Optimal Computation of Shortest Paths on Doubly Convex Bipartite Graphs

L. CHEN
FRL, P.O. Box 18345
Los Angeles, CA 90018, U.S.A.

(Received July 1998; revised and accepted April 1999)

Abstract—An optimal parallel algorithm for computing all-pair shortest paths on doubly convex bipartite graphs is presented here. The input is a (0,1)-matrix with consecutive 1’s in each of its rows and columns that represents a doubly convex bipartite graph. Our parallel algorithm runs in $O(\log n)$ time with $O(n^2 / \log n)$ processors on an EREW PRAM and is time-and-work-optimal. As a by-product, we show that the problem can be solved by a sequential algorithm in $O(n^2)$ time optimally on any adjacency list or matrix representing a doubly convex bipartite graph. The result in this paper, improves a recent work on the problem for bipartite permutation graphs which are properly contained in doubly convex bipartite graphs. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords—Shortest paths, Doubly convex bipartite graphs, Sequential and parallel algorithms, Optimality.

1. INTRODUCTION

The shortest-path problem is a well-known problem. In this work, we show that the all-pair shortest-path problem on doubly convex bipartite graphs can be solved in $O(\log n)$ time with $O(n^2 / \log n)$ EREW PRAM processors. The parallel algorithm requires no concurrent access and is time-and-work-optimal, so the corresponding sequential algorithm is also optimal. Therefore, the result here has lasting value. The shortest-path problem is a fundamental step in solving many other problems. It has been studied in at least a dozen papers. One efficient solution to the single-source shortest-path problem was discovered by Dijkstra [1]. One well-known algorithm for the all-pair shortest-path problem is due to Floyd [2], who obtained the result based on a theorem by Warshall [3]. More algorithms for the problem are presented in [4]. For unweighted graphs, the current asymptotically fastest sequential algorithm, designed by Seidel [5], runs in $O(M(n) \log n)$ time, where $M(n)$ denotes the time necessary to multiply two matrices of order $n$ and is $o(n^{2.376})$ from the work of Coppersmith and Winograd [6]. When restricted to some special classes of graphs, the problem can be solved more efficiently or even optimally. On bipartite permutation graphs, Chen [7] designed an optimal CREW PRAM algorithm for finding all-pair shortest paths, assuming input graphs are strongly ordered. In this work, we make two improvements on [7].

1. We show that the problem can be solved as efficiently for doubly convex bipartite graphs, a proper superclass of bipartite permutation graphs.
2. We show that the problem can be solved with no concurrent access within the same bounds.

Some applications of doubly convex bipartite graphs are mentioned in [8].

In next section, we give some definitions and results that are helpful for obtaining the main result of this paper. Before presenting our parallel algorithm, we present an $O(n^2)$ time sequential algorithm in Section 3. Although the trivial sequentialization of the parallel algorithm yields an $O(n^2)$ time sequential algorithm, the included sequential algorithm is more space-efficient. Moreover, reading the sequential algorithm should help understand how the parallel algorithm works. In Section 4, we describe the parallel algorithm, the main algorithm in this paper. Finally, in Section 5, we conclude the paper with some remarks.

2. PRELIMINARIES

Throughout this work, we shall use $M_{X \times Y}$ to denote the $X$ vis-à-vis $Y$ incidence matrix for a bipartite graph $G = (X, Y, E)$. For any vertex $v$, denote by $f(v)$ and $l(v)$, respectively, the indices of the first 1 and the last 1 in the row (or column) of $M_{X \times Y}$ corresponding to $v$.

We say a $(0, 1)$-matrix satisfies the consecutive 1s property for rows (or columns) if its columns (or rows) can be permuted such that the resulting matrix has consecutive 1s in each of its rows (or columns). The property was treated in our earlier work [9]. A bipartite graph $G = (X, Y, E)$ is a doubly convex bipartite graph if matrix $M_{X \times Y}$ satisfies the consecutive 1s property for both rows and columns. Recently, Chen [10] obtained a matrix characterization of doubly convex bipartite graph and showed that a graph is a doubly convex bipartite if and only if its adjacency matrix satisfies the consecutive 1s property. A bipartite graph $G = (X, Y, E)$ is strongly ordered if the 1s in each row of $M_{X \times Y}$ are consecutive and for any row except the top one, the 1s neither begin nor end before the 1s in the preceding row [11]. A bipartite permutation graph is a bipartite graph that can be strongly ordered [12]. It is now easy to see that bipartite permutation graphs are properly contained in doubly convex bipartite graphs.

