On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs

RAIMUND SEIDEL*

Computer Science Division, University of California, Berkeley, Berkeley, California 94720

Received October 5, 1992

We present an algorithm, APD, that solves the distance version of the all-pairs-shortest-path problem for undirected, unweighted n-vertex graphs in time \(O(M(n) \log n)\), where \(M(n)\) denotes the time necessary to multiply two \(n \times n\) matrices of small integers (which is currently known to be \(o(n^{2.376})\)). We also address the problem of actually finding a shortest path between each pair of vertices and present a randomized algorithm that matches APD in its simplicity and in its expected running time. © 1995 Academic Press, Inc.

1. COMPUTING ALL DISTANCES

In the following let \(G\) be an undirected, unweighted, connected graph with vertex set \(\{1, 2, \ldots, n\}\) and adjacency matrix \(A\); i.e., \(A\) is a \(0 \times 1\) matrix with entries \(a_{ij} = 1\) iff vertices \(i\) and \(j\) are adjacent in \(G\). Let \(D\) denote the distance matrix of \(G\); i.e., \(d_{ij}\) is the number of edges on a shortest path joining vertices \(i\) and \(j\) in \(G\). Our first main result is the following:

**THEOREM 1.** Given the adjacency matrix \(A\) of an undirected, unweighted, connected \(n\)-vertex graph \(G\), the algorithm APD stated below correctly computes the distance matrix \(D\) of \(G\) in time \(O(M(n) \log n)\), where \(M(n)\) denotes the time necessary to multiply two \(n \times n\) matrices of small integers (which is currently known to be \(o(n^{2.376})\)).

**FUNCTION APD** \((A : n \times n 0 \times 1\) matrix) : \(n \times n\) integer matrix.

\[
\text{let } Z = A \cdot A
\]

\[
\text{let } B \text{ be an } n \times n 0 \times 1 \text{ matrix, where}
\]

\[
b_{ij} = 1 \text{ iff } i \neq j \text{ and } (a_{ij} = 1 \text{ or } z_{ij} > 0)
\]

\[
\text{if } b_{ij} = 1 \text{ for all } i \neq j \text{ then return } \text{n \times n matrix } D = 2B - A
\]

\[
\text{let } T = \text{APD}(B)
\]

\[
\text{return } \text{n \times n matrix } D, \text{ where}
\]

\[
d_{ij} = \begin{cases} 2t_{ij} & \text{if } x_{ij} = t_{ij} \cdot \text{degree}(j) \\ 2t_{ij} - 1 & \text{if } x_{ij} < t_{ij} \cdot \text{degree}(j) \end{cases}
\]

* Supported by NSF Presidential Young Investigator Award CCR-9058440. E-mail address: seidel@cs.berkeley.edu.

The following claims establish the correctness of the algorithm APD and prove the time bound stated in Theorem 1.

**CLAIM 1.** Let \(Z = A \cdot A\). There is a path of length 2 in \(G\) between vertices \(i\) and \(j\) iff \(z_{ij} > 0\).

**Proof.** There is a length-2 path joining \(i\) and \(j\) iff there is a vertex \(k\) adjacent to both \(i\) and \(j\), which is exactly the case if \(z_{ij} = \sum_{1 \leq k < n} a_{ik}a_{kj} > 0\).

Let \(G'\) be the simple undirected \(n\)-vertex graph obtained from \(G\) by connecting every two vertices \(i\) and \(j\) by an edge if there is a path of length 1 or 2 between \(i\) and \(j\) in \(G\). Note that the 0 \(\times 1\) matrix \(B\) computed in the algorithm is the adjacency matrix of \(G'\). \(G'\) is the complete graph iff \(G\) has diameter at most 2, and in that case \(d_{ij} = 2\) if \(a_{ij} = 0\) and \(d_{ij} = 1\) if \(a_{ij} = 1\). Thus the algorithm is correct for graphs of diameter at most 2.

Let \(t_{ij}\) denote the length of a shortest path joining \(i\) and \(j\) in \(G'\).

