Matrix Multiplication and All Pairs Shortest Paths (2002; Zwick)

Tadao Takaoka, University of Canterbury
www.cosc.canterbury.ac.nz/tad.takaoka

INDEX TERMS: all pairs shortest path problem, matrix multiplication, witness, bridging set, two-phase algorithm

SYNONYMS: shortest path problem, algorithm analysis

1 PROBLEM DEFINITION

The all pairs shortest path (APSP) problem is to compute shortest paths between all pairs of vertices of a directed graph with non-negative real numbers as edge costs. We focus on shortest distances between veritices, as shortest paths can be obtained with a slight increase of cost. Classically the APSP problem can be solved in cubic time of $O(n^3)$. The problem here is to achieve a sub-cubic time for a graph with small integer costs.

A directed graph is given by $G = (V, E)$, where $V = \{1, \cdots, n\}$, the set of vertices, and E is the set of edges. The cost of edge $(i, j) \in E$ is denoted by d_{ij}. The (n, n)-matrix D is one whose (i, j) element is d_{ij}. We assume that $d_{ij} \geq 0$ and $d_{ii} = 0$ for all i, j. If there is no edge from i to j, we let $d_{ij} = \infty$. The cost, or distance, of a path is the sum of costs of the edges in the path. The length of a path is the number of edges in the path. The shortest distance from vertex i to vertex j is the minimum cost over all paths from i to j, denoted by d^*_{ij}. Let $D^* = \{d^*_{ij}\}$. We call n the size of the matrices.

Let A and B are (n, n)-matrices. The three products are defined using the elements of A and B as follows: (1) Ordinary matrix product over a ring $C = AB$, (2) Boolean matrix product $C = A \cdot B$, and (3) Distance matrix product $C = A \times B$, where

$$(1)\ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \qquad (2)\ c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}, \qquad (3)\ c_{ij} = \min_{1 \leq k \leq n} \{a_{ik} + b_{kj}\}.$$

The matrix C is called a product in each case; the computational process is called multiplication, such as distance matrix multiplication. In those three cases, k changes through the entire set $\{1, ..., n\}$. We define a partial matrix product of A and B by taking k in a subset I of V. In other words, a partial product is obtained by multiplying a vertically rectangular matrix, $A(*, I)$, whose columns are extracted from A corresponding to the set I, and similarly a horizontally rectangular matrix, $B(I, *)$, extracted from B with rows corresponding to I. Intuitively I is the set of check points k, when we go from i to j.

The best algorithm [3] computes (1) in $O(n^\omega)$ time, where $\omega = 2.376$. We carry three decimal points. To compute (2), we can regard Boolean values 0 and 1 in A and B as integers and use the algorithm for (1), and convert non-zero elements in the resulting matrix to 1. Therefore this complexity is $O(n^\omega)$. The witnesses of (2) are given in the witness matrix $W = \{w_{ij}\}$ where $w_{ij} = k$ for some k such that $a_{ik} \wedge b_{kj} = 1$. If there is no such k, $w_{ij} = 0$. The witness matrix $W = \{w_{ij}\}$ for (3) is defined by $w_{ij} = k$ that gives the minimum to c_{ij}. If we have an algorithm for (3) with $T(n)$ time, ignoring a polylog factor of n, we can solve the APSP problem in $\tilde{O}(T(n))$ time by the repeated squaring method, described as the repeated use of $D \longleftarrow D \times D$ $O(\log n)$ times.

Our definition of computing shortest paths is to give a witness matrix of size n by which we can give a shortest path from i to j in $O(\ell)$ time where ℓ is the length of the path. More specifically, if $w_{ij} = k$ in the witness matrix $W = \{w_{ij}\}$, it means that the path from i to j goes through k. Therefore a recursive function $path(i,j)$ is defined by ($path(i,k)$, k, $path(k,j)$) if $path(i,j) = k > 0$ and nil if $path(i,j) = 0$, where a path is defined by a list of vertices excluding endpoints. In the following sections, we record k in w_{ij} whenever we can find k such that a path from i to j is modified or newly set up by paths from i to k and from k to j. We introduce preceding results as a framework for the key results.

Alon-Galil-Margalit algorithm

We review the algorithm in [1]. Let the costs of edges of the given graph be ones. Let $D^{(\ell)}$ be the ℓ-th approximate matrix for D^* defined by $d_{ij}^{(\ell)} = d_{ij}^*$ if $d_{ij}^* \leq \ell$, and $d_{ij}^{(\ell)} = \infty$ otherwise. Let A be the adjacency matrix of G, that is, $a_{ij} = 1$ if there is an edge (i,j), and $a_{ij} = 0$ otherwise. Let $a_{ii} = 1$ for all i. The algorithm consists of two phases.

