An $O(n^3 \log\log n/\log n)$ Time Algorithm for the All-Pairs Shortest Path Problem

Tadao Takaoka
Department of Computer Science
University of Canterbury
Christchurch, New Zealand

October 27, 2004

1 Introduction

In this paper we consider the all-pairs shortest path (APSP) problem, which computes shortest paths between all pairs of vertices of a directed graph with non-negative real numbers as edge costs. We present an algorithm that computes shortest distances between all pairs of vertices, since shortest paths can be computed as by-products in our algorithm. It is well known that the time complexity of (n, n)-distance matrix multiplication (DMM) is asymptotically equal to that of the APSP problem for a graph with n vertices. Thus we concentrate on DMM in this paper. The computational model in this paper is the conventional RAM, where only arithmetic operations, branching operations, and random accessibility with $O(\log n)$ bits are allowed.

Fredman [4] was the first to break the cubic complexity of $O(n^3)$ under RAM, giving $O(n^3(\log\log n/\log n)^{1/3})$. This complexity was improved to $O(n^3(\log\log n/\log n)^{1/2})$ by Takaoka [6] with RAM, and to $O(n^3/(\log n)^{1/2})$ by Dobosievicz [3] with extended RAM instructions. Recently there have been some more progresses such as $O(n^3(\log\log n)/\log n)^{5/7})$ [5] and $O(n^3(\log\log n)^2/\log n)$ [9]. In this paper we improve the complexity further to $O(n^3 \log\log n/\log n)$. If edge costs are small integers, the complexity becomes more subcubic, i.e., $O(n^{3-\epsilon})$ for some $\epsilon > 0$, as shown in [7], [1], and [11].

We follow the same framework as those in [4], [6], and [9]. That is, we take a two level divide-and-conquer approach. To multiply the small matrices resulting from dividing the original matrices, we sort distance data, and use the ranks of those data in the sorted lists. As the ranks are small integers, the multiplication can be done efficiently by looking at some precomputed tables.

2 Minimum selection from a small list of small integers

To prepare for distance matrix multiplication, we introduce a basic technique for minimum selection in encoded form in this section. A sequential algorithm takes $O(n)$ time to select

the minimum of a list of integers where the size of the list is n and the magnitude of integers is up to the maximum that can be contained in single computer words. We show that it takes $O(\log m)$ time for the same problem where the size of the list is m and the magnitudes of integers are polynomial of m, and $m = O(\log n)$. We assume a pre-computed table of size $O(n)$ is available, which can be constructed in $O(n)$ time..

For simplicity we assume m is a power of 2. We can generalize our theory without changing the orders of complexities. We go through tournament for participants 1, 2, ..., m with corresponding distinct keys $k(1)$, ..., $k(m)$. If $k(i) < k(i+1)$ for odd i, i is chosen as a winner for the next stage, and $i+1$, otherwise. We keep track of winners as well as their keys for later application. The size and magnitude are so small that the list of winners and the list of their keys can be encoded into single integers. The one-to-one mappings for encoding are denoted by h_1^m and h_2^m respectively. At the 0-th stage, all of (1, 2, ..., m) are winners, encoded as $h_1^m(1, 2, ..., m) = (1-1)m^{m-1} + (2-1)m^{m-2} + ... + (m-1)$. Their keys are encoded into $h_2^m(k(1), ..., k(m)) = k(1)\mu^{m-1} + ... + k(m)$, where $\mu = m^p$ for some $p > 0$ such that $0 \leq k(i) \leq \mu - 1$ for $0 \leq i \leq m$. When there are l winners $(x_1, ..., x_l)$ in general, h_1 and h_2 are defined by $h_1(x_1, x_2, ..., x_l) = (x_1 - 1)m^{l-1} + (x_2 - 1)m^{l-2} + ... + (x_l - 1)$, and $h_2(k(x_1), ..., k(x_l)) = k(x_1)\mu^{l-1} + ... + k(x_l)$, where we omit superscripts from h_1 and h_2 for simplicity. We assume those encoded values are stored in single words. The table T^l is constructed as follows:

$$T^l(h_1(x_1, ..., x_l), h_2(k(x_1), ..., k(x_l))) = h_1(y_1, ..., y_{l/2}), \text{ where}$$
$$y_i = x_{2i} \text{ if } k(x_{2i}) < k(x_{2i+1}), y_i = x_{2i+1}, \text{ otherwise.}$$