The model of parallel computation employed in this paper, is the well-known Parallel Random Access Machine (PRAM). Our parallel algorithm is implemented on the Exclusive Read Exclusive Write (EREW) PRAM, where no concurrent access is allowed. Also mentioned is the concurrent read exclusive write (CREW) PRAM, where multiple processors can read from a memory location simultaneously but cannot write into a memory location simultaneously. For more information about the PRAM model, the reader is referred to [13].

Suppose an $X$ vis-à-vis $Y$ incidence matrix $M_{X \times Y}$ of size $s \times t$ for a bipartite graph $G = (X, Y, E)$ has consecutive 1s in each of its rows and columns. Then $f(x_i)$ and $l(x_i)$, for each $x_i$, can be easily computed in $O(1)$ time with $O(t)$ work on an EREW PRAM. So all the $f$s and $l$s can be computed in $O(1)$ time with $O(st)$ work on an EREW PRAM. With all the $f$s and $l$s, we can identify all connected components immediately. For example, two vertices $x_i$ and $x_{i+1}$ are in the same connected component if and only if two intervals $[f(x_i), l(x_i)]$ and $[f(x_{i+1}), l(x_{i+1})]$ intersect. Below we shall assume, without loss of generality, that the input graph is connected. Then, as observed in Lipski and Preparata [8], there exist two integers, $p$ and $q$, $1 < p < q < s$, such that the following conditions are satisfied.

1. Sequence $(f(x_1), \ldots, f(x_p))$ is nonincreasing, and sequence $(l(x_1), \ldots, l(x_p))$ is nondecreasing.
2. Both sequences $(f(x_{p+1}), \ldots, f(x_{q-1}))$ and $(l(x_{p+1}), \ldots, l(x_{q-1}))$ are nondecreasing or nonincreasing.
3. Sequence $(f(x_q), \ldots, f(x_s))$ is nondecreasing, and sequence $(l(x_q), \ldots, l(x_s))$ is nonincreasing.

So the matrix (or $X$) is divided into three parts: upper, middle, and lower parts. The middle part may be empty. It is always possible to make the two sequences for the nonempty middle part nondecreasing by reversing the order of the columns if necessary. In this case, the middle
part corresponds to a strongly ordered bipartite permutation graph and $G$ is called a properly ordered doubly convex bipartite graph. A doubly convex bipartite graph is a properly ordered doubly convex bipartite graph if and only if each of its connected components is. For a connected properly ordered doubly convex bipartite graph, there can be multiple distinct pairs of $(p, q)$ that satisfy the conditions mentioned above. In our procedure described below, we find the pair $(p, q)$ with maximum value. The middle part is empty if and only if $0 \leq q - p \leq 1$. It is easy to see that there exists at least one vertex in $Y$ which is adjacent to every vertex in the upper part of $X$ and there exists at least one vertex in $Y$ which is adjacent to every vertex in the lower part of $X$.

Some related properties of doubly convex bipartite graphs are given in [14], and an $O(\log^2 n)$-time $O(n^3/\log^2 n)$-processor recognition algorithm on CREW PRAM is presented there. Recently, Chen [15] obtained an improved algorithm that runs in $O(\log n)$ time with $O(M(n))$ processors, where $M(n)$ denotes the processor bound for multiplying two $n \times n$ matrices in $O(\log n)$ time and is $o(n^{2.376})$ from the work of Coppersmith and Winograd [6].

Below we present some results that help justify the correctness of our results. We shall use $d(v_i, v_j)$ to denote the distance (i.e., the number of edges in the shortest path) between $v_i$ and $v_j$. Assume $G = (X, Y, E)$ is a properly ordered doubly convex bipartite graph, where $X = \{x_1, x_2, \ldots, x_s\}$, $Y = \{y_1, y_2, \ldots, y_t\}$, and $s + t = n$.

**Lemma 1.** Suppose $i < j < k$. Then $d(x_i, x_j) \leq d(x_i, x_k)$.

**Proof.** Let $(x_i, v_1, \ldots, v_z, x_k)$ be an arbitrary shortest path from $x_i$ to $x_k$.

**Case 1.** $x_j$ is one of the vertices on the path. In other words, $x_j = v_h$ for an $h$. Then we can see easily that $d(x_i, x_j) < d(x_i, x_k)$.

