Improved Algorithm for All Pairs Shortest Paths

Yijie Han

School of Computing and Engineering
University of Missouri at Kansas City
5100 Rockhill Road
Kansas City, MO 64110
hanyij@umkc.edu
http://welcome.to/yijiehan

Abstract

We present an improved algorithm for all pairs shortest paths. For a graph of n vertices our algorithm runs in $O(n^3(\log\log n/\log n)^{5/7})$ time. This improves the best previous result which runs in $O(n^3(\log\log n/\log n)^{1/2})$ time.

Keywords: Algorithms, complexity, graph algorithms, shortest path.

1 Introduction

Given an input directed graph $G = (V, E)$, the all pairs shortest path problem (APSP) is to compute the shortest paths between all pairs of vertices of G assuming that edge costs are nonnegative real values. The APSP problem is a fundamental problem in computer science and has received considerable attention. Early algorithms such as Floyd's algorithm ([2], pp. 211-212) computes all pairs shortest paths in $O(n^3)$ time, where n is the number of vertices of the graph. Improved results show that all pairs shortest paths can be computed in $O(mn + n^2 \log n)$ time [5], where m is the number of edges of the graph. Recently Pettie showed [7] an algorithm with time complexity of $O(mn + n^2 \log\log n)$. There are also results for all pairs shortest paths for graphs with integer weights[6, 8, 10, 11]. Fredman gave the first subcubic algorithm [4] for all pairs shortest paths. His algorithm runs in $O(n^3(\log\log n/\log n)^{1/3})$ time. Later Takaoka improved the upper bounds for all pairs shortest paths to $O(n^3(\log\log n/\log n)^{1/2})$ [9]. Dobosiewicz [3] gave an upper bound of $O(n^3/(\log n)^{1/2})$ with extended operations such as normalization capability of floating point numbers in $O(1)$ time.

As shown in ([1] pp. 202-206) the time complexity of distance matrix multiplication (DMM) is asymptotically equal to that of the APSP problem. In this paper we show that the DMM can be solved in $O(n^3(\log\log n/\log n)^{5/7})$ time. Thus APSP problem for directed graphs with real edge weights can be solved also in $O(n^3(\log\log n/\log n)^{5/7})$ time.

The computation model we used for our algorithm is the conventional RAM model. In particular we assume that the bit-wise OR of two $O(\log n)$ bit integers and indexing into a $O(n)$ size table with an $O(\log n)$-bit integer take constant time.

2 Matrix Multiplication Decomposition

Let A and B be $n \times n$ matrices whose components are nonnegative real numbers. The distance matrix multiplication $C = AB$ is defined as

$$c_{ij} = \min_{1 \le k \le n} \{a_{ik} + b_{kj}\}, i, j = 1, 2, ..., n.$$

We decompose A into $n_1 = n/t$ $n \times t$ submatrices:

$$A = [A_1 \ A_2 \ ... \ A_{n_1}] .$$

We also decompose B into n_1 $t \times n$ submatrices:

$$B = \begin{bmatrix} B_1 \\ B_2 \\ ... \\ B_{n_1} \end{bmatrix} .$$

Then $C = \min\{A_1 B_1, A_2 B_2, ..., A_{n_1} B_{n_1}\}$.

Note here that each $A_i B_i$ is a $n \times n$ matrix and min is taken component-wise. Let the time complexity of computing $A_i B_i$ be $T_1(n,t)$, then the time complexity for DMM is $n^3/t + nT_1(n,t)/t$.

Now let E be an $n \times t$ matrix and F be an $t \times n$ matrix. We further decompose E into $n_2 = n/h$ $h \times t$ matrices:

$$E = \begin{bmatrix} E_1 \\ E_2 \\ ... \\ E_{n_2} \end{bmatrix} .$$

We also decompose F into n_2 $t \times h$ matrices:

$$F = [F_1 \ F_2 \ ... \ F_{n_2}] .$$

Now $D = EF$ can be decomposed into n_2^2 $h \times h$ matrices:

$$D = \begin{bmatrix} D_{11} & D_{12} & ... & D_{1n_2} \\ D_{21} & D_{22} & ... & D_{2n_2} \\ ... & ... & ... & ... \\ D_{n_2 1} & D_{n_2 2} & ... & D_{n_2 n_2} \end{bmatrix} ,$$

where $D_{ij} = E_i F_j$.

