Efficient Algorithms for Finding a Tree 3-Spanner on Permutation Graphs

Hon-Chan CHEN†, Shin-Huei WU††, and Chang-Biau YANG††, Nonmembers

SUMMARY A tree 3-spanner $T$ of a graph $G$ is a spanning tree of $G$ such that the distance between any two vertices in $T$ is at most 3 times of their distance in $G$. Madanlal et al. have presented an $O(n+m)$ time algorithm for finding a tree 3-spanner of a permutation graph. However, the complexity of their algorithm is not optimal, and their algorithm cannot be easily parallelized. In this paper, we will propose an improved algorithm to solve the same problem in $O(n)$ time. Moreover, our algorithm can be easily parallelized so that finding a tree 3-spanner of a permutation graph can be done in $O(\log n)$ time with $O(\frac{n}{\log n})$ processors on the EREW PRAM computational model.

key words: tree, spanner, permutation graph, algorithm

1. Introduction

Let $G = (V, E)$ be a connected graph with vertex set $V$ and edge set $E$. A graph $H = (V', E')$ is a spanning subgraph of $G$ if $V' = V$ and $E' \subseteq E$. If $H$ is an acyclic connected spanning subgraph, then $H$ is a spanning tree of $G$. A spanning tree $T$ is a tree $t$-spanner of $G$ if the distance between any two vertices in $T$ is at most $t$ times of their distance in $G$, where $t$ is a positive integer. The concept of the tree spanner has applications in distributed environments [8].

Cai and Corneil [2] showed that the problem to determine whether a graph has a tree $t$-spanner on unweighted graphs can be solved in polynomial time when $t \leq 2$, and it is NP-complete when $t \geq 4$. The case where $t = 3$ is an open problem; however, it is conjectured by Cai to be NP-complete [1]. Many researchers have studied the existence of a tree spanner on special types of graphs. For example, Madanlal et al. presented algorithms for finding tree 3-spanners on interval, permutation, and regular bipartite graphs [7]. Besides, the spanner problems on the hypercube and the bounded degree graphs were discussed in [4] and [3], respectively.

In the algorithm of Madanlal et al. [7], a tree 3-spanner of a permutation graph can be found in $O(n+m)$ time, where $n$ is the number of vertices and $m$ is the number of edges. However, their algorithm is not optimal, and it cannot be easily parallelized. In this paper, we will present an improved algorithm to solve the same problem in $O(n)$ time, assuming the input permutation graph is connected. Our algorithm can be easily parallelized so that finding a tree 3-spanner of a permutation graph can be done in $O(\log n)$ time with $O(\frac{n}{\log n})$ processors on the EREW PRAM computational model.

The rest part of this paper is organized as follows. In Sect. 2, we introduce the permutation graph and the $O(n+m)$ algorithm presented by Madanlal et al. In Sect. 3, we propose an $O(n)$ algorithm for finding a tree 3-spanner on permutation graphs and then show its correctness. The parallel algorithm for the same problem is described in Sect. 4. Finally, we give concluding remarks in Sect. 5.

2. Review

A permutation graph is defined as follows [5]. Let $\pi$ be a permutation sequence $\pi(1), \pi(2), \cdots, \pi(n)$ of the numbers $1, 2, \cdots, n$, and let $\pi^{-1}(i)$ be the position in the $\pi$ sequence where the number $i$ can be found. Then, a permutation graph $G = (V, E)$ is with vertex set $V = \{1, 2, \cdots, n\}$ and edge set $E$, where $(i, j) \in E$ if and only if $(i-j)(\pi^{-1}(i) - \pi^{-1}(j)) < 0$. A permutation graph can be represented by a permutation diagram. In the diagram, there are two sequences $1, 2, \cdots, n$ and $\pi(1), \pi(2), \cdots, \pi(n)$. Each vertex $i$ in $G$ corresponds to a line between $i$ and $\pi^{-1}(i)$ in the diagram. Two vertices are joined by an edge in $G$ if and only if the corresponding two lines intersect in the diagram. For example, in Fig. 1, $\pi(1) = 3, \pi^{-1}(1) = 4$, and $(1, 6) \in E$. From the definition, we know that for three vertices $i, j, k$, if $i < j < k$, if $i$ is adjacent to $k$, then $j$ is adjacent to $i$ or $k$.