**Case 2.** $x_j$ is not on the path. Then there exists a subpath, say $(x_p, y_r, x_q)$, such that $p < j < q$. Since the 1s in each column (including column $r$) of $M_{X \times Y}$ are consecutive, it follows that $(y_r, x_j)$ is an edge. Therefore, we can see that $d(x_i, x_j) \leq d(x_i, x_k)$.

**Lemma 2.** Suppose $f(x_i) < j < k \leq t$. Then $d(x_i, y_j) \leq d(x_i, y_k)$.

**Proof.** Let $(x_i, v_1, \ldots, v_z, y_k)$ be an arbitrary shortest path from $x_i$ to $y_k$.

**Case 1.** $y_j$ is one of the vertices on the path. Then we can see easily that $d(x_i, y_j) < d(x_i, y_k)$.

**Case 2.** $y_j$ is not on the path. Then there exists a subpath, say $(y_p, x_r, y_q)$, such that $p < j < q$. Since the 1s in each row (including row $r$) of $M_{X \times Y}$ are consecutive, it follows that $(x_r, y_j)$ is an edge. Therefore, we can see that $d(x_i, y_j) \leq d(x_i, y_k)$.

With the above theorems and lemma, it will not be hard to prove the correctness of the algorithms presented later in a rigorous way.

In designing PRAM algorithms, we often use the following result, usually attributed to Brent [16], to improve the processor bound without affecting the total time bound.

**Theorem 1.** If a problem can be solved in $O(T)$ time with $O(W)$ work on a PRAM, then the problem can also be solved in $O(T + W/P)$ time with $P$ processors on the same PRAM.

The use of the above theorem should be understood even though we make no explicit reference to the theorem later in this paper.

### 3. THE SEQUENTIAL ALGORITHM

Assume that the input is a matrix, $M_{X \times Y}$, for a connected properly ordered doubly convex bipartite graph $G = (X, Y, E)$. The output of the algorithm is a square matrix $M$ of order $n$, where $n$ is the number of the vertices in the graph and $m_{i,j}$ gives the length of the shortest path from node $i$ to node $j$, for $0 < i, j \leq n$. Such a matrix is called a shortest-path adjacency matrix for the graph. Initially, $M$ is set to $[M_{X \times Y}^T \ 0 \ 0 \ 0]$, where $M_{X \times Y}^T$ is the transpose of $M_{X \times Y}$. Note that the 1s in each row and in each column of $M_{X \times Y}$ are consecutive. If $v \in X$, then $0 < f(v) \leq l(v) \leq t$. If $v \in Y$, then $0 < f(v) \leq l(v) \leq s$. 
Later we shall show that, in $O(n)$ time, we can compute the lengths of the shortest paths from an arbitrary vertex, say $x_i$, in $X$ to all other vertices, or equivalently, the final value of the row (of $M$) corresponding to $x_i$.

The following three cases are considered in the procedure.

1. $x_i$ is in the upper part.
2. $x_i$ is in the middle part.
3. $x_i$ is in the lower part.

Case 3 is analogous to Case 1, so the discussion will be omitted. We observe that if $x_i$ and $x_j$ are two distinct vertices in the upper part, then the distance between them is 2.

If $x_i$ is in the upper part, we shall move down and move right as far as possible step by step until we have reached a vertex in the lower part. Suppose $x_i$ is in the middle part. Starting from $x_i$, we shall move up and move left as far as possible step by step until we have reached a vertex in the upper part. We shall also move down and move right, starting from $x_i$, as far as possible step by step until we have reached a vertex in the lower part. Then we can obtain the distance from $x_i$ to all other vertices easily. The following is the procedure.