In the first phase, $D^{(\ell)}$ is computed for $\ell = 1, ..., r$, by checking the (i,j)-element of $A^\ell = \{a_{ij}^\ell\}$. Note that if $a_{ij}^\ell = 1$, there is a path from i to j of length ℓ or less. Since we can compute Boolean mutlix multiplication in $O(n^\omega)$ time, the computing time of this part is $O(rn^\omega)$.

In the second phase, the algorithm computes $D^{(\ell)}$ for $\ell = r$, $\lceil \frac{3}{2}r \rceil$, $\left\lceil \frac{3}{2} \lceil \frac{3}{2}r \rceil \right\rceil$, $\cdots$, n' by repeated squaring, where n' is the smallest integer in this sequence of ℓ such that $\ell \geq n$. Let $T_\alpha = \{j | d_{ij}^{(\ell)} = \alpha\}$, and $I_i = T_\alpha$ such that $|T_\alpha|$ is minimum for $\lceil \ell/2 \rceil \leq \alpha \leq \ell$. The key observation in the second phase is that we only need to check k in I_i whose size is not larger than $2n/\ell$, since the correct distances between $\ell + 1$ and $\lceil 3\ell/2 \rceil$ can be obtained as the sum $d_{ik}^{(\ell)} + d_{kj}^{(\ell)}$ for some k satisfying $\lceil \ell/2 \rceil \leq d_{ik}^{(\ell)} \leq \ell$. The meaning of I_i is similar to I for partial products except that I varies for each i. Hence the computing time of one squaring is $O(n^3/\ell)$. Thus the time of the second phase is given with $N = \lceil \log_{3/2} n/r \rceil$ by $O(\sum_{s=1}^N n^3/((3/2)^s r)) = O(n^3/r)$. Balancing the two phases with $rn^\omega = n^3/r$ yields $O(n^{(\omega+3)/2}) = O(n^{2.688})$ time for the algorithm with $r = O(n^{(3-\omega)/2})$.

Witnesses can be kept in the first phase in time polylog of n by the method in [2]. The maintenance of witnesses in the second phase is straightforward.

When we have a directed graph G whose edge costs are between 1 and M where M is a positive integer, we can convert the graph G to G' by replacing each edge by up to M edges with unit cost. Obviously we can solve the problem for G by applying the above algorithm to G', which takes $O\left((Mn)^{(\omega+3)/2}\right)$ time. This time is sub-cubic when $M < n^{0.116}$. The maintenance of witnesses has an extra polylog factor in each case.

Takaoka algorithm

When the edge costs are bounded by a positive integer M, we can do better than we saw in the above. We briefly review Romani's algorithm [6] for distance matrix multiplication.

Let A and B be (n, m) and (m, n) distance matrices whose elements are bounded by M or infinite. Let the diagonal elements be 0. Then we convert A and B into A' and B' where $a'_{ij} = (m+1)^{M-a_{ij}}$, if $a_{ij} \neq \infty$, 0 otherwise, and $b'_{ij} = (m+1)^{M-b_{ij}}$, if $b_{ij} \neq \infty$, 0 otherwise.

Let $C' = A'B'$ be the product by ordinary matrix multiplication and $C = A \times B$ be that by distance matrix multiplication. Then we have

$$c'_{ij} = \sum_{k=1}^{m} (m+1)^{2M-(a_{ik}+b_{kj})}, \ c_{ij} = 2M - \lfloor \log_{m+1} c'_{ij} \rfloor.$$

We call this distance mutrix multiplication (n, m)-Romani. In this section we use the above multiplication with square matrices, that is, we use (n, n)-Romani. In the next section, we deal with the case where $m < n$.

We can compute C with $O(n^\omega)$ arithmetic operations on integers up to $(n+1)^M$. Since these values can be expressed by $O(M \log n)$ bits and Schönhage and Strassen's algorithm [7] for multiplying k-bit numbers takes $O(k \log k \log \log k)$ bit operations, we can compute C in $O(n^\omega M \log n \log(M \log n) \log \log(M \log n))$ time, or $\tilde{O}(Mn^\omega)$ time.

We replace the first phase by the one based on (n, n)-Romani, and modify the second phase based on path lengths, not distances.

