Fast algorithms for Maximum Subset Matching and All-Pairs Shortest Paths in graphs with a (not so) small vertex cover

Noga Alon [*] *Raphael Yuster* [†]

Abstract

In the *Maximum Subset Matching* problem, which generalizes the maximum matching problem, we are given a graph $G = (V, E)$ and $S \subset V$. The goal is to determine the maximum number of vertices of S that can be matched in a matching of G. Our first result is a new randomized algorithm for the Maximum Subset Matching problem that improves upon the fastest known algorithms for this problem. Our algorithm runs in $\tilde{O}(ms^{(\omega-1)/2})$ time if $m \geq s^{(\omega+1)/2}$ and in $\tilde{O}(s^{\omega})$ time if $m \leq s^{(\omega+1)/2}$, where $\omega < 2.376$ is the matrix multiplication exponent, m is the number of edges from S to $V \setminus S$, and $s = |S|$. The algorithm is based, in part, on a method for computing the rank of sparse rectangular integer matrices.

Our second result is a new algorithm for the *All-Pairs Shortest Paths* (APSP) problem. Given an undirected graph with n vertices, and with integer weights from $\{1, \ldots, W\}$ assigned to its edges, we present an algorithm that solves the APSP problem in $\tilde{O}(Wn^{\omega(1,1,\mu)})$ time where $n^{\mu} = vc(G)$ is the *vertex cover number* of G and $\omega(1, 1, \mu)$ is the time needed to compute the Boolean product of an $n \times n$ matrix with an $n \times n^{\mu}$ matrix. Already for the unweighted case this improves upon the previous $O(n^{2+\mu})$ and $\tilde{O}(n^{\omega})$ time algorithms for this problem. In particular, if a graph has a vertex cover of size $O(n^{0.29})$ then APSP in unweighted graphs can be solved in asymptotically optimal $\tilde{O}(n^2)$ time, and otherwise it can be solved in $O(n^{1.844}vc(G)^{0.533})$ time.

The common feature of both results is their use of algorithms developed in recent years for fast (sparse) rectangular matrix multiplication.

1 Introduction

A *matching* in a graph is a set of pairwise disjoint edges. A *maximum matching* is a matching of largest possible size. The problem of finding a maximum matching is fundamental in both practical and theoretical computer science.

The first polynomial time algorithm for finding a maximum matching in a general graph was obtained by Edmonds [6]. The currently fastest deterministic algorithms for this problem run in $O(mn^{1/2})$ time (see [13, 2, 19, 7]), where m and n are the number of edges and vertices, respectively, in the input graph. For dense graphs, better *randomized* algorithms are known. Lovász [12] showed that the cardinality of a maximum matching can be determined, with high probability, by computing the rank of a matrix. In particular, checking whether a graph has a perfect matching amounts to checking whether a determinant of a certain matrix, whose construction involves randomization, is nonzero.

[*]Department of Mathematics, Tel Aviv University, Tel Aviv 69978, Israel. E–mail: nogaa@post.tau.ac.il

[†]Department of Mathematics, University of Haifa, Haifa 31905, Israel. E–mail: raphy@research.haifa.ac.il

This randomized algorithm can be implemented to run in $O(n^\omega)$ time, where ω is the exponent of fast matrix multiplication. Coppersmith and Winograd [5] showed that $\omega < 2.376$. Recently, Mucha and Sankowski [14] solved a long standing open problem and showed that a maximum matching can be *found*, with high probability, in $O(n^\omega)$ time.

Our first main result in this paper concerns a natural generalization of the maximum matching problem. For a graph $G = (V, E)$ and a subset of vertices $S \subset V$, let $f(S)$ denote the maximum possible number of vertices in S that can be matched in a matching of G. Notice that if $S = V$ then $f(S)$ is simply the size of a maximum matching of G. In general, however, not every maximum matching of G saturates the maximum possible number of vertices of S (e.g. consider a triangle where S consists of two vertices) and, conversely, not every matching that saturates $f(S)$ vertices of S can be extended to a maximum matching (e.g. consider a path of length 3 where S consists of the two internal vertices). Thus, the *MAXIMUM SUBSET MATCHING* problem is a true generalization of the maximum matching problem.

Maximum Subset Matching also has natural applications. Consider a graph modeling a social network, the vertices being the members of the network and the edges being the symmetric social relations. There are two types of members: privileged members (e.g. registered customers) and non-privileged members. Our goal is to socially match the maximum number of privileged members.

We present an efficient randomized algorithm for computing $f(S)$. The running time of our algorithm is expressed in terms of $s = |S|$ and the number of edges connecting S to $V \setminus S$, denoted by m. There could be as many as $\Theta(s^2)$ edges inside S, and hence the graph may have as many as $\Theta(m + s^2)$ edges (clearly we may assume that $V \setminus S$ is an independent set). As usual in matching algorithms, we assume that the graph has no isolated vertices.

