A Parallel Algorithm for All Pairs Shortest Paths in a Random Graph

Alan Frieze [1]
Larry Rudolph [2]

Graph theory has proved to be a useful tool and there is much need for fast algorithms to solve graph problems. We consider here the problem of finding the shortest distance between all pairs of vertices of a random graph whose edge lengths are independent identically distributed nonnegative random variables. Each edge length has a distribution function F with $F(0) = 0$ and that F is differentiable at zero.

We show that with this model of randomness all shortest distances can be computed in $O(\log\log n)$ expected parallel time using n^3 parallel processors. This compares with $O(\log^2 n)$ worst-case time, with n^3 processors ([Dekel]). We can, in fact, assume that the edge distances are uniformly distributed in [0,1]. The extension to the general case follows as in Frieze and Grimmett ([Frieze]). Reif and Spirakis ([Reif]) also consider the problem of computing the all pairs shortest paths of a random graph, however, they require that the edge weights satisfy a *triangle inequality* and thus their results do not apply to the general case. Moreover, we require only n^3 processors whereas they require at least n^4 processors.

In our model of computation, we assume that comparisons, arithmetic operations and the generation of random integers all take constant time. In addition, we assume that all the processors share common memory and can read or write to any cell in a single cycle. More precisely, we assume a concurrent-read, concurrent-write ($CRCW$) model of parallel computation; if multiple processors may attempt to store into the same cell at the same time an arbitrary one succeeds

Of notable interest is the part of our algorithm that chooses the minimum of n values quickly. Parallel $CRCW$ algorithms for choosing the minimum with n processors in $O(\log\log n)$ time ([Valiant]), with $1 < p \leq n$ processors in $O(n/p + \log\log p)$ time ([Shiloach]), and with n^2 processors in constant time have been well known. Probabilistic decision tree algorithms using only n processors and constant time are also known ([Reischuk] and [Meggido]). We present a $CRCW$ parallel algorithm, i.e. we account for the processor allocation time, for this problem using n processors and show that with high probability, only three constant time steps are needed.

---

[1]GSIA, Carnegie-Mellon University and Department of Computer Science, Queen Mary College, London.

[2]Department of Computer Science, Carnegie-Mellon University

Matrix Product Algorithm (MPA)

Input: A matrix D, where $d_{i,j}$ is the length of edge (i,j).
Output: Matrix D^*, where $d^*_{i,j}$ is the shortest distance from i to j.

Begin
  $D^{(1)} := D$
  Repeat
(1) $D^{(k+1)} := D^{(k)} * D^{(k)}$
  Until $D^{(k+1)} = D^{(k)}$
  $D^* := D^k$
End

Figure 1: All Pairs Shortest Path Algorithm

1. The Matrix Product Algorithm

Let D be the original matrix for a graph $G_n = (V,E)$, where d_{ij} is the length of edge (i,j) or ∞ if there is no such edge and let $N = \{1,2,\ldots,n\}, n = |V|$. It is well known that the following 'matrix' algorithm (Figure 1) computes the all pairs shortest distances.

The * operation in line (1) denotes a special type of matrix operation defined as follows:

$$d_{ij}^{(k+1)} = \min\{\{d_{ip}^{(k)} + d_{pj}^{(k)} | p \in N\} \cup D_{ij}\} \quad i,j \in N$$

The validity of MPA follows from the easily demonstrated

$d_{ij}^{(k)}$ is the minimum length of a path from i to j using 2^k edges or less.

Let

$$m(D) = \min\{\ p \mid \text{ for all } i,j \in N \text{ there exists a shortest path from } i \text{ to } j \text{ using } p \text{ edges or less}\}. \quad (2)$$

Clearly line (1) of MPA is executed at most $\lceil \log_2 m(D) \rceil$ times. The following two Lemmas derive a bound on $m(D)$ and show how to execute line (1) in constant parallel time with high probability. To avoid confusion, we shall say that every edge has a weight (instead of saying length), a path has length x to mean that there are x edges in the path, and that a shortest path is one that has the smallest total weight, i.e. the sum of the weights on each edge of the path.

