Witnesses for Boolean Matrix Multiplication and for Shortest Paths

Noga Alon[*]
Department of Mathematics
Tel Aviv University
Tel Aviv, 69978
ISRAEL

Zvi Galil[†]
Department of Computer Science
Tel Aviv University, Tel Aviv, 69978, ISRAEL
and
Columbia University, New York, NY 10027, USA

Oded Margalit
Department of Computer Science
Tel Aviv University
Tel Aviv, 69978
ISRAEL

Moni Naor

IBM Almaden Research Center
San Jose, CA 95120
USA

Abstract

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 that runs in $\tilde{O}(n^\omega)$ time, where here $\tilde{O}(n^\omega)$ denotes $O(n^\omega(\log n)^{O(1)})$.

The subcubic methods to compute the shortest distances between all pairs of vertices also do not provide for witnesses; namely they compute the shortest distances but do not generate information for computing quickly the paths themselves. A witness for a shortest path from v_i to v_j is an index k such that v_k is the first vertex on such a path. (The last sentence is not sufficient as a definition of a witness matrix when nonpositive edges are present, see Figure 3 in page 6 for details). We describe subcubic methods to compute such witnesses for several versions of the all pairs shortest paths problem. As a result, we derive shortest paths algorithms that provide characterization of the shortest paths in addition to the shortest distances in the same time (up to a polylogarithmic factor) needed for computing the distances; namely $\tilde{O}(n^{(3+\omega)/2})$ time in the directed case and $\tilde{O}(n^\omega)$ time in the undirected case.

We also design an algorithm that computes witnesses for the transitive closure in the same time needed to compute witnesses for Boolean matrix multiplication.

[*]Research supported in part by a United states Israel BSF Grant.

[†]Work partially supported by NSF Grant CCR-9014605.

1 Introduction

Consider a Boolean matrix multiplication: $C = AB$, $C_{ij} = \bigvee_{k=1}^{n}(A_{ik} \wedge B_{kj})$. 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 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 w. The best asymptotic bound known at present is the one with the exponent $\omega < 2.376$ and is due to Coppersmith and Winograd [3].

For two functions $f(n)$ and $g(n)$ we let the notation $g(n) = \tilde{O}(f(n))$ denote the statement that $g(n)$ is $O(f(n)(\log n)^{O(1)})$.

Several researchers, including Seidel [6], Karger (personal communication) and the first three authors, discovered a simple randomized algorithm that computes witnesses in $\tilde{O}(n^\omega)$ time. In Section 2 we describe a **deterministic** algorithm for computing the witnesses in $\tilde{O}(n^\omega)$ time. 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. We also outline an alternative approach that gives slightly worse running time but may

0-8186-2900-2/92 $3.00 © 1992 IEEE

still be useful for matrices of moderate size.

Our motivation for studying the computation of witnesses for Boolean matrix multiplication is related to our work on the all pair shortest paths problem. We use the following notation. $D = \{d_{ij}\}_{ij=1}^{n}$ is the matrix of edge lengths, $d_{ij} = +\infty$ in case there is no edge from v_i to v_j. In the positive case $d_{ij} \in \{1, 2, \ldots, M, +\infty\}$ and in the unrestricted case $d_{ij} \in \{0, \pm 1, \pm 2, \ldots, \pm M, +\infty\}$. $D^* = \{d^*_{ij}\}$ is the matrix of shortest distances.

In an earlier paper [2] the first three authors designed subcubic algorithms for computing all pair shortest distances of directed graphs with integer edge lengths whose absolute value is bounded by M. We denote the problem and its time bound by $APSD(n, M)$, where n is the number of vertices in the graph. We showed that $APSD(n, M) = O((Mn)^{\nu})$, where $\nu = (3 + \omega)/2$. For $\omega < 2.376$, we have $\nu < 2.688$. In a more recent work [4] the second and third authors have improved the dependence on M and obtained better bounds for undirected graphs, in which case $APSD(n, M) = O(M^{(\omega+1)/2} n^{\omega} \log n)$. A simple $O(n^{\omega} \log n)$ algorithm for undirected $APSD(n, 1)$ was discovered independently by Seidel [6], but it does not seem to be extendable to larger edge lengths. All these algorithms do not provide any subcubic deterministic way for finding the shortest paths themselves, only the shortest distances.

One cannot have a subcubic algorithm for explicitly outputing the shortest paths between all pairs of vertices simply because in the example depicted in Figure 1 there are more than $n^3/27$ edges in all shortest paths. In fact, this holds for an exponential number of input graphs. (Replace each cycle in Figure 1 by an arbitrary connected graph of $n/3$ vertices.)

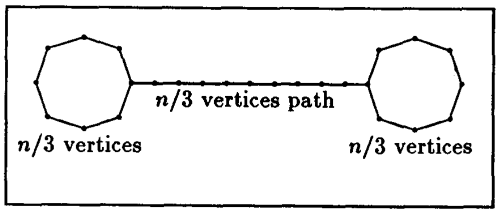

Figure 1: All shortest paths can yield a cubic output

One may use the following definition to obtain a more concise representation of all shortest paths: A *witness for a shortest path* from v_i to v_j is an 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 case of nonpositive cycles. If $d^*_{ij} = -\infty$ we want to be able to construct from the witnesses a simple path from v_i to v_j together with a vertex v_k on the path and a negative cycle containing 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 Figure 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.

