On the All-Pairs-Shortest-Path Problem

Raimund Seidel*

Computer Science Division
University of California, Berkeley
Berkeley CA 94720

Abstract

The following algorithm 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 $o(n^{2.376})$):

Input: $n \times n$ 0-1 matrix A, the adjacency matrix of undirected, connected graph G

Output: $n \times n$ integer matrix D, with d_{ij} the length of a shortest path joining vertices i and j in G

function APD(A : $n \times n$ 0-1 matrix) : $n \times n$ integer matrix
  let $Z = A \cdot A$
  let B be an $n \times n$ 0-1 matrix, where $b_{ij} = 1$ iff $i \neq j$ **and** $(a_{ij} = 1$ **or** $z_{ij} > 0)$
  if $b_{ij} = 1$ for all $i \neq j$ **then return** $n \times n$ matrix $D = 2B - A$
  let $T = \mathsf{APD}(B)$
  let $X = T \cdot A$
  return $n \times n$ matrix D, where $d_{ij} = \begin{cases} 2t_{ij} & \text{if } x_{ij} \geq t_{ij} \cdot degree(j) \\ 2t_{ij} - 1 & \text{if } x_{ij} < t_{ij} \cdot degree(j) \end{cases}$

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.

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, and let d_{ij} denote the number of edges on a shortest path joining vertices i and j in G. In this section we show that the function APD() computes all d_{ij} correctly within the claimed time bound.

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 $z_{ij} = \sum_{1 \leq k \leq n} a_{ik} a_{kj} > 0$. ◻

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

24th ANNUAL ACM STOC - 5/92/VICTORIA, B.C., CANADA

Let G' be the simple undirected n-vertex graph obtained from G by connecting every two vertices i and j by an edge iff there is a path of length 1 or 2 between i and j in G. Note that the 0-1 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 $i = i_0, i_2, i_4, \ldots, i_{2s-2}, i_{2s} = j$ is a shortest path between i and j in G' and has length s. Similarly, if $d_{ij} = 2s-1$ and $i = i_0, i_1, \ldots, i_{2s-3}, i_{2s-2}, i_{2s-1} = j$ is a shortest path in G, then $i = i_0, i_2, i_4, \ldots, i_{2s-4}, i_{2s-2}, i_{2s-1} = j$ 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 APD(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 $d_{ij} - 1 \le d_{ik} \le d_{ij} + 1$. 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.

d_{ij} even $\Longrightarrow t_{ik} \ge t_{ij}$ for all neighbors k of j in G.

d_{ij} odd $\Longrightarrow t_{ik} \le t_{ij}$ for all neighbors k of j in G, and $t_{ik} < t_{ij}$ for some neighbor k of j in G.

Proof: Assume $d_{ij} = 2s$ is even. Then, since by the last claim $d_{ik} \ge 2s - 1$ for any neighbor k of j, Claim 2 implies $t_{ik} \ge s = t_{ij}$. Similarly, if $d_{ij} = 2s - 1$ is odd, then, since $d_{ik} \le 2s$ for any neighbor k of j, we have $t_{ik} \le s = t_{ij}$, 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 $\sum_{k \text{ neighbor of } j} t_{ik} \ge t_{ij} \cdot degree_G(j)$, and

d_{ij} odd iff $\sum_{k \text{ neighbor of } j} t_{ik} < t_{ij} \cdot degree_G(j)$.

The correctness of the algorithm APD follows immediately, since $\sum_{k \text{ neighbor of } j} t_{ik} = \sum_{1 \le k \le n} t_{ik} a_{kj} = x_{ij}$.

Let $f(n, \delta)$ be the running time of APD when applied to a graph G with n vertices and of diameter δ. Since the derived graph G' clearly has diameter $\lceil \delta/2 \rceil$ we have

$$f(n,\delta) = \begin{cases} M(n) + O(n^2) & \text{if } \delta \le 2, \\ 2M(n) + O(n^2) + f(n, \lceil \delta/2 \rceil) & \text{if } \delta > 2, \end{cases}$$

where $M(n)$ denotes the time to multiply two $n \times n$ matrices. For $\delta > 1$ this solves to

$$f(n, \delta) = (2\lceil \log_2 \delta \rceil - 1) \cdot M(n) + O(n^2 \log \delta).$$

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

2. Computing All 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 only deal with the case where G is undirected, unweighted, and connected, and has vertex set $\{1, \ldots, n\}$.