Lemma 1. $\Pr\{m(D) \geq 37 \log^2 n\} = o(n^{-.1})$

Proof We use the following result from [Frieze] which says that if the edge weights are uniformly distributed in the range $(0,1)$ then, with high probability, there are no very

heavy shortest paths between nodes:

$$\Pr(\quad \text{there exists } i,j \in N \text{ such that } d^*_{ij} > 12\log n/n) = o(n^{-1}) \tag{3}$$

Let $X = \{(i,j) \in N \times N | i \neq j \text{ and } d_{ij} \leq 1/(3n)\}$ and consider the graph $H = (V, X)$. Note that this is a random digraph where independently each $(i,j) \in N \times N$ is an edge of H with probability $p = 1/(3n)$ and not an edge of H with probability $1-p$.

We next show that, with high probability, H does not contain paths with many edges.

$$\Pr(H \text{ contains a path with } w = \lceil \log_e n \rceil \text{ edges }) = o(n^{-1}). \tag{4}$$

Indeed the expected number of paths in H with w edges is

$$\leq \binom{n}{w+1}(w+1)!(\frac{1}{3n})^w \leq n(\frac{1}{3})^w = o(n^{-1}) \tag{5}$$

and (4) follows.

Suppose now that neither of the events described in (3) or (4) occurs. It is left to show that $m(D) \leq 37\log^2 n$, i.e. any path with more than $37\log^2 n$ edges is not a shortest path. Let P be any path in G_n with $t > 37\log^2 n$ edges. By (4) there are no paths in H longer than w edges and so P contains at least $t/(w+1)$ edges not in X. By construction of H, each edge of P not in X has weight greater than $1/(3n)$ and so when n is large,

$$\text{weight of } P \text{ in } G_n \ > (\frac{t}{w+1})(\frac{1}{3n}) > (\frac{12\log n}{n}),$$

and hence P is not a shortest path. ∎

Lemma 2. Line (1) in MPA can be executed in constant expected parallel time using n^3 processors.

Proof. As there are n^3 processors, n processors can be allocated to the calculation of $d^{(k+1)}_{ij}$ for each $i,j \in N$ in parallel. We show there are enough processors so that line (1) of MPA can be executed in constant expected parallel time. Line (1) consists of two steps; first n^2 sets of new shortest distances are computed and second the minimum value in each of these sets is chosen. Note that each set contains at most n elements.

Let us fix i,j,k and write $u_p = d^{(k)}_{ip} + d^{(k)}_{pj}$. Clearly $u_1, u_2, \ldots, u_n$ can be computed in parallel in constant time with n processors for each p; processor PE_p computes u_p for $p = 1,2,\ldots,n$. All that needs to be shown is that the minimum of n numbers can be computed in expected constant time with n processors. This is proved in the following section. ∎

2. Computing the Minimum in Constant Time

In this section we present a parallel algorithm for computing the minimum of n numbers, which we refer to as elements to avoid confusion, in constant expected time. Note that we present an algorithm and not just a decision tree method.

Select Minimum

Input: Set of $x_1, x_2, \ldots, x_m$ elements
Output: *minval*

Begin
/* Initialize Data Structures */
$(0 \leq i < m)$ $R[i] \leftarrow$ "Nobody is smaller"

/* Perform All Pairwise Comparisons */
$(0 \leq i, j < m)$ if $x_i < x_j$ then
$R[i] \leftarrow$ "Somebody is Bigger"

/* Find element that is smallest */
$(0 \leq i < m)$ if $R[i] =$ "Nobody is smaller" then
minval $\leftarrow x_i$

End

Figure 2: Minimum of m elements with m^2 processors

We first review the well known method to compute the minimum of m elements with m^2 processor in constant time. Let $x_1, x_2, \ldots, x_m$ be the m elements. The m^2 processors are used to perform all pairwise comparisons in parallel. A vector of size m is used to record all the x_i for which there is at least one element smaller than it. There will be one x_i for which the corresponding array element has not been set. The algorithm in the figure 2 makes this precise.