Let the time complexity of E_iF_j be $T_2(h,t)$, then $T_1(n,t) = (n/h)^2T_2(h,t)$. Therefore the time complexity for DMM is $n^3/t + (n^3/h^2)T_2(h,t)/t$. We shall show that for $k = c(\log n/\log\log n)^{1/7}$, where c is a suitable constant, $t = k^4$ and $h = 2^{4k^2\log t}$, we have that $T_2(h,t) = O(h^2)$, and therefore the upper bound of time complexity of DMM becomes $O(n^3(\log\log n/\log n)^{4/7})$. In the following we only consider the computation of E_iF_j.

Now let G be an $h \times t$ matrix and H be an $t \times h$ matrix. We further decompose G into $n_3 = h/k$ $k \times t$ matrices:

$$G = \left[\begin{array}{c} G_1 \\ G_2 \\ \dots \\ G_{n_3} \end{array}\right].$$

We also decompose H into n_3 $t \times k$ matrices:

$$H = [H_1\ H_2\ \dots\ H_{n_3}].$$

Now $L = GH$ can be decomposed into n_3^2 $k \times k$ matrices:

$$L = \left[\begin{array}{cccc} L_{11} & L_{12} & \dots & L_{1n_3} \\ L_{21} & L_{22} & \dots & L_{2n_3} \\ \dots & \dots & \dots & \dots \\ L_{n_31} & L_{n_32} & \dots & L_{n_3n_3} \end{array}\right],$$

where $L_{ij} = G_iH_j$.

We will perform computations of G_iH_j. We shall show that based on the encoding information of G and H (i.e. E_i and F_j), the computation of G_iH_j takes $O(t/k^2)$ time. Thus $T_2(h,t) = O((h/k)^2t/k^2) = O(h^2t/k^4)$. Therefore the time complexity for DMM is $O(n^3/t + (n^3/h^2)(h^2t/k^4)/t) = O(n^3/t + n^3/k^4)$. As stated above we shall let $k = c(\log n/\log\log n)^{1/7}$ for a constant c and let $t = k^4$.

3 Encoding Matrix Information

Please refer to procedure DMM in Section 4.

Let $G = \{g_{ij}\}, H = \{h_{ij}\}, L = \{l_{ij}\}$ and $L = GH$ as in Section 2. We take the lists

$$(g_{1r} - g_{1s}, \dots, g_{hr} - g_{hs}), (1 \le r < s \le t)$$

and

$$(h_{s1}-h_{r1},...,h_{sh}-h_{rh}),(1 \le r < s \le t)$$

and combine them and have them sorted. The sorting takes $O(t^2 h \log(t^2 h)) < O(h^2)$ time. Let the sorted list be N. Let $G(i,r,s)$ be the rank of $g_{ir}-g_{is}$ in N and let $H(i,r,s)$ be the rank of $h_{si}-h_{ri}$ in N. Then $N[G(i,r,s)] = g_{ir}-g_{is}$ and $N[H(i,r,s)] = h_{si}-h_{ri}$. As shown in [4], we have

$$g_{ir}+h_{rj} \le g_{is}+h_{sj} \Leftrightarrow g_{ir}-g_{is} \le h_{sj}-h_{rj}.$$

As observed in [9],

$$g_{ir}+h_{rj} \le g_{is}+h_{sj} \Leftrightarrow G(i,r,s) \le H(j,r,s)$$

.

$G(i,r,s)$ and $H(j,r,s)$ are computed at line 11 in procedure DMM.

Let $\mathcal{T} = \{T \subseteq \{1,2,3,...,t\}$ and $|T| = 3k^2\}$. Then $|\mathcal{T}| < 2^{3k^2 \log t}$. For each $T \in \mathcal{T}$ and G_i (G_i is a $k \times t$ matrix as given in section 2), we let $G_i[T]$ be the $k \times 3k^2$ matrix by deleting j-th column of G_i iff $j \notin T$. Let $G_i[T] = \{g_{ij}\}$. We obtain the encoding

$$I[G_i[T]] = \prod_{u=1}^{k} \prod_{v=1}^{3k^2-1} \prod_{w=v+1}^{3k^2} G(u,v,w),$$

where $\prod$ is the concatenation of strings. Similarly for $H_j[T]$ which is obtained by deleting i-th row of H_j iff $i \notin T$, we obtain encoding

$$J[H_j[T]] = \prod_{u=1}^{k} \prod_{v=1}^{3k^2-1} \prod_{w=v+1}^{3k^2} H(u,v,w).$$