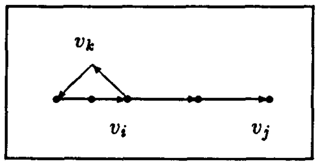

Figure 2: A naive first step is not enough for transitive closure

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 $v_i = v_{i_0}, .., v_{i_k} = v_j$ 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 an $\tilde{O}(n^{\omega})$ algorithm that computes witnesses for the transitive closure .

Coming back to the shortest paths problem we would like to compute in addition to the matrix D^* of shortest distances also

1. Witnesses for shortest paths.
2. A simple negative cycle for each i such that $d^*_{ii} = -\infty$.

Consequently, shortest paths of finite length can be easily obtained from 1. On the other hand, a shortest path of length $-\infty$ can be represented as a (possibly empty) path from 1 together with a negative cycle (from 2). We denote the problem of generating these witnesses for the $APSD(n, M)$ problem by $APSP(n, M)$ (All Pairs Shortest Paths).

In Section 4 we first give an algorithm for computing witnesses for shortest paths when edge lengths are positive, then when edge lengths are nonnegative. Finally we give an algorithm that generates the characterization of shortest paths in the general case. Its running time is $\tilde{O}(n^{(3+\omega)/2})$.

Summarizing, we get the following bounds for $APSP(n, 1)$: $\tilde{O}(n^{(3+\omega)/2})$ in the directed case and $\tilde{O}(n^{\omega})$ in the undirected case. Recall that the time bounds for $APSD(n, 1)$ are $O(n^{(3+\omega)/2})$ in the directed case and $O(n^{\omega} \log n)$ in the undirected case. Indeed, some of the Boolean matrix multiplications are now augmented to compute also witnesses, which explains our motivation to study the latter. (We believe that witnesses for Boolean matrix multiplication will be found useful elsewhere as well.) The fact that the bounds for $APSP$ are obtained from the bounds for $APSD$ by adding polylogarithmic factors is not immediate. This would be a simple consequence of our algorithm for matrix multiplication with witnesses if the algorithms just added witnesses to each Boolean matrix multiplication. However, this reason is not the only one needed to explain this coincidence. More details are given in Section 4.

2 Boolean matrix multiplication with witnesses

All the matrices in this section are n by n matrices, unless otherwise specified. If M is such a matrix, we let M_{ij} denote the entry in its ith row and jth column. Let A and B be two matrices with $\{0, 1\}$ entries, and let C be their product over the integers. Our objective is to find witnesses for all the positive entries of C, i.e., for each entry $C_{ij} > 0$ of C we wish to find a k such that $A_{ik} = B_{kj} = 1$. This is clearly equivalent to the problem of finding witnesses for the *Boolean* matrix multiplication $A \cdot B$. As observed by several researchers (including Seidel, Karger and the first three authors) there is a simple randomized algorithm that solves this problem in expected running time $\tilde{O}(n^{\omega})$. Here we consider ***deterministic*** algorithms for the problem. Our best algorithm, described in the next subsection, is, in a sense, a derandomized version of the simple randomized solution, and its running time is $\tilde{O}(n^{\omega})$. The derandomization requires several modifications in the straightforward randomized algorithm together with an interesting application of the known constructions of [5] (or [1]) of almost c-wise independent random variables in small sample spaces.

2.1 The algorithm

The first simple observation is the fact that if E and F are two matrices with $\{0, 1\}$ entries and $G = EF$ then one multiplication of matrices with entries at most n suffices for finding witnesses for all the entries of G which are precisely 1. Indeed, simply replace every 1-entry in the kth row of F by k (for all $1 \le k \le n$) to get a matrix F' and compute $G' = EF'$. Now observe that if $G_{ij} = 1$ and $G'_{ij} = k$ then k is a witness for G_{ij}.

Define $c = \lceil \log \log n + 9 \rceil$ and $\alpha = \frac{8}{2^c}$. For two matrices E and F with $\{0, 1\}$ entries define $G = E \wedge F$ by $G_{ij} = E_{ij} \wedge F_{ij}$.

Here is an outline of the algorithm. Besides A,B and $C = AB$ it uses two additional matrices: R and D. The way to perform steps 3c and 3d will be described later.

```
• While not all witnesses are known
    1. Let $L$ denote the set of all positive entries of $C$ for which there are no known witnesses.
    2. Let $R$ be the all 1 matrix.
    3. Perform the following $\lceil 1 + 3\log_{4/3} n \rceil$ times:
        (a) $D \leftarrow A \cdot (B \wedge R)$ (The matrix multiplication is over the integers)
        (b) Let $L'$ denote the set of all entries of $D$ in $L$ which are at most $c$.
        (c) Find witnesses for all entries in $L'$.
        (d) $R \leftarrow$ good matrix (see definition of good below).
```

A matrix R is *good* (in step 3d above) if the following two conditions hold:

a) The total sum of the entries of $D = A \cdot (B \wedge R)$ in L is at most 3/4 of what this sum was while using the previous matrix R. (Observe that this guarantees that after $1 + 3\log_{4/3} n$ iterations all these entries of D will vanish.)

b) The fraction of entries of D in L that go from a value bigger than c to 0 is at most α.