We prepare another mapping $f^l(h_1(x_1, ..., x_l)) = h_2(k(x_1), ..., k(x_l))$. For general discussions, we omit superscripts. T is a mapping from key values to winners, and f is a mapping from winners to key values, both in the form of encoded integers. Repeated use of T and f will finalize the winner for the whole tournament in $O(\log m)$ time, with the value of l halved at each stage. As winners and keys are distinct integers, mappings T and f are partial functions of the form of $Z \times Z \rightarrow Z$, where Z is the set of positive integers. The domain for which function values are not defined can be filled with any value.

EXAMPLE 1 *Participants and corrresponding keys are traced in unencoded form as follows:*

(1, 2, 3, 4, 5, 6, 7, 8)	*(12, 4, 7, 16, 5, 9, 8, 2)*
(2, 3, 5, 8)	*(4, 7, 5, 2)*
(2, 8)	*(4, 2)*
(8)	*(2), 8 is the final winner with key 2*

The size of the table T^l is bounded by $m^l(m^p)^l = m^l m^{pl}$ and it takes $O(m)$ time to compute each table entry, resulting in $O(m^{l(p+1)}m)$ time. Now we prepare mapping f. The size of the table is $m^l m^{pm}$, and the time for one entry is $O(m)$, resulting in

$O(m^l m^{pm} m) \leq O(m^m m^{pm} m)$ time for the whole table. The times for table construction are further multiplied by $\log m$ as there are $\log m$ tables corrsoponding superscripts l. Those times are shown to be $O(c^{m \log m})$ for some constant $c > 0$, which is further shown to be $O(n)$ when $m{=}O(\log n/(\log c \log\log n))$.

Let $\mathbf{x} = (1, 2, ..., m)$, and $\mathbf{k} = (k(1), ..., k(m))$. If the tables T and f as shown above are precomputed in $O(n)$ time, and one problem instance is given by the encoded form of $h_1(\mathbf{x})$ and $h_2(\mathbf{k})$, we can choose the minimum of m keys in $O(\log m)$ time. We call this method the Tournament.

Now we show that we can compute the minimum of $m \log m$ keys in $O(\log m)$ time with tables precomputed in $O(n)$ time. Let $\mathbf{x}$ and $\mathbf{k}$ be $m \log m$ participants and their keys. We sample m participants and corresponding keys at $\log m$ intervals, and encode them. Specifically we prepare table W that maps two sets of m arguments, $\mathbf{x}$ and $\mathbf{y}$, and a set of keys $\mathbf{k}$ in encoded form to m arguments $\mathbf{z}$ in encoded form as follows:

$$W(h_1(\mathbf{x}), h_1(\mathbf{y}), h_2(\mathbf{k})) = h_1(\mathbf{z}), \text{ where}$$
$$z_i = x_i, \text{ if } k(x_i) < k(y_i),\ z_i = y_i, \text{ otherwise.}$$

By going through $\log m$ sets of samples, using W and f alternately, we can compute the minima of m intervals of size $\log m$ each in encoded form. We assume that those $\log m$ sets of m samples are available in encoded form in advance. We call this part the preliminary part. From that point on, we can use the Tournament method described above to finalize the minimum. We call this method, that is, the preliminary part followed by the Tournament, the Better Tournament. The time for constructing W's can be shown to be $O(n)$ with $m = O(\log n/(\log c \log\log n))$ for different c. If we take the larger c, we have $O(\log m)$ time for the Better Tournament for $m \log m$ participants.

