matchings in general and bipartite graphs. Both algorithms have running time $O(n^\omega)$, where ω is the exponent of the best known matrix multiplication algorithm. Since $\omega < 2.38$, these algorithms break through the $O(n^{2.5})$ barrier for the matching problem. They both have a very simple implementation in time $O(n^3)$ and the only non-trivial element of the $O(n^\omega)$ bipartite matching algorithm is the fast matrix multiplication algorithm.

Our results resolve a long-standing open question of whether Lovász's randomized technique of testing graphs for perfect matching in time $O(n^\omega)$ can be extended to an algorithm that actually constructs a perfect matching.

1. INTRODUCTION

A *matching* in an undirected graph $G = (V, E)$ is a subset $M \subseteq E$, such that no two edges of M are incident. Let $n = |V|$, $m = |E|$. A *perfect matching* is a matching of cardinality $|V|/2$. The problems of finding a *Maximum Matching* (i.e. a matching of maximum size) and, as a special case, finding a *Perfect Matching* if one exists, are two of the most fundamental algorithmic graph problems.

Solving these problems in time polynomial in n remained an elusive goal for a long time until Edmonds [6] gave the first algorithm. Several other algorithms have been found since then, the fastest of them being the algorithm of Micali and Vazirani [12], Blum [2] and Gabow and Tarjan [7]. The first of these algorithms is in fact a modification of the Edmonds algorithm, the other two use different techniques, but all of them run in time $O(m\sqrt{n})$, which gives $O(n^{2.5})$ for dense graphs.

On the other hand Lovász [10] showed that it is possible to test whether given graph has a perfect matching in randomized time $O(n^\omega)$, where ω is the exponent of the best known matrix multiplication algorithm. Since $\omega < 2.38$, this algorithm is faster then any of the algorithms finding the perfect matching.

Much more information about a graph can be obtained in time $O(n^\omega)$ (see Cheriyan [4]). For example, it is possible to find its Gallai-Edmonds decomposition, canonical partition, identify the allowed edges, etc..

In this paper, we show that it is possible to actually construct a maximum matching in randomized time $O(n^\omega)$. Our algorithms have very simple $O(n^3)$ implementations. In case of bipartite graphs, the only non-trivial part of the algorithm is the fast matrix multiplication procedure of Coppersmith and Winograd [5]. In fact, we show that in this case perfect matchings can be found using the well-known LUP factorization algorithm of Hopcroft and Bunch [3] with minor changes. The algorithm for general graphs is more complicated, but matrix multiplication remains the most complex part.

REMARK 1.1. *In case $\omega = 2$ additional logarithmic factor appears, so in the remainder of this paper we assume for simplicity, that $\omega > 2$.*

Therefore, we not only resolve a long-standing open question of finding maximum matchings in time $O(n^{2.5-\varepsilon})$, but also provide a new and simple approach to the problem.

The rest of the paper is organized as follows. In the next section we recall some well known results useful in the matrix approach to maximum matchings. We also recall that the problem of finding a maximum matching can be reduced to the problem of finding a perfect matching. In Section 3 we introduce the idea of finding a perfect matching via Gaussian elimination. We next use this idea to develop an elementary $O(n^3)$ algorithm for finding perfect matchings in general graphs and an $O(n^\omega)$ algorithm for finding an inclusion-wise maximal allowed submatching of a given matching. In Section 4 we show that in case of bipartite graphs we can reduce the time complexity of the perfect matching algorithm to $O(n^\omega)$ by using the classic Hopcroft-Bunch LU factorization algorithm. We also achieve the same time complexity for general graphs by exploiting the structural properties of

*Research supported by KBN grant 4T11C04425

graphs having a perfect matching. Finally, we end in Section 5 with some concluding remarks and open problems.

2. PRELIMINARIES

2.1 Matchings, Adjacency Matrices and Their Inverses

Let $G = (V, E)$ be a graph and let $n = |V|$ and $V = \{v_1, \ldots, v_n\}$. A *skew symmetric adjacency matrix* of G is a $n \times n$ matrix $\tilde{A}(G)$ such that

$$\tilde{A}(G)_{i,j} = \begin{cases} x_{i,j} & \text{if } (v_i, v_j) \in E \text{ and } i < j \\ -x_{i,j} & \text{if } (v_i, v_j) \in E \text{ and } i > j \\ 0 & \text{otherwise} \end{cases},$$

where the $x_{i,j}$ are unique variables corresponding to edges of G.

Tutte [17] observed the following

THEOREM 2.1. *The symbolic determinant* $\det \tilde{A}(G)$ *is non-zero iff* G *has a perfect matching.*

Lovász[10] generalized this to

THEOREM 2.2. *The rank of the skew symmetric adjacency matrix* $\tilde{A}(G)$ *is equal to twice the size of maximum matching of* G.

