JOURNAL OF COMPLEXITY 9, 201–221 (1993)

Witnesses for Boolean Matrix Multiplication and for Transitive Closure

ZVI GALIL*

Department of Computer Science, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel, and Columbia University, New York, New York 10027

AND

OLDED MARGALIT

Department of Computer Science, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel

Received October 18, 1992

DEDICATED TO JOSEPH F. TRAUB ON THE OCCASION OF HIS 60TH BIRTHDAY

The subcubic ($O(n^\omega)$ for $\omega < 3$) algorithms to multiply Boolean matrices do not provide the witnesses; namely, they compute $C = A \cdot B$ but if $C_{ij} = 1$ they do not find an index k (a witness) such that $A_{ik} = B_{kj} = 1$. We design a deterministic algorithm for computing the matrix of witnesses which runs in $O(n^{\omega+e})$ time for any positive e. We also design an algorithm that computes witnesses for the transitive closure in the same time needed to compute witnesses for Boolean matrix multiplication. © 1993 Academic Press, Inc.

1. INTRODUCTION

Consider a Boolean matrix multiplication: $C = A \cdot B$, $C_{ij} = \bigvee_{k=1}^{n}(A_{ik} \wedge B_{kj})$. The n^3 time method that evaluates these expressions gives for every i, j for which $C_{ij} = 1$ all the k's for which $A_{ik} = B_{kj} = 1$. The subcubic

* Partially supported by NSF Grants CCR-90-14605 and CISE Institutional Infrastructure Grant CDA-90-24735.

0885-064X/93 $5.00
Copyright © 1993 by Academic Press, Inc.
All rights of reproduction in any form reserved.

methods on the other hand consider A and B as matrices of integers and do not provide any of these k's. We call a k such that $A_{ik} = B_{kj} = 1$ a *witness* (for the fact that $C_{ij} = 1$). We want to compute in addition to the matrix C a matrix of witnesses. When there is more than one witness for a given i and j we are satisfied with one such witness.

We use $O(n^\omega)$ to denote the running time of some subcubic algorithm for Boolean matrix multiplication. All our algorithms can be derived from any such algorithm yielding a corresponding time bound as a function of ω. The best asymptotic bound known at present is the one with the exponent $\omega < 2.376$ and is due to Coppersmith and Winograd (1987).

Witnesses for Boolean matrix multiplication were first defined by the authors, who gave the first subcubic algorithm to compute them (Galil and Margalit, 1991). Several researchers, including Seidel (1992), Karger, and Alon (personal communications) observed that there is a simple randomized algorithm that computes witnesses in $\tilde{O}(n^\omega)$ time, where $\tilde{O}(f(n)) = O(f(n)\log^{O(1)}(n))$. Recently Paul Dietz (private communication) improved this algorithm to run in $O(n^\omega \log(n)/\log\log(n))$ time (still, a randomized algorithm).

In Section 2 we describe a *deterministic* algorithm for computing the witnesses. 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 which are solved by calling the previous algorithm in the sequence. By choosing an appropriate algorithm from the sequence we obtain a time bound of $O(n^{\omega+\varepsilon})$ for any given ε. Recently N. Alon and M. Naor gave another deterministic algorithm that runs in time $O(n^\omega \log^6(n))$. It is essentially a derandomization of a modified version of the simple randomized algorithm, and relies heavily on the known constructions of small sample spaces with almost independent random variables. Thus their algorithm has a slightly better asymptotic running time. However, a carefully chosen algorithm from our sequence of algorithms may be better for very large but non astronomical matrices. Moreover, our method is entirely different and we hope it will lead to an $O(n^\omega)$ time algorithm.

Our motivation for studying the computation of witnesses for Boolean matrix multiplication is related to our work on the all pair shortest paths problem. One cannot have a subcubic algorithm for explicitly outputting the shortest paths between all pairs of vertices simply because in the example depicted in Fig. 1 there are more than $n^3/27$ edges in the collection of all shortest paths. In fact, this holds for an exponential number of input graphs. (Replace each cycle in Fig. 1 by an arbitrary connected graph of $n/3$ vertices.)

One may use the following definition to obtain a more concise representation of all shortest paths: A *witness for shortest path* from v_i to v_j is an

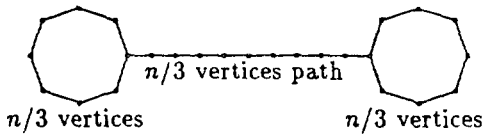

FIG. 1. All shortest paths can yield a cubic output.

index k such that v_k is the first vertex on such a path. This definition is certainly sufficient in case of positive edge lengths. A shortest path can be easily constructed from these witnesses.

This definition is insufficient in the case of negative cycles. We also want to have a concise representation for paths of length $-\infty$. One way to do it is to compute a collection of negative cycles together with information such that for $d_{ij}^* = -\infty$ we can easily compute a vertex v_k which is contained in one of the negative cycles and construct a finite path from v_i to v_j which passes through v_k (i, j, and k need not be distinct). This leads to the need to define witnesses for paths, not necessarily shortest paths.

Consider the transitive closure of a directed graph. One could try to define a witness for a path identically to the definition of a witness for a shortest path: A witness for a path from v_i to v_j is an index k such that v_k is the first vertex on such a path. Any method for computing witnesses for Boolean matrix multiplication can be immediately used for computing these "witnesses": compute witnesses for $A \cdot T$, where A is the incidence matrix and T the transitive closure. Unfortunately this definition is inappropriate as can be seen in Fig. 2: v_k is a possible witness for the path from v_i to v_j, but it is a bad choice which leads to a cycle.

We require a matrix of *witnesses for the transitive closure* to satisfy the following condition: If a path from v_i to v_j exists then such a path can be constructed by following the witnesses. Namely, there is a path $v_i = v_{i_0}, \ldots, v_{i_k} = v_j$ and for $1 \leq r \leq k$, i_r is the witness for the path from $v_{i_{r-1}}$ to v_j. In Section 3 we give an algorithm for computing witnesses for the transitive closure. It has the same time complexity as the time needed to compute witnesses for Boolean matrix multiplication.

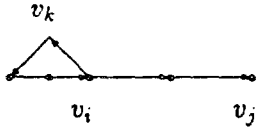

FIG. 2. A naive first step is not enough for transitive closure.