1. $p := s$; $q := s$; $j := 1$; \{initialize $p$ and $q$\}
2. $do$ $j < s$ \{compute $p$\}
   3. \hspace{1em}if $f(x_j) < f(x_{j+1}) \lor l(x_j) > l(x_{j+1})$ \{$p := j$; exit fi;\}
   4. \hspace{1em}$j := j + 1$;
   5. $od$;
3. $do$ $j < s$ \{compute $q$\}
   7. \hspace{1em}if $l(x_j) > l(x_{j+1})$ \{$q := j + 1$; exit fi\}
   8. \hspace{1em}$j := j + 1$;
   9. $od$; \{$x_i$ is in the upper part if $i < p$, and in the lower part if $i > q$\}
10. $if$ $i < q$ \{move down and move right until a vertex in the lower part is reached\}
   11. \hspace{1em}$a := i$; $b := l(x_i)$; $d := 1$;
   12. $do$ $a < q$ \{compute distance from $x_i$ to $x_j$, for $j = i - 1, \ldots, 1$\}
   13. \hspace{1em}if $m_i,j = 0$ \{$m_i,j := d + 2$;\}
   14. \hspace{1em}if $m_i,j > 0$ \{$d := m_i,j$;\}
   15. \hspace{1em}$fi$;
   16. \hspace{1em}$fi$;
   17. $if$ $i > p$ \{move up and move left until a vertex in the upper part is reached\}
   18. \hspace{1em}$a := i$; $b := f(x_i)$; $d := 1$;
   19. $do$ $a > p$ \{compute distance from $x_i$ to $x_j$, for $j = i + 1, \ldots, s$\}
   20. \hspace{1em}if $m_i,j = 0$ \{$m_i,j := d + 2$;\}
   21. \hspace{1em}if $m_i,j > 0$ \{$d := m_i,j$;\}
   22. \hspace{1em}$fi$;
   23. \hspace{1em}$fi$;
   24. $d := 0$; $j := i - 1$;
   25. $do$ $j > 0$ \{compute distance from $x_i$ to $x_j$, for $j = i - 1, \ldots, 1$\}
   26. $if$ $m_i,j = 0$ \{$m_i,j := d + 2$;\}
   27. $if$ $m_i,j > 0$ \{$d := m_i,j$;\}
   28. $fi$;
   29. $od$;
   30. $d := 0$; $j := i + 1$;
   31. $do$ $j < s$ \{compute distance from $x_i$ to $x_j$, for $j = i + 1, \ldots, s$\}
   32. $if$ $m_i,j = 0$ \{$m_i,j := d + 2$;\}
   33. $if$ $m_i,j > 0$ \{$d := m_i,j$;\}
   34. $fi$;
Consider the following sample input matrix $M_{X \times Y}$ for a properly ordered bipartite graph:

\[
\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\]

In this case, $p = 4$ and $q = 9$.

Suppose we need to compute the shortest paths from $x_2$ to all other nodes. Row 2 of the matrix has the following configuration at Line 16.

\[
\begin{array}{cccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 1 & 0 & 0 \\
3 & 0 & 1 & 1 & 1 & 0 & 0 \\
4 & 1 & 1 & 1 & 1 & 0 & 0 \\
5 & 0 & 0 & 1 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 1 & 1 & 0 \\
7 & 0 & 0 & 0 & 0 & 1 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 1 \\
9 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
\]

Lines 17–23 have no effect on the configuration, and the subsequent lines compute the shortest paths from $x_2$ to all other nodes, based on the known information. The final configuration for the row is as follows.

\[
\begin{array}{cccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \\
0 & 0 & 0 & 0 & 0 & 2 & 4 & 0 & 6 & 0 & 0 & 1 & 1 & 3 & 5 & 0
\end{array}
\]

Now suppose we need to obtain the distance from $x_7$ to all other vertices. Row 7 of the matrix has the following configuration at Line 16.

\[
\begin{array}{cccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \\
2 & 0 & 2 & 2 & 2 & 2 & 4 & 6 & 6 & 3 & 3 & 1 & 1 & 3 & 5 & 7
\end{array}
\]
When the control reaches Line 23, the new configuration for the row is as follows.

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td>x_2</td>
<td>x_3</td>
<td>x_4</td>
<td>x_5</td>
<td>x_6</td>
<td>x_7</td>
<td>x_8</td>
<td>x_9</td>
<td>y_1</td>
<td>y_2</td>
<td>y_3</td>
<td>y_4</td>
<td>y_5</td>
<td>y_6</td>
<td>y_7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The subsequent lines obtain the distance from x_7 to the rest of the vertices, based on the known information. The final configuration for the row is as follows.