Choose a number $R = n^{O(1)}$ (more on the choice of R later in this section) and substitute each variable in $\tilde{A}(G)$ with a random number taken from the set $\{1, \ldots, R\}$. Let us call the resulting matrix the *random adjacency matrix of* G and denote $A(G)$. Lovász showed that

THEOREM 2.3. *The rank of* $A(G)$ *is at most twice the size of maximum matching of* G. *The equality holds with probability at least* $1 - (n/R)$.

This gives a randomized algorithm for deciding whether a given graph G has a perfect matching: Compute the determinant of $A(G)$. With high probability, this determinant is non-zero iff G has a perfect matching. This algorithm can be implemented to run in time $O(n^\omega)$ using fast matrix multiplication (ω is the matrix multiplication exponent, currently $\omega < 2.38$, see Coppersmith and Winograd [5]).

The same applies to bipartite adjacency matrix of a bipartite graph. Let $G = (U, V, E)$ be a bipartite graph, where $|U| = |V| = n$, $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$. Let $\tilde{A}(G)_{i,j} = x_{i,j}$ if $(u_i, v_j) \in E$, otherwise $\tilde{A}(G)_{i,j} = 0$. The symbolic determinant of $\tilde{A}(G)$ is non-zero iff the graph has a perfect matching, and the rank of the matrix is equal to the size of the maximum matching. Moreover, if we substitute random numbers from $\{1, \ldots, R\}$ for the variables of $\tilde{A}(G)$, then the rank of the resulting matrix is at most equal to the size of the maximum matching and the equality holds with probability at least $1 - (n/R)$.

Let G be a graph having a perfect matching and let $A = A(G)$ be its random adjacency matrix. With high probability $\det A \neq 0$, and so A is invertible. Rabin and Vazirani [14] showed that

THEOREM 2.4. *With high probability,* $A^{-1}_{j,i} \neq 0$ *iff the graph* $G - \{v_i, v_j\}$ *has a perfect matching.*

In particular, if (v_i, v_j) is an edge in G, then with high probability $A^{-1}_{j,i} \neq 0$ iff (v_i, v_j) is *allowed*, i.e. it is contained in some perfect matching. The same is true for the bipartite adjacency matrix of a bipartite graph.

Both Theorem 2.3 and Theorem 2.4 follow from the following lemma due to Zippel [18] and Schwartz [16]

LEMMA 2.5. *If* $p(x_1, \ldots, x_m)$ *is a non-zero polynomial of degree* d *with coefficients in a field and* S *is a subset of the field, then the probability that* p *evaluates to* 0 *on a random element* $(s_1, s_2, \ldots, s_m) \in S^m$ *is at most* $d/|S|$.

For Theorem 2.3, it is enough to notice that the determinant of $\tilde{A}(G)$ is in fact a polynomial of degree n of the variables in $\tilde{A}(G)$. For Theorem 2.4, recall that $A^{-1}_{i,j} = \mathrm{adj}(A)_{i,j} / \det A$, where $\mathrm{adj}(A)_{i,j}$ — the so called adjoint of A — is the determinant of A with the j-th row and i-column removed, multiplied by $(-1)^{i+j}$.

It can be shown (see [14]), that these polynomials are non-zero over the finite field $\mathcal{Z}_p$, so both theorems work in this case as well. In fact, it follows from Lemma 2.5, that it is enough to take p polynomial in n. This is important because finite field aritmetic operations can then be performed in constant time (except for the division, but in all algorithms considered in this paper divisions are dominated by other operations).

Using Theorem 2.4, perfect matchings can be found in a straightforward manner: generete random adjacency matrix $A(G)$, compute $A^{-1}(G)$, find an allowed edge, remove this edge together with its endpoints from G, generate A for the new graph, etc.. The following theorem of Rabin and Vazirani shows that randomization is needed only in the first iteration.

THEOREM 2.6. *If* $A = A(G)$ *is non-singular, then for every* v_i *there exists a* v_j *such that* $A_{i,j} \neq 0$ *and* $(A^{-1})_{j,i} \neq 0$, *i.e.* (v_i, v_j) *is an allowed edge. Moreover, the matrix* A *with rows* i, j *and columns* i, j *removed is also non-singular.*

These considerations are enough to establish the error bounds for our algorithms which are more efficient realizations of the simple idea described above. Additional argument concerning randomization will only be needed in Subsection 3.1 and Subsection 4.2.3.

Throughout the rest of this paper, we use finite field arithmetic and omit the "with high probability" phrase in most statements.