Theorem 1.1 *Let $G = (V, E)$ be a graph and let $S \subset V$, where $|S| = s$ and there are m edges from S to $V \setminus S$. Then, there is a randomized algorithm that computes $f(S)$ w.h.p. in time*

$$\tilde{O}\left(\begin{cases} ms^{\frac{\omega-1}{2}} & \text{if } m \geq s^{\frac{\omega+1}{2}}, \\ s^\omega & \text{if } m \leq s^{\frac{\omega+1}{2}} \end{cases}\right).$$

To evaluate the running time obtained in Theorem 1.1 we compare our algorithm to known existing matching algorithms. First notice that for the current best upper bound of ω, which is less than 2.376, we have that if $m \geq s^{1.688}$ the algorithm runs in $O(ms^{0.688})$ time, and if $m \leq s^{1.688}$ the algorithm runs in $O(s^{2.376})$ time. Notice that the running time is *not* expressed as a function of $|V| = n$ and that n may be as large as $s+m$. Maximum Subset Matching can be solved via maximum *weighted* matching algorithms as follows. Assign to every vertex with both endpoints in S weight 2, and edges from S to $V \setminus S$ weight 1. Then, clearly, a maximum weighted matching has weight $f(S)$. The algorithm of Gabow and Tarjan [7] is currently the fastest algorithm for maximum weighted matching in general graphs. In our setting, it runs in $\tilde{O}(\sqrt{n}(m + s^2))$ time (here $m + s^2$ is our upper bound on the total number of edges), which is worse than the running time of the algorithm of Theorem 1.1 for a wide range of parameters (in fact, if $\omega = 2 + o(1)$ as conjectured by many researchers, then it is never better, for any valid combination of the parameters s, m, n). It is not known how to apply Lovás'z randomized maximum cardinality matching algorithm to the weighted case. Even if it were possible, the obtained running time would only be $\tilde{O}(n^\omega)$.

We now turn to our second main result. In the *All-Pairs Shortest Paths* (APSP) problem, which is one of the most fundamental algorithmic graph problems, we are given a graph $G = (V, E)$ with

$V = \{1, \ldots, n\}$, and our goal is to compute the *distance matrix* of G. This is an $n \times n$ matrix D where $D(i, j)$ is the length of a shortest path from i to j, for all $i, j = 1, \ldots, n$. We also require a concise $n \times n$ data structure so that given a pair i, j, a shortest path from i to j can be constructed in time which is proportional to the number of edges it contains. In the most fundamental case, the graph is undirected and unweighted; this is the main case we address in this paper (more generally, we allow the weights to be positive integers). The currently fastest algorithms for the APSP problem are either the simple obvious $O(nm)$ algorithm (here $m = |E|$) consisting of breadth first search from each vertex, or, as m gets larger, a randomized algorithm of Seidel [16] that solves the problem in $\tilde{O}(n^{\omega})$ time. This algorithm also has a deterministic version [8, 1]. Shoshan and Zwick [17] proved that if the graph has positive weights from $\{1, \ldots, W\}$ then APSP can be solved, deterministically, in $\tilde{O}(Wn^{\omega})$ time.

Is there a natural graph parameter for which there exists an algorithm whose running time is expressed in terms of it and which always performs at least as well as the $\tilde{O}(n^{\omega})$ algorithm, and is generally faster? Our main result shows that there is one.

Let $vc(G)$ be the *vertex cover number* of G, that is the smallest possible cardinality of a subset $S \subset V$ so that $V \setminus S$ is an independent set. The fact that $vc(G)$ is NP-Hard to compute is well known and its decision version is one of the canonical examples for NP-Completeness. It is also well known, and trivial, that $vc(G)$ can be approximated to within a factor of two, in linear time (simply take all the vertices in a maximal matching with respect to containment). Our main result shows that if G is an undirected graph with n vertices and $vc(G) = n^{\mu}$ then the APSP problem can be solved in time $\tilde{O}(Wn^{\omega(1,1,\mu)})$, where each weight is taken from $\{1, \ldots, W\}$. Here $\omega(1, 1, \mu)$ is the rectangular matrix multiplication exponent. Namely, it is the number of algebraic operations needed to multiply an $n \times n$ matrix with an $n \times n^{\mu}$ matrix over an arbitrary ring (hence, $\omega = \omega(1, 1, 1)$).