**CLAIM 2.** For any pair \(i, j\) of vertices, \(d_{ij}\) even implies \(d_{ij} = 2t_{ij}\), and \(d_{ij}\) odd implies \(d_{ij} = 2t_{ij} - 1\).

**Proof.** Observe that if for a pair \(i, j\) of vertices \(d_{ij} = 2s\) and \(i = i_0, i_1, \ldots, i_{2s-1}, i_{2s} = j\) is a shortest path in \(G\), then \(i = i_0, i_1, \ldots, i_{2s-1}, i_{2s} = j\) is a shortest path between \(i\) and \(j\) in \(G'\) and has length \(s\). Similarly, if \(d_{ij} = 2s - 1\) and \(i = i_0, i_1, \ldots, i_{2s-2}, i_{2s-1} = j\) is a shortest path in \(G\), then \(i = i_0, i_2, i_4, \ldots, i_{2s-2}, i_{2s-1} = j\) is a shortest path between \(i\) and \(j\) in \(G'\) and has length \(s\).

Thus after the \(t_{ij}\)'s have been computed recursively by \(\text{ADP}(B)\), one only needs to determine the parities of the \(d_{ij}\)'s in order to deduce their values from the respective \(t_{ij}\)'s. How those parities can be determined efficiently is shown by the following claims, the first of which is trivial.

**CLAIM 3.** Let \(i\) and \(j\) be a pair of distinct vertices in \(G\). For any neighbor \(k\) of \(j\) in \(G\) we have \(d_{ij} - 1 \leq d_{ik} \leq d_{ij} + 1\). Moreover, there exists a neighbor \(k\) of \(j\) with \(d_{ik} = d_{ij} - 1\).
Claim 4. Let \( i \) and \( j \) be a pair of distinct vertices in \( G \):
\[
\begin{align*}
d_{ij} \text{ even} & \implies t_{ik} \geq t_{ij} & \text{for all neighbors } k \text{ of } j \text{ in } G. \\
d_{ij} \text{ odd} & \implies t_{ik} \leq t_{ij} & \text{for all neighbors } k \text{ of } j \text{ in } G, \text{ and} \\
t_{ik} & \leq t_{ij} & \text{for some neighbor } k \text{ of } j \text{ in } G.
\end{align*}
\]

Proof. Assume that \( d_{ij} = 2s \) is even. Then, since by the last claim that \( d_{ik} \geq 2s - 1 \) for any neighbor \( k \) of \( j \), Claim 2 implies \( t_{ik} \geq s = t_{ij} \). Similarly, if \( d_{ij} = 2s - 1 \) is odd, then, since \( d_{ik} \leq 2s \) for any neighbor \( k \) of \( j \), we have \( t_{ik} \leq s = t_{ij} \) and \( d_{ik} = 2s - 2 \) for some neighbor \( k \) of \( j \). But for that neighbor \( t_{ik} = s - 1 < s = t_{ij} \) holds.

As a straightforward consequence of Claim 4 we have

Claim 5. \( d_{ij} \) even iff \( \sum_{k \text{ neighbor of } j} t_{ik} \geq \deg_G(j) \), and \( d_{ij} \) odd iff \( \sum_{k \text{ neighbor of } j} t_{ik} \leq \deg_G(j) \).

The correctness of the algorithm \textsc{APD} follows immediately, since \( \sum_{k \text{ neighbor of } j} t_{ik} = \sum_{1 \leq k \leq \deg_G(j)} a_{kj} = x_{ij} \).

Let \( f(n, \delta) \) be the running time of \textsc{APD} when applied to a graph \( G \) with \( n \) vertices and of diameter \( \delta \). Since the derived graph \( G' \) clearly has diameter \( \lceil \delta/2 \rceil \) we have
\[
f(n, \delta) = \begin{cases} M(n) + O(n^2) & \text{if } \delta \leq 2, \\ 2M(n) + O(n^2) + f(n, \lceil \delta/2 \rceil) & \text{if } \delta > 2, \end{cases}
\]

where \( M(n) \) denotes the time to multiply two \( n \times n \) matrices. For \( \delta > 1 \) the solution is
\[
f(n, \delta) = (2 \lceil \log_2 \delta \rceil - 1) \cdot M(n) + O(n^2 \log \delta).
\]

Since \( \delta \leq n - 1 \) and since \( M(n) = \Omega(n^2) \) this means that the running time of \textsc{APD} is \( O(M(n) \log n) \), which by the results on fast matrix multiplication by Coppersmith and Winograd \cite{Coppersmith1987} is \( O(n^{2.376}) \).