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td>x_2</td>
<td>x_3</td>
<td>x_4</td>
<td>x_5</td>
<td>x_6</td>
<td>x_7</td>
<td>x_8</td>
<td>x_9</td>
<td>y_1</td>
<td>y_2</td>
<td>y_3</td>
<td>y_4</td>
<td>y_5</td>
<td>y_6</td>
<td>y_7</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

It can be readily seen that the time complexity of the algorithm is $O(n)$. Slightly modifying the preceding procedure, we can give a procedure which obtains the final value of the row (of M) corresponding to a vertex of Y in $O(n)$ time. By now, we can see that all the elements of M can be obtained in $O(n^2)$ time. If a doubly convex bipartite graph is not properly ordered, it can be turned into a properly ordered one within the same time bound (see, e.g., [8]). Therefore, we can conclude the following.

**Theorem 2.** The all-pair shortest-path problem on doubly convex bipartite graphs can be solved in $O(n^2)$ time.

### 4. THE PARALLEL ALGORITHM

Below we describe an efficient parallel algorithm that obtains the final value of the row (of M) corresponding to a vertex in X. Let $x_i$ be an arbitrary vertex in X. To obtain the final value of the row corresponding to $x_i$, we construct a directed and ordered tree of size $n$ with $x_i$ as the root and each node of the tree corresponds to a vertex of the input properly ordered doubly convex bipartite graph. The order of the children of a node of the tree follows the vertex order of the input bipartite graph. The tree has the property that any path between the root and another node coincides with a shortest path between these two vertices in the doubly convex bipartite graph.

The tree is constructed based on the following rules.

1. $x_i$ is the root of the tree.
2. If $y_j$ is adjacent to $x_i$, then $y_j$’s parent is $x_i$.
3. If $x_i$ is in the upper part (i.e., $i \leq p$), then:
   - 3.1. if $y_j$ is not adjacent to $x_i$, then $y_j$’s parent is $x_f(y_j)$;
   - 3.2. if $j \neq i$ and $f(x_i) \in \{f(x_j), l(x_j)\}$, then $x_j$’s parent is $y_f(x_i)$;
   - 3.3. if $f(x_i) \notin \{f(x_j), l(x_j)\}$, then $x_j$’s parent is $y_f(x_j)$.
4. If $x_i$ is in the middle part (i.e., $p < i < q$), then:
   - 4.1. if $y_j$ is not adjacent to $x_i$ and $l(y_j) < i$, then $y_j$’s parent is $x_l(y_j)$;
   - 4.2. if $y_j$ is not adjacent to $x_i$ and $l(y_j) > i$, then $y_j$’s parent is $x_f(y_j)$;
   - 4.3. if $j < i$, then $x_j$’s parent is $y_l(x_j)$;
   - 4.4. if $j > i$, then $x_j$’s parent is $y_f(x_j)$.
5. If $x_i$ is in the lower part (i.e., $i \geq q \neq p$), then the tree construction is analogous to the case in which $x_i$ is in the upper part.
Figure 1. Explicit tree, I.

Suppose we need to compute the length of the shortest paths from $x_2$ to all other nodes. The constructed tree is in Figure 1. Now, suppose we need to find the shortest paths from $x_7$ to all other nodes. The constructed tree is given in Figure 2.

The tree is conceptually constructed in a bottom-up fashion, and is obviously a rooted spanning tree. In our sequential algorithm presented earlier, we also implicitly construct a rooted tree, but in a top-down fashion. Moreover, the implicit tree constructed by the sequential algorithm seldom includes all the vertices of the input doubly convex bipartite graph. The implicit trees corresponding to the explicit trees in Figures 1 and 2 are depicted in Figure 3.

To obtain an efficient implementation of the parallel algorithm, we make some further investigations.

**Lemma 3.** If $x_i$ is in the upper part, then the constructed tree satisfies the following properties.

1. $x_1, \ldots, x_{l(y_j(x_i))}$, except $x_i$, are all the children of $y_{f(x_i)}$.
2. If $l(y_j) > l(y_{j-1})$ and $f(x_j) < j$, then $x_{l(y_{j-1})+1}, \ldots, x_{l(y_j)}$ are all the children of $y_j$.
3. If $f(x_j) < f(x_{j-1}) \leq f(x_i)$, then $y_{f(x_i)}, \ldots, y_{f(x_{j-1})-1}$ are all the children of $x_j$.
4. If $l(x_i) \leq l(x_{j-1}) < l(x_j)$, then $y_{l(x_{j-1})+1}, \ldots, y_{l(x_j)}$ are all the children of $x_j$.