The improvement in which the minimum of n elements can be found in constant time with n processors is based on a lovely idea independently discovered by Reischuk and Meggido. The idea is to randomly choose square root of n of the elements and compute their minimum in constant time with the n processors according to the above description. With very high probability, there should remain only about $\sqrt{n}$ elements that are smaller than this minimum. A second and possible a third iteration will almost surely find the real minimum of the whole set. Reischuk examined the more general case of choosing the k^{th} largest and presents an upper bound on the expected number of phases of comparison steps (i.e. a decision tree model). Meggido also assumes a decision tree model and shows that the minimum can be found in constant time "almost surely".

We are concerned with a parallel algorithm and not a decision tree result, and so, we must consider the processor allocation problem. Moreover, we precisely compute exactly the expected number of iterations required to compute the minimum.

Let $m = \lfloor \sqrt{n} \rfloor$ with the n elements denoted $u_1, u_2, \ldots, u_n$. The first step of the decision tree method poses no problem for processor allocation; the first m processors randomly

Improved Minimum Selection

Input: n numbers $u_1, u_2, \ldots, u_n$ and n PE's,
Output: the minimum value, *minval*.

```
                Loop
                    /* Initialize Data Structures */
(0 < i ≤ m)         Test[i] := nil
                    IsMin[i] := "Nobody is smaller"

(i = 1)             Done := true
                    minval := ∞

                    /* Choose m Elements */
(0 < i ≤ n)         if u_i < minval then
                    r_i := Random Index between 1 and m
                    Test[r_i] := u_i

                    /*   Compare all pairs and record results   */
(0 < ij ≤ m)        if Test[i] < Test[j] then
                        IsMin[j] := "Somebody is smaller"

                    /* Pick minimum */
(0 < i ≤ m)         if IsMin[i] = "Nobody is smaller" then
                        minval := Test[i]

                /*   Are we done?   */
(0 < i ≤ n)         if u_i < minval then done := false
                Until Done
```

Figure 3: Minimum of n elements with n processors

choose elements with the element chosen by PE_i assigned to x_i. After the first iteration, however, there may be many candidate elements remaining although in no apparent order. It is not clear how to assign at most m of the remaining elements to the array x. We propose the following method (see Figure 3):

Instead of choosing a candidate element, u_j to assign to x_i, we invert the procedure and have each candidate element u_j choose a x_i. That is, for each element, u_j, that is smaller than the current minimum processor j randomly chooses an index i and assigns u_j to x_i. If more than one element chooses the same index then we assume that an arbitrary one of the assignments succeed. Since about $1/e$ of the indices will be nonempty, the iteration can proceed choosing a new minimum.

More precisely, we present a parallel program outline, called *APSPA* and then derive its expected running time.

We now proceed to prove that with high probability (*i.e.*$1 - o(n^{-3})$) only a small number (three or four) of iterations are required to find the minimum. (Note that due to our method of assigning elements to the array Test we cannot use the proofs of Meggido or Reischuk.)

Let T_k denote the set of elements successfully stored in Test during the kth iteration and let C_k denote the set of candidates, that is $C_k = \{u_i < minval_{k-1}\}$, where $minval_{k-1}$ is the value of *minval* after the $k-1$ iteration. We examine the probable sizes of these sets and prove:

Theorem 1: The following five statements about the sets Test and Candidates are all true. (We precede each statement with an informal, intuitive description in parentheses.)