Permutation graphs were proposed by Pnueli et al. [9], who also described an $O(n^3)$ algorithm for testing if a given undirected graph is a permutation graph. An

---

Manuscript received January 15, 2003.
Manuscript revised May 2, 2003.
†The author is with the Department of Information Management, National Chin-Yi Institute of Technology, Taichung, Taiwan.
††The authors are with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan.

Fig. 1 An example of a permutation graph. (a) A permutation graph. (b) The corresponding diagram.
improved $O(n^2)$ algorithm for the recognition of a permutation graph was presented by Spinrad [10].

The algorithm of Madanlal et al. constructs a tree 3-spanner $T$ of a permutation graph by several stages. Let $A_i = \{j|j > i$ and $j$ is adjacent to $i\}$ for $1 \leq i \leq n$. Define

$$P_{\text{max}}(i) = \max \left\{ \max A_i : i \neq \emptyset, \text{otherwise.} \right\}$$

Initially, let $i = 1$, $r_1 = 1$, $\alpha_0 = 1$, and $\alpha_1 = P_{\text{max}}(1)$. In stage $i$, add edge $(r_i, \alpha_i)$ to $T$ and find vertex sets $S_i$ and $R_i$, where $S_i = \{v | v \text{ is in interval } (\alpha_{i-1}, \alpha_i) \}$ and $v$ is adjacent to $r_i$ and $R_i = \{v | v \text{ is in interval } (\alpha_{i-1}, \alpha_i) \}$ and $v$ is not adjacent to $r_i$. For all vertices $v \in S_i$, add edge $(v, r_i)$ to $T$; for all vertices $v \in R_i$, add edge $(v, \alpha_i)$ to $T$. If $\alpha_n = n$, then the resulting graph $T$ is a tree 3-spanner of the input permutation graph and stop this algorithm. Otherwise, determine $r_{i+1}$ and $\alpha_{i+1}$ as follows for the next stage. Let $\alpha_{i+1} = \max_{v \in R_i} P_{\text{max}}(v)$.

Moreover, let $T_i = \{v | v \in R_i$ and $v$ is adjacent to $\alpha_{i+1}\}$. Then, choose $r_{i+1}$ from $T_i$ such that if $v$ is not adjacent to $r_{i+1}$ for some $v$ in interval $(\alpha_{i}, \alpha_{i+1})$, then $q$ is not adjacent to $v$ for all $q \in T_i$. Let $i = i + 1$ and do the next stage.

We use Fig. 1 to illustrate the algorithm of Madanlal et al. In stage 1, $r_1 = 1$, $\alpha_1 = P_{\text{max}}(1) = 6$, $S_1 = \{2, 3\}$, $R_1 = \{4, 5\}$, and $(1, 6)$ is an edge of $T$. We add edges $(2, 1)$, $(3, 1)$, $(4, 6)$, and $(5, 6)$ to $T$. In addition, $\alpha_2 = \max\{P_{\text{max}}(4), P_{\text{max}}(5)\} = \max\{8, 12\} = 12$, $T_1 = \{5\}$, and $r_2 = 5$. In stage 2, $S_2 = \{8, 10\}$, $R_2 = \{7, 9, 11\}$, and $(5, 12)$ is an edge of $T$. We add edges $(8, 5)$, $(10, 5)$, $(7, 12)$, $(9, 12)$, and $(11, 12)$ to $T$. Since $\alpha_2 = 12$, we stop the algorithm. The resulting tree 3-spanner is shown in Fig. 2.

In the algorithm of Madanlal et al., it needs to determine $\alpha_{i+1} = \max_{v \in R_i} P_{\text{max}}(v)$ in each stage $i$. Since finding $P_{\text{max}}$ for all vertices takes $O(m)$ time [7], the complexity of their algorithm is $O(n + m)$ even when the permutation diagram is given.

<p>| Table 1 | The values of $\pi(i)$, $\pi^{-1}(i)$, $L(i)$, and $U(i)$. |
|--------|---------------|---------------|---------------|---------------|</p>
<table>
<thead>
<tr>
<th>$i$</th>
<th>$1$</th>
<th>$2$</th>
<th>$3$</th>
<th>$4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi(i)$</td>
<td>$3$</td>
<td>$6$</td>
<td>$2$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\pi^{-1}(i)$</td>
<td>$4$</td>
<td>$3$</td>
<td>$1$</td>
<td>$6$</td>
</tr>
<tr>
<td>$L(i)$</td>
<td>$3$</td>
<td>$6$</td>
<td>$6$</td>
<td>$8$</td>
</tr>
<tr>
<td>$U(i)$</td>
<td>$4$</td>
<td>$4$</td>
<td>$4$</td>
<td>$6$</td>
</tr>
</tbody>
</table>