**Proof.** Suppose $j \neq i$ and $f(x_j) \in [f(x_j), l(x_j)]$. Then we can conclude from Rule 3.2 that $x_j$ is a child of $y_{f(x_i)}$ in the constructed tree. Such a $j$ must be one of the numbers in the ordered
set \{1, \ldots, (i-1), (i+1), \ldots, l(y_{f(x_i)})\}, since \(f(x_i) \in [f(x_j), l(x_j)]\) if and only if \(x_j\) is adjacent to \(y_{f(x_i)}\). Therefore, \(x_1, \ldots, x_{l(y_{f(x_i)})}\), except \(x_i\), are all the children of \(y_{f(x_i)}\).

Suppose \(l(y_j) > l(y_{j-1})\) and \(f(x_1) < y\). Then \(f(x_k) = j\) if \(l(y_{j-1}) < k \leq l(y_j)\). We can now conclude from Rule 3.3 that \(x_k\) is a child of \(y_j\) if \(l(y_{j-1}) < k \leq l(y_j)\). Since all \(y_j\)'s children are constructed using Rule 3.3, we conclude that \(y_{f(y_j)}, \ldots , y_{f(y_j)-1}\) are all the children of \(y_j\).

Suppose \(f(x_j) < f(x_{j-1}) \leq f(x_i)\). Then \(f(y_k) = j\) if \(f(x_j) \leq k < f(x_{j-1})\). We can now conclude from Rule 3.1 that \(y_k\) is a child of \(x_j\) if \(f(x_j) \leq k < f(x_{j-1})\). Since all \(x_j\)'s children are constructed using Rule 3.1, we conclude that \(y_{f(x_j)}, \ldots , y_{f(x_{j-1})-1}\) are all the children of \(x_j\).

Note that in this case, \(x_j\) is in the upper part, but it is below \(x_i\) (i.e., \(j > i\)).

Suppose \(l(x_i) \leq l(x_{j-1}) < l(x_j)\). Then we can conclude from Rule 3.1 that \(y_k\) is a child of \(x_j\) if \(l(x_{j-1}) < k \leq l(x_j)\). Since all \(x_j\)'s children are constructed using Rule 3.1, we conclude that \(y_{l(x_{j-1})+1}, \ldots , y_{l(x_j)}\) are all the children of \(x_j\). Note that in this case, \(x_j\) is in the lower part.

**Lemma 4.** If \(x_i\) is in the middle part, then the tree satisfies the following properties.

1. If \(f(x_j) < f(x_{j+1})\) and \(i > j\), then \(y_{f(x_j)}, \ldots , y_{f(x_{j+1})-1}\) are all the children of \(x_j\).
2. If \(l(x_{j-1}) < l(x_j)\) and \(i < j\), then \(y_{l(x_{j-1})+1}, \ldots , y_{l(x_j)}\) are all the children of \(x_j\).
3. If \(f(y_j) < f(y_{j+1})\) and \(f(y_j) < i\), then \(x_{f(y_j)}, \ldots , x_{\min\{f(y_{j+1}), i\}-1}\) are \(y_j\)'s children.
4. If \(l(y_j) > l(y_{j-1})\) and \(l(y_j) > i\), then \(x_{\max\{l(y_{j-1}), i\}+1}, \ldots , x_{l(y_j)}\) are \(y_j\)'s children.

**Proof.** Suppose \(f(x_j) < f(x_{j+1})\) and \(i > j\). Then we can conclude from Rule 4.1 that \(y_k\) is a child of \(x_j\) if \(f(x_j) \leq k < f(x_{j+1})\). Since all the children of \(x_j\) are constructed using Rule 4.1 if \(x_j\) is above \(x_i\), we conclude that \(y_{f(x_j)}, \ldots , y_{f(x_{j+1})-1}\) are all the children of \(x_j\).

Suppose \(l(x_{j-1}) < l(x_j)\) and \(i < j\). Then we can conclude from Rule 4.2 that \(y_k\) is a child of \(x_j\) if \(l(x_{j-1}) < k \leq l(x_j)\). Since all the children of \(x_j\) are constructed using Rule 4.2 if \(x_j\) is below \(x_i\), we conclude that \(y_{l(x_{j-1})+1}, \ldots , y_{l(x_j)}\) are all the children of \(x_j\).
Figure 3. Implicit trees, I and II.