Lemma 1 *If $R \leftarrow R \wedge S$ in step 3d where S is a random* 0, 1 *matrix, then the new R is good with probability at least* 1/6.

The lemma follows from the following three claims:

Claim 1 *The probability that the sum of entries of D in L goes down by at least a factor of $3/4$ is at least $1/3$.*

To see this, observe that the expected sum of entries of D in L goes down by $1/2$. Thus, the claim follows from Markov's Inequality. □

Claim 2 *The probability that a fixed entry of D which is at least c drops down to 0 is at most $1/2^c$.*

This is obvious. Observe that the claim holds even if we only assume that every c entries of S are independent. □

Claim 3 *The probability that more than a fraction α of the entries of D in L drop from at least c to 0 is at most $\frac{1}{2^c}\frac{1}{\alpha} = \frac{1}{8}$.*

This follows from Claim 2 by Markov's Inequality. Since $1/3 - 1/8 > 1/6$ the lemma follows. □

Define $\epsilon = \frac{1}{2^{c+1}}$. The crucial point is to observe that the proof of the above lemma still holds, with almost no change, if the matrix S is not totally random but its entries are chosen from a c-wise ϵ-dependent distribution in the sense of [5], [1]. Recall that if m random variables whose range is $\{0, 1\}$ are *c-wise ϵ-dependent* then every subset of $i \leq c$ of them attains each of the possible 2^i configurations of 0 and 1 with probability that deviates from $1/2^i$ by at most ϵ.

Lemma 2 *If $R \leftarrow R \wedge S$ in step 3d where the entries of S are chosen as n^2 random variables that are c-wise ϵ-dependent, then the new R is good with probability at least $1/12 - 2\epsilon$.*

We note that in fact it is sufficient to choose only one column and copy it n times. The proof is by the following modified three claims, whose proof is analogous to that of the corresponding previous ones.

Claim 4 *The probability that the sum of entries of D in L goes down by at least a factor of $3/4$ is at least $1/3 - 2\epsilon$.* □

Claim 5 *The probability that a fixed entry of D which is at least c drops down to 0 is at most $1/2^c + \epsilon$.* □

Claim 6 *The probability that more than a fraction α of the entries of D in L drop from at least c to 0 is at most $(\frac{1}{2^c} + \epsilon)\frac{1}{\alpha} < \frac{2}{2^c}\frac{1}{\alpha} = 1/4$.* □

The lemma follows from the above three claims. □

As shown in [5] and in [1] there are explicit probability spaces with n^2 random variables which are c-wise ϵ-dependent, whose size is

$$\left(\log n \cdot c \cdot \frac{1}{\epsilon}\right)^{2+o(1)},$$

which is less than, e.g., $O((\log n)^5)$. Moreover, these spaces can be easily constructed in time negligible with respect to the total running time of our algorithm. Now suppose that in step 3d all the matrices S defined by such a probability space are searched, until a good one is found. Checking whether a matrix is good requires only matrix multiplication plus $O(n^2)$ operations. Therefore the inner loop (starting at step 3) takes polylog n times matrix multiplication time. It is important to note that during the performance of step 3d, while considering all possible matrices S provided by our distribution, we can accomplish step 3c as well. This is true since c-wise ϵ-dependence guarantees that every entry in L' will drop to precisely 1 for some of the matrices S and hence, by the observation in the beginning of this subsection, if we replace each matrix multiplication in the search for a good S by two matrix multiplications as described in that observation, we complete steps 3c and 3d together.

In every iteration of the inner loop 3 at most α fraction of the entries of L are "thrown" (i.e. their witness will not be found in this iteration of the outer loop). Therefore at least $(1-\alpha)^{1+3\log_{4/3} n}$ fraction of the entries of D in L will *not* be thrown during the completion of these iterations. For those entries, which are at least $1/2$ of the entries in L, a witness is found. Therefore, only $O(\log n)$ iterations of the outer loop are required, implying the desired $\tilde{O}(n^\omega)$ total running time.

We have thus proved the following:

Theorem 1 *The witnesses for the Boolean multiplication of two n by n matrices can be found in deterministic $\tilde{O}(n^\omega)$ time.*

2.2 An alternative approach

The witnesses for Boolean matrix multiplication can be computed in a different manner. Although the running time obtained is slightly worse than that of our previous algorithm, it may give better performance for matrices of moderate size.