Note that we cannot compute all those shortest paths explicitly in $o(n^3)$ 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 "successor" matrix S, where for each vertex pair $i \ne j$ 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 three instances of 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

$$w_{ij} = \begin{cases} \text{some } k \text{ such that } a_{ik} = 1 \text{ and } b_{kj} = 1, \text{ and} \\ 0 \text{ iff no such } k \text{ exists.} \end{cases}$$

Now assume that we have distance matrix D and adjacency matrix A of graph G at our disposal, and let i and j be two vertices with $d_{ij} = d > 0$. The entry s_{ij} in the successor matrix will be some neighbor k of i with $d_{kj} = d - 1$. In other words, we want to find

some k such that $(a_{ik} = 1)$ and $(d_{kj} = d - 1)$.

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 $b^{(d)}_{\mu\nu} = 1$ iff $d_{\mu\nu} = d - 1$. 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 $d_{ij} - 1 \le d_{kj} \le d_{ij} + 1$ for any neighbor k of i it suffices to find

some k such that $(a_{ik} = 1)$ and $(d_{kj} \equiv d-1 \pmod 3)$.

Thus for each $r = 0, 1, 2$ determining the successors s_{ij} for all vertex pairs i, j with $d_{ij} \bmod 3 = r$ 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 $d^{(r)}_{\mu\nu} = 1$ iff $d_{\mu\nu} + 1 \bmod 3 = r$.

```
Function APSP(A: n × n 0-1 matrix): n × n successor matrix
    let D := APD(A)
    for each r = 0,1,2 do
        let D^(r) be the n × n 0-1 matrix with d_ij^(r) = 1 iff d_ij + 1 mod 3 = r
        let W^(r) := BPWM(A, D^(r))
    return n × n matrix S, where s_ij = w_ij^(ρ), with ρ = d_ij mod 3
```

```
Function BPWM(A, B: n × n 0-1 matrices): n × n witness matrix
    let W := −A · B
    for each d = 2^ℓ where ℓ = 0, ..., ⌈log_2 n⌉ − 1 repeat ⌈3.42 · log_2 n⌉ times
        choose d independent random numbers k_1, k_2, ..., k_d, drawn uniformly from {1, ..., n}
        let X be an n×d matrix with columns k_i a_{*k_i} and Y a d×n matrix with rows b_{k_i,*} (1≤i≤d)
        let C = X · Y
        for each (i, j) s.t. w_ij < 0 and c_ij is a witness for (i, j) do w_ij := c_ij
    foreach (i, j) s.t. w_ij < 0 do w_ij := some witness k for (i, j), found by trying each k
    return W.
```

The function APSP above 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 BPWM, which is also outlined above and is analyzed in the next section. From that analysis we can conclude that if two $n \times n$ matrices can be multiplied in time $O(n^\omega)$, then APSP constructs shortest paths for all pairs of vertices in expected time $O(n^\omega \log n)$ if $\omega > 2$, and in time $O(n^2 \log^2 n)$ if $\omega = 2$.

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) iff $a_{ik} = 1$ and $b_{kj} = 1$. We say that an $n \times n$ integer matrix W is a *boolean product witness matrix* for A and B iff

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

Above we give the description of a randomized algorithm that computes a boolean product witness matrix for A and B in expected time $O(n^\omega \log n)$, assuming that the time necessary to multiply two $n \times n$ small integer matrices is $O(n^\omega)$, with $\omega > 2$. (If $\omega = 2$, the expected running time of our algorithm is $O(n^2 \log^2 n)$.)

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.

Let us first argue that 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 $0 \le i, j \le n$ the entry c_{ij} counts the number of witnesses for (i, j).

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

Thus if some entry w_{ij} of matrix W in BPWM is zero, then there is no 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 assures that this happens to every negative w_{ij}, BPWM(A, B) indeed returns a boolean product witness matrix for A and B.

What about the running time of BPWM? For each $d = 2^\ell$ the body of the big 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 $O(n^2 d^{\omega-2})$ (apply the $O(n^\omega)$ square matrix mulplication 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

$$\sum_{0 \le \ell < \lceil \log_2 n \rceil} O(n^2 (2^\ell)^{\omega-2}) = O(n^2) \sum_{0 \le \ell < \lceil \log_2 n \rceil} 2^{\ell(\omega-2)},$$

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 BPWM, if we can show that the expected running time of the last **for each** loop is $O(n^2)$, 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 BPWM and let $C = X \cdot Y$.

If for some pair (i,j) exactly one index k_λ in S is a witness for (i,j), then $c_{ij} = k_\lambda$.

