On minimum witnesses for Boolean matrix multiplication

Keren Cohen [*] Raphael Yuster [†]

Abstract

Minimum witnesses for Boolean matrix multiplication play an important role in several graph algorithms. For two Boolean matrices A and B of order n, with one of the matrices having at most m nonzero entries, the fastest known algorithms for computing the minimum witnesses of their product run in either $O(n^{2.575})$ time or in $O(n^2 + mn \log(n^2/m)/\log^2 n)$ time. We present a new algorithm for this problem. Our algorithm runs either in time

$$\tilde{O}(n^{\frac{3}{4-\omega}} m^{1-\frac{1}{4-\omega}})$$

where $\omega < 2.376$ is the matrix multiplication exponent, or, if fast rectangular matrix multiplication is used, in time

$$O(n^{1.939} m^{0.318}) .$$

In particular, if $\omega - 1 < \alpha < 2$ where $m = n^\alpha$, the new algorithm is faster than both of the aforementioned algorithms.

Keywords: Boolean matrix multiplication, minimum witness

1 Introduction

Let A and B be two Boolean $n \times n$ matrices. A *witness matrix* for the Boolean product $C = AB$ is an $n \times n$ matrix W with the following properties. If $C[i,j] = 0$, then $W[i,j] = 0$. Otherwise, $W[i,j]$ is equal to *some* index k such that $A[i,k] = B[k,j] = 1$. We say that W is a *minimum witness matrix* if $W[i,j]$ is the least index k such that $A[i,k] = B[k,j] = 1$. A maximum witness matrix is defined analogously. Notice that minimum and maximum witness matrices are unique.

Witness matrices and minimum/maximum witness matrices arise naturally in various algorithmic settings. For example, witness matrices can be used to compute shortest paths and fixed sized graphs; see, e.g., [1, 7, 10]. Minimum/maximum witness matrices are useful for computing lowest common ancestors in directed acyclic graphs [5], for computing bottleneck paths [11], and for computing minimum and maximum weight triangles and other fixed graphs [13].

Clearly, witness matrices, as well as minimum and maximum witness matrices, can be computed naively in $O(n^3)$ time as a by-product of naive matrix multiplication that computes the Boolean product of each row and column pair separately. If one of the matrices has $m < n^2$ nonzero entries, then naive matrix multiplication, and hence witness computation, requires $O(nm)$ time (folklore).

[*]Department of Mathematics, University of Haifa, Haifa 31905, Israel.

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

However, it is well-known that matrices can be multiplied faster. Coppersmith and Winograd [4], improving earlier results beginning with the seminal result of Strassen [12], have shown how to multiply two $n \times n$ matrices over an arbitrary ring using $O(n^{2.376})$ arithmetic operations. This easily implies that two Boolean matrices can be multiplied in $O(n^{2.376})$ time. Let ω denote the infimum over all exponents such that the Boolean product of two $n \times n$ matrices can be computed in $O(n^{\omega})$ time. The result of [4] implies that $\omega < 2.376$ and clearly $\omega \geq 2$. We note that recently Vassilevska Williams [14], following a result of Stothers (in his Ph.D. Thesis), showed that $\omega < 2.373$[1].

Unfortunately, algorithms such as [4, 12] that compute matrix products non-naively, do not directly provide witnesses for the resulting product. A simple randomized algorithm for computing a witness matrix for a Boolean product in essentially $O(n^{\omega})$ time is given by Seidel [10]. Alon and Naor [1] gave a deterministic $O(n^{\omega})$ time algorithm for the problem. An alternative, slightly slower, deterministic algorithm was given by Galil and Margalit [7]. These algorithms are faster than the naive $O(nm)$ algorithm whenever $n^{\omega-1} = o(m)$. However, computing a minimum witness matrix seems to be a more difficult problem[2]. Kowaluk and Lingas [9] gave an algorithm that runs in $O(n^{2+1/(4-\omega)}) \leq O(n^{2.616})$. This was slightly improved in [5] to $O(n^{2.575})$ using the same algorithm but utilizing improved bounds for *rectangular* matrix multiplication. This algorithm is faster than the naive $O(nm)$ time algorithm whenever $m > n^{1.575}$. We mention here that an interesting combinatorial algorithm that slightly improves upon the naive $O(nm)$ algorithm is given by Blelloch et al. [2]. It runs in $O(n^2 + mn\log(n^2/m)/\log^2 n)$ time.

Improving upon the algorithms from [2, 5] is an open problem mentioned in both of these papers. In the present paper we design an algorithm that improves upon both of them whenever $\omega - 1 < \alpha < 2$ where $m = n^{\alpha}$.