2. A Deterministic Algorithm for Computing Witnesses for Boolean Matrix Multiplication

In this section we give a deterministic algorithm for the witnessed Boolean matrix multiplication problem. In Subsection 2.1 we describe the algorithm. It can be viewed as a sequence of algorithms ALG_s; the first one, ALG_1 is the naive cubic algorithm, and each of the others computes a fast Boolean matrix multiplication and uses it to partitions the problem into several smaller problems which are solved by calling a previous algorithm. In Subsection 2.2. we analyze the time complexity of the algorithm and show that the exponent of ALG_s is $\leq \omega + (3 - \omega)/\sqrt{s}$, but the constants tend to infinity as s increases. So there is some optimal s for which we get the best complexity.

We generalize the problem by introducing more parameters to the problem. We solve a problem with four parameters: n, m, u, l_0. The problem is defined as:

n: We are given two $n \times n$ matrices
u: and a set U of u pairs for each of which we want to find a witness.
m: We are given also a set of m indices and know that all u pairs have a witness in one of these m indices.
l_0: We are given a partition of the m indices above into l_0 equal parts. We know for each pair that there is only one part which contains witnesses for it (and the part is known) so there are only m/l_0 possibilities (out of m) for the witness.

We denote the time complexity of solving the above problem by $f(n, m, u, l_0)$. The naive algorithm ALG_1 checks each one of the m/l_0 candidates for witness for every pair $(ij) \in U$ and thus

$$f(n, m, u, l_0) \leq f_1(n, m, u, l_0) = \frac{mu}{l_0}. \tag{1}$$

In Subsection 2.1 we show a way of generating faster and faster algorithms starting from this one.

2.1. *The Sequence of Algorithms*

The (n, m, u, l_0) problem is clearly easier than computing a witnessed $n \times m$ by $m \times n$ matrix multiplication. Such matrix multiplication can be computed naively, giving us all the witnesses, but requires $\Theta(n^2m)$ time; using a fast matrix multiplication method yields no witnesses. So we combine the two methods by splitting the input matrices into blocks of size $m/L \times m/L$ (the parameter L is fixed later), multiplying the blocks fast, and using a naive way to combine the block multiplications into the

desired result. We split m into L parts in orthogonal way to the given l_0 partition, i.e., each m/l_0 segment is divided into L equal subsegments and for each i, $1 \leq i \leq L$, the ith block consists of l_0 subsegments, the ith subsegment for each segment. Thus, we now know for each pair, not only m/l_0 possibilities, but also which of the L parts contains a witness, i.e., m/Ll_0 possibilities.

The algorithm ALG_{s+1} for $s > 0$, defines a Boolean matrix κ with u rows and L columns. An entry $\kappa(ij, l)$, $(ij) \in U$, $1 \leq l \leq L$, is one if and only if the lth block contains a witness for the (ij) pair. κ is computed from the input matrice using a fast Boolean matrix multiplication. Then the algorithm deactivates columns and/or rows of κ. A row (ij) is deactivated when we are no longer looking for a witness for it (at this stage). A column l is deactivated when we are not interested in getting witnesses from the lth block (at this stage).

We call a 1 entry in κ, $\kappa(ij, l)$, *unique* if $\kappa(ij, l) = 1$ and the (ij) row is active and for each active column $l' \neq l$, $\kappa(ij, l') = 0$. We start by deactivating columns: every column which contains only zeros in active rows can be deactivated since it does not contain any witnesses for (ij) pairs which interest us. Even if a column l does contain some 1s in active rows, we still can, sometimes, deactivate it: when it does not contain any unique entry:

$\forall$ active (ij), if $\kappa(ij, l) = 1$ then there exists $l' \neq l$ such that $\kappa(ij, l') = 1$.

We can deactivate column l since other columns can produce witnesses for the active rows. Furthermore, even if a column does contain at most α, (α is fixed later) unique entries, we deactivate it, along with the rows of the unique entries. We solve them later. After deactivating a column, entries may become unique and we look for other possible column deactivations. When no column can be deactivated we switch to row deactivation: Every column now has at least α unique entries. We call ALG_s, the previous algorithm in the sequence, to solve all these rows with unique ones at once. The four parameters for ALG_s are: n stays the same; m becomes ml_r/L, where l_r is the number of columns left; u is λ_r which is at least αl_r; and l_0 is $l_r l_0$ (apart from the partition to l_0 parts, which was known as input, we now have partition into l_r parts, where the witness for every entry in U can lie only in one part.)

After this row deactivating, we can switch back to columns deactivating since there are no unique entries. Hence we can deactivate at least one column (actually, we can even choose any column we like). We repeat this process of deactivating rows and columns again and again until there are fewer than δ (the parameter δ is fixed later) columns left. Then we solve the remaining rows using ALG_s. We should not forget the rows

which were taken out during the columns deactivating. We solve all of them too using the previous algorithm.

A more formal description of the algorithm is given below. All the algorithms share the two $n \times n$ Boolean matrices A and B. Each algorithm has additional input:

Input

1. $U \subseteq [1, n] \times [1, n]$, $|U| = u$.
2. $M \subseteq [1, n]$, $|M| = m$.
3. A function $\mathbf{L}_0$: $M \mapsto [1, l_0]$ (the partition to blocks).
4. A function $\mathbf{K}_0$: $U \mapsto [1, l_0]$ (the only block containing a witness)
5. An index s used to fix the behavior of the algorithm. We sometime refer to the running of the algorithm with the parameter s as ALG_s (hence the algorithm becomes a sequence of similar algorithms with four parameters).

The input satisfies:

1. $\forall\ (ij) \in U$, $\exists\ k \in M$, such that $a_{ik} = b_{kj} = 1$.
2. $\forall\ l \in [1, l_0]$, $|\mathbf{L}_0^{-1}(l)| = m/l_0$. Note that we assume that l_0 divides m.
3. If $k \in M$ satisfies $a_{ik} = b_{kj} = 1$ for $(ij) \in U$, then $\mathbf{L}_0(k) = \mathbf{K}_0(ij)$.

The input for ALG is

1. $U = \{(ij)\colon (A \cdot B)_{ij} = 1\}$.
2. $M = [1, n]$.
3. $\mathbf{L}_0 \equiv 1$.
4. $\mathbf{K}_0 \equiv 1$.
5. s_0, which is fixed later.

Clearly this input satisfies the three conditions and the five parameters (n, m, u, l_0, and s) are (n, n, u, 1, and s_0), where $u \leq n^2$.

Output

A function K: $U \mapsto M$ such that $\forall\ (ij) \in U$, $k = \mathbf{K}(ij)$ satisfies $a_{ik} = b_{kj} = 1$. We represent K as a set of pairs (ij, k) and build it gradually by joining subfunctions on disjoint subsets of U.