Here is a rough outline of the approach: We design a sequence of algorithms, the first algorithm ALG_0, is the naive cubic way: test all the n possible witnesses for every positive entry C_{ij}. The next algorithm ALG_1, is the following: Consider each of the two matrices A and B as an $L \times L$ block matrix where each block is of size $n/L \times n/L$. Multiply the two block matrices using the trivial L^3 time algorithm, and using fast matrix multiplication for any multiplication of two blocks. Now we know for each positive entry C_{ij}, a product of a block of A and a block of B which contains a witness. Use ALG_0 for finding witnesses inside that block. The running time is

$$O\left(L^3\left(\frac{n}{L}\right)^{\omega} + L^2\left(\frac{n}{L}\right)^3\right).$$

An appropriate choice of L gives an $O(n^{\frac{9-2\omega}{4-\omega}})$ time algorithm.

The sequence starting with these two algorithms can be extended, where each algorithm uses the previous one and the time complexity converges to $O(n^{\omega+O(\log^{-1/3}(n))})$. This requires several additional ideas including a generalization of the problem to that of finding witnesses for a prescribed subset of entries of the product of two rectangular matrices, given certain information on the location of these witnesses. The details are complicated and since the running time is inferior to that of our previous algorithm we do not include them. For any given problem, one can apply any of the algorithms from the sequence above. It seems that for certain possible sizes, one of the algorithms ALG_s for some small integer s may actually be faster than the algorithm in the previous subsection.

3 Computing witnesses for the transitive closure

In the introduction we explained why the immediate solution that computes witnesses for $A \cdot T$ does not work. Another simple solution is to add lengths to the edges and compute witnesses for shortest paths. However, the best time for computing only the distances in the directed case (even without the witnesses for the paths) is $O(n^{(\omega+3)/2})$.

The only reason that the immediate solution does not work are the cycles. So we first find the strongly connected components of G, then we contract them into new vertices. Now we can use the immediate algorithm to solve the new problem. Lastly, we "open" the contracted vertices and transform the solution to a solution for the original problem. *More formally:*

Algorithm

1. Compute the strongly connected components of the input graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$. Denote by $V' = \{v'_1, v'_2, \ldots, v'_m\}$ the set of strongly connected components of G, where $v'_i = \{v_{i1}, v_{i2}, \ldots, v_{ir_i}\}$. We build the contracted graph $G' = (V', E')$, where $E' = \{(v'_i, v'_j) : \exists (v_{ix}, v_{jy}) \in E\}$. Each edge $(v'_i, v'_j) \in E'$ is arbitrarily associated with one edge $(v_{ix}, v_{jy}) \in E$. This can be done in $O(n^2)$ time.
2. Solve the transitive closure problem of the graph G', denote the solution by T'. Compute witnesses for the Boolean matrix multiplication $T' \cdot A'$ by W'. This step can be done in $\tilde{O}(n^{\omega})$ time.
3. For each strongly connected component we find witnesses for the transitive closure (which is a clique). Denote the witnesses matrix for that problem by $\hat{W}$; 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 Algorithm 3.1.
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 Algorithm 3.2.

Theorem 2 *The algorithm above computes the matrix of witnesses in time $\tilde{O}(n^{\omega})$.*

3.1 Computing witnesses for a strongly connected graph

The algorithm has two stages. In the first, we perform breadth first search (BFS) from one of the vertices v_0. In the process we generate a BFS tree T. For each edge $(u, v) \in T$ and every descendant w of v we set $W(u, w) \leftarrow v$. In the second stage, we use the reverse edges and perform another BFS from v_0. We process a vertex when it is first visited. Assume we enter first u using edge (u, v). We then consider all $w \in V$ and if $W(u, w)$ is undefined we set it to v.

Obviously, each stage takes $O(n^2)$ time. Correctness follows by induction. The induction hypothesis states that for every processed vertex u, and every w, starting with u and following W we obtain a simple path from u to w. The base is true because the first stage essentially processes v_0. For the induction step, assume we process u. If $W(u, w) = z$ is defined, it was defined in the first stage and (u, z) is in the BFS tree of the first stage and following W we follow a path on the tree from u to w. If it is undefined, we set $W(u,w) \leftarrow v$, where v was processed before. The claim now follows from the induction hypothesis.

3.2 Joining solutions

Examine the solution for G'. Suppose that $W'(i,j)$-$= k$; by the definition of a witness, there exists an edge (v'_i, v'_k). Let (v_{ix}, v_{ky}) be the edge of G associated with it.

$$W(v_{ik_1}, v_{jk_2}) = \begin{cases} v_{ky} & k_1 = x \\ \hat{W}(v_{ik_1}, v_{ix}) & \text{otherwise.} \end{cases}$$

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

4 Finding paths

In this section we solve the $APSP(n,1)$ problem. Solving the $APSP(n,M)$ problem is similar, since the treatment of 'large' edges is the same as in the $APSD$ problem. We first show what cannot be done. Then we solve the positive case (subsection 4.1). We solve the nonnegative case in subsection 4.2. The solution for the unrestricted case is complicated, and due to space limitations we omit most of the proofs for this case in subsection 4.4. In subsection 4.3 we consider the simpler special case of undirected graphs.

As explained in the introduction, one way to avoid the cubic bottleneck is to compute witnesses. For each pair (i,j) we compute an index k of a first vertex on a shortest path from v_i to v_j. This certainly works in the positive case.