Theorem 1.1 *Given two Boolean matrices of order n where one of them has at most m nonzero entries, there is an algorithm that computes the minimum witness matrix of their product in time* [3]

$$\tilde{O}(n^{\frac{3}{4-\omega}} m^{1-\frac{1}{4-\omega}})$$

or, if fast rectangular matrix multiplication is used, in time

$$O(n^{1.939} m^{0.318}) .$$

The algorithm of Theorem 1.1 is faster than the algorithms of [5, 9] whenever $m = o(n^2)$, and matches their complexity whenever $m = \Theta(n^2)$. It also improves upon the naive algorithm as well as its slight improvement in [2] whenever $m > \tilde{\Omega}(n^{\omega-1})$. In particular, if $\omega = 2$, the new algorithm is preferable to both of these algorithm for any density $m = n^{\alpha}$ where $1 < \alpha < 2$. It is interesting to note that algorithms using fast matrix multiplication usually have a phase transition at a certain input density where they become the algorithms of choice, and from that point, the complexity remains the same regardless of the growth in density. On the other hand, the growth in the running time of our algorithm is smooth with respect to the growth in density.

The main ingredient of the proof of Theorem 1.1 is a procedure for iterative partitioning of the Boolean matrices that are to be multiplied. The algorithms of [5, 9] correspond to the initial

[1]We use the 2.376 bound in this paper in order to make it easy to compare with related results; one can easily plug in the value 2.373 whenever ω is used to obtain slightly improved exponents.

[2]Computing maximum witnesses is computationally equivalent to computing minimum witnesses, so we only consider minimum witnesses.

[3]Throughout this paper, $\tilde{O}(f(n))$ stands for $O(f(n)\log^c n)$ for some $c > 0$.

partitioning of our algorithm. We show that we can refine the partitions further and further, where each level of partitions reveals more and more minimum witnesses, causing the next partitions to become sparser, compensating for the increase in the number of parts. The final partitioning reveals the yet remaining witnesses. We note that the algorithm of Galil and Margalit mentioned earlier [7] has a similar flavor, although the two algorithms are completely different. It is obtained from a sequence of algorithms, the first is the naive cubic algorithm, and each subsequent algorithm computes a fast Boolean matrix multiplication and uses it to partition the problem into several smaller problems that are solved by calling the previous algorithm in the sequence.

The next section presents a warm-up for our algorithm, where only two levels of partitions are used. Already this approach yields improved running times, but this is still not optimal. In Section 3 we generalize our method and present the full algorithm, proving Theorem 1.1. Section 4 contains some concluding remarks and examples of applications of the main result.

2 A warm-up algorithm: two levels of partitions

In this section we construct an algorithm that is based on two level partitioning of the input matrices. This algorithm and its analysis serve as a basis for the general algorithm.

Let A and B be two $n \times n$ Boolean matrices, and assume that A has at most m nonzero entries. The case where B has at most m nonzero entries is analogous, as can be seen from the fact that the minimum witness matrix of AB is the transpose of the minimum witness matrix of $B^T A^T$. We may also assume that $m \geq n^{\omega-1}$ since if $m < n^{\omega-1}$ the naive algorithm achieves better running time than our algorithm. Our goal is to compute the minimum witness matrix W of the Boolean product $C = AB$. We first describe the algorithm and then analyze its complexity.

2.1 The algorithm

Let $0 < \delta < \gamma < 1$ be two parameters whose values will be chosen later after optimization.

As in [5], we partition A into $r = n^{1-\gamma}$ rectangular sub-matrices of dimensions $n \times n^{\gamma}$ each[4]. We denote the rectangular matrices by $A_1, ..., A_r$, where A_k consists of the columns of A in the range $(k-1)n^{\gamma}+1, \ldots, kn^{\gamma}$. Similarly, we partition B into r sub-matrices of dimensions $n^{\gamma} \times n$ each, denoted by $B_1, ..., B_r$, where B_k consists of all the rows of B in the range $(k-1)n^{\gamma}+1, \ldots, kn^{\gamma}$.

Using fast rectangular matrix multiplication, we compute the Boolean products $C_k = A_k B_k$ for $k = 1, \ldots, r$. Observe that $C = \vee_{k=1}^{r} C_k$.

For each ordered pair of indices (i, j), we locate the least index $k = k(i, j)$ such that $C_k[i, j] = 1$. If no such index exists then we set $W[i, j] = 0$, as $C[i, j] = 0$. Otherwise, the minimum witness can be located by examining the product $A_k B_k = C_k$. Specifically, if ℓ is the least index such that $A_k[i, \ell] = 1$ and $B_k[\ell, j] = 1$, then $W[i, j] = \ell + (k-1)n^{\gamma}$. The easiest way to locate ℓ, as in [5], is to just scan for ℓ sequentially. This requires ℓ operations, but it may be that $\ell = n^{\gamma}$, so this is not very efficient.