Algorithm

We use the superscript s to denote the parameters of ALG_s. So, U^s means the input U for ALG_s. We use also non-superscripted variables, U

for example. Whenever they are mentioned in context of ALG_{s+1} calling ALG_s, they should be regarded as the corresponding variables of ALG_{s+1}. The parameters L, α, and δ used in the algorithm are chosen later.

1. If $m^{3-\omega}u < l_0 n^2$ then call ALG_1 with the same parameters, else continue to the next step.
2. L-partition step:
   (a) *Round* L up: $\tilde{L} \stackrel{\text{def}}{=} \lceil L \rceil$. Note that $L \le \tilde{L} < L + 1$.
   (b) Add dummy elements to M, if necessary, to make it a multiple of $l_0\tilde{L}$.

$$\tilde{m} \stackrel{\text{def}}{=} l_0\tilde{L}\left\lceil \frac{m}{l_0\tilde{L}} \right\rceil.$$

   Note that $m \le \tilde{m} < m + l_o\tilde{L}$.
   (c) Define L: $M \longmapsto [1, \tilde{L}]$ such that

$$\forall\ 1 \le l' \le \tilde{L},\ 1 \le l \le l_0, \qquad |\mathbf{L}^{-1}(l') \cap \mathbf{L}_0^{-1}(l)| = \frac{\tilde{m}}{l_0\tilde{L}}.$$

   (d) Permute the columns of A and arrange them in L equal blocks: For $1 \le l \le \tilde{m}/\tilde{L}$, define $A(l) = \{a(l)_{ik}\}_{i=1\,k=1}^{n\,\tilde{L}}$ as

$$a(l)_{ik} = a_{ij}, \qquad \text{such that } \mathit{Index}(j, \mathrm{L}^{-1}(l)) = k.$$

   Similarly, define $B(l) = \{b(l)_{kj}\}_{k=1\,j=1}^{\tilde{L}\,n}$ as

$$b(l)_{kj} = b_{ij}, \qquad \text{such that } \mathit{Index}(i, \mathrm{L}^{-1}(l)) = k.$$

   (e) Note that $C \stackrel{\text{def}}{=} A \cdot B$, $C = \{c_{ij}\}_{i=1\,j=1}^{n\,n}$ satisfies

$$\forall\ (ij) \in U, \qquad c_{ij} = \bigvee_{l=1}^{\tilde{L}} \bigvee_{\mathrm{L}(k)=l} a_{ik} \wedge b_{kj},$$

   so, restricting to $(ij) \in U$, $c_{ij} = \tilde{c}_{ij}$, where

$$\tilde{C} = \bigvee_{l=1}^{\tilde{L}} A(l) \cdot B(l). \tag{2}$$

   Compute (2) using fast $\{\wedge, \vee\}$ Boolean matrix multiplication for $A(l) \cdot B(l)$ and let

$$\kappa(ij, l) \stackrel{\text{def}}{=} (A(l) \cdot B(l))_{ij}.$$

3. κ-reducing step:
   (a) Initialize:
      i. Activate all columns: $D \leftarrow [1, \tilde{L}]$,
      ii. The postponed (unsolved) part is empty: $U_1 \leftarrow \emptyset$ and
      iii. The partial solution is empty: $\mathbf{K} \leftarrow \emptyset$.
   (b) Define the function $\mathbf{Q}$: $U \mapsto [0, \tilde{L}]$ as

$$\mathbf{Q}(ij) = \begin{cases} l' & \forall l \in D,\ \kappa(ij, l) = 1 \Leftrightarrow l = l', \\ 0 & \text{otherwise.} \end{cases}$$

   $\mathbf{Q}$ marks the unique entries: $\mathbf{Q}(ij) \neq 0$ means that $(ij, Q(ij))$ is a unique entry. Note that by this definition $\mathbf{Q}$ is changed every time D is updated.
   (c) while $(|D| > \delta)$ do
      i. (Columns deactivating) While $\exists\ l \in D$, such that $|\mathbf{Q}^{-1}(l)| \leq \alpha$, do

Postpone unique entries	$U_1 \leftarrow U_1 \cup \mathbf{Q}^{-1}(l)$,
Remove them from U	$U \leftarrow U/\mathbf{Q}^{-1}(l)$,
Deactivate l	$D \leftarrow D/\{l\}$.

      ii. (Rows deactivating) Else (no such l exists) Call ALG_s to solve all the unique entries. We call it with the input
         A. $U^s \leftarrow \mathbf{Q}^{-1}(D)$.
         B. $M^s \leftarrow \mathbf{L}^{-1}(D)$.
         C. $\mathbf{L}_0^s$: $\mathbf{L}_0^s(k) \leftarrow |D|(\mathbf{L}_0^{s+1}(k) - 1) + Index(\mathbf{L}(k), D)$.
         D. $\mathbf{K}_0^s$: $\mathbf{K}_0^s(ij) \leftarrow |D|(\mathbf{K}_0^{s+1}(ij) - 1) + Index(\mathbf{Q}(ij), D)$.
         (Solution accumulating $\mathbf{K} \leftarrow \mathbf{K} \cup \mathbf{K}_s$ and $U \leftarrow U/\mathbf{Q}^{-1}(D)$.
   (d) (Rest of U) Call ALG_s with input:
      i. $U^s \leftarrow U$ (the elements which were left after the updates).
      ii. $M^s \leftarrow \mathbf{L}^{-1}(D)$.
      iii. $\mathbf{L}_0^s \leftarrow \mathbf{L}_0^{s+1}$.
      iv. $\mathbf{K}_0^s \leftarrow \mathbf{K}_0^{s+1}$.
      $\mathbf{K} \leftarrow \mathbf{K} \cup \mathbf{K}_s$.
   (e) (U_1, the rows deactivated in Step 3(c)i) Call ALG_s with the input:
      i. $U^s \leftarrow U_1$.
      ii. $M^s \leftarrow U^{s+1}$.
      iii. $\mathbf{L}_0^s \leftarrow \mathbf{L}_0^{s+1}$.
      iv. $\mathbf{K}_0^s \leftarrow \mathbf{K}_0^{s+1}$.
      $\mathbf{K} \leftarrow \mathbf{K} \cup \mathbf{K}_s$.

The exact values for the parameters L, α, and δ are chosen later.

Correctness Proof

LEMMA 1. *For all* $(ij) \in U$, *there exists a witness* $k \in \mathbf{L}^{-1}(D)$ *such that* $a_{ik} = b_{kj} = 1$.