Theorem 1.2 *There is an APSP algorithm that, given an undirected n-vertex graph G with weights from $\{1, \ldots, W\}$ and with $vc(G) = n^{\mu}$, runs in $\tilde{O}(Wn^{\omega(1,1,\mu)})$ time. In particular, if W is constant and $vc(G) \leq n^{0.294}$ then the algorithm runs in asymptotically optimal $\tilde{O}(n^2)$ time and, if $vc(G) > n^{0.294}$ then the algorithm runs in $O(n^{1.844} vc(G)^{0.533})$ time.*

In our proof of Theorem 1.2 we construct the distance matrix in the guaranteed running time. It is also possible to obtain a concise data structure representing the paths, by using witnesses for (rectangular) matrix multiplication, as shown in [1]; the details of this latter construction, which is quite standard, will appear in the full version of this paper. In Section 2 we list the known results for $\omega(1, 1, \mu)$. In particular, Coppersmith [4] proved that $\omega(1, 1, \mu) = 2 + o(1)$ for $\mu < 0.294$, and that, assuming $\omega = \omega(1, 1, 1) > 2$, then $\omega(1, 1, \mu) < \omega(1, 1, 1)$ for all $\mu < 1$ (see Lemma 2.3 in the next section). Thus, at present, our algorithm is faster than the $O(n^{\omega})$ algorithm for all unweighted graphs having a vertex cover of size $n^{1-\epsilon}$. We also note that the proof of Theorem 1.2 does not assume that we can compute $vc(G)$ precisely. As mentioned earlier, there is a $\Theta(m + n)$ 2-approximation algorithm for $vc(G)$, which suffices for our purposes.

The rest of this paper is organized as follows. As both of our main results, although having completely different proofs, rely on fast rectangular matrix multiplication, we present in Section 2 the facts we need about fast rectangular matrix multiplication algorithms. In Section 3 we address our first main result on the maximum subset matching problem and prove Theorem 1.1. The APSP problem in undirected graphs is addressed in Section 4 where Theorem 1.2 is proved. The final section contains some concluding remarks and open problems.

2 Fast (sparse) rectangular matrix multiplication

We start this section by presenting some parameters related to fast rectangular matrix multiplication. Our assumption is that the input matrices are given to us in sparse representation, that is as a collection of triplets (row, column, value) listing all positions where the matrix is non-zero. Thus, the size of the representation of an input matrix is linear in the number of non-zero entries of the matrix.

Let $M(a,b,c)$ be the minimal number of algebraic operations needed to multiply an $a \times b$ matrix by a $b \times c$ matrix over an arbitrary ring R. Let $\omega(r,s,t)$ be the minimal exponent for which $M(n^r,n^s,n^t) = O(n^{\omega(r,s,t)})$. Recall that $\omega = \omega(1,1,1) < 2.376$ [5]. The best bounds available on $\omega(1,\mu,1)$, for $0 \le \mu \le 1$ are summarized in the following results. Before stating them we need to define two more constants, α and β, related to rectangular matrix multiplication.

Definition 2.1 $\alpha \;=\; \max\{\; 0 \le \mu \le 1 \;\mid\; \omega(1,\mu,1) = 2 + o(1) \;\} \quad , \quad \beta \;=\; \dfrac{\omega - 2}{1-\alpha}\,.$

Lemma 2.2 (Coppersmith [4]) $\alpha > 0.294$ *.*

It is not difficult to see that Lemma 2.2 implies the following theorem. A proof can be found, for example, in Huang and Pan [10].

Lemma 2.3 $\omega(1,\mu,1) = \omega(1,1,\mu) = \omega(\mu,1,1) \;\le\; \begin{cases} 2+o(1) & \text{if } 0 \le \mu \le \alpha, \\ 2+\beta(\mu-\alpha)+o(1) & \text{otherwise.} \end{cases}$

Notice that Coppersmith's result implies that if A is an $n \times n^{0.294}$ matrix and B is an $n^{0.294} \times n$ matrix then AB can be computed in $O(n^{2+o(1)})$ time (assuming ring operations take constant time), which is essentially optimal since the product is an $n \times n$ matrix that may contain no zero elements at all. Note that with $\omega = 2.376$ and $\alpha = 0.294$ we get $\beta \simeq 0.533$. If $\omega = 2 + o(1)$, as conjectured by many, then $\alpha = 1$. (In this case β is not defined, but also not needed.)

We now present results for fast sparse matrix multiplication. The running times of these algorithms are expressed in terms of α, β, and ω.

Lemma 2.4 (Yuster and Zwick [21]) *The product of two $n \times n$ matrices over a ring R, each with at most m non-zero elements, can be computed in time*

$$O(\min\{\; m^{\frac{2\beta}{\beta+1}}\, n^{\frac{2-\alpha\beta}{\beta+1}} + n^{2+o(1)}\;,\; n^{\omega}\;\}).$$

Note that for $\omega = 2.376$, $\alpha = 0.294$, and $\beta = 0.533$, the algorithm runs in $O(\min\{m^{0.7}n^{1.2} + n^{2+o(1)}\;,\; n^{2.376}\})$ time (in fact, by "time" we assume that each algebraic operation in the ring takes constant time; if not, then the running time should be multiplied by the time needed for an algebraic operation).