In the above descriptions, we ignored encoding time. In later sectons we will see this is a reasonable assumption as all the work for table construction and encoding are done at the beginning, which will be shared by subproblems, and absorbed in the main complexity of DMM.

3 Distance matrix multiplication by the Better Tournament

The distance matrix multiplication is to compute the following distance product $C = AB$ for two (n, n)-matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ whose elements are real numbers.

$$c_{ij} = min_{k=1}^{n}\{a_{ik} + b_{kj}\}, (i, j = 1, ..., n) \quad (1)$$

The operation in the right-hand side of (1) is called distance matrix multiplication of the min version, and A and B are called distance matrices in this context. If we use max instead we call it the max version.

Now we divide A, B, and C into (m, m)-submatrices for $N = n/m$ as follows:

$$\begin{pmatrix} A_{1,1} & ... & A_{1,N} \\ ... & & \\ A_{N,1} & ... & A_{N,N} \end{pmatrix} \begin{pmatrix} B_{1,1} & ... & B_{1,N} \\ ... & & \\ B_{N,1} & ... & B_{N,N} \end{pmatrix} = \begin{pmatrix} C_{1,1} & ... & C_{1,N} \\ ... & & \\ C_{N,1} & ... & C_{N,N} \end{pmatrix}$$

Matrix C can be computed by

$$C_{ij} = min_{k=1}^{N}\{A_{ik}B_{kj}\}(i,j = 1,...N), \ (2)$$

where the product of submatrices is defined similarly to (1) and the "min" operation is defined on submatrices by taking the "min" operation componentwise. Since comparisons and additions of distances are performed in a pair, we omit counting the number of additions for measurement of the time complexity. We have N^3 multiplications of distance matrices in (2). Let us assume that each multiplication of (m,m)-submatrices can be done in $T(m)$ computing time, assuming precomputed tables are available. The time for constructing the tables is reasonable when m is small. Then the total time excluding table construction is given by $O(n^3/m + (n/m)^3T(m))$.

In [9], it is shown that $T(m) = O(m^2(m\log m)^{1/2})$ with $m = O(\log n/(\log\log m)^3)$ by the Tournament. Thus the time becomes $O(n^3(\log m/m)^{1/2})$. The Tournament method is shown first. Then in the subsequent sections, we show m can be greater by the Better Tournament.

Now we further divide the small (m,m)-submatrices into rectangular matrices in the following way. We rename the matrices A_{ik} and B_{kj} in (2) by A and B. Let $M = m/l$, where $1 \leq l \leq m$. Matrix A is divided into M (m,l)-submatrices A_1, ..., A_M from left to right, and B is divided into M (l,m)-submatrices B_1, ..., B_M from top to bottom. Note that A_k are vertically rectangular and B_k are horizontally rectangular. Then the product $C = AB$ can be given by

$$C = min_{k=1}^{M}\{A_kB_k\} \ (3)$$

The values of m and l were determined in [9], using the Tournament, to achieve the claimed complexity.

4 How to multiply rectangular matrices

We rename again the matrices A_k and B_k in (3) by A and B. In this section we show how to compute AB, that is,

$$min_{r=1}^{l}\{a_{ir} + b_{rj}\}, \text{ for } i = 1,...,m; j = 1,...,m. \ (4)$$

We assume that the lists of length m, $(a_{1r} - a_{1s}, ..., a_{mr} - a_{ms}), (1 \leq r < s \leq l)$, and $(b_{s1} - b_{r1}, ..., b_{sm} - b_{rm}), (1 \leq r < s \leq l)$ are already sorted for all r and s such that $1 \leq r < s \leq l$. The time for sorting will be mentioned in Section 7. Let E_{rs} and F_{rs} be the corresponding sorted lists. For each r and s, we merge lists E_{rs} and F_{rs} to form list G_{rs}.