2. Computing Shortest Paths

Let us now consider the problem of computing for each pair of vertices in graph \( G \) a shortest connecting path, and not just the length of such a path. Again we deal only with the case where \( G \) is undirected, unweighted, and connected and has vertex set \( \{1, \ldots, n\} \).

Note that we cannot compute all shortest paths explicitly in \( O(n^2) \) time, since there are graphs with \( \Theta(n^2) \) pairs of vertices whose connecting paths have lengths \( \Theta(n) \) each. Thus we only compute a data structure that allows shortest connecting paths to be reconstructed in time proportional to their lengths. This data structure will be the so-called shortest path successor matrix \( S \), where for each vertex pair \( i \neq j \) the entry \( s_{ij} \) is a neighbor \( k \) of \( i \) that lies on a shortest path from \( i \) to \( j \).

Our strategy will be to compute the successor matrix \( S \) from the distance matrix \( D \). In particular, we will show that computing \( S \) from \( D \) essentially amounts to solving the boolean product witness matrix problem, which asks to compute for any given two \( n \times n \) \( 0 \)-1 matrices \( A \) and \( B \) an \( n \times n \) integer "witness" matrix \( W \) so that
\[
w_{ij} = \begin{cases} 1 \text{ if } k \text{ such that } a_{ik} = 1 \text{ and } b_{kj} = 1, \\ 0 \text{ if no such } k \text{ exists.} \end{cases}
\]

Theorem 2. Given the adjacency matrix \( A \) and the distance matrix \( D \) of an undirected, unweighted, connected \( n \)-vertex graph \( G \), a shortest path successor matrix of \( G \) can be computed by solving three instances of the boolean product witness matrix problem plus an additional \( O(n^2) \) time.

Proof. Suppose we have distance matrix \( D \) and adjacency matrix \( A \) of graph \( G \) at our disposal and we let \( i \) and \( j \) be two vertices with \( d_{ij} = d > 0 \). Then entry \( s_{ij} \) in the successor matrix will be some neighbor \( k \) of \( i \) with \( d_{ik} = d - 1 \). In other words, we want to find

some \( k \) such that \( (a_{ik} = 1) \text{ and } (d_{ij} = d - 1) \).

This means that determining the successors \( s_{ij} \) for all vertex pairs \( i, j \) with \( d_{ij} = d \) can be achieved by solving the boolean product witness matrix problem for \( A \) and \( B^{(d)} \), where \( A \) is the adjacency matrix of \( G \) and \( B^{(d)} \) is the \( n \times n \) \( 0 \)-1 matrix with \( b_{ij}^{(d)} = 1 \) iff \( d_{ij} = d \). Thus all entries of the successor matrix \( S \) can be found by solving a boolean product witness matrix problem for each \( d, 0 < d < n \).

Of course, solving \( n - 1 \) instances of this problem is too expensive. However, it suffices to deal with only three instances. The key observation is that since \( d_{ij} \leq d_{ij} + 1 \) for any neighbor \( k \) of \( i \) it suffices to find

some \( k \) such that \( (a_{ik} = 1) \text{ and } (d_{ij} = d - 1 \text{ mod } 3) \).

Thus for each \( r = 0, 1, 2 \) determining the successors \( s_{ij} \) for all vertex pairs \( i, j \) with \( d_{ij} \text{ mod } 3 = r \) can be achieved by solving the boolean product witness matrix problem for \( A \) and \( D^{(r)} \), where \( D^{(r)} \) is the \( n \times n \) \( 0 \)-1 matrix with \( d_{ij}^{(r)} = 1 \) iff \( d_{ij} + 1 \text{ mod } 3 = r \).