3. An $O(n)$ Algorithm for Finding a Tree 3-Spanner

In this section, we shall present an improved $O(n)$ algorithm for finding a tree 3-spanner of a permutation graph, which bases on the idea of the algorithm of Madanlal et al. In their algorithm, the most con-
and \(\alpha_{i+1}\) defined in the algorithm of Madanlal et al. can be substituted by our \(r_{i+1}\) and \(\alpha_{i+1}\) in each stage \(i\) of Algorithm A, \(1 \leq i \leq k - 1\).

**Lemma 1**: For any vertex \(i\), \(1 \leq i \leq n\), \(P_{\text{max}}(i) = L(\pi^{-1}(i))\).

**Proof**: By the definition of \(P_{\text{max}}(i)\), we have \(\pi^{-1}(P_{\text{max}}(i)) \leq \pi^{-1}(i)\). Since \(L(\pi^{-1}(i)) = \max\{\pi(1), \pi(2), \ldots, \pi(\pi^{-1}(i))\}\), we obtain \(L(\pi^{-1}(i)) = P_{\text{max}}(i)\). Assume to the contrary that \(L(\pi^{-1}(i)) = j > P_{\text{max}}(i)\) for some \(j\). Then, \(\pi^{-1}(j) < \pi^{-1}(i)\) and \(j\) is adjacent to \(i\). It contradicts the definition of \(P_{\text{max}}(i)\) that \(P_{\text{max}}(i)\) is the vertex of the maximal label adjacent to \(i\). Therefore, \(P_{\text{max}}(i) = L(\pi^{-1}(i))\). \(\square\)

**Lemma 2**: Each \(\alpha_{i+1}, 1 \leq i \leq k - 1\), defined in the algorithm of Madanlal et al. is equal to \(L(U(\alpha_i))\).

**Proof**: In the algorithm of Madanlal et al., \(\alpha_{i+1} = \max_{e \in R_i} P_{\text{max}}(v)\), where \(R_i = \{j \mid j\) is in interval \(\alpha_{i-1}, \alpha_i\) and \(j\) is adjacent to \(\alpha_i\}\). Since \(P_{\text{max}}(v) = L(\pi^{-1}(v))\), if \(\pi^{-1}(u) > \pi^{-1}(v)\) for any two vertices \(u, v\) and \(P_{\text{max}}(u) \geq P_{\text{max}}(v)\). Thus, \(\alpha_{i+1} = P_{\text{max}}(u)\) for some \(u \in R_i\) with \(\pi^{-1}(u) = \max_{j \in R_i} \pi^{-1}(j)\). Since any vertex in \(R_i\) is of smaller label than \(\alpha_i\) and is adjacent to \(\alpha_i\), we obtain \(\pi^{-1}(u) = \max\{\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(\alpha_i)\} = U(\alpha_i)\) and \(\alpha_{i+1} = L(U(\pi^{-1}(u))) = L(U(\alpha_i))\). \(\square\)

**Lemma 3**: Each \(r_{i+1}, 1 \leq i \leq k - 1\), defined in the algorithm of Madanlal et al. can be substituted by \(\pi(U(\alpha_i))\).

**Proof**: In their algorithm, they choose \(r_{i+1}\) from \(T_i\), where \(T_i = \{j \mid j \in R_i\) and \(j\) is adjacent to \(\alpha_{i+1}\}\), such that if \(v\) is not adjacent to \(r_{i+1}\) for some \(v\) in interval \(\alpha_i, \alpha_{i+1}\), then \(q\) is not adjacent to \(v\) for any \(q \in T_i\). In fact, the simplest way to choose \(r_{i+1}\) is to choose the vertex \(u\) from \(T_i\) with \(\pi^{-1}(u) = \max\{\pi^{-1}(q) \mid q \in T_i\}\). The reason is that if \(u\) is not adjacent to some vertex \(v\) in interval \(\alpha_i, \alpha_{i+1}\), then for any vertex \(q \in T_i\), \(q \neq u\), we have \(q < v\) and \(\pi^{-1}(q) < \pi^{-1}(u) < \pi^{-1}(v)\) and then \(q\) not adjacent to \(v\). Since any vertex of \(T_i\) is of smaller label than \(\alpha_i\) and is adjacent to \(\alpha_i\), \(\pi^{-1}(u) = \max\{\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(\alpha_i)\} = U(\alpha_i)\). Therefore, \(u = \pi(U(\alpha_i))\). In the algorithm of Madanlal et al., there may be more than one vertex which can be \(r_{i+1}\). However, \(u = \pi(U(\alpha_i))\) must be able to be \(r_{i+1}\). We claim that \(r_{i+1}\) defined in the algorithm of Madanlal et al. can be substituted by \(\pi(U(\alpha_i))\). \(\square\)