Suppose $f(y_j) < f(y_{j+1})$ and $f(y_j) < i$. Then we can conclude from Rule 4.3 that $x_k$ is a child of $y_j$ if $f(y_j) \leq k < \min\{f(y_{j+1}), i\}$.

Suppose $l(y_j) > l(y_{j-1})$ and $l(y_j) > i$. Then we can conclude from Rule 4.4 that $x_k$ is a child of $y_j$ if $\max\{l(y_{j-1}), i\} < k \leq l(y_j)$.

Note that if $f(y_j) < f(y_{j+1})$ and $f(y_j) < i$, $x_{f(y_j)}, \ldots, x_{\min\{f(y_{j+1}), i\}}$ are not necessarily the only children of $y_j$. If, in addition, $l(y_j) \leq l(y_{j-1})$ or $l(y_j) \leq i$, then those nodes are the only children of $y_j$; otherwise, $x_{f(y_j)}, \ldots, x_{\min\{f(y_{j+1}), i\}}$ and $x_{\max\{l(y_{j-1}), i\}+1}, \ldots, x_{l(y_j)}$ are all the children of $y_j$.

From Rule 2, it is easy to see that $y_{f(x_i)}, \ldots, y_{l(x_i)}$ are all the children of $x_i$. If $x_i$ is in the lower part, we can have a lemma analogous to Lemma 3. It is worthwhile to point out that if no node has been referred to as a child of a node, then that node has no children. So now we can tell easily, for each node, how many children the node has and what they are. Based on this information, we can construct, for one $x_i$, directed ordered tree optimally on an EREW PRAM using some standard techniques (see, e.g., [17,18]). Once we have obtained the tree, we shall use the Euler tour technique to find the level numbers of all the nodes of the tree. All this can be done in $O(\log n)$ time with $O(n/\log n)$ processors on a PRAM without concurrent access [19,20]. We can find the shortest paths from a vertex in $Y$ to all other vertices in a similar way. Below we elaborate on some implementation details.

The first step is to obtain, from the input matrix $M_{X \times Y}$ representing a properly ordered doubly convex bipartite graph, all the $f$s and $l$s. This can be easily done in $O(1)$ time and $O(n^2)$ work on an EREW PRAM. With $f$s and $l$s, the rest of the computation can be done without access to the matrix. We can easily compute $p$ and $q$ from $f$s and $l$s in $O(\log n)$ time with $O(n)$ work on an
EREW PRAM. Now, the trees can be constructed from these data based on the rules described earlier in this section.

Suppose we have a node $x_1$ in the upper part. From Lemma 3 we can see that, to construct a tree rooted at $x_1$, we need read $f(x_1)$, $f(x_i)$, and $l(x_i)$ $O(n)$ times and read other $f$s and $l$s $O(1)$ times. By making $O(n)$ copies of $f(x_1)$, $f(x_i)$ and $l(x_i)$ and proper synchronization of program execution, we can remove all concurrent access. The data duplication can be easily done in $O(\log n)$ time and $O(n)$ work on an EREW PRAM.

Now suppose we have a node $x_i$ in the middle part. From Lemma 4 we can see that, to construct a tree rooted at $x_i$, we need read integer $i$ $O(n)$ times and read $f$s and $l$s $O(1)$ times. By making $O(n)$ copies of integer $i$ and proper synchronization of program execution, we can remove all concurrent access. The duplication of integer $i$ can be easily done in $O(\log n)$ time and $O(n)$ work on an EREW PRAM.