The function \textsc{APSP} below details our algorithm for finding all shortest paths. For the solution of the three instances of the boolean product witness matrix problem it uses the function \textsc{BPWM}, which is also outlined below and is analyzed in the next section. From that analysis we can conclude the following corollary to Theorem 2:

Corollary 1. If two \( n \times n \) matrices can be multiplied in time \( O(n^\omega) \), then \textsc{APSP} constructs shortest paths for all pairs of vertices in expected time \( O(n^\omega \log n) \) if \( \omega > 2 \) and in expected time \( O(n^2 \log^2 n) \) if \( \omega = 2 \).
FUNCTION $\text{APSP}(A: n \times n \ 0 \ 1 \text{ matrix})$: $n \times n$ successor matrix.

let $D := \text{APD}(A)$
for each $r = 0, 1, 2$ do
let $D^{(r)}$ be the $n \times n$ $0 \ 1$ matrix
with $d^{(r)}_{ij} = 1$ if $d_{ij} + 1 \mod 3 = r$
let $W^{(r)} := \text{BPWM}(A, D^{(r)})$
return $n \times n$ matrix $S$, where $s_{ij} = w^{(r)}_{ij}$, with $r = d_{ij} \mod 3$

FUNCTION $\text{BPWM}(A, B: n \times n \ 0 \ 1$ matrices)$: n \times n$ witness matrix.

let $W := -A \cdot B$
for each $d = 2^l$, where $l = 0, \ldots, \lceil \log_2 n \rceil - 1$ repeat
choose $d$ independent random numbers $k_1, k_2, \ldots, k_d$, drawn uniformly from $\{1, \ldots, n\}$
let $X$ be an $n \times d$ matrix with columns $k_1a_{k1}, \ldots, k_da_{kd}$, and $Y$
with rows $b_{k_1}, (1 \leq i \leq d)$
let $C = X \cdot Y$
for each $(i, j)$ s.t. $w_{ij} < 0$ and $c_{ij}$ is a witness for $(i, j)$
do $w_{ij} := c_{ij}$
for each $(i, j)$ s.t. $w_{ij} < 0$ do $w_{ij} :=$ some witness $k$ for $(i, j)$, found by trying each $k$
return $W$.

3. WITNESSES FOR 0–1 MATRIX PRODUCTS

Given two $n \times n$ 0–1 matrices $A$ and $B$ we say that index $k$ is a witness for the index pair $(i, j)$ if $a_{ik} = 1$ and $b_{kj} = 1$.

We say that an $n \times n$ integer matrix $W$ is a boolean product witness matrix for $A$ and $B$ if

$w_{ij} = \begin{cases} 0 & \text{if there is no witness for } (i, j), \\ \text{some witness } k \text{ for } (i, j) \text{ otherwise.} \end{cases}$

Above we describe a randomized algorithm $\text{BPWM}$ that computes a boolean product witness matrix. In its description we refer to column $k$ of a matrix $Z$ as $z^*_k$, to row $k$ as $z_{k*}$. The expression $A \cdot B$ denotes the normal matrix product between $A$ and $B$.

**Theorem 3.** If two $n \times n$ matrices can be multiplied in time $O(n^k)$, then $\text{BPWM}$ computes a boolean product witness matrix for two $n \times n$ matrices $A$ and $B$ in expected time $O(n^k \log n)$ if $k > 0$ and in expected time $O(n^2 \log^2 n)$ if $k = 0$.

Let us first argue that $\text{BPWM}$ correctly computes a witness matrix.

**Claim 6.** If $A$ and $B$ are $n \times n$ 0–1 matrices and $C = A \cdot B$, then for each $0 \leq i, j \leq n$ the entry $c_{ij}$ counts the number of witnesses for $(i, j)$.

**Proof.** Trivial, since $c_{ij} = \sum_{1 \leq k \leq n} a_{ik}b_{kj}$.

Thus if some entry $w_{ij}$ of matrix $W$ in $\text{BPWM}$ is zero, then there is not witness for pair $(i, j)$. Any initially negative $w_{ij}$ is explicitly reset to some witness for $(i, j)$. Since the last for each loop ensures that this happens to every negative $w_{ij}$, $\text{BPWM}(A, B)$ indeed returns a boolean product witness matrix for $A$ and $B$.