Instead, we distinguish between two cases. Let $d(k, i)$ be the number of nonzero entries in the i'th row of A_k. We consider a representation of A_k as a sparse matrix, with the nonzero elements of each row occupying a linked list. Hence, the i'th list of A_k has length $d(k, i)$. If $d(k, i) \leq n^{\delta}$ then ℓ will be located sequentially. This requires only $O(d(k, i)) \leq O(n^{\delta})$ operations.

[4]Rounding issues are ignored in this paper as they have no effect on the asymptotic running times.

It remains to show how to locate ℓ for pairs (i,j) for which $d(k,i) > n^\delta$ where $k = k(i,j)$. Let A'_k denote the matrix obtained from A_k by removing all the rows with at most n^δ nonzero entries and let n_k denote the number of rows of A'_k. We partition A'_k into $s = n^{\gamma-\delta}$ rectangular sub-matrices of dimensions $n_k \times n^\delta$ each, denoted by $A'_{k,1}, \ldots, A'_{k,s}$. We partition B_k into s rectangular sub-matrices of dimensions $n^\delta \times n$ each, denoted by $B_{k,1}, \ldots, B_{k,s}$. Using rectangular matrix multiplication, we compute the Boolean products $C_{k,t} = A'_{k,t}B_{k,t}$ for $k = 1, \ldots, r$ and $t = 1, \ldots, s$. Observe that $C_k = \vee_{t=1}^{s} C_{k,t}$.

For each ordered pair indices (i,j) such that $k = k(i,j)$ and $d(k,i) > n^\delta$, we locate the smallest index $t = t(i,j)$ such that $C_{k,t}[i,j] = 1$. Locating t can be done sequentially using $O(s)$ operations but there is a faster way using binary search which we now describe. Assume for simplicity that $s = 2^p$ is a power of 2 (again, this assumption does not affect the asymptotic running time). Consider the matrices $C_{k,1} \ldots, C_{k,s}$. Each is an $n_k \times n$ Boolean matrix. For every $q = 0, \ldots, p-1$ and for every $d = 0, \ldots, 2^{p-q} - 1$ let

$$C^*_{k,q,d} = \vee_{\ell=2^q d+1}^{2^q(d+1)} C_{k,\ell} \ .$$

Binary searching, starting from $q = p-1$ downwards we can locate $t = t(i,j)$ in $O(\log s)$ time. The overhead (both in time and space) in computing the Boolean matrices $C^*_{k,q,d}$ for a given k is just a factor of $O(\log s)$ over the cumulative size of all the $C_{k,t}$. Once we have computed t, we know that the minimum witness can be located by examining the product $A'_{k,t}B_{k,t} = C_{k,t}$. Hence, it can now be located sequentially using $O(n^\delta)$ operations, as this is the column dimension of $A'_{k,t}$.

2.2 Running time

We start by recalling a few facts on fast (possibly rectangular) matrix multiplication. These facts will also be useful for the analysis of the main result in the next section.

Let $T(\ell, m, n)$ be the time needed to compute the Boolean product of an $\ell \times m$ matrix by an $m \times n$ matrix. Let $\omega(r,s,t)$ be the infimum of all the exponents ω' for which $T(n^r, n^s, n^t) = O(n^{\omega'})$. We let $\omega = \omega(1,1,1)$ be the exponent of square matrix multiplication. Recall that by [4] we have $\omega < 2.376$. Another useful fact is that $\omega(r,s,t) = \omega(r,t,s) = \omega(s,r,t) = \omega(s,t,r) = \omega(t,r,s) = \omega(t,s,r)$ (see [8]). By performing rectangular matrix multiplication using square block matrices one obtains the following straightforward bounds:

Lemma 2.1

$$\begin{aligned} \omega(1,r,1) = \omega(r,1,1) &\le 2 + (\omega-2)r \qquad \textit{for } 0 \le r \le 1 \ , \\ \omega(s,r,1) = \omega(r,s,1) &\le s + 1 + (\omega-2)r \qquad \textit{for } 0 \le r \le s \le 1 \ . \end{aligned}$$

Coppersmith [3] proved that if α is the supremum over all constants r for which $\omega(1,r,1) = 2$ then $\alpha > 0.294$. The following lemma is obtained by decomposing a given matrix product into smaller products. (See, e.g., Huang and Pan [8].)