We can now conclude that an ordered tree rooted at $x_i$, for any $x_i$, can be constructed in $O(\log n)$ time and $O(n)$ work on an EREW PRAM. To find all-pair shortest paths, we need construct a tree for each $x_i$. As has just been shown, each tree can be constructed from the input graph represented by $n$ $f$s and $n$ $l$s in $O(\log n)$ time and $O(n)$ work on an EREW PRAM. Since we need construct $n$ trees, we make $n$ copies of all the $f$s and $l$s, which can be done in $O(\log n)$ time and $O(n^2)$ work on an EREW PRAM. It follows that all-pair shortest paths can be computed in $O(\log n)$ time and $O(n^2)$ work on an EREW PRAM. The algorithm is obviously work-optimal since the processor-time product matches the trivial lower work bound of $\Omega(n^2)$. In fact, the algorithm is also time-optimal. A frequently used technique of obtaining lower bound is by reduction (see, e.g. [21]). Well known is the tight lower bound for computing the OR or AND of $n$ bits, for which Cook, Dwork and Reischuk [22] showed that $\Omega(\log n)$ time is required on exclusive-write machines, no matter how many processors are used. By constructing an adjacency matrix for an $n$-vertex strongly ordered bipartite permutation graph from $n$ bits, Chen [7] established the lower time bound of $\Omega(\log n)$ for solving the shortest path problem on strongly ordered bipartite permutation graphs. By further constructing a permutation function, Chen [23] showed that the lower bound still holds if the graph is represented by a permutation function. By definition, a strongly ordered bipartite permutation graph is also a properly ordered doubly convex bipartite graph. So $\Omega(\log n)$ time is also required if we need to compute shortest paths from a matrix representing a properly ordered doubly convex bipartite graph. As is mentioned earlier in this section, all the $f$s and $l$s can be obtained from a matrix representing a properly ordered doubly convex bipartite graph in constant time with optimal work on EREW PRAM. It follows that $\Omega(\log n)$ time lower bound for computing shortest paths is still valid if we are given the pair of $f$ and $l$ for each vertex of a properly ordered doubly convex bipartite graph. Since the upper and lower time bounds coincide, we conclude that our parallel algorithm is time-optimal. To summarize the main result of this paper, we give the following theorem.

**Theorem 3.** The all-pair shortest-path problem on doubly convex bipartite graphs can be solved in $O(\log n)$ time with $O(n^2/\log n)$ processors on an EREW PRAM. The algorithm is time-and-work-optimal.

5. **CONCLUDING REMARKS**

In this paper, we have shown that all-pair shortest paths of a doubly convex bipartite graph can be computed in $O(\log n)$ time and $O(n^2)$ work by a time-and-work-optimal EREW PRAM algorithm. In the algorithm, we assume the input graph is properly ordered. In case such an order is not given, we can obtain one by invoking a procedure for the consecutive 1s property [15]. Chen [15] has shown that adjacency matrices for doubly convex bipartite graphs can be turned into ones with consecutive 1s in each row and column in $O(\log n)$ time with $O(M(n))$ processors on exclusive-write PRAM. It then follows easily that $O(\log n)$ time bound for solving the all-pair shortest-path problem on doubly convex bipartite graphs on exclusive-write PRAM is optimal
even if the input graphs are not properly ordered. However, in this case, the question of whether the problem can be solved by a time-optimal parallel algorithm as efficiently in work as the best sequential algorithm is open.

We note that in addition, to the fact that solving the all-pair shortest-path problem is interesting in its own right, a shortest-path adjacency matrix can also help in solving the graph isomorphism problem, a famous open problem. Recently, the concept of identification matrices was introduced (see, e.g., [24]). Let $M_1$ and $M_2$ be two matrices representing, respectively, two graphs $G_1$ and $G_2$ of a certain class $C$, according to a certain relation $\mathcal{R}$. Suppose $G_1$ and $G_2$ are isomorphic if and only if there exist two permutation matrices $P_1$ and $P_2$ such that $M_1 = P_1 M_2 P_2$. Then $M_1$ and $M_2$ are said to be identification matrices for $G_1$ and $G_2$ of $C$, with respect to $\mathcal{R}$.

Now, to test isomorphism of two graphs, given two identification matrices with respect to a certain relation, it suffices to test if, by permuting the columns, two (resulting) matrices can have the same set of rows. Several kinds of matrices including augmented adjacency matrices satisfying the consecutive 1s property have been shown to be identification matrices [24]. In a more recent work, Chen [10] has also shown that shortest-path adjacency matrices are identification matrices for all directed graphs.

Note that undirected graphs can be regarded as a special case of directed graphs with an undirected edge, say $\{v_i, v_j\}$, interpreted as two directed edges $(v_i, v_j)$ and $(v_j, v_i)$. For constructing shortest-path adjacency matrices from arbitrary graphs, no provably optimal sequential algorithm has been obtained. It will be interesting to find out how much more efficiently the problem can be solved.

REFERENCES