4.1 Positive $APSP$

Consider the algorithm for the positive $APSD(n,1)$ problem [2]. It uses Boolean matrix multiplications to compute short distances. Computing Boolean matrix multiplication with witnesses gives witnesses for these shortest paths.

For computing large distances, we use the separator trick: We consider in turn each vertex as a source and the layered graph obtained by single source shortest paths. We take a block of consecutive layers, choose the smallest one and use it as a separator. Each path that goes beyond the separator must go through the separator. Hence we minimize over the choice of the vertex on the separator. This part is performed naively and provides witnesses: If a shortest path from v_i to v_j goes through v_k, where v_k belongs to the separator, then the i,j witness can be taken to be the i,k witness that has already been computed.

Consequently, to obtain the time bound we can substitute $\tilde{O}(n^\omega)$ (the time for witnessed Boolean matrix multiplication) for $O(n^\omega)$ in the bound for the positive $APSD(n,1)$:

Theorem 3 *The positive $APSP(n,1)$ problem can be solved in $\tilde{O}(n^{(3+\omega)/2})$ time.*

4.2 Nonnegative $APSP$

Consider the nonnegative $APSD(n,1)$ problem. One way to solve it is to eliminate the zero length edges first. We first compute Z, the transitive closure of the zero edges. It gives us all zero (shortest) distances. Let D' be obtained from D by replacing the $+\infty$ entries by zero. Compute the Boolean matrix multiplication $E \overset{\text{def}}{=} (Z+I)D'(Z+I)$. An edge in E corresponds to a path of length 1. Solve the $APSP$ problem for E to obtain all shortest nonzero distances.

A first attempt to obtain witnesses for shortest paths is simply to combine three sets of witnesses: the ones obtained from the solution of the positive $APSP$, the witnesses implied by the definition of E (two per entry) and the witnesses of the transitive closure Z. This approach does not give witnesses for the shortest paths as the following example shows.

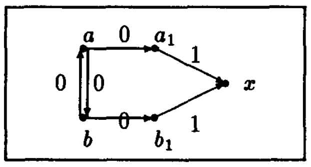

Figure 3: An example of failure of the naive solution

Consider Figure 3. E contains a direct edge of length 1 from a to x and from b to x. Suppose that the witness which was chosen in the $(Z+I)D'(Z+I)$ multiplication for the $a \to x$ pair is b_1 and for the $b \to x$ pair is a_1. Simply combining the three sets of witnesses would lead us to the infinite path $a, b, a, b, \ldots$ for the pair $a \to x$.

The difficulty above is caused by zero length cycles in G. But we overcome it as in Section 3. We find the strongly connected components of Z, compute witnesses for them and contract them in G, forming the contracted graph G'. We avoid multiple edges by always choosing a minimum length edge. We next find the witnesses for shortest paths in G' by the method described above. (In G' there are no zero length cycles.) Finally, we combine the two sets of witnesses as in Section 3 to obtain the witnesses for shortest paths of G. The full details are omitted.

Theorem 4 *The nonnegative $APSP(n,1)$ problem can be solved in $\tilde{O}(n^{(3+\omega)/2})$ time.*

4.3 Undirected *APSP*

In the undirected case, zero edges or negative edges pose no problem and can be eliminated easily. The special algorithm for the undirected $APSP$ uses essentially only Boolean matrix multiplications, which can easily be replaced by ones with witnesses. A *randomized* algorithm that solves the undirected $APSP(n,1)$ problem in (expected) time $\tilde{O}(n^{\omega})$ has been found by Seidel as well as by ourselves. Our result in Section 2 enable us to obtain a ***deterministic*** algorithm with the same running time.

Theorem 5 *The undirected $APSP(n,1)$ problem can be solved in $\tilde{O}(n^{\omega})$ time.*

4.4 The general case

We consider the case of $d_{ij} \in \{0, \pm 1\}$. We first solve the corresponding $APSD$ problem, so we know D^*. We reduce the unrestricted problem step by step: In Subsubsection 4.4.1 we show how to eliminate the $-\infty$ distances. In Subsubsection 4.4.2 we show how to eliminate the zero cycles. In Subsubsection 4.4.3 we solve the reduced problem where all the cycles are positive.

4.4.1 Removing the negative cycles

To handle the $-\infty$ distances, we start with the negative cycles. A ***negative component*** is a maximal set of vertices C such that if $v_i, v_j \in C$ then $d^*_{ij} = d^*_{ji} = -\infty$. The output for the unrestricted $APSP$ problem consists of the following parts:

1. $D^{*'}$ a matrix of distances which matches D^* in all its entries which are not $-\infty$.
2. A matrix of witnesses W: for every $1 \le i, j \le n$ starting from v_i and using W one obtains a simple path π_{ij} from v_i to v_j if one exists. Otherwise W_{ij} is undefined. In case d^*_{ij} is finite π_{ij} is a shortest path and in case $d^*_{ij} = -\infty$, π_{ij} passes through a negative component. Its length, in any case is $d^{*'}_{ij}$.
3. An array Neg representing a subgraph G' of G of outdegree ≤ 1. G' contains a simple negative cycle σ_C for every negative component C and a simple path from every vertex in C to σ_C.