**Theorem 4**: Algorithm A finds a tree 3-spanner of a permutation graph in \(O(n)\) time.

**Proof**: By the above lemmas, we know that \(r_{i+1}\) and \(\alpha_{i+1}\) defined in the algorithm of Madanlal et al. can be substituted by our \(r_{i+1}\) and \(\alpha_{i+1}\) in each stage \(i\) of Algorithm A, \(1 \leq i \leq k - 1\). For the vertices \(v\) in interval \((\alpha_{i-1}, \alpha_i)\), our algorithm uses the same manner of Madanlal et al. to add edge \((v, r_i)\) or \((v, \alpha_i)\) to \(T\). Therefore, the resulting tree found by Algorithm A is a tree 3-spanner of a permutation graph. To compute \(L(i)\) and \(U(i)\), \(i = 1, 2, \ldots, n\), we can scan the \(\pi\) and \(\pi^{-1}\) sequences from left to right and keep each latest maximal label during the scanning. Thus, \(L(i)\) and \(U(i)\) can be computed in \(O(n)\) time. Step 2 can be done in \(O(1)\) time, and Step 3 takes \(O(n)\) time totally after \(k\) stages. The complexity of Algorithm A is therefore \(O(n)\). \(\square\)

4. **A Parallel Algorithm for Finding a Tree 3-Spanner**

In the algorithm of Madanlal et al., it is not easy to parallelize the computation of \(r_{i+1}\) and \(\alpha_{i+1}\), especially on computing \(r_{i+1}\) which needs to test vertices in \(T_i\) and vertices in interval \((\alpha_i, \alpha_{i+1})\). Based on Algorithm A, we can easily derive a parallel algorithm for finding a tree 3-spanner of a permutation graph in \(O(\log n)\) time with \(O(\frac{n}{\log n})\) processors on the EREW PRAM computational model. Instead of executing stages, we find all \(r_i\) and \(\alpha_i\) at first, then add edges to construct a tree 3-spanner.

Let \(s(i) = \pi(U(i))\) and \(t(i) = L(U(i))\) for all \(i, i = 1, 2, \ldots, n\). We use \(t(i)\) as the parent of vertex \(i\), \(i = 1, 2, \ldots, n - 1\), to construct a tree \(T_n\). Since the input permutation graph is connected, \(T_n\) is rooted at vertex \(n\). For instance, all \(t(i)\) of Fig. 1, \(i = 1, 2, \ldots, n - 1\), are 6, 6, 6, 8, 12, 12, 12, 12, 12, 12, 12, respectively. We can use edges \((i, t(i))\) to represent a rooted tree as shown in Fig. 3. In \(T_n\), there is a path from vertex 1 to the root. We call such a path the \((1, n)\)-path, denoted by \(1-v_1-v_2-\ldots-n\). In Fig. 3, the \((1, n)\)-path is 1-6-12.

Suppose the input permutation graph \(G\) needs \(k\) stages in executing Algorithm A. Then, we have the following lemmas.

**Lemma 5**: In the \((1, n)\)-path \(1-v_1-v_2-\ldots-n\), vertices \(v_1, v_2, \ldots, n\) are equal to \(\alpha_1, \alpha_2, \ldots, \alpha_k\) of Algorithm A, respectively.

**Proof**: Since \(v_1\) is the parent of vertex 1, \(v_1 = t(1) = L(U(1)) = \alpha_1\). Similarly, \(v_2\) is the parent of \(v_1\), and \(v_2 = t(v_1) = L(U(v_1)) = L(U(\alpha_1)) = \alpha_2\). With the argument, we can conclude that \(v_1 = \alpha_1, v_2 = \alpha_2, \ldots, n = \alpha_k\). \(\square\)

**Corollary 6**: The length of the \((1, n)\)-path is equal to

![Fig. 3](image-url) The rooted tree constructed by edges \((i, t(i))\), \(i = 1, 2, \ldots, n\).
the number of stages in executing Algorithm A.