Let H_{rs} be the list of ranks of $a_{ir} - a_{is}$ $(i = 1, ..., m)$ in G_{rs} and L_{rs} be the list of ranks of $b_{sj} - b_{rj}$ $(j = 1, ..., m)$ in G_{rs}. Let $H_{rs}[i]$ and $L_{rs}[j]$ be the ith and jth components of H_{rs} and L_{rs} respectively. Then we have

$$G_{rs}[H_{rs}[i]] = a_{ir} - a_{is},\ G_{rs}[L_{rs}[j]] = b_{sj} - b_{rj}$$

The lists H_{rs} and L_{rs} for all r and s can be made in $O(l^2 m)$ time, when the sorted lists are available.

We have the following obvious equivalence.

$$a_{ir} + b_{rj} \leq a_{is} + b_{sj} \Longleftrightarrow a_{ir} - a_{is} \leq b_{sj} - b_{rj} \Longleftrightarrow H_{rs}[i] \leq L_{rs}[j]$$

Fredman [4] observed that the information of ordering for all i, j, r, and s in the rightmost side of the above formula is sufficient to determine the product AB by a precomputed table. This information is essentially packed in the three dimensional space of $H_{rs}[i](i = 1, ..., m; r = 1, ..., l; s = r + 1, ..., l)$, and $L_{rs}[j](j = 1, ..., m; r = 1, ..., l; s = r + 1, ..., l)$. We call this the three-dimensional packing.

Takaoka [6] proposed that to compute each (i, j) element of AB, it is enough to know the above ordering for all r and s. We call this the two-dimensional packing. Note that the precomputed table must be obtained within the total time requirement. The two-dimensional packing will therefore allow a larger size of m, leading to a speed-up.

In [9], it is shown that a one-dimensional packing scheme is possible by the Tournament, resulting in the complexity of $O(n^3(\log\log n)^2)/\log n)$.

We first describe the Tournament, and then proceed to the Better Tournament. We extend the tournament schemes in Section 2 with two sets of keys H's and L's. To choose the minimum of $x_r^0 = a_{ir} + b_{rj}, (r = 1, ..., l)$ we go through the Tournament. Later we will initialize $x_r^0, (r = 1, ..., l)$ differently for the Better Tournamnet. We assume m and l are a power of 2. Our theory can be generalized into other cases easily. Specifically we compare in the following pairs. This is the 0th comparison stage.

$$(x_1^0, x_2^0), (x_3^0, x_4^0), ..., (x_{l-1}^0, x_l^0)$$

Suppose the minima of those pairs, that is, winners, are $x_1^1, x_2^1, ..., x_{l/2}^1$. Each x_j^1 has two possibilities of being x_{2j-1}^0 or x_{2j}^0. Then we compare in the following pairs. This is the 1st comparison stage.

$$(x_1^1, x_2^1), (x_3^1, x_4^1), ..., (x_{l/2-1}^1, x_{l/2}^1)$$

The winner x_j^2 of (x_{2j-1}^1, x_{2j}^1) has four possiblities of being x_i^0 for $4j - 3 \leq i \leq 4j$. By repeating these stages, we can finish in $\log l - 1$ stages. Comparing in the pair (x_r^0, x_{r+1}^0), that is, testing "$x_r^0 \leq x_{r+1}^0$?", is equivalent to comparing in the pair $(H_{r,r+1}[i], L_{r,r+1}[j])$. Thus if we pack two tuples $(H_{12}[i], H_{34}[i], ..., H_{l-1,l}[i])$ and $(L_{12}[j], L_{34}[j], ..., L_{l-1,l}[j])$ into single integers, we can know the $l/2$ winners in $O(1)$ time from a precomputed table in the form of an encoded integer. We can take a similar approach for stages 1, 2, ... Thus the time

```
H   1  2  3  4  5  6  7  8        L   1  2  3  4  5  6  7  8
   ----------------------           ----------------------
1 |    1 13* 5  6  7  3  4        1 |   11  3* 2  7  4  5  6
2 |      12  4  5  8  2  5        2 |       4  1  5  6  7  8
3 |          5  6  1#10  7        3 |         15  3  8# 1  5
4 |             2  4  5  1        4 |            11  3  7 14
5 |                12  3  9        5 |               10  5  6
6 |                   4* 8        6 |                  7* 9
7 |                      3        7 |                     10
8 |                               8 |
```