What about the running time of $\text{BPWM}$? For each $d = 2^l$ the body of the loop is executed $O(\log n)$ times and each execution involves the multiplication of an $n \times d$ with a $d \times n$ matrix plus additional $O(n^2)$ work (note that testing whether a number is a witness for $(i, j)$ can be done in constant time). The matrix multiplication can be performed in time $O(n^2d^\omega \log^c)$ (apply the $O(n^3)$ square matrix multiplication algorithm to $d \times d$ submatrices of $A$ and $B$ in turn) and thus dominates the running time. It follows that the time necessary to perform the entire first for each loop is $O(\log n)$ times

$$\sum_{0 \leq l \leq \lceil \log_2 n \rceil} O(n^2(2^l)^{\omega - 2}) = O(n^2) \sum_{0 \leq l \leq \lceil \log_2 n \rceil} 2^{kl} \omega - 2l,$$

which is $O(n^\omega \log n)$ if $\omega > 2$ and $O(n^2 \log^2 n)$ if $\omega = 2$. This is also the expected running time of the entire function $\text{BPWM}$, if we can show that the expected running time of the last for each loop is $O(n^\omega)$; i.e., for each pair $(i, j)$ the expected work is constant. For this it suffices to prove that for any $(i, j)$ for which a witness exists, the first for each loop fails to find a witness with probability at most $1/n$.

**Claim 7.** Let $A$ and $B$ be $n \times n$ 0–1 matrices, let $S$ be a sequence of $d$ integers $k_1, k_2, \ldots, k_d$, each between 1 and $n$, and let matrices $X$ and $Y$ be defined as in the algorithm $\text{BPWM}$ and let $C = X \cdot Y$. If for some pair $(i, j)$ exactly one index $k_j$ in $S$ is a witness for $(i, j)$, then $c_{ij} = k_j$.

**Proof.** If $k_j$ is the only index $k_j$ in $S$ so that $a_{ik_j} = 1$ and $b_{kj} = 1$, then $c_{ij} = \sum_{1 \leq k \leq d} k_a_{ik_j} b_{kj} = k_j$.

Let us now concentrate on some fixed pair $(i, j)$ for which witnesses exist, say, $c$ of them. The previous claim implies that if during one of the iterations of the big for each loop there is exactly one witness for $(i, j)$ among the randomly chosen indices $k_1, \ldots, k_d$, then a witness for $(i, j)$ is found and assigned to $w_{ij}$.

We now need to argue that it is very unlikely that this fails to happen. Consider the iterations for which $1/2 \leq c \leq \frac{1}{2} \leq n$ holds. The following claim implies that each of these iterations fails to produce a witness for $(i, j)$ with probability at most $1 - 1/2e$. Thus no witness is produced in all these iterations with probability at most $(1 - 1/2e)^{\approx 24 \log n^2} \leq 1/n$, and hence a witness for $(i, j)$ has to be found in the last for each loop with probability at most $1/n$, as claimed.

**Claim 8.** Let $I$ be a set of $n$ balls $c$ of which are colored crimson. Assume that $d$ times a ball is drawn from $I$ uniformly
at random and put back, where d satisfies n/2 ≤ ed ≤ n. Then
the probability that exactly once a crimson ball was drawn is
at least 1/2e.

Proof. The desired probability is \( d(c/n)(1-c/n)^{d-1} \). Since by
the assumptions on d we have \( dc/n \geq 1 \) and \(-c/n \geq -1/d\) it follows that
\( (dc/n)(1-c/n)^{d-1} \geq \frac{1}{2}(1-1/d)^{d-1} > \frac{1}{2}e^{-1} \).