It is possible to extend the method of [21] to rectangular matrices. This was explicitly done in [11]. To simplify the runtime expressions, we only state the case where A and B^T (where A and B are the two matrices being multiplied) have the same dimensions, as this case suffices for our purposes.

Lemma 2.5 (Kaplan, Sharir, and Verbin [11]) *Let A be an $n \times r$ matrix and let B be an $r \times n$ matrix, over a ring R, each with at most m non-zero elements. Then AB can be computed in time*

$$O\left(\begin{cases} mn^{\frac{\omega-1}{2}} & \textit{if } m \geq n^{\frac{\omega+1}{2}}, \\ m^{\frac{2\beta}{\beta+1}}n^{\frac{2-\alpha\beta}{\beta+1}} & \textit{if } n^{1+\frac{\alpha}{2}} \leq m \leq n^{\frac{\omega+1}{2}}, \\ n^{2+o(1)} & \textit{if } m \leq n^{1+\frac{\alpha}{2}} \end{cases}\right).$$

Notice that the value of r in the statement of Theorem 2.5 is irrelevant since our sparse representation (and the fact that a common all-zero column of A and B^T can be discarded without affecting the product) implies that $r = O(m)$.

Finally, a word about Boolean matrix multiplication. Although not properly a ring, all of the results in this section also apply to Boolean matrix multiplication. This is easy to see; simply perform the operations over Z_{n+1}. A non-zero value is interpreted as 1. As each algebraic operation in Z_n costs only $\Theta(\log n)$ time it is not surprising that some researchers actually define ω as the exponent of Boolean matrix multiplication.

3 Maximum Subset Matching

Let $G = (V, E)$ be an undirected graph with $V = \{1, \ldots, n\}$. With each edge $e \in E$ we associate a variable x_e. Define the *skew adjacency matrix* (also known as the Tutte matrix) $A_t(G) = (a_{ij})$ by

$$a_{ij} = \begin{cases} +x_e, & \text{if } e = ij,\ i < j \text{ and } (i,j) \in E; \\ -x_e, & \text{if } e = ij,\ i > j \text{ and } (i,j) \in E; \\ 0, & \text{otherwise.} \end{cases}$$

Tutte [18] showed that $A_t(G)$ is non-singular if and only if G has a perfect matching. This was generalized later by Lovász [12] who proved that:

Lemma 3.1 *The size of a maximum matching in G is $\frac{1}{2}rank(A_t(G))$.*

This fact, together with some additional non-trivial ideas, leads to an $O(n^\omega)$ time randomized algorithm for deciding whether a graph has a perfect matching, and, much later, to $O(n^\omega)$ time randomized algorithms that actually *find* a maximum matching [14, 9].

Our first lemma is a generalization of Lemma 3.1. For a graph $G = (V, E)$ and a subset $S \subset V$, let $A_t(S, G)$ denote the sub-matrix of $A_t(G)$ obtained by taking only the rows corresponding to vertices of S. Thus, $A_t(S, G)$ is an $s \times n$ matrix where $s = |S|$ and $n = |V|$. Recalling the definition of $f(S)$ from the introduction we prove:

Lemma 3.2

$$f(S) = rank(A_t(S, G)).$$

Proof. We first prove that $f(S) \leq rank(A_t(S, G))$. Consider the case where $s - f(S)$ is even (the odd case is proved analogously). We may assume that $n - s$ is even since otherwise, we can alway add an additional isolated vertex to $V \setminus S$ without affecting the rank of $A_t(S, G)$ nor the value $f(S)$. We add edges to G so that $V \setminus S$ induces a complete graph, and denote the new graph by G'.

Clearly, the cardinality of the maximum matching of G' is $(n-s+f(S))/2$. Thus, by Lemma 3.1, $rank(A_t(G')) = n-s+f(S)$. In particular, we must have that

$$rank(A_t(S,G')) \geq rank(A_t(G')) - (n-s) = f(S).$$

But $A_t(S,G')$ and $A_t(S,G)$ are the same matrix, hence $rank(A_t(S,G)) \geq f(S)$.

We next prove that $f(S) \geq rank(A_t(S,G))$. Suppose that $rank(A_t(S,G)) = r$. We have to show that at least r vertices of S can be saturated by a matching. As the rank is r there is an $r \times r$ sub-matrix of $A_t(S,G))$, call it B, which is nonsingular. Let the rows of B correspond to S' (notice that $S' \subset S$), and the columns to U, which is some set of vertices, possibly intersecting S'. It suffices to show that there is a matching covering the vertices in S'. In the expansion of the determinant of B we get $r!$ products (with signs). Each product corresponds to an oriented subgraph of G obtained by orienting, for each x_{ij} in the product, the edge from i (the row) to j (the column). This gives a subgraph in which the out-degree of every vertex of S' is 1 and the indegree of every vertex of U is 1. Thus any connected component is either a directed path, all of whose vertices are in S' besides the last one which is in $U \setminus S'$, or a cycle, all of whose vertices are in S'. The crucial point now is that if there is an odd cycle in this subgraph, then the contribution of this term to the determinant is zero, as we can orient the cycle backwards and get the same term with an opposite sign (we do it only for the lexicographically first such cycle in the subgraph, to make sure this is well defined; this will pair the terms that cancel). As the determinant is nonzero, there is at least one such subgraph in which all components are either paths or even cycles, and hence there is a matching saturating all vertices in S', as needed. ■

