A Note of an $O(n^3/ \log n)$ Time Algorithm for All Pairs Shortest Paths*

Yijie Han
School of Computing and Engineering
University of Missouri at Kansas City
Kansas City, MO 64110
hanyij@umkc.edu

Abstract

We improve the all pairs shortest path algorithm given by Takaoka to time complexity $O(n^3/ \log n)$. Our improvement is achieved by using a smaller table and therefore saves time for the algorithm.

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 [6], where $m$ is the number of edges of the graph. Recently Pettie showed [10] 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[7, 11, 14, 15]. Fredman gave the first subcubic algorithm [5] 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 $O(n^3 (\log \log n/ \log n)^{1/2})$ [12]. Dobosiewicz [4] 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. In 2004 we obtained an algorithm with time complexity $O(n^3 (\log \log n/ \log n)^{5/7})$ [8]. Later Takaoka obtained an algorithm with time $O(n^3 \log \log n/ \log n)$ [13] and Zwick gave an algorithm with time $O(n^3 \sqrt{\log \log n/ \log n})$ [16].

*Research supported in part by NSF grant 0310245.
In [13] Takaoka raised the question whether the factor \( \log \log n \) can be removed from the time complexity of his algorithm. In this paper we show an algorithm with time complexity \( O(n^3 / \log n) \). This algorithm uses word length of \( O(\log n \log \log n) \) bits and therefore is not directly comparable to Takaoka and Zwick’s results [13, 16]. It only shows that if we use word length of \( O(\log n) \) bits then our algorithm has the same time complexity as Takaoka’s algorithm [13]. However, if we allow larger word length \( O(\log n \log \log n) \) bits) then we can do in \( O(n^3 / \log n) \) time.

We note that in 2005 Chan [3] first obtained an algorithm with time complexity \( O(n^3 / \log n) \). Chan’s algorithm does not use tabulation and bit-wise parallelism. His algorithm also runs on a pointer machine. We were unaware of Chan’s result [3] when we submitted this paper for publication. Since Chan published his result before us the result of \( O(n^3 / \log n) \) time should be fully attributed to Chan. We present this paper here only for the purpose of showing that we applied a technique different than Chan’s [3] to achieve \( O(n^3 / \log n) \) time.

Very recently we have achieved \( O(n^3(\log \log n / \log n)^{5/4}) \) time complexity [9]. This is the currently best result for the all pairs shortest path problem. We gave reasons in [9] that this \( O(n^3(\log \log n / \log n)^{5/4}) \) time represents a intrinsic bound and shall be very difficult to improve on.

2 Computation by Table Lookup

In Takaoka’s algorithm [13] a table \( T \) is needed for comparing \( r \) pairs of numbers \( a_1, a_2, ..., a_r \) and \( b_1, b_2, ..., b_r \), each of which is a positive integer \( \leq 2m \), for \( r = l/2, l/4, l/8, ..., 1 \), to find out \( c_1, c_2, ..., c_r \) where \( c_i = a_i \) if \( a_i < b_i \) and otherwise \( c_i = b_i \). These numbers are very small and \( a_1, a_2, ..., a_r, b_1, b_2, ..., b_r \) can be encoded into one integer. Takaoka’s algorithm uses \( \log l \) tables of total size \( m^l(2m)^l = O(c^l \log m) \) and requires \( O(c^l \log m) \) time to build the table, where \( c \) is a suitable constant. We build tables for the same purpose. Our tables use \( O(c^l \log m) \) space but only \( O(c^l) \) entries need to be initialized and therefore our tables can be built in \( O(c^l) \) time.

Initially there are \( l \) numbers. After the first round of comparison \( l/2 \) numbers remain, for each of these \( l/2 \) numbers we need a number with 2 possibilities to indicate the winner. After the \( i \)-th round of comparison \( l/2^i \) numbers remain, for each of these \( l/2^i \) numbers we need a number with \( 2^i \) possibilities to indicate the winner. Therefore for the \( i \)-th round, we need \( l/2^i \) numbers each having \( i \) bits to indicate \( 2^i \) possibilities of the winner. Thus we use \( li/2^i \) bits to indicate the winners. In the \( i \)-th round, there are \( l/2^i \)
numbers remain, each being $\leq 2m$ and therefore using $\log m + 1$ bits. The total number of bits used is therefore $O(l + l \log m)$. Thus tables of size $O(d + l \log m) = O(d \log m)$ is needed.

However, we show that only $O(d)$ entries of the table needs to be initialized. When we encode $a_j$’s ($b_j$’s), $1 \leq j \leq r$, we concatenate the bits in $a_j$ ($b_j$) but add one bit with value 0 at the most significant bit of each number. These added bits with 0 values are called test bits. Thus encoded number would become

$$0a_10a_20...0a_r0b_10b_2...0b_r.$$  

Before we index into the lookup table we do some manipulation of the coded words. We extract $0b_10b_20...0b_r$ out into another word $W_1$ using a mask and then shift it so it aligns with $0a_10a_20...0a_r$. We then turn the test bits in the word $W_0$ containing $0a_10a_20...0a_r$ to 1’s by ORing $W_0$ with $M$, where $M$ is the mask $10^{\log m+1}10^{\log m+1}1...$ assuming that each $a_j$ and $b_j$ has $\log m + 1$ bits, and get $W_0 = 1a_11a_21...1a_r$. We then do $W_2 = (W_0 - W_1) \text{AND } M$, where \text{AND} is the bitwise and operation. Now all bits for $a_j$ and $b_j$ in $W_2$ are 0’s except the test bit which could be 1 or 0. If the corresponding test bit is 1 then $a_j \geq b_j$ otherwise $a_j < b_j$. We then use $W_2$ to index into the lookup table. Here we omitted the fact that word $W_0$ and $W_1$ contain the at most $l$ bits to identify the winners.

Therefore each of the $l/2^k$ numbers in the comparison in the $i$-th round uses $\log m + 1$ bits but with only 2 possibilities (test bit is either 0 or 1). Thus the table we construct uses $O(d \log m)$ space but only $O(d)$ entries are used and therefore can be built in $O(d)$ time.

3 Improving the Time Complexity

Refer to section 7 of Takaoka’s paper[13], one part of Takaoka’s algorithm takes $O((m^3/l) \log m)$ time, another part takes $O(lm^2)$ time. To balance these two parts, $l$ is set to $(m \log m)^{1/2}$. Instead of setting $m = \log^2 n/(\log^2 c \log \log n)$ in section 7 of Takaoka[13], we set $m = \log^2 n \log \log n / \log^2 c$. Then $l = (m \log m)^{1/2} = O(\log n \log \log n / \log c)$ and $l / \log l = O(\log n / \log c)$. Since our table construction takes $O(d)$ time and we substitute $l / \log l$ for $l$ as did by Takaoka, we got $O(n)$ time for constructing the table. The time complexity of the all pairs shortest path algorithm is $O(n^3 (\log m / m)^{1/2})$ as analyzed by Takaoka. In our case this is $O(n^3 / \log n)$. Therefore we have:

**Theorem 1:** All pairs shortest paths of directed graphs can be computed in $O(n^3 / \log n)$ time.
References