Proof. At the beginning of ALG_{s+1}, $\mathbf{L}^{-1}(D) = U^{s+1}$, so the claim of the lemma follows from the first condition on the input of ALG_{s+1}. Updating U certainly does not violate the property since U only becomes smaller.

The only place we should check is the column deactivating (Step 3(c)i). Suppose that after l is removed from D, the property is violated for the first time. So there exists $(ij) \in U$ such that there is no witness k for it in $\mathbf{L}^{-1}(D)$, so

$$\forall\, l \in D, \qquad \kappa(ij, l) = 0.$$

From the minimality (first time), there exists $k \in \mathbf{L}^{-1}(D \cup \{l\})$, and clearly $\mathbf{L}(k) = l$, so $\kappa(ij, l) = 1$. By the definition of $\mathbf{Q}$, $\mathbf{Q}(ij)$ was l before l was removed and therefore $(ij) \notin U$ after the removing of l. ■

LEMMA 2. *The input for* ALG_s *is valid.*

Proof. By induction: the input for the first algorithm is obviously valid and we show that every time that ALG_{s+1} recursively calls ALG_s it calls it with a valid input.

The call in Step 3(c)ii is valid since

1. By Lemma 1, for $(ij) \in U^s$, there exists $k \in \mathbf{L}^{-1}(D)$ such that $a_{ik} = b_{kj} = 1$. From the definition of M^s, the first condition on the input of ALG_s follows.

2. If $\mathbf{L}_0^s(l_1) = \mathbf{L}_0^s(l_2)$, then from the definition of $\mathbf{L}_0^s$, $\mathbf{L}_0^{s+1}(l_1) = \mathbf{L}_0^{s+1}(l_2)$, and $\mathbf{L}(l_1 = \mathbf{L}(l_2)$. Therefore,

$$|(\mathbf{L}_0^s)^{-1}(l)| = \frac{\tilde{m}}{l_0^{s+1}\tilde{L}} = \frac{|D|\tilde{m}/\tilde{L}}{|D|l_0^{s+1}} = \frac{m^s}{l_0^s}.$$

3. It follows from the third condition on the input of ALG_{s+1} that $\mathbf{L}_0^{s+1}(k) = \mathbf{K}_0^{s+1}(ij)$. From the definition of $\mathbf{Q}$, $\mathbf{Q}(ij) = \mathbf{L}(k)$ (note that $\mathbf{Q}(ij) \neq 0$ since $(ij) \in \mathbf{Q}^{-1}(D)$), so the third condition on the input of ALG_s follows.

The call in Step 3d is valid

1. Follows from Lemma 1.
2. Immediate from the second condition on the input of ALG_{s+1}.
3. Follows from the third condition on the input of ALG_{s+1} ($M^{s+1} \subseteq M^s$).

The call in Step 3e is valid since all the three conditions easily follow from the condition on the input of ALG_{s+1}. ■

THEOREM 1. *ALG_s is correct, i.e., it finds a witness for every $(ij) \in U$.*

Proof. By induction on s. ALG_1 is correct since it finds all the witnesses which exist, and the conditions on the input guarantees that every $(ij) \in U$ has a witness. We assume that ALG_s is correct, and prove that ALG_{s+1} is correct. By correctness of ALG_s, Step 3d computes $\mathbf{K}$ for $(ij) \in U$ (the elements which are left in U when we reach Step 3d) and Step 3e computes $\mathbf{K}$ for $(ij) \in U_1$. So it remains to show that Step 3c computes $\mathbf{K}$ for (ij) in the final value of $U^{s+1} \backslash (U \cup U_1)$. We show by induction on the number of iterations of Step 3c that at any moment we have $\mathrm{K}(ij)$ for pairs (ij) in the current value of $U^{s+1} \backslash (U \cup U_1)$. Initially $U_1 = \varnothing$ and $U = U^{s+1}$ so $U^{s+1} \backslash (U \cup U_1) = \varnothing$. There are two types of updates: in Step 3(c)i, $U \cup U_1$ is kept the same, and in Step 3(c)ii, $U^{s+1} \backslash (U \cup U_1)$ is enlarged by $\mathbf{Q}^{-1}(D)$, but from the correctness of ALG_s, $\mathbf{K}^s$ contains solutions for $\mathbf{Q}^{-1}(D)$. ■

Time Estimation

Given an algorithm ALG_s, whose time complexity is $f_s(n, m, u, l_0)$, we choose three parameters, L, α and δ, use them as described above and get that if $m^{3-\omega} u < l_0 n^2$ then

$$f_{s+1}(n, m, u, l_0) = f_1(n, m, u, l_0) = mu/l_0 \tag{3}$$

else

$$f_{s+1}(n, m, u, l_0) \leq n^2 \tilde{m}^{\omega-2} \tilde{L}^{3-\omega} + f_s(n, \tilde{m}, \tilde{L}\alpha, l_0) + f_s\left(n, \frac{\tilde{m}\delta}{\tilde{L}}, u, l_0\right) + \Sigma^{\uparrow} f_s\left(n, \frac{\tilde{m} l_r}{\tilde{L}}, \lambda_r, l_0 l_r\right) \tag{4}$$

Where $\Sigma^{\uparrow}$ is a sum over iteration numbers r and we have $\delta \leq l_r$, $\alpha l_r \leq \lambda_r$ and $\Sigma_r \lambda_r \leq u$. The first term is the time for Step 2, which is

$$\left(\frac{\tilde{m}}{\tilde{L}}\right)^{\omega} \left(\frac{n}{\tilde{m}/\tilde{L}}\right)^2 \tilde{L}.$$

The second term is for Step 3e (every time U_1 is updated, no more than α elements are added to it, and there are at most $\tilde{L}$ steps since each column can be deactivated at most once). The third term is for Step 3d which is invoked when no more than δ columns are left, giving rise to $\tilde{m}\delta/\tilde{L}$ potential witnesses. The term in the summation is for each call in Step 3(c)ii with l_r columns and λ_r unique entries in them. We have $l_r \geq \delta$, $\lambda_r \geq \alpha l_r$ because each column has at least α unique entries and $\Sigma_r \lambda_r \leq u$ because all these entries are distinct.

2.2. *Recursion Analysis*

We prove inductively that

1. The time complexity of ALG_s satisfies

$$f_s(n, m, u, l_0) \le 13^s n^2 m^{\omega-2} \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{c_s}.$$

2. The parameter c_s satisfies $0 \le c_s \le 1$ and for $s > 1$, $c_s \le (3 - \omega)/(4 - \omega)$.

We will also show that the sequence of parameters c_s converges to 0. The base, ALG_1, has $c_1 = 1$ (See (1)). Next, we prove the induction step, but first we prove several lemmas.