The probabilistic algorithms presented so far, as well as the algorithms presented in the remainder of this paper, are all Monte Carlo algorithms. Using techniques from [14] and [4] all these algorithms can be made Las Vegas.

Notice that the matching algorithm for the bipartite case uses randomization in a very limited manner, namely to find a full-rank substitution for a skew-symmetric adjacency matrix $\tilde{A}(G)$. Other than that, the algorithm is deterministic.

2.2 Maximum vs. Perfect Matchings

The problem of finding a maximum matching is not harder than the problem of finding a perfect matching, as the following theorem shows

THEOREM 2.7. *The problem of finding a maximum matching can be reduced in randomized time $O(n^\omega)$ to the problem of finding a perfect matching.*

This was proved by Ibarra and Moran [9][1] in the bipartite case and by Rabin and Vazirani [14] in the general case.

In the remainder of this paper we only consider the problem of finding a perfect matching.

3. CONSTRUCTING MATCHINGS VIA GAUSSIAN ELIMINATION

For any matrix A, let $A_{R,C}$ denote a submatrix of A corresponding to rows R and columns C. Consider a random bipartite adjacency matrix $A = A(G)$ of a bipartite graph $G = (U, V, E)$, where $|U| = |V| = n$, $U = \{u_1, u_2, \ldots, u_n\}$, $V = \{v_1, v_2, \ldots, v_n\}$. If $(u_1, v_1) \in E$ and $A^{-1}_{1,1} \neq 0$, then (u_1, v_1) is an allowed edge. We may thus choose this edge as a matching edge and try to find a perfect matching in $G' = G - \{u_1, v_1\}$. The problem with this approach is that edges that were allowed in G might not be allowed in G'. Computing the matrix $A(G')^{-1}$ from scratch is out of the question as this would lead to a $O(n^{\omega+1})$ algorithm for perfect matchings. There is however another way shown in the following well known property of the Schur complement:

THEOREM 3.1 (ELIMINATION THEOREM). *Let*

$$A = \begin{matrix} a_{1,1} & v^T \\ u & B \end{matrix} \qquad A^{-1} = \begin{matrix} \hat{a}_{1,1} & \hat{v}^T \\ \hat{u} & \hat{B} \end{matrix} \quad ,$$

where $\hat{a}_{1,1} \neq 0$. Then $B^{-1} = \hat{B} - \hat{u}\hat{v}^T/\hat{a}_{1,1}$.

PROOF. Since $AA^{-1} = I$, we have

$$\begin{matrix} a_{1,1}\hat{a}_{1,1} + v^T\hat{u} & a_{1,1}\hat{v}^T + v^T\hat{B} \\ u\hat{a}_{1,1} + B\hat{u} & u\hat{v}^T + B\hat{B} \end{matrix} = \begin{matrix} I_1 & 0 \\ 0 & I_{n-1} \end{matrix} \quad .$$

Using these equalities we get

$$B(\hat{B} - \hat{u}\hat{v}^T/\hat{a}_{1,1}) = I_{n-1} - u\hat{v}^T - B\hat{u}\hat{v}^T/\hat{a}_{1,1} =$$
$$I_{n-1} - u\hat{v}^T + u\hat{a}_{1,1}\hat{v}^T/\hat{a}_{1,1} = I_{n-1} - u\hat{v}^T + u\hat{v}^T = I_{n-1}.$$

and so $B^{-1} = \hat{B} - \hat{u}\hat{v}^T/\hat{a}_{1,1}$ as claimed. □

[1]Schrijver suggests in [15], that this paper shows how to actually construct the matching, but this is not true.

The modification of $\hat{B}$ described in this theorem is in fact a single step of the well known Gaussian elimination procedure. In this case, we are eliminating the first variable (column) using the first equation (row). Similarly, we can eliminate from A^{-1} any other variable (column) j using any equation (row) i, such that $A^{-1}_{i,j} \neq 0$.

As an immediate consequence of Theorem 3.1 we get simple $O(n^3)$ algorithms for finding perfect matchings in bipartite (Algorithm 1) and general (Algorithm 2) graphs.

Algorithm 1 Simple algorithm for perfect matchings in bipartite graphs.

SIMPLE-BIPARTITE-MATCHING(G):
$B = A^{-1}(G)$
$M = \emptyset$
for $c = 1$ to n do

1. find a row r, not yet eliminated, and such that $B_{r,c} \neq 0$ and $A(G)_{c,r} \neq 0$ (i.e. (u_c, v_r) is an allowed edge in $G - V(M)$);
2. eliminate the r-th row and the c-th column of B;
3. add (u_c, v_r) to M;

Algorithm 2 Simple algorithm for perfect matchings in general graphs.