Lemma 7: In the (1, n)-path 1-v1-v2-...-vk-1-n, vertices \( s(1), s(v_1), s(v_2), \ldots, s(v_{k-1}) \) are equal to \( r_1, r_2, \ldots, r_k \) of Algorithm A, respectively.

Proof: By Lemma 5, we have \( v_1 = \alpha_1, v_2 = \alpha_2, \ldots, v_{k-1} = \alpha_{k-1} \). For vertex 1, \( s(1) = \pi(U(1)) = 1 \). For vertex \( v_1, s(v_1) = \pi(U(v_1)) = \pi(U(\alpha_1)) = r_2 \).

Therefore, we can deduce that \( s(1) = r_1, s(v_1) = r_2, \ldots, s(v_{k-1}) = r_k \).

When all \( r_i \) and \( \alpha_i, i = 1, 2, \ldots, k \), are obtained, we shall construct a tree 3-spanner by adding edges of \( G \). In Algorithm A, each vertex \( u \) in interval \( (\alpha_{i-1}, \alpha_i) \) in stage \( i \) is adjacent to either \( r_i \) or \( \alpha_i \). For parallel processing, we will find the adjacency pair \( [u, \alpha_u] \) of each vertex \( u, u = 1, 2, \ldots, n \), such that \( u \) is adjacent to either \( s(u) \) or \( b_u \). The following is our parallel algorithm.

Algorithm B

Input: A connected permutation graph with the \( \pi \) sequence.

Output: A tree 3-spanner \( T \).

Step 1. Let \( L(i) = \max\{\pi(1), \pi(2), \ldots, \pi(i)\} \) and \( U(i) = \max\{\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(i)\} \) for \( i = 1, 2, \ldots, n \).

Step 2. Let \( s(i) = \pi(U(i)) \) and \( t(i) = L(U(i)) \) for \( i = 1, 2, \ldots, n \).

Step 3. For \( i = 1, 2, \ldots, n-1 \), use \( t(i) \) as the parent of vertex \( i \) to construct a tree \( T \) rooted at vertex \( u \).

Step 4. Find the \( (1, n)-path \) 1-v1-v2-...-n on \( T \). Suppose the length of the \( (1, n) \)-path is \( k \).

Step 5. Transfer the interval \( [1, v_1], [v_1, v_2], \ldots, [k, v_k] \) to the tree \( T \).

Step 6. For each vertex \( u \leq v_i \), let the adjacency pair \( [u, \alpha_u] \) to \( v_i \). For each vertex \( u > v_i \), let the adjacency pair \( [u, \alpha_u] \) of \( u \) be some \( P_i = [v_{i-1}, v_i] \), \( 2 \leq i \leq k \), such that \( v_{i-1} < u \leq v_i \).

Step 7. For \( u = 1, 2, \ldots, n \), if \( u \) is adjacent to \( s(u) \), then add edge \((u, s(u)) \) to \( T \); otherwise, add edge \((u, b_u) \) to \( T \).

In the example of Fig. 1, the \( (1, n) \)-path is 1-6-12, and the intervals are \([1, 6]\) and \([6, 12]\). For vertices 1, 2, 3, 4, 5, and 6, their adjacency pairs are all \([1, 6]\), and these vertices are adjacent to vertex \( s(1) = \pi(U(1)) = 1 \) or vertex 6. For other vertices, their adjacency pairs are \([6, 12]\), and these vertices are adjacent to vertex 5 or 12, where \( s(6) = \pi(U(6)) = 5 \). The resulting tree is the same as the tree shown in Fig. 2.

Now, we consider the complexity of Algorithm B. It is trivial that Steps 3, 5, and 7 can be done in \( O(\log n) \) time with \( O(\frac{n}{\log n}) \) processors on the EREW PRAM model. By the prefix maxima computation [6], Step 1 can be done in \( O(\log n) \) time with \( O(\frac{n}{\log n}) \) processors on the EREW PRAM model.

Step 2 needs additional computation to avoid concurrent read and write. Let \( g_i = \pi(U(i)) \). For \( i = 2, 3, \ldots, n \), let \( g_i = 0 \) if \( U(i) = U(i-1) \) and let \( g_i = \pi(U(i)) \) otherwise. Then, use the prefix maxima computation to obtain all \( s(i) = \max\{g_1, g_2, \ldots, g_i\} \), \( i = 1, 2, \ldots, n \). Similarly, we can let \( h_k = L(U(1)) \) and obtain all \( t(i) = \max\{h_1, h_2, \ldots, h_i\} \) by the same technique, where \( h_k = 0 \) if \( U(i) = U(i-1) \) and \( h_i = L(U(i)) \) otherwise, \( i = 2, 3, \ldots, n \).