Our algorithm computes $f(S)$ by computing, with high probability, $rank(A_t(S,G))$, which, by Lemma 3.2, amounts to the same thing. Computing the rank of a symbolic matrix (such as $A_t(S,G)$) directly is costly. Each algebraic operation is performed in a ring of multivariate polynomials of high degree, and cannot, therefore, be performed in constant (or, close to constant) time. By using a result of Zippel [23] and Schwartz [15], Lovász [12] proved the following.

Lemma 3.3 *If G is a graph with n vertices and we replace each variable of $A_t(G)$ with a random integer from $\{1,\ldots,R\}$ then the rank of the resulting matrix equals $rank(A_t(G))$ with probability at least $1-n/R$. Similarly, if B is any given sub-matrix of $A_t(G)$ then the rank of the resulting sub-matrix equals the rank of B, with probability at least $1-n/R$.*

For a complex matrix A, let A^* denote, as usual, the Hermitian transpose of A (some researchers also denote it by A^H). If A is a real matrix then $A^* = A^T$. We need to recall the following simple fact from linear algebra.

Fact 3.4 *Let A be a complex matrix, then A^*A and A have the same kernel.*

Indeed, suppose $A^*Ax = 0$, then, using the Hermitian product, $< Ax, Ax >=< A^*Ax, x >= 0$, whence $Ax = 0$. Notice that the assertion may fail for general fields, as can be seen, for instance, by the $p \times p$ matrix over F_p, all of whose entries are equal to 1.

We need the following result of Hopcroft and Bunch [3] which asserts that Gaussian Elimination of a matrix requires asymptotically the same number of algebraic operations needed for matrix multiplication. Notice that a by-product of Gaussian elimination is the matrix rank.

Lemma 3.5 *Let A be an $n \times n$ matrix over an arbitrary field. Then $rank(A)$ can be computed using $O(n^\omega)$ algebraic operations. In particular, if each field operation requires $\Theta(K)$ time then $rank(A)$ can be computed in $O(Kn^\omega)$ time.*

An important proposition which is obtained by combining Lemma 3.5, Lemma 2.5, Fact 3.4, and one additional idea, is the following:

Theorem 3.6 *Let A be an $s \times n$ matrix having at most m non-zero integer entries located in an $s \times (n-s)$ sub-matrix (the other s^2 entries may be all non-zero), and suppose that the largest absolute value of an entry is R. Then, $rank(A)$ can be computed, w.h.p., in time*

$$\tilde{O}\left(\left\{\begin{array}{ll} (\log R)ms^{\frac{\omega-1}{2}} & if\ m \geq s^{\frac{\omega+1}{2}}, \\ (\log R)s^{\omega} & if\ m \leq s^{\frac{\omega+1}{2}} \end{array}\right.\right).$$

Proof: Let A_2 be the $s \times (n-s)$ sub-matrix containing at most m non-zero entries, and let A_1 be the remaining $s \times s$ sub-matrix. Clearly,

$$B = AA^T = A_1A_1^T + A_2A_2^T.$$

We first compute $A_1A_1^T$ using $O(s^\omega)$ algebraic operations and in $O((\log R)s^\omega)$ time, as each algebraic operation requires $O(\log R)$ time. We compute $A_2A_2^T$ using Lemma 2.5. This can be done in the time stated in Lemma 2.5, multiplied by $\log R$. We have therefore computed B. Notice that B is an $s \times s$ matrix, and each entry in B has absolute value at most nR^2. Furthermore, by Fact 3.4, $rank(B) = rank(A)$. Now, suppose $rank(B) = t$. Thus, B has a $t \times t$ sub-matrix B' whose rank is t, and hence $\det(B') \neq 0$. On the other hand, by Hadamard Inequality $|det(B')| \leq (tn^2R^4)^{t/2} < (nR)^{2n}$. Since an integer x has $O(\log x)$ prime divisors, choosing a random prime $p = O((nR)^2)$ guarantees that, w.h.p., $det(B') \neq 0$ also in F_p, and, in particular, $rank(B) = t$ also in F_p. We compute $rank(B)$ using Lemma 3.5, using $O(s^\omega)$ algebraic operations, where each algebraic operation in F_p requires $O(\log n + \log r)$ time. Thus, in time $\tilde{O}((\log R)s^\omega)$. The overall running time of the algorithm is