SIMPLE-GENERAL-MATCHING(G):
$B = A^{-1}(G)$
$M = \emptyset$
for $c = 1$ to n do
if column c is not yet eliminated then

1. find a row r, not yet eliminated, and such that $B_{r,c} \neq 0$ and $A(G)_{c,r} \neq 0$ (i.e. (v_c, v_r) is an allowed edge in $G - V(M)$);
2. eliminate the r-th row and the c-th column of B;
3. eliminate the c-th row and the r-th column of B;
4. add (v_c, v_r) to M;

3.1 Matching Verification

We now consider a particularly simple case of Gaussian elimination with no pivoting and show how to use it to find an inclusion-wise maximal allowed submatching of any matching in time $O(n^\omega)$.

Assume that we are performing a Gaussian elimination on a $n \times n$ matrix X and we always have $X_{i,i} \neq 0$ after eliminating the first $i-1$ rows and columns. In this case, we can avoid any row or column pivoting, and the algorithm SIMPLE-ELIMINATION (Algorithm 3) performs Gaussian elimination of X in time $O(n^\omega)$. By "lazy elimination" we mean storing the expression of the form uv^T/c describing the changes required in the remaining submatrix without actually performing them. These changes are then executed in batches during the calls to UPDATE(R, C) which updates the $X_{R,C}$ submatrix. Suppose that k changes where accumulated for the submatrix $X_{R,C}$ and then UPDATE(R, C) was called. Let these changes be $u_1v_1^T/c_1, u_2v_2^T/c_2, \ldots, u_kv_k^T/c_k$.

Algorithm 3 Gaussian elimination with no pivoting

SIMPLE-ELIMINATION(X):
for $i = 1$ to n do

1. lazily eliminate the i-th row and i-th column of X;
2. let j be the largest integer such that $2^j | i$;
3. UPDATE($\{i+1, ..., i+2^j\}, \{i+1, ..., n\}$);
4. UPDATE($\{i+2^j+1, ..., n\}, \{i+1, i+2^j\}$);

Then the accumulated change of $X_{R,C}$ is

$$u_1 v_1^T/c_1 + u_2 v_2^T/c_2 + \ldots, u_k v_k^T/c_k = UV$$

where U is a $|R| \times k$ matrix with columns $u_1, u_2, \ldots, u_k$ and V is a $k \times |C|$ matrix with rows $v_1^T/c_1, v_2^T/c_2, \ldots, v_k^T/c_k$. The matrix UV can be computed using fast matrix multiplication.

This algorithm is an iterative description of the standard recursive factorization algorithm. It has time complexity $O(n^\omega)$ because of the following lemma

LEMMA 3.2. *The number of changes performed by Algorithm 3 in step 3. and 4. is at most 2^j.*

PROOF. In the i-th iteration rows $i+1, \ldots, i+2^j$ and columns $i+1, \ldots, i+2^j$ are updated. Since $2^j | i$, we have $2^{j+1} | i - 2^j$, so these rows and columns were also updated in step $i - 2^j$. Thus, the number of changes is at most 2^j. □

It follows from this lemma, that the cost of the update in i-th iteration is proportional to the cost of multiplying the $2^j \times 2^j$ matrix by a $2^j \times n$ matrix. By splitting the second matrix into $2^j \times 2^j$ square submatrices, this can be done in time $n/2^j (2^j)^\omega = n(2^j)^{\omega-1}$. Now, every j appears $n/2^j$ times, so we get the total time complexity of

$$\sum_{j=0}^{\lceil \log n \rceil} n/2^j n(2^j)^{\omega-1} = n^2 \sum_{j=0}^{\lceil \log n \rceil} (2^{\omega-2})^j = O(n^2 (2^{\omega-2})^{\lceil \log n \rceil}) = O(n^\omega)$$

Thus, we get

THEOREM 3.3. *Naive iterative Gaussian elimination without row or column pivoting can be implemented in time $O(n^\omega)$ using a lazy updating scheme.*

The SIMPLE-ELIMINATION algorithm is similar to the classic Hopcroft-Bunch algorithm [3]. Being iterative (and much simpler), it is better suited for proving Theorem 3.4, where we need to skip row-column pairs if the corresponding diagonal element is zero.