Note that the bound M is replaced by ℓM in the distance matrix multiplication in the first phase. Ignoring polylog factors, the time for the first phase is given by $\tilde{O}(n^\omega r^2 M)$. We assume that M is $O(n^k)$ for some constant k. Balancing this complexity with that of second phase, $O(n^3/r)$, yields the total computing time of $\tilde{O}(n^{(6+\omega)/3} M^{1/3})$ with the choice of $r = O(n^{(3-\omega)/3} M^{-1/3})$. The value of M can be almost $O(n^{0.624})$ to keep the complexity within sub-cubic.

2 KEY RESULTS

Zwick improved the Alon-Galil-Margalit algorithm in several ways. The most notable is an improvement of the time for the APSP problem with unit edge costs from $O(n^{2.688})$ to $O(n^{2.575})$. The main accelerating engine in Alon-Galil-Margalit [1] was the fast Boolean matrix multiplication and that in Takaoka [8] was the fast distance matrix multiplication by Romani, both powered by the fast matrix multiplication of square matrices.

In this section, the engine is the fast distance matrix multiplication by Romani powered by the fast matrix multiplication of rectangular matrices given by Coppersmith [4], and Huang and Pan [5]. Suppose the product of (n, m) matrix and (m, n) matrix can be computed with $O(n^{\omega(1,\mu,1)})$ arithmetic operations, where $m = n^\mu$ with $0 \leq \mu \leq 1$. Several facts such as $O(n^{\omega(1,1,1)}) = O(n^{2.376})$ and $O(n^{\omega(1,0.294,1)}) = \tilde{O}(n^2)$ are known. To compute the product of (n, n) square matrices we need $n^{1-\mu}$ matrix multiplications, resulting in $O(n^{\omega(1,\mu,1)+1-\mu})$ time, which is reformulated as $O(n^{2+\mu})$, where μ satisfies the equation $\omega(1, \mu, 1) = 2\mu + 1$. Currently the best known value for μ is $\mu = 0.575$, so the time becomes $O(n^{2.575})$, which is not as good as $O(n^{2.376})$. So we use the algorithm for rectangular matrices in the following.

We incorporate the above algorithm into (n, m)-Romani with $m = n^\mu$ and $M = n^t$, and the computing time of $\tilde{O}(Mn^{\omega(1,\mu,1)})$. The next step is how to incorporate (n, m)-Romani into the APSP algorithm. The first algorithm is a mono-phase algorithm based on repeated squaring, similar to the second phase of the algorithm in [1]. To take advantage of rectangular matrices in (n, m)-Romani, we need the following definition of the bridging set, which plays the role of the set I in the partial distance matrix product in Section 1.

Let $\delta(i, j)$ be the shortest distance from i to j, and $\eta(i, j)$ be the minimum length of all shortest paths from i to j. A subset I of V is an ℓ-bridging set if it satisfies the condition that if $\eta(i, j) \geq \ell$, there exists $k \in I$ such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$. I is a strong ℓ-bridging set if it satisfies the condition that if $\eta(i, j) \geq \ell$, there exists $k \in I$ such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$ and $\eta(i, j) = \eta(i, k) + \eta(k, j)$. Note that those two sets are the same for a graph with unit edge costs.

We note that if $(2/3)\ell \leq \mu(i, j) \leq \ell$ and I is a strong $\ell/3$-bridging set, there is a $k \in I$ such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$ and $\mu(i, j) = \mu(i, k) + \mu(k, j)$. With this property of strong bridging sets, we can use (n, m)-Romani for the APSP problem in the following way. By repeated squaring in a similar way to Alon-Galil-Margalit, the algorithm computes $D^{(\ell)}$ for $\ell = 1, \lceil \frac{3}{2} \rceil, \left\lceil \frac{3}{2} \lceil \frac{3}{2} \rceil \right\rceil, \cdots, n'$, where n' is the first value of ℓ that exceeds n, using a various types of set I described below. To compute the bridging set, the algorithm maintains the witness matrix with extra polylog factor in the complexity. In [9], there are three ways for selecting the set I. Let $|I| = n^r$ for some r sucn that $0 \leq r \leq 1$.