Obviously the witnesses generate all finite length shortest paths. In case of $d^*_{ij} = -\infty$ this output can be used to generate a path from v_i to v_j of length under any specified bound: Given a bound L on the length of the path, start generating the path and update the bound. If, after reaching v_k, $D^{*'}_{kj}$ is less than the bound L, or if the Neg pointer is not defined, then follow the witnesses matrix W. Otherwise follow the Neg pointer. In any case update the bound by subtracting the length of the current edge $(v_k \rightarrow v_{k'})$.

The matrix $D^{*'}$ can be computed from the witnesses matrix W using the algorithm in Figure 4.

```
Set $D^{*'}_{ij} \leftarrow \begin{cases} \infty & i \neq j, \\ 0 & \text{otherwise} \end{cases}$
For all pairs $i, j$ such that $d^*_{ij} < \infty$ do Upd$(i,j)$

Upd$(i,j)$: If $(D^{*'}_{ij} = \infty)$ { $k \leftarrow W_{ij}$; Upd$(k,j)$;
    $D^{*'}_{ij} \leftarrow d_{ik} + D^{*'}_{kj}$ }
```

Figure 4: Algorithm for computing $D^{*'}$

Lemma 3 *The algorithm in Figure 4 computes $D^{*'}$ in $O(n^2)$ time.*

The computation of the array Neg is the most complicated. The reason for that is that we can not make any use of the distance matrix inside a negative component; all distances are $-\infty$. Let $P(n,m)$ be the problem of testing whether a graph with n vertices and m edges has a negative cycle and let $\widehat{P}(n,m)$ be the problem of constructing a negative cycle given the matrix D^* of shortest distances (and thus knowing that one exists).

Lemma 4 *The time complexity of P, $T(n,m)$ satisfies $T(n,m) \le \widehat{T}(2n, m+3n-2) + O(n)$, where $\widehat{T}$ is the time complexity of $\widehat{P}$.*

The proof construcs for an input G to P a new graph $\widehat{G}$ input to $\widehat{P}$ with only constant factor more vertices and edges. Thus any subcubic algorithm will be transformed into a subcubic algorithm. (Actually the complexity in terms of number of vertices and number of edges is preserved.) Therefore, since we believe that the problem of finding whether a contains a negative cycle or not is a difficult one (no subcubic algorithm was known before), we get that the problem of explicitly finding even a single negative cycle given all the shortest distances is a difficult problem. We solve it below.

The subcubic algorithm for the unrestricted $APSP$ is a modification of the unrestricted $APSD$ of [2]. The latter is a recursive algorithm that uses Boolean matrix multiplications, transitive closure computations and the separator trick. Some recursive calls construct a new graph and then operate on it. We now show how to compute the array Neg in $\tilde{O}(n^{\omega})$ time.

First we run the $APSD$ algorithm and replace the Boolean matrix multiplications by witnessed multiplications. The resulting recursive structure has logarithmic depth and each layer contains at most n edges. More precisely, we have $\log(n)$ types of edges, the zero type is an edge which appears in the input graph, and every type t edge $v_i \xrightarrow{t} v_j$ is a path $\pi^t_{ij} = \{v_{k_r}\}_{r=0}^{m}$ such that $k_0 = i$, $k_m = j$, $m \leq n$ and its length d^t_{ij} satisfies

$$d^t_{ij} \geq \sum_{r=0}^{m-1} d^{t-1}_{k_r k_{r+1}}. \qquad (1)$$

Call the process of replacing a type t edge with a type $t-1$ path a *refinement* of that edge.

Consider i with $d^*_{ii} = -\infty$. The APSD algorithm finds it by dicovering the existence of a triangle $(v_i, v_{k_1}, v_{k_2}, v_i)$ such that $d^t_{ik_1}, d^t_{k_2 i} \leq 0$, $d^0_{k_1 k_2} < 0$ for some $t \leq \log n$ (the recursion depth). It discovers the triangle by two Boolean matrix multiplications and computing them with witnesses yields also k_1 and k_2. We start with such a triangle and maintain a characterization of the path from v_i to v_i. We keep a linked list which specifies the refinement degree for every edge.

Repeat t times refining all the edges to edges which are one type smaller. The problem with this process is that we can end up with a path of $\Omega(n^{\log(n)})$ edges. To avoid this, in every refinement we keep the number of vertices to be no more than n by making sure that no vertex will appear more than once in the current path. To do this we also keep for every vertex v_k on the linked list a bound $dist(k)$ on the distance from v_i to v_k along the path which is being built and a unique index $num(k)$ which enables us to determine which vertex comes first on the path.

The refinement step is simple: we use the witnesses from the witnessed $APSD$ algorithm to convert a type t edge into a path of type $t-1$. Note that we always use type $t-1$ edges, it can be that some of these edges have lower type, say $t' < t-1$; e.g. in the triangles mentioned above. In this case we regard them as a type $t-1$ edge which consists of a path of one edge of type $t-2$ etc. We maintain the linked list and update the functions $dist$ and num. If while refining some edge $v_i \rightarrow v_j$ of type t, we get a vertex v_k which already was on that path, thus forming a cycle, there are two cases:

Negative cycle: We change our target: set $i \leftarrow k$ and start with the cycle between k and k which we currently have. We release the subpaths from v_i to the first occurrence of v_k and from the second occurrence of v_k to v_i (mark them as not in the current path).