THEOREM 3.4. *Let G be a graph having a perfect matching. For any matching M of G, an inclusion-wise maximal allowed (i.e. extendable to a perfect matching) submatching M' of M can be found in time $O(n^\omega)$.*

PROOF. Let $M = \{(v_1, v_2), (v_3, v_4), \ldots, (v_{k-1}, v_k)\}$ and let $v_{k+1}, v_{k+2}, \ldots, v_n$ be the unmatched vertices. Compute the inverse $A(G)^{-1}$ of random adjacency matrix of G and permute its rows and columns so that the row order is v_1,v_2,v_3, $v_4, \ldots, v_n$ and the column order is v_2,v_1,v_4,v_3,$\ldots$,v_n,v_{n-1}. Now, perform Gaussian elimination of the first k rows and k columns using the SIMPLE-ELIMINATION algorithm, but if the eliminated element is zero just skip to the next iteration. The eliminated row-column pairs correspond to a maximal submatching M' of M. □

This verification algorithm is interesting in itself but it is also a key ingredient in our matching algorithm for general graphs.

REMARK 3.5. *Theorem 2.6 does not guarantee that $A(G)^{-1}$ correctly encodes the allowed edges. Fortunately, we can still use Theorem 2.4. For sufficiently large (but polynomial) p, we get the correct non-zero structure throughout the algorithm.*

4. ALGORITHMS FOR FINDING A PERFECT MATCHING

4.1 Bipartite Graphs

We start with the simple case, that is with bipartite graphs. Let $G = (U, V, E)$, $|U| = |V| = n$, be a bipartite graph having a perfect matching and let $A = A(G)$ be its bipartite adjacency matrix.

Assume that we fix the column order and eliminate the columns from left to right. When eliminating the i-th column we have to find a row j, not already eliminated, such that $(u_i, v_j) \in E$ and the updated value of $A^{-1}(j, i)$ is non-zero. We cannot use the SIMPLE-ELIMINATION algorithm, because we do not know the elimination order of the rows in advance. There is, however, a bit more complicated algorithm that allows row pivoting and still works in time $O(n^\omega)$ — the classic LU factorization algorithm of Hopcroft and Bunch (see [3] and [1]). This algorithm looks for a non-zero element in the currently eliminated column and it is possible to modify it so that it only chooses elements corresponding to edges of G.

For completeness we present a simplified version of the Hopcroft-Bunch algorithm in Appendix A.

4.2 General Graphs

4.2.1 Basic Idea

In case of general graphs, the removal of vertices $u, v \in V$, $(u, v) \in E$ corresponds to first eliminating the u-th row and the v-th column and then eliminating the v-th row and the u-th column. The lazy computation mechanism of Hopcroft-Bunch algorithm does not work any more.

The idea of our algorithm (Algorithm 4) is that, if $|M'| \geq n/8$, then GENERAL-MATCHING is called for a graph smaller by a constant factor, and if $|M'| \leq n/8$ then it is possible to use some structural properties to partition G into smaller pieces, and find a perfect matching in each of them separately.

Algorithm 4 $O(n^\omega)$ algorithm for perfect matchings in general graphs.

GENERAL-MATCHING(G):

1. Find a matching M of size $\geq n/4$ using the greedy algorithm.
2. Verify M using the algorithm from Theorem 3.4. Let M' be the maximal allowed subset of M.
3. Match the vertices of M' and remove them from G.
4. If $|M'| \geq n/8$, call GENERAL-MATCHING($G - V(M')$), where $V(M')$ is the set of vertices matched by M'.
5. Otherwise, call PARTITION($G - V(M')$).

4.2.2 Canonical Partition

We now describe the details of the PARTITION(G) procedure. Let us start with a few definitions. A graph G is called *elementary* if G has a perfect matching and allowed edges of G form a connected spanning subgraph of G. Let us define a relation $\sim_G$ on the set V as follows: $u \sim_G v$ iff either $u = v$ or $G - \{u, v\}$ does not have a perfect matching. The following theorem is due to Lovász (see [11])

THEOREM 4.1. *If G is elementary, then $\sim_G$ is an equivalence relation.*

By $P(G)$ we denote the set of equivalence classes of $\sim_G$, the so-called *canonical partition* of G. Recall that $G - \{u, v\}$ has a perfect matching iff $A(G)^{-1} \neq 0$, so the matrix $A(G)^{-1}$ encodes the canonical partition. $P(G)$ has very nice structural properties as the following theorem shows (see [11] for details)

THEOREM 4.2. *Let G be elementary, let $S \in P(G)$ with $|S| \geq 2$ and let C be any component of $G - S$. Then:*

1. *the bipartite graph G'_S obtained from G by contracting each component of $G - S$ to a single vertex and deleting edges in S is elementary;*
2. *the graph C is factor-critical, i.e. for any vertex $v \in V(C)$, $C - v$ has a perfect matching;*
3. *the graph C' obtained from $G[V(C) \cup S]$ by contracting the set S to a single vertex u_C is elementary;*
4. *$P(C') = \{\{u_C\}\} \cup \{T \cap V(C) | T \in P(G)\}$.*

In particular, this means that the number of connected components in $G - S$ is equal to $|S|$ and that every perfect matching of G matches vertices of S with vertices in different components of $G - S$. Moreover, any such matching of vertices of S can be extended to a perfect matching of G.