Lemma 2.2

$$\omega(1,r,1) = \omega(r,1,1) \le \begin{cases} 2 & \textit{if } 0 \le r \le \alpha, \\ 2 + \frac{\omega-2}{1-\alpha}(r-\alpha) & \textit{if } \alpha \le r \le 1. \end{cases}$$

$$\omega(s,r,1) = \omega(r,s,1) \le \begin{cases} 1+s & \textit{if } 0 \le r \le \alpha s \le \alpha, \\ 1 + s + \frac{\omega-2}{1-\alpha}(r-\alpha s) & \textit{if } \alpha s \le r \le s \le 1. \end{cases}$$

Observe that if we use either the upper bound from Lemma 2.1 or the upper bound from Lemma 2.2 as our estimate, we can assume that for given $0 \leq \delta \leq 1$, the function $f(\beta) = \omega(\beta, \delta, 1)$ is *convex* for $0 \leq \beta \leq 1$.

We now turn to the analysis of the algorithm. The first stage of our algorithm computes $C_1, \ldots, C_r$ in time

$$O(rn^{\omega(1,\gamma,1)}) = O(n^{1-\gamma+\omega(1,\gamma,1)}) \ . \tag{1}$$

Constructing sparse representations of $A_1, \ldots, A_r$ using row lists can be done sequentially in $O(n^2)$ time. At the same time, we can also compute $d(k, i)$ for all $k = 1, \ldots, r$ and for all $i = 1, \ldots, n$. Computing $k(i, j)$ for each pair of indices i, j can be done naively in time

$$O(rn^2) = O(n^{3-\gamma}) \ . \tag{2}$$

For a pair (i, j) such that $k = k(i, j)$ and $d(k, i) < n^\delta$, computing the minimum witness takes $O(n^\delta)$ time. Hence, for all such pairs the time required is

$$O(n^{2+\delta}) \ . \tag{3}$$

Constructing the truncated matrices $A'_1, \ldots, A'_r$ takes at most $O(n^2)$ time. Recall that n_k denotes the number of rows of A'_k. Let β_k be such that $n^{\beta_k} = n_k$. Computing the products $C_{k,t} = A'_{k,t}B_{k,t}$ for $k = 1, \ldots, r$ and $t = 1, \ldots, s$ can therefore be performed in time

$$O(\sum_{k=1}^{r} sn^{\omega(\beta_k,\delta,1)}) = O(n^{\gamma-\delta} \sum_{k=1}^{r} n^{\omega(\beta_k,\delta,1)}) \ .$$

In order to estimate this quantity we use the fact that each row of each A'_r contains at least n^δ nonzero entries. Since the overall number of nonzero entries in all of the A'_r together is at most m, we have

$$\sum_{k=1}^{r} n_k = \sum_{k=1}^{r} n^{\beta_k} \leq \frac{m}{n^\delta} \ .$$

By convexity of the estimates for $\omega(\beta_k, \delta, 1)$, we have that $\sum_{k=1}^{r} n^{\omega(\beta_k,\delta,1)}$ subject to $\sum_{k=1}^{r} n^{\beta_k} \leq \frac{m}{n^\delta}$ is maximized when $m/n^{\delta+1}$ of the β_k are 1 and all the remaining β_k equal zero. Hence,

$$O(n^{\gamma-\delta} \sum_{k=1}^{r} n^{\omega(\beta_k,\delta,1)}) \leq O(n^{\gamma-\delta} \frac{m}{n^{\delta+1}} n^{\omega(1,\delta,1)}) = O(mn^{\gamma-2\delta-1+\omega(1,\delta,1)}) \ . \tag{4}$$

For each ordered pair indices (i, j) such that $k = k(i, j)$ and $d(k, i) > n^\delta$, computing $t = t(i, j)$ takes $O(\log s)$ time, so for all such pairs, the required time is only $O(n^2 \log n)$. However, we must first compute all the Boolean matrices $C^*_{k,q,d}$. For a given k, the time and space for constructing them is $O(n_k ns \log n)$ because the size of each $C_{k,t}$ is $n_k \times n$ and there are s of them. Summing this up for all k the required time is

$$O(ns \log n \sum_{k=1}^{r} n_k) = O(mn^{1+\gamma-2\delta} \log n) \ . \tag{5}$$

Once $t(i, j)$ is computed, the minimum witness of the pair can be found in $O(n^\delta)$ time, so for all such pairs, in $O(n^{2+\delta})$ time, as in (3).

It now remains to optimize the choices of γ and δ so that the maximum of (1),(2),(3),(4),(5) is minimized.

Let us first observe that (2) is redundant as it never exceeds (1). Indeed, comparing both exponents we see that

$$1-\gamma+\omega(1,\gamma,1) \geq 3-\gamma$$

which follows from the the fact that $\omega(1,\gamma,1) \geq 2$. Likewise, (5) is redundant (up to the logarithmic factor) as it never exceeds (4). Comparing exponents we see that

$$\gamma-2\delta-1+\omega(1,\delta,1) \geq 1+\gamma-2\delta$$

which follows from the the fact that $\omega(1,\delta,1) \geq 2$. It remains to optimize with respect to (1),(3),(4).