LEMMA 3. *If* $0 \le c \le 1$, $a_r \ge y$ *for* $1 \le r \le n$ *and* $\sum_{r=1}^{n} a_r \le A$ *then*

$$\sum a_r^c \le \frac{A}{y^{1-c}}.$$

Proof.

$$\sum_{r=1}^{n} a_r^c = \sum_{r=1}^{n} \frac{a_r}{a_r^{1-c}} \le \sum_{r=1}^{n} \frac{a_r}{y^{1-c}} \le \frac{A}{y^{1-c}}. \quad \blacksquare$$

In Section 2.1 we proved recursion (4). Examine the summation term:

$$\Sigma^{\uparrow} f_s\left(n, \frac{\bar{m}l_r}{\bar{L}}, \lambda_r, l_0 l_r\right) \le \Sigma^{\uparrow} 13^s n^2 \left(\frac{\bar{m}l_r}{\bar{L}}\right)^{\omega-2} \left(\frac{(\bar{m}l_r/\bar{L})^{3-\omega}\lambda_r}{l_0 l_r \cdot n^2}\right)^{c_s},$$

taking constants out of the sum we get,

$$\le 13^s n^2 \left(\frac{\bar{m}}{\bar{L}}\right)^{\omega-2} \left(\frac{(\bar{m}/\bar{L})^{3-\omega}}{l_0 \cdot n^2}\right)^{c_s} \Sigma^{\uparrow} l_r^{\omega-2+(3-\omega)c_s-c_s} \lambda_r^{c_s},$$

using the condition $\alpha l_r \le \lambda_r$ yields,

$$\le 13^s n^2 \left(\frac{\bar{m}}{\bar{L}}\right)^{\omega-2} \left(\frac{(\bar{m}/\bar{L})^{3-\omega}}{l_0 \cdot n^2}\right)^{c_s} \alpha^{-(\omega-2)(1-c_s)} \sum_{\substack{\delta \le l_r \\ \sum \lambda_r \le u}} \lambda_r^{\omega-2+(3-\omega)c_s}.$$

The exponent $b = \omega - 2 + (3 - \omega)c_s$ of λ_r satisfies $0 \le b \le 1$ (from the induction hypothesis on ALG_s), so we can apply Lemma 3 with $c = b$, $a_r = \lambda_r$, $y = \alpha\delta$ (since $\lambda_r \ge \alpha l_r \ge \alpha\delta$) and $A = u$. We get

$$\Sigma^{\uparrow} f_s\left(n, \frac{\tilde{m} l_r}{\tilde{L}}, \lambda_r, l_0 l_r\right)$$

$$\leq 13^s n^2 \left(\frac{\tilde{m}}{\tilde{L}}\right)^{\omega-2} \left(\frac{(\tilde{m}/\tilde{L})^{3-\omega}}{l_0 \cdot n^2}\right)^{c_s} \alpha^{-(\omega-2)(1-c_s)} \frac{u}{(\alpha\delta)^{1-(\omega-2+(3-\omega)c_s)}},$$

or

$$\Sigma^{\uparrow} f_s\left(n, \frac{\tilde{m} l_r}{\tilde{L}}, \lambda_r, l_0 l_r\right) \leq 13^s \frac{u}{\alpha\delta} n^2 \left(\frac{\tilde{m}\delta}{\tilde{L}}\right)^{\omega-2} \left(\frac{(\tilde{m}\delta/\tilde{L})^{3-\omega}\alpha}{l_0 n^2}\right)^{c_s}. \tag{5}$$

Denote

$$d_s \stackrel{\text{def}}{=} (3-\omega)(\omega-2) + 2(3-\omega)^2 c_s + (\omega - 2 - (3-\omega)^2)c_s^2 + (3-\omega)c_s^3. \tag{6}$$

LEMMA 4. $d_s > 0$.

Proof. Rewrite the definition (6) as

$$d_s = (3-\omega)(c_s^2 + (\omega - 2 + (4-\omega)c_s)(1-c_s)) + (\omega-2)c_s^2 + (3-\omega)c_s^3.$$

$2 \leq \omega < 3$, $0 < c_s \leq 1$ so the first term is positive hence

$$d_s \geq (\omega-2)c_s^2 + (3-\omega)c_s^3.$$

Applying $2 \leq \omega < 3$ and $0 < c_s$ again proves that $d_s > 0$. ∎

Choose the parameters L, δ, α as

$$\begin{aligned} L &\stackrel{\text{def}}{=} \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{((\omega-2)c_s + 2(3-\omega)c_s^2 - (3-\omega)c_s^3)/d_s}, \\ \tilde{L} &\stackrel{\text{def}}{=} [L], \\ \delta &\stackrel{\text{def}}{=} \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{((\omega-2)c_s + 2(3-\omega)c_s^2 - (4-\omega)c_s^3)/d_s}, \\ \alpha &\stackrel{\text{def}}{=} u \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{(-(\omega-2)c_s - (4-\omega)c_s^2)/d_s}. \end{aligned} \tag{7}$$

LEMMA 5. $L \geq 1$.

Proof. $(\omega-2)c_s + 2(3-\omega)c_s^2 - (3-\omega)c_s^3 = c_s(1-c_s)(\omega - 2 + (4-\omega)c_s) + c_s^3$, $0 < c_s \leq 1$ and $2 \leq \omega < 3$ so, the numerator of the exponent of

L is non-negative. The denominator is positive (Lemma 4) and the base of the exponent is ≥ 1 (otherwise Step 1 returns immediately), so $L \geq 1$. ■

LEMMA 6. $L \leq m/l_0$.

Proof. L of ALG_{s+1} satisfies

$$L = \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{(c_s/(3-\omega))(1-((\omega-2)c_s^2+(3-\omega)c_s^3/d_s))}.$$

If $s = 1$, then $c_s = 1$, $d_s = 4 - \omega$ and hence

$$L = \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{1/(4-\omega)}.$$

Otherwise ($s > 1$), $2 \leq \omega < 3$ and $c_s \leq (3 - \omega)/(4 - \omega)$, so

$$L \leq \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{c_s/(3-\omega)} \leq \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{1/(4-\omega)}.$$

$1/(4 - \omega) \leq 1$, hence in any case,

$$L \leq \frac{u}{n^2} \cdot \frac{m^{3-\omega}}{l_0} \leq \frac{u}{n^2} \cdot \frac{m}{l_0}.$$

n is not changed during the recursive calls, and u only decreases. In the beginning, $u \leq n^2$, hence it is true all the time and therefore $L \leq m/l_0$. ■

Recall that $m \leq \tilde{m} < m + l_0\tilde{L}$ and $\tilde{L} < 2L$ (by Lemma 5). Thus, by Lemma 6 we have

COROLLARY 7. $m \leq \tilde{m} < 3m$.