The PARTITION algorithm (Algorithm 5) breaks G down into bipartite and non-bipartite pieces and reduces the problem of finding a perfect matching in G to problem of finding perfect matching in all pieces using Algorithm 7 and Algorithm 4. We now show, that the total size of the pieces is

Algorithm 5 The partitioning algorithm.

PARTITION(G):

1. If G is not elementary, compute the elementary compononents of G and perform partition of each component;
2. Let $S \in P(G)$ with $|S| = k \geq 2$ and let $C_1, \ldots, C_k$ be connected components of $G - S$. Assume that C_1 is the largest component. Any perfect matching of G is a sum of $k + 1$ perfect matchings. One of them is a perfect bipartite matching between S and a set C containing a single vertex c_i from each of the C_i. The other k matchings are perfect matchings in $C_i - c_i$.
3. Let C'_1 be the graph $G[S \cup V(C_1)]$ with S contracted to a single *artificial* vertex s. We have $P(C'_1) = \{\{s\}\} \cup \{S_i \cap V(C_1) | S_i \in P(G)\}$, so we can infer $P(C'_1)$ from $P(G)$.
4. If $P(C'_1)$ contains a non-trivial class, call PARTITION(C'_1). Otherwise call GENERAL-MATCHING(C'_1).
5. Let M'_1 be the matching found.
6. M'_1 matches the contracted vertex s with some vertex c. Let $v \in S$ be any neighbour of c. Remove s from C'_1 and match c with v.
7. Extend $\{(c, v)\}$ to a perfect bipartite matching between S and some set C containing a single vertex from each of the C_i. This is possible since $\{(c, v)\}$ is an allowed edge (if it would not be allowed, we would have $c \sim_G v$, and so $c \in S$).
8. Remove the matched vertex from each of the C_i and find perfect matchings in each of the resulting graphs.

equal to n, and that the non-bipartite pieces are by a constant factor smaller than G. It follows that the complexity of both Algorithm 4 and Algorithm 5 is $O(n^\omega)$.

LEMMA 4.3. *The total number of vertices of graphs in which PARTITION finds perfect matchings by calling the BIPARTITE-MATCHING algorithm or the GENERAL-MATCHING algorithm is equal to n.*

PROOF. The only problem here is the creation of artificial vertices in step 3. But notice that the matching of every such vertex is then used in step 6. to match some other vertex "for free". While going up the partition tree, all these artificial vertices are eventually exchanged with real ones and so the total number of vertices matched is equal to the number n of real vertices. □

LEMMA 4.4. *The PARTITION algorithm calls the GENERAL-MATCHING algorithm for graphs with $\leq 7/9n$ vertices.*

PROOF. The components that are not the largest components at their stage of partition can have at most $n/2$ vertices since otherwise they would be the largest components.

The largest component, on the other hand, is subject to further partition, so we only have to prove that when the parti-

tion stops in step 4., C_1' has at most $7/9n$ vertices. Observe that whenever PARTITION(G) is called from GENERAL-MATCHING, graph G has a matching $\hat{M}$ consisting of only unallowed edges and spanning at least $n/3$ vertices. Let $\hat{V}$ be the set of these vertices. The edges of $\hat{M}$ can never go across the partition, so when the decomposition stops in step 4., C_1' does not contain any vertices of $\hat{V}$. Now, notice that with each step of the partition, the largest component loses at least three vertices (at least two vertices in S and at least one vertex in some smaller component) and gains exactly one artificial vertex. Thus, the number of vertices in C_1' when partition stops is at most $n-2/3|\hat{V}| \leq n-2/9n = 7/9n$. □

We have thus reduced the problem of finding a perfect matching in a graph of size n to the problem of finding perfect matchings or bipartite perfect matching in its pieces. The size of each non-bipartite piece is at most $\frac{7}{9}n$. Since we can find the perfect matchings in these pieces in time $O(s^\omega)$, where s is the size of the piece, the complexity of the PARTITION algorithm is $O(n^\omega)$.

Note that the size of a bipartite piece may be $\Theta(n)$, but this is not a problem since we already know how to find bipartite perfect matchings in time $O(n^\omega)$.

Theorem 2.6 does not guarantee that $A(G)^{-1}$ correctly encodes the canonical partition, compare Remark 3.5.