Proof: If k_λ is the only index k_ν in S so that $a_{ik_\nu} = 1$ and $b_{k_\nu j} = 1$, then $c_{ij} = \sum_{1 \le \nu \le d} k_\nu a_{ik_\nu} b_{k_\nu j} = k_\lambda$. □

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 numbers $k_1, \ldots, k_d$, 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 $n/2 \le cd \le n$ 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 $(1 - 1/2e)^{\lceil 3.42 \log_2 n \rceil} \le 1/n$, 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 $n/2 \le cd \le n$.

Then the probability that exactly once a crimson ball was drawn is at least $1/2e$.

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

4. Discussion

Please note that our algorithms only involve integer matrices* whose entries are less than n^2. Thus the $O(M(n)\log n)$ 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 [15, 14] 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 and +) via ordinary multiplication of matrices whose entries have representation size not logarithmic, but superlinear in n. See Pan's book [13, 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 [8], who were the first to achieve a substantially 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_{ij}'s. Initially [8] they had an algorithm for the undirected, unweighted case with running time $O(n^{2+\omega/3})$, assuming $M(n) = o(n^\omega)$. In collaboration with Alon [1] they subsequently altered their approach and improved their result to $O(n^{(3+\omega)/2})$, also generalizing it to the case of directed graphs with edgeweights in $\{-1, 0, +1\}$.

Alon, Galil, Margalit [1], together with Karger, Koller, Phillips [11], and Feder, Motwani [3] have to be credited for the recent resurgence of interest in all-pairs-shortest-path problems, an area that has seen relatively little action [4, 6, 9, 5] since the classic results had been established in the early sixties. Alon, Galil, and Margalit seem to have embarked on a rather comprehensive investigation of the entire area and apparently have obtained a number of impressive new results [7]: in particular, an algorithm that solves the distance verstion of the all-pairs-shortest-path problem on undirected graphs with integer edge weights between 0 and B in time $O(B^2 M(n) \log n)$;† a randomized algorithm for the boolean product witness matrix problem similar to BPWM, although slightly slower, and very recently also a deterministic algorithm with comparable worst case running time.

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

†We should point out that APSP can easily be adapted to compute the successor matrix from the distance matrix also in this case, adding a multiplicative factor of B^2 to its running time. David Karger [10] has obtained a similar result, but not based on boolean product witness matrices.

References

[1] N. Alon, Z. Galil, and O. Margalit, *On the Exponent of the All Pairs Shortest Path Problem*, Proc. 32nd FOCS (1991), 569–575.

[2] D. Coppersmith and S. Winograd, *Matrix Multiplication via Arithmetic Progressions*, J. of Symbolic Computation 9 (1990), 251–280.

[3] T. Feder and R. Motwani, *Clique Partitions, Graphs Compression and Speeding-up Algorithms*, Proc. STOC (1991), 123–133.

[4] M. L. Fredman, *New Bounds on the Complexity of the Shortest Path Problem*, SIAM J. Comp. 5 (1976), 83–89.

[5] M. L. Fredman and R. E. Tarjan, *Fibonacci Heaps and thier Uses in Improved Network Optimization Algorithms*, JACM (1987), 596–615.

[6] D. B. Johnson, *Efficient Algorithms for Shortest Paths in Sparse Networks*, JACM (1977), 1–13.

[7] Z. Galil, *Private Communication.*

[8] Z. Galil and O. Margalit, *On the Exponent of the All Pairs Shortest Path Problem*, Manuscript, April 1991.

[9] H. N. Gabow, *Scaling Algorithms for Network Problems*, J. Comput. System Sci. 31 (1985), 148–168.

[10] D. R. Karger, *Private Communication.*

[11] D. R. Karger, D. Koller, and S. J. Phillips, *Finding the Hidden Path: Time Bounds for All-Pairs Shortest Paths*, Proc. 32nd FOCS (1991), 560–568.

[12] C. C. McGeoch, *Finding Shortest Paths with the Optimal Subgraph*, Manuscript, 1992.

[13] V. Pan, **How to Multiply Matrices Faster**, Springer Lecture Notes in Computer Science 179 (1984).

[14] F. Romani, *Shortest-Path Problem is not Harder than Matrix Multiplication*, IPL 11 (1980) 134–136.

[15] G. Yuval, *An Algorithm for Finding All Shortest Paths Using $N^{2.81}$ Infinite-Precision Multiplications*, IPL 4 (1976) 155–156.