Actually, as an anonymous referee pointed out, one can prove that $\tilde{m} < 2m$, but this will only improve the constants which are quite loose anyway.

LEMMA 8. *Part* 1 *of the induction step holds.*

Proof. Estimate the time complexity of ALG_{s+1}. If $m^{3-\omega}u < l_0 n^2$, then (3) applies and

$$f_{s+1}(n, m, u, l_0) = f_1(n, m, u, l_0) = \frac{mu}{l_0}$$

$$= 13^s n^2 m^{\omega-2} \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{c_s} \left(\left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{1-c_s} / 13^s\right).$$

Using $m^{3-\omega}u < l_0 n^2$ and $c_s \leq 1$ we get that the right-hand parenthesized term is ≤ 1 so

$$f_{s+1}(n, m, u, l_0) \leq 13^s n^2 m^{\omega-2} \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{c_s}.$$

Otherwise, we have $m^{3-\omega}u \geq l_0 n^2$ and (4) applies. We use induction for the first two recursive terms and (5) as estimate for the summation, and we get

$$\begin{aligned} f_{s+1}(n, m, u, l_0) \leq\ & n^2 \tilde{m}^{\omega-2} \tilde{L}^{3-\omega} + 13^s n^2 \tilde{m}^{\omega-2} \left(\frac{\tilde{m}^{3-\omega} \cdot \tilde{L}\alpha}{n^2 l_0}\right)^{c_s} \\ & + 13^s n^2 \left(\frac{\tilde{m}\delta}{\tilde{L}}\right)^{\omega-2} \left(\frac{(\tilde{m}\delta/\tilde{L})^{3-\omega}u}{n^2 l_0}\right)^{c_s} \\ & + 13^s \frac{u}{\alpha\delta} n^2 \left(\frac{\tilde{m}\delta}{\tilde{L}}\right)^{\omega-2} \left(\frac{(\tilde{m}\delta/\tilde{L})^{3-\omega}\alpha}{l_0 n^2}\right)^{c_s}. \end{aligned}$$

Using Corollary 7 and the fact that $L \leq \tilde{L} < 2L$ yields:

$$\begin{aligned} f_{s+1}(n, m, u, l_0) \leq\ & 3^{\omega-2} 2^{3-\omega} n^2 m^{\omega-2} L^{3-\omega} \\ & + 13^s 3^{\omega-2+(3-\omega)c_s} 2^{c_s} n^2 m^{\omega-2} \left(\frac{m^{3-\omega} \cdot L\alpha}{n^2 l_0}\right)^{c_s} \\ & + 13^s 3^{\omega-2+(3-\omega)c_s} n^2 \left(\frac{m\delta}{L}\right)^{\omega-2} \left(\frac{(m\delta/L)^{3-\omega}u}{n^2 l_0}\right)^{c_s} \\ & + 13^s 3^{\omega-2+(3-\omega)c_s} \frac{u}{\alpha\delta} n^2 \left(\frac{m\delta}{L}\right)^{\omega-2} \left(\frac{(m\delta/L)^{3-\omega}\alpha}{l_0 n^2}\right)^{c_s}. \end{aligned}$$

Let

$$c_{s+1} = \frac{(3-\omega)(\omega-2)c_s + 2(3-\omega)^2 c_s^2 - (3-\omega)^2 c_s^3}{d_s}. \tag{8}$$

Substitute the values chosen for L, α and δ in (7) and get

$$\begin{aligned} f_{s+1}(n, m, u, l_0) \leq\ & n^2 m^{\omega-2} \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{c_{s+1}} \\ & \times (3^{\omega-2} 2^{3-\omega} + 13^s 3^{\omega-2+(3-\omega)c_s} (2^{c_s} + 1 + 1)). \end{aligned}$$

$2 \le \omega < 3$ and $c_s \le 1$ yields $2^{3-\omega} \le 2$, $3^{\omega-2} \le 3$ and $3^{\omega-2+(3-\omega)c_s} \le 3$ so

$$f_{s+1}(n, m, u, l_0) \le n^2 m^{\omega-2} \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{c_{s+1}}$$

$$\times (6 + 12 \cdot 13^s) < 13^{s+1} n^2 m^{\omega-2} \left(\frac{m^{3-\omega}u}{l_0 n^2}\right)^{c_{s+1}}.$$

This proves Part 1 of the induction. ■

COROLLARY 9. $c_2 = (3 - \omega)/(4 - \omega)$.

Proof. Substitute $c_1 = 1$ in (8). ■

LEMMA 10. $0 < c_s \le 1/\sqrt{s}$ *and* $c_{s+1} \le c_s$.

Proof. By induction: the base is trivial

$$0 < c_1 = 1 \le \frac{1}{\sqrt{1}}.$$

Assume that $0 < c_s \le 1/\sqrt{s}$, then rewriting (8) we get

$$c_{s+1} = c_s \Big/ \left(1 + \frac{c_s^2((3 - \omega)c_s + \omega - 2)}{(\omega - 2)(3-\omega) + (3 - \omega)^2 c_s(2 - c_s)}\right).$$

Using $2 \le \omega \le 3$ and the induction hypothesis $0 < c_s$ we get

$$c_{s+1} \le c_s \Big/ \left(1 + \frac{c_s^2((3 - \omega)c_s + \omega - 2)}{(\omega - 2) \cdot 1 + 1 \cdot (3 - \omega)c_s \cdot 2}\right) \le \frac{c_s}{1 + (c_s^2/2)}.$$

Since $1 + x \ge \sqrt{1 + 2x}$,

$$c_{s+1} \le c_s \frac{1}{\sqrt{1 + c_s^2}} = \frac{1}{\sqrt{(1/c_s^2) + 1}}$$

so clearly $c_{s+1} \le c_s$ and from the induction hypothesis $c_s \le 1/\sqrt{s}$ we get

$$c_{s+1} \le \frac{1}{\sqrt{s + 1}}. \tag{9}$$

As for the sign of c_{s+1}, by Lemma 4, it is enough to show that the numerator is positive;

$$(3-\omega)(\omega-2)c_s + 2(3-\omega)^2 c_s^2 - (3-\omega)^2 c_s^3 = (3-\omega)(\omega-2)c_s + (3-\omega)^2 c_s^2(2-c_s), \tag{10}$$

and since $0 < c_s < 2$, all the terms in the right-hand side of (10) are positive. ■

Corollary 9 and Lemma 10 prove Part 2 of the induction. Consequently, the complexity of ALG_s is $O(13^s n^{\omega+1/\sqrt{s}})$. Choosing $s = \log^{2/3} n$ yields the complexity of $O(n^{\omega+\log^{-1/3} n})$.