(a) (With high probability, during the first iteration, all of Test gets filled.)

$$\Pi_A = \Pr(|T_1| < m) = o(n^{-1}) \quad o(n^{-3})$$

(b) (With high probability, the minimum chosen at the end of the first iteration is no larger than $\frac{4}{3}m\log n$ of the elements.)

$$\Pi_B = \Pr(|C_2| \geq \tfrac{4}{3}m\log n \text{ given that } |T_1| = m) = o(n^{-1}) \quad o(n^{-3})$$

(c) (With high probability, during the second iteration, 1/4 of Test gets filled or, if C_1 is small then Test gets at least 1/4 of C_1.)

$$\Pi_C = Pr(|T_2| \leq .25\min(m, |C_2|) \text{ given that } |C_2| < \tfrac{4}{3}m\log n) = o(n^{-1}) \quad o(n^{-3})$$

(d) (With high probability, the minimum chosen at the end of the second iteration is no larger than $33\log^2 n$ of the elements.)

$$\Pi_D = \Pr(\ |C_3| \geq 33\log^2 n \text{ given that } |C_2| < 2m\log n \text{ and } |T_2| > .25\min(m, |C_2|)) = o(n^{-1}) \quad o(n^{-3})$$

(e) (With high probability, during the third iteration all but at most 6 of the remaining candidates are assigned to Test.)

$$\Pi_E = \Pr(|C_3 - T_3| \geq 3 \text{ given that } |C_3| < 17\log^2 n) = o(n^{-1})$$

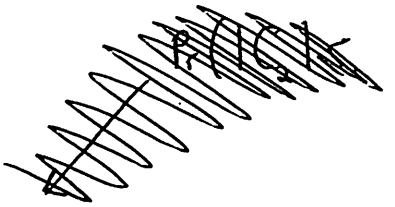
We deduce then that with high probability after three iterations the set of candidates is reduced to at most six elements and thus the minimum will be found after no more than four iterations.

Proof. of (a): Consider during the first iteration how elements are assigned to the array Test. The probability that some index of Test is not chosen is:

$$\Pr(\text{ Test[i] = nil }) = (1 - 1/m)^n \leq e^{-m}$$

and so

$$\begin{aligned} \Pi_A &= \Pr(|T_1| < m) \\ &\leq \textstyle\sum_{i=1}^{m} \Pr(\text{ Test[i] = nil }) \leq me^{-m} \end{aligned} \quad = o(n^{-d}) \text{ for any } d > 0.$$



∎

Proof. of (b): This result follows from the fact that the number of ways to form the array Test without choosing any of the $4m\log n$ smallest elements is very small compared to the number of ways to form it choosing from all n elements. More formally:

$$T_1 \text{ is a random } m\text{-subset of } N \text{ and if } \quad |C_2| \geq s \doteq \lceil 4m\log n \rceil$$

then there are only $\binom{n-s}{m}$ choices for T_1. Thus

$$\Pi_B \leq \binom{n-s}{m} / \binom{n}{m} \leq (1-s/n)^m \leq e^{-sm/n} < e^{-4\log n} = o(n^{-1})$$

∎

Proof. of (c): If $|\ C\ |$ is big then we expect about 1/e of Test to be filled. We use the value 1/4 since it is less than 1/e and also works for when C is small. ~~Also note that the summation is begun from 4 since the probabilities for $k < 4$ are all zero.~~

$$\Pi_C \leq \sum_{k=9}^{s-1} \Pi_k / \Pr(|C_2| < s)$$

where

$$\Pi_k = \Pr(|T_2| \leq .25\min(m,k) \text{ given that } |C_2| = k)$$

Let $k_1 = \lfloor k/4 \rfloor$, then for $k \leq m$,

$$\begin{aligned}
\Pi_k &\leq \Pr(\text{ there exists a } k_1\text{-subset } S \text{ of } \{1,2,\ldots,m\} \\
&\qquad \text{such that for each } i \in C_2 \text{ processor } PE_i \text{ chooses a random integer in } S) \\
&\leq \binom{m}{k_1}(k_1/m)^k \\
&\leq (me/k_1)^{k_1}(k_1/m)^k \\
&\leq (ke^{1/3}/4m)^{3k/4}.
\end{aligned}$$