Nonnegative cycle: We bypass the section between the two copies of v_k: define the successor of v_k in the list as the successor of the copy which is further in the path, and release the cycle.

It is easy to verify that while performing this algorithm we maintain a negative cycle which is refined all the time, but never has more than n vertices. So finally we will have a type zero negative cycle.

Theorem 6 *The algorithm sketched above finds a negative cycle in $O(n^2 \log n)$ time.*

After one negative cycle is build: $v_{i_0}, v_{i_1}, \ldots, v_{i_{m-1}}$, we fix Neg to follow it: $Neg(i_r) \leftarrow i_{r+1 \pmod m}$. To define Neg on the other vertices of the negative component C, remove all outgoing edges from the cycle (all edges $u \rightarrow w$ where u is in the cycle and w not). After the change there is still a path from every vertex of C to the cycle. Then run a BFS on the reverse edges from v_{i_0}. Set $Neg(v_p) = v_q$ if you reach v_p from v_q in the BFS. Clearly, Neg will not change on the cycle, and following it will lead from any vertex in C to v_{i_0}.

The time complexity of the algorithm is $O(n^2)$ per refining step (we refine no more than n edges of type t, each one of them into $O(n)$ edges of type $t-1$ and each takes $O(1)$ time), therefore the total time is $O(n^2 \log n)$. We may run this algorithm several times, once for each strongly connected component with n_i vertices. The complexity will be $\sum_i n_i^2 \log n_i$, where $\sum_i n_i = n$. From the convexity of $n^2 \log n$ we get that the time complexity is $O(n^2 \log n)$.

Now, that we have the array Neg, we remove all the $-\infty$ distances. The reduction is as follows:

1. Identify the negative components and find the Neg pointers for each one of them. All these pointers fit into a single array Neg.

2. Solve the witnessed transitive closure problem for each one of the negative components. Denote the witness matrix by T.

3. Construct a new graph $\hat{G}$ by contracting the negative components. In case of multiple edges we keep only one. In $\hat{G}$ we give edge length zero to original edges and -1 to edges incident with new vertices. Note that $\hat{G}$ does not have negative cycles. In $\hat{G}$ we compute witnesses for shortest paths (as explained in the next subsubsections). Denote the witnesses matrix by $\hat{W}$.

4. Join the last two matrices into W as follows: For every pair of vertices i and j, let $\hat{i}$ and $\hat{j}$ be the contracted negative components which contains i and j if such components exists, or the original vertices otherwise. Let $\hat{k} = \hat{W}_{\hat{i}\hat{j}}$ and let $i' \rightarrow k'$ be the edge in the original graph which was chosen to represent the contracted edge $\hat{i} \rightarrow \hat{k}$. Set

$$W^-_{ij} \leftarrow \begin{cases} T_{ij} & \hat{i} = \hat{j}, \\ k' & \hat{i} \neq \hat{j} \text{ and } i = i' \\ T_{ii'} & i \neq i'. \end{cases}$$

This completes the computation of witnesses for the $-\infty$ distances. At this point we can delete all edges which are adjacent to any vertex in a negative component, resulting in a graph $\widetilde{G}$ with no negative cycles. We solve the *APSP* problem for $\widetilde{G}$ and merge the solution $\widetilde{W}$ with that of the $-\infty$ distances W^- by taking

$$W_{ij} = \begin{cases} \widetilde{W}_{ij} & \tilde{d}^*_{ij} \text{ finite} \\ W^-_{ij} & \text{otherwise.} \end{cases}$$

It is easy to verify that defining the witness matrix W in this way, we can follow it and get the finite distances as $\widetilde{W}$ gives them, and get for the $-\infty$ distances a simple finite path which passes though at least one negative component.

It remains to show how to compute witnesses for graphs with no negative cycles. Because of the presence of negative length edges we cannot use the simple algorithm that contracts the zero length cycles of Subsection 4.2.

4.4.2 Removing the zero cycles

We define $v_i R v_j$ if and only if $d^*_{ij} + d^*_{ji} = 0$ and refer to the equivalence classes of R as *zero components.* We then compute witnesses as follows.

1. Find witnesses inside zero components.

2. Replace each zero component with a new structure called a U–structure, so that the resulting graph has no non–trivial zero components and compute witnesses for the resulting graph.

3. Combine the two sets of witnesses.

To find witnesses inside a zero component we define a subgraph G_1 with edges (v_i, v_j) such that v_i and v_j are in the same zero component and $d_{ij} = d^*_{ij}$. Thus each edge in G_1 is a shortest path, furthermore

Lemma 5 *Each path in G_1 is a shortest path in G.*

Given a component with m vertices, its corresponding U–structure has a set of vertices containing two vertices v'_i and v''_i for each original vertex v_i and the edges are:

1. An edge from v'_i to v'_j of length -1 for every v_i and v_j such that $d^*_{ij} = -1$.

2. An edge from v''_i to v''_j of length $+1$ for every v_i and v_j such that $d^*_{ij} = +1$.

3. Zero length edges from v'_i to v'_j and from v''_i to v''_j for every pair of vertices v_i and v_j such that $d^*_{ij} = 0$ and $i < j$.