It will be convenient to denote $m = n^x$ where $\omega - 1 \leq x \leq 2$ and work only with exponents. Equating (1) and (3) we obtain that

$$\delta = \omega(1,\gamma,1) - \gamma - 1\ . \tag{6}$$

As (1) is a decreasing function of γ and (4) is an increasing function of γ, their minimum is attained when equal. Equating (1) and (4) using the bound from Lemma 2.1, we obtain

$$1-\gamma+2-2\gamma+\omega\gamma = x+\gamma-2\delta-1+2-2\delta+\omega\delta\ .$$

Plugging in (6) this translates to

$$3-(3-\omega)\gamma = x+\gamma+1-(4-\omega)(1-(3-\omega)\gamma)\ .$$

This implies that the values of γ and δ that equate (1),(3),(4) are

$$\gamma = \frac{6-\omega-x}{(4-\omega)^2}\ , \qquad \delta = \frac{\omega-2}{(4-\omega)^2} + \frac{(3-\omega)x}{(4-\omega)^2}\ .$$

The overall running time of the algorithm is therefore

$$\tilde{O}(n^{2+\delta}) = \tilde{O}(n^{2+\frac{\omega-2}{(4-\omega)^2}} m^{\frac{(3-\omega)}{(4-\omega)^2}}) = O(n^{2.143}m^{0.237}) \tag{7}$$

where the r.h.s. uses $\omega < 2.376$.

Finally, we repeat our optimization for (1),(3),(4) using fast rectangular matrix multiplication and the bound from Lemma 2.2 with $\alpha = 0.294$. In this case we obtain a slightly improved running time of $O(n^{2.112}m^{0.232})$.

3 Proof of the main result

The algorithm in the previous is based on refining the original partition of the matrices A_k and B_k into a sub-partition of matrices $A'_{k,t}$, $B_{k,t}$. In other words, the algorithm is a two-stage refinement of the original matrices A, B. In this section we extend the algorithm to a multi-stage refinement, resulting in an algorithm with improved running time.

3.1 The algorithm

The parameters $0 < \delta < \gamma < 1$ have the same function they had in the previous section; in other words, γ controls the initial refinement and δ is the threshold for which pairs become "uninteresting" for the next refinement, as they are determined using the current refinement.

Let $h \geq 1$ be an integer whose purpose is to control the number of stages of refinements. Set

$$\epsilon = \frac{\gamma - \delta}{h} , \qquad \gamma_z = \gamma - \epsilon z \text{ for } z = 0, \ldots, h .$$

Thus, $\gamma_0 = \gamma$ and $\gamma_h = \delta$.

We now describe a generic stage z of the algorithm starting with $z = 0$, and show how to proceed from the current stage to the next one. At each stage we will have a set of *relevant* matrix pairs denoted by $\mathcal{M}_z = \{(A_1^{(z)}, B_1^{(z)}), \ldots, (A_{r_z}^{(z)}, B_{r_z}^{(z)})\}$ where $r_z = n^{1-\gamma_z}$. The dimensions of each $B_t^{(z)}$ will be $n^{\gamma_z} \times n$, and the dimension of each $A_t^{(z)}$ will be $n_{z,t} \times n^{\gamma_z}$, where $n_{z,t} \leq n$. In fact, it will always be the case that the union of all the rows of all the $B_t^{(z)}$ is B, and each $B_t^{(z)}$ represents n^{γ_z} consecutive rows of B. On the other hand, each $A_t^{(z)}$ represents n^{γ_z} consecutive columns of A, with some $n - n_{z,t}$ rows removed. In fact, it might be the case that some $n_{z,t} = 0$, namely some $A_t^{(z)}$ might be empty matrices.

As a special case, notice $\mathcal{M}_0$ is just the set of pairs of matrices (A_k, B_k) for $k = 1, \ldots, n^{1-\gamma}$ as in the initial refinement of the algorithm of the previous section. Also notice that for the case $h = 1$ (this corresponds to the algorithm of the previous section), $\mathcal{M}_1$ is the set of pairs of matrices $(A'_{k,t}, B_{k,t})$ of the previous section.

Another object that we hold in each stage is the set of relevant index pairs $\mathcal{W}_z$ for which the minimum witness has not yet been determined. For each $(i,j) \in \mathcal{W}_z$ we also have an associated interval $I(i,j,z)$ of indices so that the (yet undetermined) minimum witness is guaranteed to be determined by considering only the products of the pairs $(A_t^{(z)}, B_t^{(z)})$ for $t \in I(i,j,z)$. It will always be the case that $|I(i,j,z)| \leq n^{\gamma_{z-1} - \gamma_z}$ where for completeness we define $\gamma_{-1} = 1$.