Figure 1: $H_{rs}[i]$ and $L_{rs}[j]$ for $l = 8$

for computing (4) becomes $O(\log l)$. Since the ranks are between 1 and $2m$, and there are up to l remaining winners, the size of each table is bounded by $m^l(2m)^{l/2}(2m)^{l/2} = m^l(2m)^l$, as discussed in Section 5. The time for computing those tables will be mentioned in Section 7. In the following, winners are specified by indices.

EXAMPLE 2 *Let $H_{rs}[i]$ and $L_{rs}[j]$ be given in Figure 1. First we have the two lists*

$$(H_{1,2}[i], H_{3,4}[i], H_{5,6}[i], H_{7,8}[i]) = (1, 5, 12, 3)$$

$$(L_{1,2}[j], L_{3,4}[j], L_{5,6}[j], L_{7,8}[j]) = (11, 15, 10, 10)$$

Since $1 < 11, 5 < 15, 12 > 10$, and $3 < 10$, the winners at the first stage are (1, 3, 6, 7). The next lists are thus

$$(H_{1,3}[i], H_{6,7}[i]) = (13, 4),\ (L_{1,3}[j], L_{6,7}[j]) = (3, 7),\ \textit{(shown by *).}$$

Since $13 > 3$ and $4 < 7$, the winners at the second stage are (3, 6), (shown by #). The last lists are $(H_{3,6}[i]) = (1)$ and $(L_{3,6}[j]) = (8)$, from which we conclude $a_{i3} + b_{3j}$ is the minimum. All tuples above are encoded in single integers in actual computation. If we had different H and L, we might have other winners such as (2, 4, 5, 8) at the first stage, and (2, 8) at the second stage, etc. We have all preparations for those situations in the form of precomputed tables, so each step is done in $O(1)$ time.

5 How to compute the tables

In Section 2 we used h_1 for encoding winners given in indices, and h_2 for encoding keys. In this section, we use the same h_1 for winners, but modify h_2 by setting $\mu = 2m$, and encoding ranks $H's$ or $L's$, instead of the set of keys.

When $h_2^{-1}(\alpha) = (a_1, ..., , a_k)$ for positive integer α, we express the jth element of $h^{-1}(\alpha)$ by $h_2^{-1}(\alpha)[j]$, that is, $h_2^{-1}(\alpha)[j] = a_j$.

Now we construct tables $T^l, T^{l/2}, ...$ The role of table T^r is to determine winners $(x_1, ..., x_{r/2})$ for the next stage when there are r remaining winners. Let integers α and β

represent encoded forms of $l/2$ ranks in H's and L's. That is, $\alpha = h_2(H_{1,2}, ..., H_{l-1,l})$, and $\beta = h_2(L_{1,2}, ..., L_{l-1,l})$. The role of z is to represent winners in an encoded form. At the beginning all are winners. The table T^l is defined by

$$T^l(z, \alpha, \beta) = h_1(x_1, x_2, ..., x_{l/2}),$$

where $x_j = 2j - 1$ if $h_2^{-1}(\alpha)[j] < h_2^{-1}(\beta)[j]$, $x_j = 2j$ otherwise, and $z = h_1(1, 2, ..., l)$. The value of z has just one possibility. Tables $T^{l/2}, T^{l/4}, ...$ can be defined similarly using $l/2$ winners, $l/4$ winners, ... as follows:

Let α and β be encoded forms of $r/2$ ranks each and $z = h_1(z_1, ..., z_r)$. T^r is defined by

$$T^r(z, \alpha, \beta) = h_1(x_1, x_2, ..., x_{r/2}),$$

where $x_j = z_{2j-1}$ if $h_2^{-1}(\alpha)[j] < h_2^{-1}(\beta)[j]$, $x_j = z_{2j}$ otherwise.