$$\tilde{O}\left(\left\{\begin{array}{ll} (\log R)(s^\omega + ms^{\frac{\omega-1}{2}}) & \text{if } m \geq s^{\frac{\omega+1}{2}}, \\ (\log R)(s^\omega + m^{\frac{2\beta}{\beta+1}}s^{\frac{2-\alpha\beta}{\beta+1}}) & \text{if } s^{1+\frac{\alpha}{2}} \leq m \leq s^{\frac{\omega+1}{2}}, \\ (\log R)(s^\omega + s^{2+o(1)}) & \text{if } m \leq s^{1+\frac{\alpha}{2}} \end{array}\right.\right) = \tilde{O}\left(\left\{\begin{array}{ll} (\log R)ms^{\frac{\omega-1}{2}} & \text{if } m \geq s^{\frac{\omega+1}{2}}, \\ (\log R)s^\omega & \text{if } m \leq s^{\frac{\omega+1}{2}} \end{array}\right.\right).$$

■

Completing the proof of Theorem 1.1: We are given a graph $G = (V, E)$ and a subset $S \subset V$, where $|S| = s$ and there are m edges between S and $V \setminus S$. Our goal is to compute $f(S)$.

We construct the Tutte sub-matrix $A_t(S, G)$ and replace each variable with an integer from $\{1, \ldots, n^2\}$, uniformly at random. Denote the obtained integer matrix by A. By Lemma 3.3, with probability at least $1 - 1/n$, $rank(A) = rank(A_t(S, G))$. Thus, we need to compute $rank(A)$. Notice, however, that A is an $s \times n$ integer matrix with at most m non-zero entries in an $s \times (n-s)$ sub-matrix. Furthermore, the absolute value of each entry of A is at most $R = n^2$. Thus, by Theorem 3.6 we can compute $rank(A)$ in the stated running time. ■

It is important to notice that computing $rank(A)$ directly in Theorem 1.1, without computing AA^T, is costly. The fastest algorithms for computing the rank of an $s \times n$ matrix directly require the use of Gaussian elimination, and can be performed using $O(ns^{\omega-1})$ algebraic operations [3]. Gaussian elimination, however, cannot make use of the fact that the matrix is sparse (namely, in our terms, make use of m as a parameter to its running time). This can be attributed to the negative result of Yannakakis [20] who proved that controlling the number of fill-ins (entries that were originally zero and become non-zero during the elimination process) is an NP-Hard problem.

4 All Pairs Shortest Paths in graphs with an s-vertex cover

In this section we prove Theorem 1.2. Suppose $G = (V, E)$ is an undirected graph and $S \subset V$ is a vertex cover of G. The weight of each edge is an integer from $\{1, \ldots, W\}$. We denote $S = \{1, \ldots, s\}$ and $T = V \setminus S = \{s + 1, \ldots, n\}$. Our goal is to obtain the distance matrix $D = D_{n\times n}$ whose rows and columns are indexed by V, where $D(x, y)$ is the length of a shortest path connecting x and y.

For an $n^\alpha \times n^\beta$ matrix A and an $n^\beta \times n^\gamma$ matrix B, both with entries in $\{0, \ldots, K\} \cup \{\infty\}$, we define the *distance product* $C = A \star B$ to be $C(i, j) = Min_{k=1}^{n^\beta} A(i, k) + B(k, j)$. Yuval [22] observed that C can be computed in $O(Kn^{\omega(\alpha,\beta,\gamma)})$ time. The idea of his proof is to replace each entry z with $(n^\beta + 1)^z$ (and infinity with 0), compute the usual product C' of the resulting matrices A' and B' , and deduce $C(i, j)$ by considering $C'(i, j)$ as a number written in base $n^\beta + 1$. In fact, this argument can be stated more generally as follows:

Lemma 4.1 *For an $n^\alpha \times n^\beta$ matrix A and an $n^\beta \times n^\gamma$ matrix B, both with entries in $\{0, \ldots, K\} \cup \{\infty\}$, let $c(i, j, q)$ denote the number of distinct indices k for which $A(i, k) + B(k, j) = q$, Then, all the numbers $c(i, j, q)$ for $i = 1, \ldots, n^\alpha$, $j = 1, \ldots, n^\gamma$ and $q = 0, \ldots, 2K$ can be computed in $\tilde{O}(Kn^\omega(\alpha, \beta, \gamma))$ time.*

Denote by A the adjacency matrix of G where the rows are indexed by S and the columns by V. Thus, A is an $s \times n$ matrix and $A(i, j) = w(i, j)$ if $ij \in E$, $A(i, i) = 0$, and otherwise $A(i, j) = \infty$, for $i = 1, \ldots, s$ and $j = 1, \ldots, n$. We first compute the distance product $B = A \star A^T$. This can be done in $\tilde{O}(Wn^{\omega(\mu,1,\mu)})$ time where $s = n^\mu$, using Lemma 4.1. We consider B as the adjacency matrix of a weighted undirected graph G' whose vertex set is S. Notice that the weight of each edge of G' is between 1 and $2W$. The weight $w'(i, j)$ corresponds to a shortest path connecting i with j in G, among all paths with at most two edges.