We initially set $\mathcal{W}_0 = \{(i,j) \mid 1 \leq i \leq n \ , \ 1 \leq j \leq n\}$ and $I(i,j,z) = \{1, \ldots, r_0\}$. This corresponds to that fact that we have still not computed any witness and all the products of all the pairs in $\mathcal{M}_0$ must be considered in order to deduce the minimum witness of (i,j).

We begin stage z by computing all the Boolean products of all pairs in $\mathcal{M}_z$. This results in a set of matrices $\mathcal{C}_z = \{C_1^{(z)}, \ldots, C_{r_z}^{(z)}\}$ where $C_t^{(z)} = A_t^{(z)} B_t^{(z)}$.

For each pair $(i,j) \in \mathcal{W}_z$, we locate the smallest index $t = t(i,j,z) \in I(i,j,z)$ such that $C_t^{(z)}[i,j] = 1$. This can be performed sequentially using at most $|I(i,j,z)|$ operations. Observe that in the case $z = 0$ it may be that no such index exists, indicating that (i,j) has no witness. As pairs with no witnesses are determined already in stage 0, we have that for all $z \geq 1$, a witness for $(i,j) \in \mathcal{W}_z$ must exist.

Once we have located $t = t(i,j,z)$, we know that the minimum witness can be located by examining the product of row i of $A_t^{(z)}$ with column j of $B_t^{(z)}$. We cannot afford to search for it sequentially as the column dimension of $A_t^{(z)}$ is n^{γ_z} which may be too large. Instead, we use the same argument as in the previous section. We construct a representation of $A_t^{(z)}$ as a sparse matrix. If row i has less than n^δ nonzero entries then the minimum witness of (i,j) can be located using $O(n^\delta)$ operations. Otherwise, we pass (i,j) to the next stage of the algorithm, namely we place (i,j) in $\mathcal{W}_{z+1}$.

In order to handle pairs (i,j) with $t = t(i,j,z)$ for which row i of $A_t^{(z)}$ has at least n^δ nonzero entries, we further partition the matrices $A_t^{(z)}$ and $B_t^{(z)}$ as follows. The matrix $B_t^{(z)}$ is partitioned by splitting its rows consecutively into $n^{\gamma_z - \gamma_{z+1}}$ equal parts. This results in $n^{\gamma_z - \gamma_{z+1}}$ matrices with dimensions $n^{\gamma_{z+1}} \times n$. These matrices are carried over to the next level. Likewise, after removing the rows of $A_t^{(z)}$ that have at most n^δ nonzero entries, we partition the remaining $A_t^{(z)}$ by splitting its columns consecutively into $n^{\gamma_z - \gamma_{z+1}}$ equal parts. Observe that each of the partitioned parts of $A_t^{(z)}$ has a corresponding partitioned part of $B_t^{(z)}$. These $n^{\gamma_z - \gamma_{z+1}}$ pairs of partitions are added to $\mathcal{M}_{z+1}$. Since for each pair $(i,j) \in \mathcal{M}_{z+1}$ we have computed $t(i,j,z)$, we know that in the next level, this witness can be determined by examining the products of the pairs corresponding to the partitions of $A_t^{(z)}$ and $B_t^{(z)}$. As there are $n^{\gamma_z - \gamma_{z+1}}$ such pairs, we have that $|I(i,j,z+1)| \le n^{\gamma_z - \gamma_{z+1}}$, as required.

This completes the description of a general stage of the algorithm. Observe that in the final stage $z = h$, the column dimensions of the matrices $A_t^{(h)}$ is n^δ and hence in this stage all the witnesses remaining in $\mathcal{W}_h$ are determined, and no additional stage is required.

3.2 Running time

We compute the running time required by each stage of the algorithm.

Computing the Boolean products of all pairs in $\mathcal{M}_z$ in order to obtain the set of matrices $\mathcal{C}_z$ is just the sum of the times to compute each of these r_z products. Each product is a rectangular multiplication where the three dimensions involved are $n_{z,t}, n^{\gamma_z}, n$. Defining $\beta_{z,t}$ as $n^{\beta_{z,t}} = n_{z,t}$, the required time is therefore

$$\sum_{t=1}^{r_z} n^{\omega(\beta_{z,t},\gamma_z,1)} . \tag{8}$$

Observe that for $z = 0$, the sum in (8) becomes the same as (1), since $r_0 = n^{1-\gamma}$, $\beta_{0,t} = 1$, and $\gamma_0 = \gamma$. Hence, in the case $z = 0$ the required time is