Let $z_1, z_2, ..., z_r$ be r winners where r is an even integer. We introduce two mappings f^r and g^r for $1 \leq r \leq l$ to determine which ranks to use next after we have those r winners. We omit the subscripts i and j from $H_{rs}[i]$ and $L_{rs}[j]$ for simplicity in the following.

$$f^r(h_1(z_1, z_2, ..., z_r)) = h_2(H_{z_1,z_2}, H_{z_3,z_4}, ..., H_{z_{r-1},z_r})$$

$$g^r(h_1(z_1, z_2, ..., z_r)) = h_2(L_{z_1,z_2}, L_{z_3,z_4}, ..., L_{z_{r-1},z_r})$$

Those mappings can be computed in $O(r)$ time.

To compute $T^{l/2}(z, \alpha, \beta) = h_1(x_1, x_2, ..., x_{l/4})$, for example, we decode z, α, and β in $O(l)$ time, then test $h_2^{-1}(\alpha)[j] < h_2^{-1}(\beta)[j]$ to get $x_1, ..., x_{l/4}$, and finally encode $x_1, ..., x_{l/4}$ spending $O(l)$ time. We do this for all possible z, α, and β. Other tables can be computed similarly in $O(l)$ time for each entry. Tables f^r and g^r can also be computed in $O(l)$ time for each entry. The total time for f's and g's are not greater than that for T's.

The total time for computing those tables is thus $O(m^l(2m)^l l)$. Observe

$$O(m^l(2m)^l l) = O(c^{l \log m}), \text{ for some constant } c > 0 \text{ (5)}.$$

6 Algorithm by table look-up

Using tables $T^0, T^1, ...$, we can compute $min_r\{a_{ir} + b_{rj}\}$ by repeating the following $\log l$ steps. We start from $f^l(z^0) = h_2(H_{1,2}, H_{3,4}, ..., H_{l-1,l})$ and $g^l(z^0) = h_2(L_{1,2}, L_{3,4}, ..., L_{l-1,l})$. The last $z^{\log l}$ is the solution index.

$$z^0 = h_1(1, 2, ..., l) \text{ (z^0 was precomputed)}$$
$$z^1 = T^0(z^0, f^l(z^0), g^l(z^0))$$
$$z^2 = T^1(z^1, f^{l/2}(z^1), g^{l/2}(z^1))$$
$$...$$
$$z^{\log l} = T^{\log l - 1}(z^{\log l - 1}, f^2(z^{\log l - 1}), g^2(z^{\log l - 1}))$$

Now we describe the preliminary part of the Better Tournament. We assume there are $l \log l$ participants 1, 2, ..., $l \log l$ at the beginning. Suppose participants and their ranks are encoded using h_1 and h_2 at $\log l$ intervals. Corresponding to $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{k}$ in Section 2, the first two sets of participants and ranks are

$$h_1(1, \log l + 1, ..., (l-1)\log l + 1),\ h_1(2, \log l + 2, ..., (l-1)\log l + 2),$$
$$h_2(H_{1,2}, H_{\log l+1, \log l+2}, ..., H_{(l-1)\log l+1,(l-1)\log l+2}),$$
$$h_2(L_{1,2}, L_{\log l+1, \log l+2}, ..., L_{(l-1)\log l+1,(l-1)\log l+2}).$$

In the Tournament as applied to DMM, we extended table T with two arguments of α and β. We can similarly extend table W with α and β. By using W and f alternately, we can select m winners in encoded form, and pass them to the Tournament. This preliminary part takes $O(\log l)$ time. As in Section 2, the time for constructing W's can be estimated in the same formula with that for T's with different c. By taking the larger c, we include this time for W and f in (5).