We solve the APSP problem in G'. This can be done in $\tilde{O}(Ws^\omega)$ time using the algorithm of Shoshan and Zwick [17]. Denote the output distance matrix by D'. Clearly, $D'(i, j) = D(i, j)$ for $i = 1, \ldots, s$ and $j = 1, \ldots, s$. Indeed, in a shortest path from i to j in G we can short-circuit any two consecutive edges (i_1, i_2, i_3) where $i_2 \in T$ with the direct edge (i_1, i_3) which is an edge of G', without increasing the length of the path.

We now remain with the problem of computing $D(i, j)$ where at least one of i or j is in T. By symmetry we shall assume that $i < j$. Consider first the case $i \in S$ and $j \in T$. In the beginning of the algorithm we can choose (in linear time), for each $j \in T$, an arbitrary neighbor $f(j) \in S$. We have already computed $D(i, f(j))$; hence suppose that $D(i, f(j)) = \ell$. As our graph is undirected we have that if $\ell = \infty$ then also $D(i, j) = \infty$. Otherwise, for each neighbor v of j,

$$|D(i, v) - \ell| \leq 2W.$$

Let x_h denote the number of neighbors v of j with $D(i, v) + w(v, j) = \ell - 2W + h$, for $h = 1, \ldots, 5W$. Clearly, if we can determine all the x_h then, if h' is the smallest index for which $x_{h'} > 0$ then $D(i, j) = \ell - 2W + h$.

We propose two methods for computing the x_h's. The first method is suitable (in our setting) when W is constant, but is presented here since it is also applicable in situations where the set of possible distances is *not consecutive* and not necessarily constant, thus we believe it may find other applications. Let $D^{(k)}$ be the matrix obtained from D' by replacing each entry z with z^k. We shall demonstrate the method in case $W = 1$ (the unweighted case). The generalization is not difficult.

Consider the regular integer products $C_k = A^T D^{(k)}$ for $k = 0, \ldots, 4$ (for the sake of accuracy, we replace infinities with zeros when performing the integer product). Thus, $C_0(j,i)$ is just the number of neighbors of j that can reach i. Namely, $C_0(j,i) = x_1 + x_2 + x_3 + x_4 + x_5$. Similarly, $C_k(j,i)$ is just the sum of the distances from each of these neighbors to i, each distance taken to the k'th power. Namely,

$$C_k(j,i) = (\ell-2)^k x_1 + (\ell-1)^k x_2 + \ell^k x_3 + (\ell+1)^k x_4 + (\ell+2)^k x_5.$$

Considering the x_1, x_2, x_3, x_4, x_5 as unknown variables, we have a system of five linear equations whose coefficient matrix is just the 5×5 Vandermonde matrix with generators $\ell-2, \ell-1, \ell, \ell+1, \ell+2$. It is well known that this system has a unique solution (all of our generators are distinct, and even consecutive). This Vandermonde method becomes more inefficient if W is not constant, but each linear system can be solved in $O(W^2)$ time.

The second method uses truncated distance matrices. Let $\hat{D}'$ be the matrix obtained from D' by replacing each entry z with $z \bmod 5W$. We apply Lemma 4.1 to the product $A^T \star \hat{D}'$, and obtain, in particular, the values $c(i,j,q)$ for $q = 0, \ldots, 6W-1$. Notice that each finite entry in A is between 0 and W and each finite entry in $\hat{D}'$ is between 0 and $5W-1$. We claim that we can determine the x_h from these values. Indeed, each contribution to $c(i,j,q)$ is due to a neighbor v so that $(D(i,v) \bmod 5W) + w(v,j) = q$. But this determines $D(i,v) + w(v,j)$, and hence a corresponding x_h, uniquely, since we know that all the possible $D(i,v)$ are between $\ell - 2W$ and $\ell + 2W$, thus, in a $4W$ interval, and we know that $1 \leq w(v,j) \leq W$, which is another interval of length W.

We have thus shown how to compute all of the distances $D(i,j)$ for $i \in S$ and $j \in T$ precisely. In particular, we have determined the first s rows of D in $\tilde{O}(Wn^{\omega(1,\mu,\mu)})$ time. Denote these first s rows of D by D''. To determine the distances $D(i,j)$ where $i \in T$ and $j \in T$, we simply repeat the above procedure using the truncated distance matrix $\hat{D}''$ (instead of $\hat{D}'$) and applying Lemma 4.1 to the product $A^T \star \hat{D}''$. The running time now is $\tilde{O}(Wn^{\omega(1,\mu,1)})$.