A practical note: while the fastest algorithm for Boolean matrix multiplication is highly impractical, there are some practical algorithms for moderate size matrices. Our best algorithm introduces intolerable constants even when we use the practical algorithms for Boolean matrix multiplication, however we can get a practical improvement by choosing ALG_s for some small constant s.

3. Computing Witnesses for Transitive Closure

In this section we compute witnesses for transitive closure: For all i and j if a path from v_i to v_j exists, we compute W_{ij}, a witness to the path. The witnesses satisfy the following property. A path from v_i to v_j can be constructed by following the witnesses. We saw in the introduction (Fig. 2) that the trivial way (witnessing $A \cdot T$) does not work. The reason it does not work is the existence of directed cycles, so we first find the witnesses for paths inside strongly connected components (Subsection 3.2), then we contract them into new vertices. Now we can use the trivial algorithm to solve the new problem (Subsection 3.3). Lastly, we "open" the contracted vertices and transform the solution to a solution for the original problem (Subsection 3.4). This algorithm is described more formally in Subsection 3.1.

3.1. *The Main Algorithm*

1. Compute strongly connected components of the input graph G. Let $V = \{v_1, \ldots, v_n\}$ be the set of vertices of G. Denote by $V' = \{v'_1, v'_2, \ldots, v'_{n'}\}$ the set of strongly connected components of G, $v'_i = \{v_{i1}, v_{i2}, \ldots, v_{ir_i}\}$ where $v_{ij} \in V$. We build a contracted graph G' whose set of edges is

$$E' = \{v'_i \to v'_j : \exists\, x, y \text{ such that } v_{ix} \to v_{jy} \in E\}.$$

This can be done in $O(n^2)$ time.

2. For each strongly connected component find witnesses for the transitive closure (which is a clique). Denote the witnesses matrix for that problem by W^*; this matrix is defined only for pairs which are in the same strongly connected component. This can be done in $O(n^2)$ time, as described in the algorithm of Subsection 3.2.

3. Solve the transitive closure problem on the graph G'; denote the solution by T'. Compute W', the witnesses for the Boolean matrix multiplication $A \cdot T'$. We show in Subsection 3.3. that it indeed solves the problem for G' (and does not run into problems like the one in Fig. 2.)

4. Expand the solution of the contracted problem into a solution for the whole problem. This can be done in $O(n^2)$ time as described in the algorithm of Subsection 3.4.

3.2. *Finding Witnesses for Strongly Connected Graphs*

The general idea behind the algorithm is to run two Breadth First Searches (BFSs). The first one runs from some arbitrary vertex v_0 on the outgoing edges. The second one runs from the same vertex on the ingoing edges (reversed BFS). In the first (forward) BFS we actually solve the Single Source problem. In the second (backward) BFS we transform the solution to all the other vertices. We use the fact that the transitive closure is a clique in the proof. The algorithm is shown in Fig. 3.

To prove the correctness of the algorithm we prove several invariants. First we prove the invariants of the first half (lines 1–19).

LEMMA 11. *While in the first half of the algorithm, the graph T_1 whose set of vertices is the vertices which has a nonempty Pre pointer and with edges $v \rightarrow w$ if and only if $Pre(v) = w$ is a tree rooted at v_0 plus a self loop at v_0.*

Proof. At the beginning (line 3), the tree contains only the root v_0. Every time a *Pre* pointer is changed (line 8) it is changed from nil (outside the tree) to point to a vertex which is already in the tree. Hence, by induction on the number of vertices in the graph it is a rooted tree. ■

Note that the *Pre* pointers are reversed edges of G (and T_1).

LEMMA 12. *While in the first half of the algorithm, W_{uv} is defined if and only if v is a descendant of u in T_1 and if so, it points to the son of u which is the ancestor of v (or v itself). Furthermore, W_{uv} is updated only once.*

Proof. Induction on the number of vertices in T_1. The base is trivial. Suppose that it is true so far, and we add another vertex w to T_1 (line 8). From Lemma 11, following the *Pre* pointers from w will lead us to v_0 through all the vertices u which are ancestors of w and clearly line 13 will make W point to the right son. We only update W_{uw} entries, none of them had a value before since w was not in T_1 and therefore was not a descendant of u in T_1. ■

LEMMA 13. *Each time the while condition (line 4) is tested Queue contains all the vertices v of T_1 such that there exists a vertex u outside T_1 such that $v \rightarrow u$ is an edge in G.*

```
1   for all v ∈ V do Pre[v] ← nil;
2   Queue ← v0;
3   Pre[v0] ← v0;
4   while (Queue ≠ ∅) do begin
5         delete v from Queue;
6         for all edges v → w do begin
7               if Pre[w] = nil then begin
8                  Pre[w] ← v;
9                  insert w into Queue;
10                 u ← w;
11                 u1 ← Pre[u];
12                 while u ≠ u1 do begin
13                       W_{u1 w} ← u;
14                       u ← u1;
15                       u1 ← Pre[u];
16                 end (while)
17              end (if)
18        end (for)
19  end (while Queue ≠ ∅)

20  for all v ∈ V do Post[v] ← nil;
21  Queue ← v0;
22  Post[v0] ← v0;
23  while (Queue ≠ ∅) do begin
24        delete v from Queue;
25        for all edges w → v do begin
26              if Post[w] = nil then begin
27                 Post[w] ← v;
28                 insert w into Queue;
29                 for all u ∈ V do
30                       if W_{wu} = nil then W_{wu} ← v;
31              end (if)
32        end (for)
33  end (while)
```

FIG. 3. Strongly connected witnessing.