Now we resize l. Suppose we have l participants at the beginning. Then the value of l in the preliminary part and the Tournament is replaced by $l/\log l$. The time for computing the final winner is still $O(\log l)$, but m can be larger as seen in the next section.

7 Analysis of computing time

The time for this algorithm to compute (4) by the Better Tournament is $O(m^2 \log l) = O(m^2 \log m)$.

Let us evaluate the time for (3). Since there are m/l products $A_k B_k$ in (3), we need $O((m/l)m^2 \log m)$ time. To compute minimum component-wise in $min_{k=1}^{m/l}\{A_k B_k\}$, we need $O((m/l)m^2)$ time. We also need $O(Ml^2 m) = O(lm^2)$ time for Ml^2 mergings as described in Section 3.

We set $l = (m \log m)^{1/2}$ to balance the first and the third of the above three complexities. Then to compute the product of (m, m)-matrices, we take $T(m) = O(m^2 (m \log m)^{1/2})$ time.

We determine the size of submatrices in Section 3. Let m be given by $m = \log^2 n/(\log^2 c \log\log n)$. Then we have $l \leq \log n/\log c$ for sufficiently large n. As we substitute $l/\log l$ for l in the $O(c^{l \log m})$ time for making the tables given in equation (5), the time for constructing tables is $O(c^{(l/\log l)\log m}) = O(n)$. Substituting the value of m for $O(n^3(\log m/m)^{1/2})$, we have the overall computing time for distance matrix multiplication as $O(n^3 \log\log n/\log n)$.

We note that the time for sorting to obtain the lists E_{rs} and F_{rs} for all k in (3) in Section 3 is $O(Ml^2 m \log m)$. This task of sorting, which we call presort, is done for all A_{ij} and B_{ij} in Section 3 in advance, taking $O((n/m)^2 (m/l) l^2 m \log m) = O(n^2 l \log m)$ time, which is absorbed in the main complexity.

The encoding of winners and ranks for the preliminary part of the Better Tournament is $O(l)$, and encoding can be done only at the beginning of the Better Tournament. If we

do this for all the submatrices, the time is not greater than that for the above described sorting.

8 Concluding remarks

We showed an asymptotic improvemenet on the time complexity of the all-pairs shortest path problem. The results will have consequences for application areas, such as the diameter of a graph, where DMM or the APSP problem is used. As another example, we note that the maximum subarray problem [2], [10], [8] can be solved in the same complexity as that of DMM.

References

[1] Alon, N, Galil, and Margalit, On the Exponent of the All Pairs Shortest Path Problem, Jour. Comp. Sys. Sci., vol 54, no. 2, pp 255-262, 1997

[2] Bentley, J, Programming Pearls - Perspective on Performance, Comm. ACM, 27 (1984) 1087-1092

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

[4] Fredman, M, New bounds on the complexity of the shortest path problem, SIAM Jour. Computing, vol. 5, pp 83-89, 1976

[5] Han, Y, Improved algorithms for all pairs shortest paths, Info. Proc. Lett., 91 (2004) 245-250

[6] Takaoka, T., A New upper bound on the complexity of the all pairs shortest path problem, Info. Proc. Lett., 43 (1992) 195-199

[7] Takaoka, T, Subcubic algorithms for the all pairs shortest path problem, Algorithmica, vol. 20, 309-318, 1998

[8] Takaoka, T, Subcubic algorithms for the maximum subarray probelm, Proc. Computing:Australasian Theory Symposium (CATS 2002), pp 189-198

[9] Takaoka, T., A Faster Algorithm for the All Pairs Shortest Path Problem and its Application, COCOON 2004

[10] Tamaki, H and T. Tokuyama, Algorithms for the maximum subarray problem based on matrix multiplication, Proc. 9-th SODA (Symposium on Discrete Algorithms), (1998) 446-452

[11] Zwick, U, All pairs shortest paths in weighted directed graphs - exact and almost exact algorithms, 39th FOCS, pp 310-319, 1998.