4. DISCUSSION

Please note that our algorithms only involve integer
matrices\(^1\) whose entries are less than \( n^2 \). Thus the
\( O(n \log n) \) time bound holds for the usual RAM model
that assumes constant time primitive arithmetic and
comparison operations on integers whose values are polynomial
in \( n \). This is in contrast to previous methods [17, 16] that
solve the all-pairs-shortest-path problem by emulating so-called
“funny matrix multiplication” (i.e., matrix multiplication
over a semiring whose operations are \( \min + \) ) via
ordinary multiplication of matrices whose entries have
representation size not logarithmic, but superlinear in \( n \). See
Pan’s book [15, Theorems 18.10, 23.6].

The main algorithm APD is somewhat of a curiosity. It
applies to the case of unweighted, undirected graphs, but it
does not seem to admit ready generalization to the weighted
and/or directed case. Algorithm APD owes a lot to work by
Galil and Margalit [9], who were the first to achieve a sub-
stantially subcubic bound for a dense version of the all-
pairs-shortest-path problem. They also used the derived
graph \( G' \) but then employed a much more complicated
method to determine the parities of the \( d_i's \).

Randomized algorithms similar to BPWM have also
been discovered by Karger [12] and by Alon et al. [2]. The
latter group of authors has also managed to apply ran-
domization and has obtained a deterministic algorithm for
the Boolean product witness matrix problem with a worst
case running time of \( O(M(n \log n) \log \Theta(n)) \). Recently Dietz [4]
has obtained an \( o(M(n \log n)) \) expected time bound.

Since the classic results had been established in the early
sixties, research on the all-pairs-shortest-paths problem saw
relatively little action [6, 8, 11, 7] until interest surged in
the nineties [5, 13, 14, 1]. Galil and Margalit have been
undertaking a rather comprehensive study of the entire
area, and together with Alon and Naor they have achieved
a number of impressive results. In particular they [10] can
find distance matrices for undirected graphs with integer
edge weights with absolute values smaller than some con-
stant \( B \) in time \( O(n \log n) \). For directed graphs with
such edge weights they [1] can find the distance matrix
in time \( O(n^{\omega+3/2}) \). For \( B = 1 \) they [2] can also find
a successor matrix in time \( O(n \log n) \) for undirected graphs
and in time \( O(n^{\omega+3/2}) \) for directed graphs. Here \( O(f(n)) \)
denotes \( O(f(n) \log^{(O)(1))} n, \) the constant \( \omega \) < 2.376 denotes
the exponent for matrix multiplication, and the depend-
ence on \( B \) in the first two bounds is a small poly-
nomial.

REFERENCES

1. N. Alon, Z. Galil, and O. Margalit, On the exponent of the all pairs
shortest path problem, in “Proceedings, 32nd FOCS, 1991,”
pp. 569–575.
2. N. Alon, Z. Galil, O. Margalit, and M. Naor, Witnesses for Boolean
matrix multiplication and for shortest paths, in “Proceedings, 33rd
3. D. Coppersmith and S. Winograd, Matrix multiplication via
5. T. Feder and R. Motwani, Clique partitions, graphs compression
6. M. L. Fredman, New bounds on the complexity of the shortest path
7. M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in
(1987), 596–615.
8. D. B. Johnson, Efficient algorithms for shortest paths in sparse
9. Z. Galil and O. Margalit, On the exponent of the all pairs shortest path
problem for undirected graphs, manuscript, April 1991.
10. Z. Galil and O. Margalit, All pairs shortest distances for graphs with
small integer edge lengths, submitted.
Time bounds for all-pairs shortest paths, in “Proceedings, 32nd FOCS,
14. C. C. McGeoch, Finding shortest paths with the optimal subgraph,
manuscript, 1992.
15. V. Pan, “How to Multiply Matrices Faster,” Lecture Notes in
16. F. Roman, Shortest-path problem is not harder than matrix
17. G. Yuval, An algorithm for finding all shortest paths using \( n^{2.81} \)
155–156.

\(^1\) For APD it is actually not too hard to come up with a variant that only
uses Boolean matrix multiplication (no more than \( 4 \log_2 \delta \) – 3 of them)
plus \( O(n^2 \log \delta) \) overhead: use the Mod 3 trick of APSP.