4. For every k, a zero length edge from $v'_{max(k)}$ to $v''_{min(k)}$ where $max(k) \overset{\text{def}}{=} \max\{\ell : d^*_{\ell k} = 0\}$ and $min(k) \overset{\text{def}}{=} \min\{\ell : d^*_{\ell k} = 0\}$. Note that we may count the same edge several times (there are no multiple edges).

The U–structure replaces the corresponding component by replacing any edge from v_i of some component to w_j of another component by an edge from v''_i to w'_j. An example is shown in Figure 5. In this example, all down going edges in the U–structure have length -1, all up going edges have length $+1$ and all horizontal edges have length 0. The thick edges are the edges from $v'_{\max(k)}$ to $v''_{\min(k)}$.

Lemma 6 *Let G be a graph with no negative cycles. The graph G_2 obtained from G by replacing each zero component by a U structure satisfies the following properties*

1. *Its edge lengths are in* $\{-1, 0, 1\}$,

2. *it contains no zero length cycles,*

3. *it has twice the number of vertices as G and*

4. *it contains the shortest distances structure of G.*

Every shortest path in G_2 can be translated into a shortest path in G by replacing the edges in the U-structures by paths in G (which exists from Lemma 6). Thus, we have reduced the unrestricted case to the case in which all cycles have positive lengths.

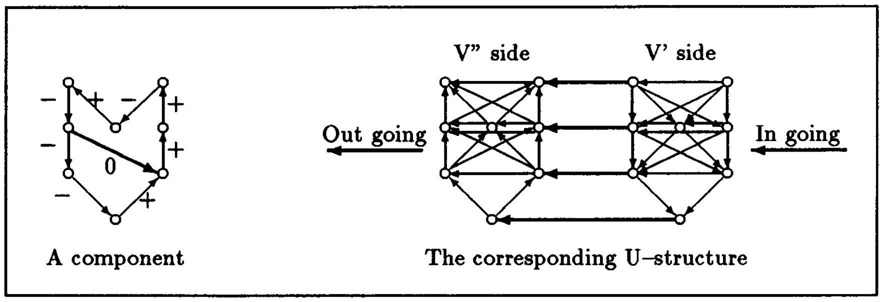

Figure 5: An example of a U–structure

4.4.3 Positive cycles only

Here, we find witnesses by unwinding the recursive *APSD* algorithm taking always the first edge.

Lemma 7 *Given a graph G with no directed nonpositive cycles, we can solve the APSP problem by replacing all the Boolean matrix multiplications in the APSD algorithm by witnessed Boolean matrix multiplications.*

Theorem 7 *The unrestricted $APSP(n,1)$ problem can be solved in $\tilde{O}(n^{(3+\omega)/2})$ time.*

Proof. Subsubsection 4.4.1 reduced the general problem into one without negative cycles. Subsubsection 4.4.2 reduced the problem even further to the case where there is no nonpositive length cycle. In this subsubsection we solved this case. □

5 Open Problems

Of course one would like to improve the time bounds for computing various types of witnesses. In particular, can we compute the witnesses for Boolean matrix multiplication in time $O(n^\omega)$? Alternatively, one would like to improve the best algorithms for $APSD(n,M)$ and $APSP(n,M)$, for directed graphs and for undirected graphs, for the nonnegative case and for the unrestricted case, by improving the exponent (in the undirected case this is not possible without improving Boolean matrix multiplication) or by improving the dependence on M.

We are also looking for other problems for which one may need the witnesses in addition to Boolean matrix multiplication or to transitive closure. It seems very plausible that these will find additional applications in the future.

Given the shortest distances, how hard is it to compute witnesses for the shortest paths? Possibly, this can be solved in $O(n^2)$ time, but all our algorithms need additional matrix multiplications.

References

[1] N. Alon, O. Goldreich, J. Hastad and R. Peralta, *Simple constructions of almost k-wise independent random variables*, Proc. 31st IEEE FOCS, St. Louis, Missouri, IEEE (1990), pp. 544-553.

[2] N. Alon, Z. Galil and O. Margalit, *On the exponent of the All Pairs Shortest Path problem*, Proc. 32th IEEE FOCS, IEEE (1991), pp. 569-575.

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

[4] Z. Galil and O. Margalit, *A faster algorithm for the all pairs shortest path problem for undirected graphs*, August 1991.

[5] J. Naor and M. Naor, *Small-bias probability spaces: efficient constructions and applications*, Proc. 22nd annual ACM STOC, ACM Press (1990), pp. 213-223.

[6] R. Seidel, *On the All-Pairs-Shortest-Path Problem*, Proc. 24th ACM STOC, ACM Press (1992), pp. 745–749.