Proof. The claim holds initially. Every time a vertex v is deleted from *Queue* (line 5) all the vertices w such that there exists an edge $v \rightarrow w$ in G join T_1 and the claim holds again. ■

COROLLARY 14. *When Queue is empty, all the vertices in G which are reachable from v_0 are in T_1.*

LEMMA 15. *Every vertex v enters T_1 and Queue at most once.*

Proof. v can enter *Queue* (line 9) only after it enters T_1 (line 8) and it cannot leave T_1. It can enter T_1 only if it was not there before (line 7). ■

LEMMA 16. *The time complexity of the first half (lines* 1–19) *is* $O(n^2)$.

Proof. Lines 4–5 take $O(n)$ time by Corollary 14. Lines 6–7 take $O(m) = O(n^2)$ time since each edge is examined at most once. Lines 8–11 take $O(n)$ time by Corollary 14. Lines 12–16 take $O(n^2)$ time since W is updated only once (Lemma 12). ■

LEMMA 17. *After the first half of the algorithm,* W_{v_0w} *is defined for every* w.

Proof. Recall that we are working on a strongly connected graph. So from Corollary 14 and Lemma 12, the first half finds witnesses from v_0 to every other vertex. ■

LEMMA 18. *The graph* T_2 *whose set of vertices is the vertices which has a non empty Post pointer and with edges* $v \rightarrow w$ *if and only if* $Post(w) = v$ *is a rooted tree where the edges point towards the root* v_0 *plus a self loop in* v_0.

Proof. At the beginning (line 22), the tree contains only the root v_0. Every time a *Post* pointer is changed (line 27) it is changed from nil (outside the tree) to point to a vertex which is already in the tree. Hence, by induction on the number of vertices in the graph it is a rooted tree, where the edges point toward the root v_0. ■

Note that the *Post* pointers are edges of G (and T_2).

LEMMA 19. *The witnesses matrix* W *always satisfies that if* W_{uv} *is defined, then following the witnesses from* u *to* v *would yield a simple path. Furthermore, if* v *is in* T_2 *then* W_{vw} *is defined for all* w.

Proof. By Lemma 12 and Lemma 17, when we complete the first half, the statement of the lemma holds. We use induction on the size of T_2 to show that the second part maintains this invariant. When we update W (line 30), we do not change any non nil value, so we need to check the property only for the pair wu which we update in line 30. $Post(v)$ was non nil, because v was in *Queue*, so by the induction hypothesis, W_{vu} is non nil. Furthermore, the path from v to u does not pass through w, otherwise, W_{wu} would have been non nil. There exists an edge $w \rightarrow v$, and we can fix W_{wu} to point to v and the resulting path is simple. W_{wu} will be defined for all u after line 29. ■

As for the time complexity, lines 27–30 are executed only n times since each vertex enters T_2 only once.

THEOREM 2. *The algorithm in Fig.* 3 *is correct and runs in* $O(n^2)$ *time.*

3.3. *Witnessed Transitive Closure on Acyclic Graphs*

In this subsection we prove why the naive way of witnessing $A \cdot T$ which does not work in the general case (see Fig. 2), does work for G'.

FACT 20. *G' is acyclic.*

Proof. Immediate from the construction, a cycle would merge several strongly connected components together. ■

LEMMA 21. *Witnessing the Boolean matrix multiplication $A \cdot T$ for acyclic graphs solves the witnessed transitive closure.*

Proof. By definition, if $W_{ij} = k$ then there exists an edge $v_i \rightarrow v_k$ and a path from v_k to v_j. Define $k_0 = i$ and $k_{r+1} = W_{k_r j}$. If $r \neq r'$ then $v_r \neq v_{r'}$ because the graph is acyclic. Therefore there cannot be more than n different k_rs and thus for some finite r, $k_r = j$. ■

3.4. *Algorithm for Joining Solutions*

Examine the solution for G'. Suppose that $W'_{ij} = k$. By the definition of witness, there exists an edge $v'_j \rightarrow v'_k$. By the definition of G', there exists an edge of the form $v_{ix} \rightarrow v_{ky}$ in G; we choose one such edge arbitrarily.

$$W_{v_{ik_1},\, v_{jk_2}} = \begin{cases} v_{ky} & k_1 = x \\ W^*_{v_{ik_1},\, v_{ix}} & \text{otherwise.} \end{cases}$$

The time complexity of this algorithm is $O(n^2)$.

THEOREM 3. *The algorithm above solves the witnessed transitive closure problem.*

Proof. We prove that given v_{ix} and v_{jk}, using W we obtain a path from v_{ix} to v_{jk}. $v_{ix} \in v'_i$ and $v_{jk} \in v'_j$, where v'_i and v'_j are strongly connected components of G and are vertices of G'. In G' the witnesses W' generate a path from v'_i to v'_j of length (number of edges) r. We prove the correctness of W by induction on r. The base is for vertices in the same strongly connected component, where we use W^* which was proved to work correctly in Subsection 3.2. Suppose that we proved for paths with r edges in G' and for some two vertices v_i and v_j the path (given by W') from v'_i to v'_j has $r + 1$ edges. Let v'_k be the first vertex in G' (the first strongly connected component in G). Tracing the witnesses matrix W from v_i toward v_j, we use W^* to reach v_{ix} as long as we stay in v'_i, the strongly connected component of v_i. It follows from Theorem 2 that we get there after finite number of steps. Then, by using W we get to v_{ky} in the strongly connected component v'_k. The length of the path in G' from v'_k to v_j is only r so we can use the induction hypothesis to complete the path. ■

4. Open Problems

Several algorithms: the randomized solution, our deterministic algorithm, and the derandomized solution compute witnesses and require time complexity close to $O(n^\omega)$ but there is still a small gap. We believe that our approach, perhaps in a simpler form, will close the remaining gap.

Another question is what is the time complexity of computing witnesses for Boolean matrix multiplication, given the result of the multiplication. Possibly, this can even be solved in $O(n^2)$ time, but all our algorithms use additional matrix multiplications.

Acknowledgment

We thank an anonymous referee for his comments, including the simple proof for Lemma 3, and for not letting us get away with a "shortcut" in one of the proofs.

References

Coppersmith, D., and Winograd, S. (1987), Matrix multiplication via arithmetic progressions, *in* "Proc. 19th Annual Symp. on Theory of Computing," pp. 1–6.

Dietz, P. (private communication), A faster randomized algorithm for computing the Boolean product witness matrix.

Galil, Z., and Margalit, O. (1991), On witnesses for Boolean matrix multiplication, unpublished manuscript, August.

Seidel, R. (1992), On the all-pairs-shortest-path problem, *in* "Proc. 24th Annual Symp. on Theory Of Comp.," pp. 745–749.