We have shown how to correctly compute the final distance matrix D. What remains is to determine the running time of the algorithm. As noted in the introduction, if $vc(G) = n^\mu$ then finding a vertex cover S with $s \leq 2n^\mu$ vertices can be easily done in $O(n^2)$ time. Overall, the algorithm consists of three distance products (three applications of Lemma 4.1), and one application of the algorithm of Shoshan and Zwick. The most time consuming operation is $\tilde{O}(Wn^{\omega(1,\mu,1)})$, and hence the result follows. ■

5 Concluding remarks

We presented two new algorithms for two fundamental problems in algorithmic combinatorics. Both of these algorithms are based on rectangular matrix multiplication, but each is combined with different tools from combinatorics and linear algebra. It is plausible that for Maximum Subset Matching this is not the end of the road. The possibility of a faster algorithm remains (whether using fast matrix multiplication or not). This, however, is not the case for our APSP result. Namely, no algorithm that is expressed in terms of n and $vc(G)$ can outperform the algorithm of Theorem 1.2. Assuming $vc(G) = s$, no algorithm can perform faster than the time needed to multiply two Boolean matrices of orders $n \times s$ and $s \times n$.

To see this, consider the following simple reduction. Let A be an $n \times s$ Boolean matrix and let B be an $s \times n$ Boolean matrix. Create a graph with $2n + s$ vertices, consisting of sets $X = \{x_1, \ldots, x_n\}$,

$Y = \{y_1, \dots, y_s\}$, $Z = \{z_1, \dots, z_n\}$. There is an edge from $x_i \in X$ to $y_j \in Y$ if an only if $A(i,j) = 1$. Similarly, there is an edge from $y_i \in Y$ to $z_j \in Z$ if an only if $B(i,j) = 1$. Notice that Y is a vertex cover of the created graph. Clearly, the shortest path from x_i to z_j has length 2 if and only if in the product $C = AB$ we have $C(i,j) = 1$.

Another interesting open problem is whether Theorem 3.6 can be extended to finite fields. That is, given an $s \times n$ matrix A over F_p (p constant) with at most m non-zero entries, is there an algorithm that computes $rank(A)$ in $O(ms^{(\omega-1)/2})$ time if $m \geq s^{(\omega+1)/2}$ and in $O(s^{\omega})$ time if $m \leq s^{(\omega+1)/2}$?

References

[1] N. Alon and M. Naor. Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16:434–449, 1996.

[2] N. Blum. A New Approach to Maximum Matching in General Graphs. *Proceedings of the* 47*th* *International Conference on Automata, Languages, and Programming (ICALP)*, 586–597, 1990.

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

[4] D. Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13:42–49, 1997.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[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. *Journal of the ACM*, 38(4):815–853, 1991.

[8] Z. Galil and O. Margalit. All Pairs Shortest Paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54:243–254, 1997.

[9] N. Harvey. Algebraic Structures and Algorithms for Matching and Matroid Problems. *Proceedings of the* 47*th* *IEEE Symposium on Foundations of Computer Science (FOCS)*, Berkeley, CA, October 2006.

[10] X. Huang and V.Y. Pan. Fast rectangular matrix multiplications and applications. *Journal of Complexity*, 14:257–299, 1998.

[11] H. Kaplan, M. Sharir, and E. Verbin. Colored intersection searching via sparse rectangular matrix multiplication. *Proceedings of the* 22*nd* *ACM Symposium on Computational Geometry (SOCG)*, 52–60, 2006.

[12] L. Lovász. On determinants, matchings, and random algorithms. *In: Fundamentals of computation theory, Vol. 2 pages 565–574*, Akademie-Verlag, Berlin, 1979.

[13] S. Micali and V.V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. *Proceedings of the* 21^{st} *IEEE Symposium on Foundations of Computer Science (FOCS)*, 17–27, 1980.

[14] M. Mucha and P. Sankowski. Maximum Matchings via Gaussian Elimination. *Proceedings of the* 45^{th} *IEEE Symposium on Foundations of Computer Science (FOCS)*, 248–255, 2004.

[15] J.T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM*, 27(4):701–717, 1980.

[16] R. Seidel. On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[17] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. *Proceedings of the* 40^{th} *Symposium of Foundations of Computer Science (FOCS)*, 605–614, 1999.

[18] W.T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 22:107–111, 1947.

[19] V.V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. *Combinatorica*, 14(1):71–109, 1994.

[20] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

[21] R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1:2–13, 2005.

[22] G. Yuval. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Information Processing Letters*, 4:155–156, 1976.

[23] R. Zippel. Probabilistic algorithms for sparse polynomials. *Proceedings of Symbolic and Algebraic Computation (EUROSAM)*, Lecture Notes in Computer Science 72:216–226, 1979.