Note that since each $G(u,v,w)$ and $H(u,v,w)$ can be encoded by $O(\log h)$ bits, $I[G_i[T]]$ and $J[H_j[T]]$ can be encoded by $O(k^5 \log h)$ bits. The computation of the encoding takes $O(|\mathcal{T}|(h/k)k^5) = O(2^{O(3k^2 \log t)}(h/k)k^5)$ time because for each $T \in \mathcal{T}$ there are $2h/k$ submatrices ($G_i[T]$'s and $H_j[T]$'s) for which the encoding needs to be done and for each submatrix the encoding can be done in $O(k^5)$ time because there are only $O(k^5)$ integers encoded into the encoding. Since $h = 2^{4k^2 \log t}$ the time for encoding is no more than $O(h^2)$. Now each encoded value of $I[G_i[T]]$ and $J[H_j[T]]$ takes $O(k^5 \log h) = O(k^7 \log t) = O(k^7 \log k)$ bits.

$I[G_i[T]]$ is computed at line 15 of procedure DMM. $J[H_j[T]]$ is computed at line 19 of procedure DMM.

4 The Improving Technique

Let $L_{ij} = \{l_{ab}\}, a, b = 1, 2, ..., k$, where $l_{ab} = \min_{1 \le c \le t}\{g_{ac} + h_{cb}\}$ and g_{ac} is from G_i and h_{cb} is from H_j. Let $m(a, b)$ be the index of c for which the minimum is attained in $\min_{1 \le c \le t}\{g_{ac} + h_{cb}\}$. It is obvious that $1 \le m(a, b) \le t$. Let $M(G_i, H_j) = \{m(a, b) | 1 \le a, b \le k\}$. Since $L_{ij} = G_i H_j$ is a $k \times k$ matrix and $t = k^4$, therefore $|\{1, 2, ..., t\} - M(G_i, H_j)| \ge k^4 - k^2$. This shows that many values of $\{1, 2, ..., t\}$ are not used in computing $G_i H_j$. We use a loop of $m = O(\log t)$ rounds to reduce the possible values for $M(G_i, H_j)$ from $\{1, 2, ..., t\}$ to a set M_m with $|M_m| \le 3k^2$. Once there are only $3k^2$ values left to choose from, the computation of L_{ij} will become easy. This loop is at line 25 to 36 in procedure DMM.

With m rounds we reduce $M_0 = \{1, 2, ..., t\}$ to M_m. In the i-th round we reduce from M_{i-1} to M_i. To keep track of M_i's, we maintain all possible subsets of $\{1, 2, ..., t\}$.

For each L_{ij} we keep an integer $l(i, j)$ which encodes T, where $T = M_a$ at the end of the a-th round. That is, $l(i, j)$ has t bits and the i-th bit is 1 iff $i \in T$.

We partition each $T = \{a_1, a_2, ..., a_j\}$, where $a_1 < a_2 < ... < a_j \le t$, into $\lceil |T|/(3k^2) \rceil$ sets $T_1 = \{a_1, a_2, ..., a_{3k^2}\}$, $T_2 = \{a_{3k^2+1}, a_{3k^2+2}, ..., a_{6k^2}\}$, ... Every set except the last one has $3k^2$ elements. We use an encoded integer to denote each T_i. Such an integer has t bits and the j-th bit is 1 iff $a_j \in T_i$. Let these encoded integers be $t_1, t_2, ..., t_{\lceil |T|/(3k^2) \rceil}$. We now build a table P of 2^t entries. The value of $l(i, j)$ is used to index into P. If $l(i, j)$ corresponds to T then $P[l(i, j)] = \{t_1, t_2, ..., t_{\lceil |T|/(3k^2) \rceil}\}$. This table has size $2^t \lceil t/(3k^2) \rceil$. Since $k = O((\log n / \log\log n)^{1/7})$ and $t = k^4$, P can be built in $O(n)$ time (here each entry of P can be built in $O(t)$ time). This table can be built once and be used throughout the whole algorithm. In procedure DMM, P is built ar line 4. $t_1, t_2, t_3 ...$ are obtained at line 27.

For each possible value of $I[G_i[T]]$ and $J[H_j[T]]$, where $|T| = 3k^2$, we build a table K. $K[I[G_i[T]], J[H_j[T]], T]$ gives an encoded integer containing the at most k^2 values in $M(G_i[T], H_j[T])$. This encoded integer is an integer having t bits and the i-th bit is 1 iff $i \in M(G_i[T], H_j[T])$. Table K can be built in $O(2^{O(k^7 \log k)} 2^t k^2 k^2)$ time for a suitable constant c because there are only $O(2^{O(k^7 \log k)} 2^t)$ possible entries (here T is represented by a t-bit integer and therefore has 2^t possible values), the value for each entry has k^2 integers which are the values in $M(G_i[T], H_j[T])$ and each such integer can be determined in $|T|$ time. Thus if we let $k^7 \log k = c \log n$ or $k = c(\log n / \log\log n)^{1/7}$ for a suitable constant c, we can build table K in $O(n)$ time. K is built at line 2 in procedure DMM.