4.2.3 Implementation Details

So far we have ignored the problem of performing the canonical partition and concentrated on the complexity of finding perfect matchings in its parts. The algorithm performing the whole partition in time $\tilde{O}(n^2)$ is implicit in the work of Cheriyan [4], but for completeness we present the main ideas here.[2]

First of all, we use the dynamic connectivity algorithm of Holm, de Lichtenberg and Thorup [8]. Their algorithm supports the following operations on a dynamic graph:

- INSERT(e) — inserts edge e into G;
- DELETE(e) — deletes edge e from G;
- SIZE(v) — returns the size the connected component of v;
- CONNECTED(u, v) — tests if u and v are connected by a path in G;

All these operations require only polylogarithmic time.

Single stage of PARTITION is presented as Algorithm 6. Steps 3., 4. and 6. take time $\tilde{O}(n)$. Since partition has at most $O(n)$ stages, the total time required to perform these steps is $\tilde{O}(n^2)$.

The complexity of step 1. is $O(n^2)$ for the whole partition. To see this, notice that we only have to test if v is an element of a non-trivial set $S \in P(G)$ once. If v is an element of such S, then we can use S as a basis for partition and C will not contain v. If, on the other hand, v is not an element of a non-trivial set $S \in P(G)$, that it will never be, because of the property 4. in Theorem 4.2.

[2] $\tilde{O}$ denotes the so-called "soft O" notation, i.e. $f(n) = \tilde{O}(g(n))$ iff $f(n) = O(g(n)\log^k n)$ for some constant k.

Algorithm 6 Implementation details of the PARTITION algorithm

PARTITION(G): /* implementation details */

1. choose a non-trivial set $S \in P(G)$, if one exists;
2. call DELETE(e) for all edges between $G[S]$ and $G-S$; let T be the set of all the $G-S$ endpoints of these edges;
3. call SIZE(v) for all $v \in T$ and let u be a vertex for which the returned value is the largest;
4. call CONNECTED(u, v) for all $v \in T$ and let D be the set of vertices in T disconnected from u; these vertices represent the smaller components and u represents the largest component;
5. call DFS starting from vertices in D thus identifying all the components except the largest one; the largest component is identified without performing a DFS as a complement of S and the smaller components; let C be the largest component;
6. create additional vertex s and call INSERT(s, v) for all $v \in T - D$;

Steps 2. and 5. require time proportional to the number of edges between $G[S]$ and $G-S$ and edges inside smaller components. These sets of edges are disjoint for different phases so the total complexity of steps 2. and 5. is $\tilde{O}(m) = \tilde{O}(n^2)$.

The partition algorithm can be thus implemented to run in time $\tilde{O}(n^2)$. This does not increase the time complexity of the matching algorithm.

5. CONCLUDING REMARKS AND OPEN PROBLEMS

We have presented $O(n^\omega)$ randomized algorithms for the maximum matching problem, therefore breaking the long-standing $O(n^{2.5})$ barrier. Using similar techniques together with separator decomposition and fast nested dissection, maximum matchings in planar graphs can be found in time $O(n^{\omega/2})$. These results will be published elsewhere [13].

In the bipartite case, the only non-trivial part of the algorithm is the fast matrix multiplication procedure. The general case is more complex because of performing the canonical partition. Is this structural approach really neccessary?

Acknowledgments

The authors would like to thank their favourite supervisor Krzysztof Diks for numerous useful discussions.

6. REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.*

Addison-Wesley Longman Publishing Co., Inc., 1974.

[2] N. Blum. A new approach to maximum matching in general graphs. In *Proc. 17th ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 586–597. Springer-Verlag, 1990.

[3] J. Bunch and J. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[4] J. Cheriyan. Randomized $\Theta(M(|V|))$ algorithms for problems in matching theory. *SIAM Journal on Computing*, 26(6):1635–1655, 1997.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 1–6. ACM Press, 1987.

[6] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[7] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4):815–853, 1991.

[8] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[9] O. H. Ibarra and S. Moran. Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication. *Inf. Process. Lett.*, 13(1):12–15, 1981.

[10] L. Lovász. On determinants, matchings and random algorithms. In L. Budach, editor, *Fundamentals of Computation Theory*, pages 565–574. Akademie-Verlag, 1979.

[11] L. Lovász and M. Plummer. *Matching Theory*. Akadémiai Kiadó, 1986.

[12] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the twenty first annual IEEE Symposium on Foundations of Computer Science*, pages 17–27, 1980.

[13] M. Mucha and P. Sankowski. Maximum matching in planar graphs via gaussian elimination. *12'th Annual Euroepan Symposium on Algorithms, accepted*, 2004.

[14] M. O. Rabin and V. V. Vazirani. Maximum matchings in general graphs through randomization. *Journal of Algorithms*, 10:557–567, 1989.

[15] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer-Verlag, 2003.

[16] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.