$$O(n^{1-\gamma+\omega(1,\gamma,1)}) . \tag{9}$$

In order to estimate (8) for $z \ge 1$ we shall first estimate $\sum_{t=1}^{r_z} n_{z,t}$. Recall that each $A_t^{(z)}$ is the result of some column partitioning of some $A_s^{(z-1)}$ from which the rows with at most n^δ nonzero entries have been removed. The total number of rows in all the $A_s^{(z-1)}$ that have more than n^δ nonzero entries is, obviously, at most m/n^δ. Since each $A_s^{(z-1)}$ is split into $n^{\gamma_{z-1}-\gamma_z}$ parts, we have

$$\sum_{t=1}^{r_z} n_{z,t} = \sum_{t=1}^{r_z} n^{\beta_{z,t}} \le \frac{m}{n^\delta} n^{\gamma_{z-1}-\gamma_z} .$$

By convexity of the estimates for $\omega(\beta_{z,t},\gamma_z,1)$, we have that $\sum_{t=1}^{r_z} n^{\omega(\beta_{z,t},\gamma_z,1)}$ subject to the bound $\sum_{t=1}^{r_z} n^{\beta_{z,t}} \le \frac{m}{n^\delta} n^{\gamma_{z-1}-\gamma_z}$ is maximized when $mn^{-1-\delta+\gamma_{z-1}-\gamma_z}$ of the $\beta_{z,t}$ are 1 and all the remaining $\beta_{z,t}$ equal zero. Hence (8) is at most

$$\sum_{t=1}^{r_z} n^{\omega(\beta_{z,t},\gamma_z,1)} \le O(mn^{-1-\delta+\gamma_{z-1}-\gamma_z+\omega(1,\gamma_z,1)}) = O(mn^{-1-\delta+\epsilon+\omega(1,\gamma_z,1)}) . \tag{10}$$

For each pair $(i,j) \in \mathcal{W}_z$, locating the smallest index $t = t(i,j,z) \in I(i,j,z)$ such that $C_t^{(z)}[i,j] = 1$ is done sequentially using at most $|I(i,j,z)| \le n^{\gamma_{z-1}-\gamma_z}$ operations. The overall time required for this step is therefore

$$O(|\mathcal{W}_z|n^{\gamma_{z-1}-\gamma_z}) = \begin{cases} O(n^{3-\gamma}) & \text{if } z = 0, \\ O(n^{2+\epsilon}) & \text{if } z = 1,\ldots,h. \end{cases} \tag{11}$$

Constructing a representation of $A_t^{(z)}$ as a sparse matrix takes $O(n^2)$ time for all the $t = 1,\ldots,r_z$ together. As in the previous section, minimum witnesses for pairs (i,j) for which row i of $A_t^{(z)}$ (where $t = t(i,j,z)$) has at most n^δ nonzero entries is located using $O(n^\delta)$ operations. Otherwise, the pair is moved on to the list $\mathcal{W}_{z+1}$. The overall time for this step is therefore at most

$$O(|\mathcal{W}_z|n^\delta) = O(n^{2+\delta}) \; . \tag{12}$$

The final step of stage z is the partitioning of the matrices $B_t^{(z)}$ and $A_t^{(z)}$ (recall that for $A_t^{(z)}$ we first remove the rows with at most n^δ nonzero entries) into $n^{\gamma_z-\gamma_{z+1}}$ sub-matrices. This is a straightforward operation that takes time proportional to the overall size of these matrices which is $O(n^2)$. Likewise, determining $|I(i,j,z+1)|$ takes $O(1)$ time for each pair so at most $O(n^2)$ time overall. This final step is therefore not a bottleneck in the running time of the algorithm.

It remains to optimize γ, δ, h (which also define ϵ) so that none of (9),(10),(11),(12) exceed the running time stated in Theorem 1.1. As in the previous section, we see immediately that (11) is redundant as it never exceeds (9). So we only need to optimize using (9),(10),(12). Observe that (10) is maximized for $\gamma_z = \gamma_0 = \gamma$. Also, we will use $h = \log n$ so $\epsilon = O(1/\log n)$ making ϵ asymptotically negligible in the exponent of (10). Observe, however, that $h = \log n$ causes the logarithmic factor in the running time. It remains to determine γ and δ so that the following three exponents are equated:

$$1 - \gamma + \omega(1,\gamma,1) = 2 + \delta = x - 1 - \delta + \omega(1,\gamma,1)$$

where we define $m = n^x$ (and recall that we assume $\omega - 1 \le x \le 2$). This is optimized when $\delta = x - 2 + \gamma$ and $x - 1 = \omega(1,\gamma,1) - 2\gamma$. Using the estimate from Lemma 2.1 for $\omega(1,\gamma,1)$ gives

$$\gamma = \frac{3-x}{4-\omega} \; , \qquad \delta = x - 2 + \frac{3-x}{4-\omega} \; .$$

The overall running time of the algorithm is therefore

$$\tilde{O}(n^{2+\delta}) = \tilde{O}(n^{\frac{3}{4-\omega}} m^{1-\frac{1}{4-\omega}}) = O(n^{1.848} m^{0.384}) \tag{13}$$

where the r.h.s. uses $\omega < 2.376$.