Also $k > m$ implies $\Pi_k \leq \Pi_m$ and so

$$\Pi_C \leq \left(\frac{1}{\Pr(|C_2| < s)}\right) \sum_{k=9}^{m} (ke^{1/3}/4m)^{3k/4} + 2m\log n(e^{1/3}/4)^{3m/4} = o(m^{-3}).$$

∎

Proof. of (d): We will use the following shorthand notation:

$$\begin{aligned}
p &: \lceil 33\log^2 n \rceil \\
A_k &: (|C_2| = k) \\
B_j &: (|T_2| = l) \\
W_c &: (|C_2| < 2m\log n) \\
W_t &: (|T_2| > \min\{m, |C_2|\}/4)
\end{aligned}$$

Using this notation, we rewrite and proceed as follows:

$$\begin{aligned}
\Pi_D &= \Pr(|C_3| \geq p \mid W_c \ \& \ W_t) \\
&= \sum_{k=p}^{2m\log n} \sum_{l=.25\min\{m,|k|\}}^{k} \frac{\Pr(|C_3|\geq p \mid A_k \ \& \ B_l)\ \Pr(A_k \ \& \ B_l)}{\Pr(W_c \ \& \ W_t)} \\
&\leq \sum_{k=p}^{2m\log n} \sum_{l=.25\min\{m,|k|\}}^{k} \binom{k-p}{l} / \binom{k}{l} \\
&\leq \sum_{k=p}^{2m\log n} \sum_{l=.25\min\{m,|k|\}}^{k} (1-p/k)^l \leq \sum\sum e^{-33\log n/8}
\end{aligned}$$

and (d) follows. ∎

Proof. of (e): Imagine that $r_i, i \in C_3$, elements are chosen sequentially. Then $|C_3 - T_3|$ is the number of times an r_i, just generated, has been chosen before. But the probability of this happening is never more than p/m regardless of the previous values of the r_i's. As $C_3 < p$ we have

$$\Pr(|(C_3 - T_3)| \geq 3) \leq \sum_{t=3}^{p} \left(\binom{p}{t}\right)(p/m)^t(1-p/m)^{p-t} = O(p^{14}/m^{7})$$

(the terms in the sum reduce by at least a factor of p^2/m as k increases.) ∎

The result of this paper can be summed up as follows:

Theorem 2: The expected running time of MPA is $O(\log\log n)$.

Proof. From Lemmas 1 and 2, the expected running time is $O(\log(37\log^2 n) + n^{-.1}\log^2 n)$. ∎

Bibliography

Dekel, E., D. Nassimi, S. Sahni, "Parallel Matrix and Graph Algorithms," *SIAM Journal of Computing*, Vol. 10, (1981).

Frieze, A. and G. R. Grimmett, "The Shortest Path Problem for Graphs with Random Arc-Lengths" *to appear in Discrete Applied Mathematics.*

Megiddo, N. "Parallel Algorithms for Finding the Maximum and the Median Almost Surely in Constant Time," Preliminary Manuscript, GSIA, Carnegie-Mellon University, Pittsburgh, PA (1982).

Reif, J. and P. Spirakis, "Expected Parallel Time and Sequential Space Complexity of Graph and DiGraph Problems," Technical Report No. 91, New York University, Oct. 1983.

Reischuk, R., "A Fast Probabilistic Parallel Sorting Algorithm," *22nd Symposium on Foundations of Computer Science*, (1981), 212-219.

Shilaach, Y. and V. Vishkin, "Finding the Maximum, Merging, and Sorting in a Parallel Computation Model," *Journal of Algorithms*, Vol. 2, (1981), 88-102.

Valiant, L. G., "Parallelism in Comparison Problems," *SIAM Journal of Computing*, Vol. 4, No. 3, (1975) 348-355.

Bibliography