Now for each $l(i, j)$ and corresponding T value we index into P and get $g = \lceil |T|/(3k^2) \rceil$ values representing g sets $T_1, T_2, ..., T_g$. This is done at line 27 in procedure DMM. Each of these sets except the last one has $3k^2$ elements. For each $1 \le i, j \le h/k$ we take $g-1$ steps to index into

$$K[I[G_i[T_v]], J[H_j[T_v]], T_v], 1 \le v \le g-1.$$

This is done at lines 29 to 33 in procedure DMM. Because $|T_v| = 3k^2$ and the value of table K returns at most k^2 values we have reduced the size of T_v by 1/3. Therefore we have reduced the size of $T - T_g$ by 1/3. That is, we have reduced T by 2/3 if $|T| > 3k^2$. We can now combine (by using OR operations) the above $g-1$ K values and t_g into one integer which is the new value of $l(i,j)$ for the next round. This is done at line 34 of procedure DMM. For the $O(\log t)$ rounds the value of g is decreasing geometrically and therefore the time is dominated by the first round which takes $O((h/k)^2 t/k^2)$. Since we let $t = k^4$ the time is $O(h^2)$.

When $|T| \le 3k^2$ we simply add arbitrary elements in $\{1, 2, ..., t\}$ to make $|T| = 3k^2$. We index into a table K' similar to table K. The difference between K' and K is that K' gives an encoded integer which contains k^2 values with each value giving the index of c for which $\min_{1 \le c \le t}\{g_{ac} + h_{cb}\}$, $1 \le a, b \le k$, is attained. And therefore we can obtain $G_i H_j$. This is done at lines 38 to 40 in procedure DMM.

Thus we have shown that DMM can be computed in $O(n^3/t) = O(n^3(\log\log n/\log n)^{4/7})$.

Theorem 1: All pairs shortest paths can be computed in $O(n^3(\log\log n/\log n)^{4/7})$ time.

We list our algorithm below.

Procedure **DMM**

1. { /* Initialization, done once only. */
2.   Create table K;
3.   Create table K';
4.   Create table P;
5.
6.   /* Now work on G and H. */
7.   for each pair of G and H do
8.   { /* This is repeated $\frac{n^3}{h^2 t}$ times as there are that many pairs of G and H.*/
9.   for $1 \le i \le h$ and $1 \le r < s \le t$ do
10.   {
11.   Compute $G(i,r,s)$ and $H(i,r,s)$; /*By sorting*/
12.   }
13.   for $1 \le i \le h/k$ and each $T \in \mathcal{T}$ do
14.   {

compute $I[G_i[T]]$;

}

for $1 \le j \le h/k$ and each $T \in \mathcal{T}$ do

{

compute $J[H_j[T]]$;

}

/* Now work on G_i and H_j. */

for each pair of G_i and H_j do

{/* This is repeated $(h/k)^2$ times as there are that many pairs of G_i and H_j when multiplying G and H. */

$T = \{1, 2, ..., t\}$ /* T is represented by a t-bit vector. */

for$(a = 1;\ a \le \lceil \log_{4/3}(t/(3k^2)) \rceil;\ a++)$

{ /* Each iteration reduces the size of T by a factor of 3/4. After $\lceil \log_{4/3}(t/(3k^2)) \rceil$ iterations T is of size $3k^2$. */

Let $(T_1, T_2, ..., T_g) = P[T]$;

/* Index into table P to get $T_1, T_2, ..., T_g$. Each T_b is a t-bit vector. $g \le \lceil \frac{t}{(3k^2)(4/3)^a} \rceil$. */

for$(v = 1;\ v \le g - 1;\ v++)$

{

$S_v = K[I[G_i[T_v]], J[H_j[T_v]], T_v]$;

/* Look up into table K to reduce the number of 1's in T_v by a factor of 3/4. */

}

$T = (\bigvee_{v=1}^{g-1} S_v) \bigvee T_g$;

/* The number of 1's in T is reduced by a factor of 3/4. */

}

/* Now T has no more than $3k^2$ bits which are 1's. */

$S = K'[I[G_i[T]], J[H_j[T]], T]$;