Finally, using the estimate from Lemma 2.2 for $\omega(1,\gamma,1)$ with $\alpha = 0.294$ we obtain a slightly improved running time of $O(n^{1.939} m^{0.318})$. ■

4 Concluding remarks

Any algorithm that uses witness matrix computations as an ingredient, and where (at least) one of the matrices involved can be assumed to have m nonzero entries may benefit from the algorithm of Theorem 1.1. We give two examples.

In [13] it was shown, in particular, how to compute, for any pair of vertices in vertex-weighted graph, a minimum weight triangle containing the pair, if one exists. (If a globally minimum weight triangle is required, a faster algorithm that does not rely on witnesses in given in [6].) The main and most costly ingredient of the algorithm in [13] uses just *one* witness computation of matrices whose number of non-zeros correspond to the number of edges of the graph. It hence runs in $O(n^{2.575})$ time. Our new algorithm shows that this problem can be solved, for graphs with m edges, in time $O(n^{1.939}m^{0.318})$, which is faster for $m = O(n^{2-\epsilon})$. Notice that in this application, *both* matrices have at most m nonzero entries, so it does not exploit the full power of Theorem 1.1.

In [11] it was shown how to compute the All Pairs Bottleneck Paths in directed vertex weighted graphs using several maximum witness computations as (the most costly) ingredient. In this problem the goal is to find a path connecting the pair of vertices, where the smallest weight of a vertex on the path is maximized. Again, the running time they obtain is $O(n^{2.575})$. The problem is that in some of the boolean products involved, both matrices may have $\Theta(n^2)$ nonzero entries even if the initial graph only has $m \ll n^2$ edges. However, if one settles for bottleneck paths consisting of at most 4 vertices, then in each boolean product at least one of the matrices has m nonzero entries (see Section 2.2 in [11]). Hence, this case can be solved in $O(n^{1.939}m^{0.318})$ time.

The most notable open problem in this area is whether or not a faster algorithm exists. In fact, we cannot rule out the existence of a faster algorithm for minimum witnesses even when $m = \Theta(n^2)$. At present, the fastest algorithm for this dense case still runs in $O(n^{2.575})$ time. Interestingly, the fastest algorithm for All-Pairs Shortest Paths in unweighted directed graphs also has this complexity [15], but for the latter we still do not know how to exploit the fact that the graph may be sparser to obtain faster running times. In fact, at present, APSP runs in $O(n^{2.575})$ time for graphs with $m > n^{1.576}$ edges. On the other hand, for minimum witnesses we do know how to exploit sparseness, even if only one of the matrices is sparse.

References

[1] N. Alon and M. Naor. Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4):434–449, 1996.

[2] G.E. Blelloch, V. Vassilevska, and R. Williams. A new combinatorial approach for sparse graph problems. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming*, pages 108–120. Springer-Verlag, 2008.

[3] D. Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13(1):42–49, 1997.

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[5] A. Czumaj, M. Kowaluk, and A. Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theoretical Computer Science*, 380(1-2):37–46, 2007.

[6] A. Czumaj and A. Lingas. Finding a heaviest vertex-weighted triangle is not harder than matrix multiplication. *SIAM Journal on Computing*, 39(2):431–444, 2009.

[7] Z. Galil and O. Margalit. Witnesses for boolean matrix multiplication and for transitive closure. *Journal of Complexity*, 9(2):201–221, 1993.

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

[9] M. Kowaluk and A. Lingas. LCA queries in directed acyclic graphs. In *Proceedings of the 32nd international colloquium on Automata, Languages and Programming*, pages 241–248. Springer Verlag, 2005.

[10] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[11] A. Shapira, R. Yuster, and U. Zwick. All-pairs bottleneck paths in vertex weighted graphs. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 978–985. Society for Industrial and Applied Mathematics, 2007.

[12] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[13] V. Vassilevska, R. Williams, and R. Yuster. Finding heaviest h-subgraphs in real weighted graphs, with applications. *ACM Transactions on Algorithms*, 6(3):1–23, 2010.

[14] V. Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Symposium on Theory of Computing, New York*, pages 887–898. ACM Press, 2012.

[15] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.