(1) Select $9n \ln n/\ell$ vertices from V at random. In this case we can show that the algorithm solves the APSP problem with high probability, i.e., with $1 - 1/n^c$ for some constant $c > 0$. In our case $c = 3$. In other words, I is a strong $\ell/3$-bridging set with high probability. The time T is dominated by (n, m)-Romani. We have $T = \tilde{O}(\ell M n^{\omega(1,r,1)})$, since the magnitude of matrix elements can be up to ℓM. Since $m = O(n \ln n/\ell) = n^r$, we have $\ell = \tilde{O}(n^{1-r})$, and thus $T = O(Mn^{1-r} n^{\omega(1,r,1)})$. When $M = 1$, this bound on r is $\mu = 0.575$, and thus $T = O(n^{2.575})$. When $M = n^t \geq 1$, the time becomes $O(n^{2+\mu(t)})$, where $t \leq 3 - \omega = 0.624$ and $\mu = \mu(t)$ satisfies $\omega(1, \mu, 1) = 1 + 2\mu - t$. It is determined from the best known $\omega(1, \mu, 1)$ and the value of t. As the result is correct with high probability, this is a randomized algorithm.
(2) We consider the case of unit edge costs here. In (1), the computation of witnesses is an extra thing, i.e., not necessary if we need only shortest distances. If we want to achieve the same complexity in the sense of an exact algorithm, not a randomized one, the computation of witnesses is essential. As mentioned earlier, maintenance of witnesses, that is, matrix W, can be done with an extra polylog factor, meaning the analysis can be focused on Romani within the $\tilde{O}$-notation. Specifically we select I as an $\ell/3$-bridging set, which is strong with unit edge costs. To compute I as an $O(\ell)$-bridging set, we obtain the vertices on the shortest path from i to j for each i and j using the witness matrix W in $O(\ell)$ time. After obtaining those n^2 sets spending $O(\ell n^2)$ time, it is shown in [9] how to obtain a $O(\ell)$-bridging set of $O(n \ln n/\ell)$ size within the same time complexity. The process of obtaining the bridging set must stop at $\ell = n^{1/2}$ as the process is too expensive beyond this point, and thus we use the same bridging set beyond this point. The time before this point is the same as that in (1), and that after this point is $\tilde{O}(n^{2.5})$.
(3) When edge costs are positive and bounded by $M = n^t > 0$, we can use a similar procedure to compute an $O(\ell)$-bridging set of $O(n \ln n/\ell)$ size in $\tilde{O}(\ell n^2)$ time. Using the brdging set, we can solve the APSP problem in $\tilde{O}(n^{2+\mu(t)})$ time in a similar way to (1). The result can be generalized into the case where edge costs are between $-M$ and M within the same time

complexity by modifying the procedure for computing an ℓ-bridging set, provided there is no negative cycle. The details are shown in [9].

3 APPLICATIONS

The eccentricity of a vertex v of a graph is the greatest distance from v to any other vertices. The diameter of a graph is the greatest eccentricity of any vertices. In other words, the diameter is the greatest distance between any pair of vertices.

The center of a graph is the vertex with the smallest eccentricity. It is obvious that those problems can be solved through the APSP problem. The center is mainly defined on an undirected graph. It is related to the facility location problem, since the center is the best place to locate a facility that serves the community modelled by a graph.

4 OPEN PROBLEMS

We state two major challenges here among others. The first is to improve the complexity of $\tilde{O}(n^{2.575})$ for the APSP with unit edge costs. The other is to improve the bound of $M < O(n^{0.624})$ for the complexity of the APSP with integer costs up to M to be sub-cubic.

5 RECOMMENDED READING

[1] N. Alon, Z. Galil and O. Margalit, On the exponent of the all pairs shortest path problem, Proc. 32th IEEE FOCS (1991) 569–575. Also JCSS 54 (1997) 255-262.

[2] N. Alon, Z. Galil, and O. Margalit and M. Naor, Witnesses for Boolean matrix multiplication and for shortest paths, Proc. 33th IEEE FOCS (1992) 417–426.

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

[4] D. Coppersmith, Rectangular matrix multiplication revisited, Jour. Complex. 13 (1997) 42-49.

[5] X. Huang and V. Y. Pan, Fast rectangular matrix multiplications and applications, Jour. Complex. 14 (1998) 257-299.

[6] F. Romani, Shortest-path problem is not harder than matrix multiplications, Info. Proc. Lett. 11 (1980) 134–136.

[7] A. Schönhage and V. Strassen, Schnelle Multiplikation Großer Zahlen, Computing 7 (1971) 281–292.

[8] T. Takaoka, Sub-cubic time algorithms for the all pairs shortest path problem, Algorithmica, 20 (1998) 309–318

[9] U. Zwick, All pairs shortest paths using bridging sets and rectangular matrix multiplication, Jour. ACM, 49, 3 (2002) 289–317.