[17] W. T. Tutte. The factorization of linear graphs. *J. London Math. Soc.*, 22:107–111, 1947.

[18] R. Zippel. Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226, Berlin, 1979. Springer-Verlag.

APPENDIX

A. THE HOPCROFT-BUNCH ALGORITHM

We now present a slight modification of the Hopcroft-Bunch algorithm that can be used to find perfect matchings in bipartite graphs (see Subsection 4.1). Hopcroft and Bunch present their algorithm in a recursive manner. We have decided to use iterative approach to emphasize the fact that this algorithm is an extension of the SIMPLE-BIPARTITE-MATCHING algorithm.

The basic idea is that since columns are eliminated from left to right, we do not need to update the whole rows, only the parts that will be needed soon. Notice that in this case the

Algorithm 7 $O(n^\omega)$ algorithm for perfect matchings in bipartite graphs.

BIPARTITE-MATCHING(G):
$B = A^{-1}(G)$
$M = \emptyset$
for $c = 1$ to n do
1. find row r such, that $A(c,r) \neq 0$ and $B(r,c) \neq 0$, i.e. (u_c, v_r) is an allowed edge $G - V(M)$;
2. lazily eliminate the r-th row and c-th column of B;
3. let j be the largest integer such that $2^j | c$;
4. update columns $c+1, c+2, \ldots, c+2^j$;

update operation is a bit more complicated than in Algorithm 3. Suppose we update columns $c+1, c+2, \ldots, c+2^j$. These columns were updated in the $(c-2^j)$-th iteration, so we have to perform updates resulting from elimination of columns $c-2^j+1, c-2^j+2, \ldots, c$. Let us assume without loss of generality that the corresponding rows have the same numbers. The first update comes from elimination of the $(c-2^j+1)$-th column and $(c-2^j+1)$-th row. The second update comes from elimination of the $(c-2^j+2)$-th column and $(c-2^j+2)$-th row, but we do not know the values $B(c-2^j+2, c+1), B(c-2^j+2, c+2), \ldots, B(c-2^j+2, c+2^j)$ in this row without performing the first update. Fortunately, we can use the lazy computation trick once again! We lazily perform the postponed updates one after another and after performing the i-th update, we only compute the actual values for the rows used in the next 2^l updates, where l is the largest number, such that $2^l | i$.

What is the time complexity of updating columns $c+1, c+2, \ldots, c+2^j$? We have to perform 2^j updates and performing the i-th update requires multiplication of a $2^l \times 2^l$ matrix by a $2^l \times 2^j$ matrix, where l is as before. This is the same situation as in the analysis of Algorithm 3, but now we have 2^j instead of n. The complexity is thus $O(2^{j\omega})$. We also have to count the time required to update the rows $c+1, c+2, \ldots, c+2^j$ and this requires multiplication of a $(n-c) \times 2^j$ matrix by a $2^j \times 2^j$ matrix. This can be done in time $O(n/2^j 2^{j\omega}) = O(n(2^j)^{\omega-1})$. We now have to sum it up over all j, but again, this is the same sum as before and we get $O(n^\omega)$.

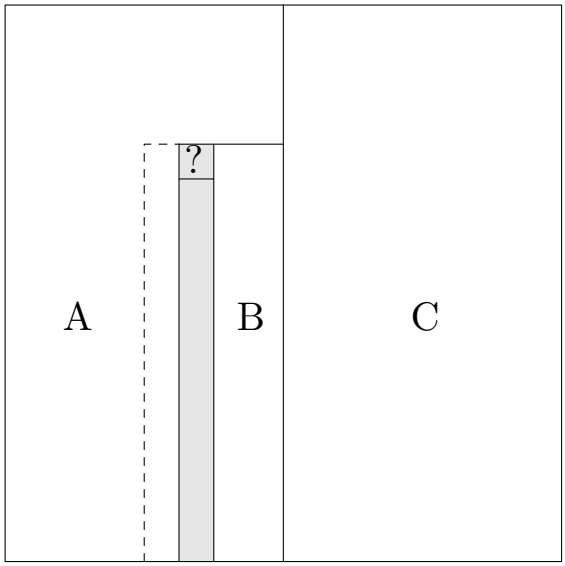

Figure 1: Update in the bipartite perfect matching algorithm. The shaded region is being updated. Changes in the square labelled "?" depend on each other, while the changes to the entries below the "?" can be computed using straight matrix multiplication. The rest of the matrix can be divided into three parts: the part marked A has been fully updated and further elimination will not affect the entries in A, the part marked B has already been updated but further elimination will affect the entries in B and additional updates will be required, the part marked C has not been updated yet. The region outlined with dashed lines corresponds to the columns used in the update being performed.