/* S now contains the k^2 indices for multiplying G_i and H_j, that is, the set $M(G_i, H_j)$. */

Compute $L_{ij} = G_i H_j$ using $M(G_i, H_j)$;

}

}

}

5 Further Improvement

We shall let $k = c(\log n/\log\log n)^{1/7}$ as in the previous sections and we shall let $t = k^3$ and $\mathcal{T} = \{\{1, 2, ..., t\}\}$. Thus $|\mathcal{T}| = 1$. We encode the matrix information as in section 3 for G and H. Since G_i is a $k \times t$ matrix and H_j is a $t \times k$ matrix, $I[G_i]$ and $J[H_j]$ can be encoded by $O(k^7 \log h)$ bits. The computation of the sorting of lists $(g_{1r} - g_{1s}, ..., g_{hr} - g_{hs}), (1 \le r < s \le t)$ and $(h_{s1} - h_{r1}, ..., h_{sh} - h_{rh}), (1 \le r < s \le t)$ takes $O(ht^2 \log(ht^2)) = O(hk^6 \log(hk))$ time and the time of the encoding is $O((h/k)kt^2) = O(hk^6)$ time. Thus if we let $h = k^9$ then the computation of GH is within $O(h^2/k^2)$ time. Since there are $O((n/h)^2(n/t)) = O(n^3/(h^2k^3))$ L's for the whole input matrix the computation for the whole matrix is $O(n^3/k^5)$.

Because encoding I and J has $O(k^7 \log h) = O(k^7 \log k)$ bits, table K has only $2^{O(k^7 \log k)}$ entries. Thus if we let $k = c(\log n/\log\log n)^{1/7}$ for a suitable constant c then table K can be built in $O(n)$ time. The value of an entry of table K is an integer of k^3 bits with at most k^2 bits setting to 1's because $|M(G_i, H_j)| \le k^2$. Our computation of G_iH_j's returns $O((n/k)^2 n/t) = O(n^3/k^5)$ K values in $O((n/k)^2 n/t) = O(n^3/k^5)$ time. Thus we have reduced each $M(G_i, H_j)$ from k^3 to k^2 and we have done this in $O(n^3/k^5)$ time.

Now we use this algorithm in section 4. We let $t = k^5$ and $h = 2^{4k^2 \log t}$. For all G_i's and H_j's there we take only $O(((n/k)^2 \frac{t/k}{k^2} + n^2)(n/t))$ time, where we have $(n/k)^2(n/t)$ L_{ij}'s and for each of them we take $\frac{t/k}{k^2}$ time because $3k^2$ values in M_a are looked in one step and we have t/k values instead of t values because we have reduced the elements in M_0 by a factor of k. Thus the total time for our algorithm is $O(n^3/t) = O(n^3/k^5) = O(n^3(\log\log n/\log n)^{5/7})$.

Theorem 2: All pairs shortest paths can be computed in $O(n^3(\log\log n/\log n)^{5/7})$ time.

6 Conclusions

Although we have improved the upper bound for all pairs shortest paths from previous results, the problem is far from closed. We expect that better bounds be derived in the future.

Acknowledgment

Earlier version of this paper did not list out the code for DMM. We are thankful to a referee's suggestion to list out procedure DMM which makes this paper much easier to understand.

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman. The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[2] A. V. Aho, J. E. Hopcroft, J. D. Ullman. Data Structures and Algorithms, Addison-Wesley, Reading, MA, 1983.

[3] W. Dobosiewicz. A more efficient algorithm for min-plus multiplication. *Inter. J. Comput. Math.* **32**, 49-60(1990).

[4] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Computing* **5**, 83-89(1976).

[5] M. L. Fredman, R. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34, 596-615, 1987.

[6] Z. Galil, O. Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134, 103-139(1997).

[7] S. Pettie. A faster all-pairs shortest path algorithm for real-weighted sparse graphs. *Proceedings of 29th International Colloquium on Automata, Languages, and Programming (ICALP'02), LNCS Vol. 2380*, 85-97(2002).

[8] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51, 400-403(1995).

[9] T. Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters* **43**, 195-199(1992).

[10] M. Thorup. Undirected single source shortest paths in linear time. *Proc. 38th IEEE Symposium on Foundations of Computer Science*, Miami Beach, Florida, 12-21(1997).

[11] U. Zwick. All pairs shortest paths in weighted directed graphs - exact and almost exact algorithms. *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science*, Palo Alto, California, 310-319(1998).