In Step 4, since the successor (i.e., the parent) of each vertex is known, we can apply the list ranking technique [6] to find the \( (1, n) \)-path in \( O(\log n) \) time with \( O(\frac{n}{\log n}) \) processors on the EREW PRAM model.

Step 5 also needs additional computation to avoid concurrent read and write. Let \([x_u, y_u]\) be the temporary pair of vertex \( u \). For each vertex \( u \) of \( G \), if \( u \) is a vertex \( v_i \) of the \( (1, n) \)-path 1-v1-v2-...-v_k for some \( i, 1 \leq i \leq k \), then let \([x_u, y_u] = P_i = [v_{i-1}, v_i] \); otherwise, let \([x_u, y_u] = P_k \). We say \([x_i, y_i] = [x_j, y_j] \) if \( x_i < x_j \) and \( y_i < y_j \). By the suffix minima computation [6], we can obtain all adjacency pairs \([u, \alpha_u]\), where \([u, \alpha_u] = \min\{[x_u, y_u], [x_{u+1}, y_{u+1}], \ldots, [x_{n}, y_{n}]\}, u = 1, 2, \ldots, n \). Thus, Step 6 can be done in \( O(\log n) \) time with \( O(\frac{n}{\log n}) \) processors on the EREW PRAM computational model. Table 2 shows how to find \( s(i), t(i) \), and the adjacency pair \([u, \alpha_u]\) for each vertex \( i \) of Fig. 1.

Theorem 8: Algorithm B finds a tree 3-spanner of a permutation graph in \( O(\log n) \) time with \( O(\frac{n}{\log n}) \) processors on the EREW PRAM computational model.

Proof: The complexity of Algorithm B is shown in the above paragraphs. We shall prove that the graph constructed by Algorithm B is the same as the tree constructed by Algorithm A. For each vertex \( u \) of \( G \), the adjacency pair of \( u \) is \([u, \alpha_u] = [v_{i-1}, v_i] = [\alpha_{i-1}, \alpha_i] \), \( 1 \leq i \leq k \) such that \( \alpha_{i-1} < u \leq \alpha_i \). Consider the following two cases.

Case 1. \( \alpha_i = \alpha_i \). Since \( \alpha_i \) is adjacent to \( r_i \), and \( r_i = \pi(U(\alpha_{i-1})) = \pi(U(v_{i-1})) = s(v_{i-1}) = s(u) \), \( u \) is adjacent to \( s(u) \).

Case 2. \( \alpha_{i-1} < \alpha_i \). Since \( r_i = s(u) \) and \( \alpha_i = b_u \), \( u \) is adjacent to \( s(u) \) or \( b_u \).

Therefore, if \( u \) is adjacent to \( s(u) \), then add edge \((u, s(u)) \) to \( T \); otherwise, add edge \((u, b_u) \).

Table 2

<table>
<thead>
<tr>
<th>( \cdot )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_1 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>( \pi(s(i)) )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>( h_i )</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>( \pi(t(i)) )</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>( x_i )</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>( y_i )</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>( \pi(a_i) )</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>( b_i )</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
to $T$. Since all vertices of $G$ are in intervals $[1, v_1], [v_1, v_2], \ldots, [v_{k-1}, n]$ and we add edges for all vertices, the resulting tree $T$ found by Algorithm B is the same as the tree found by Algorithm A. □

5. Concluding Remarks

In this paper, we propose an $O(n)$ algorithm for finding a tree 3-spanner of a permutation graph. Our algorithm can be easily parallelized so that the same problem can be done in $O(\log n)$ time with $O(\frac{n}{\log n})$ processors on the EREW PRAM computational model. Comparing with our algorithm, the algorithm of Madanlal et al. is not efficient and can not be parallelized.

Madanlal et al. have presented an algorithm for finding a tree 3-spanner on interval graphs [7]. Since the classes of interval graphs and permutation graphs are subclasses of trapezoid graphs, there may exist a polynomial time algorithm for finding a tree 3-spanner on trapezoid graphs. It is interesting to study the tree 3-spanner problem on such a type of graphs.

References


Hon-Chan Chen received the BS, MS, and PhD degrees in Information Management from National Taiwan University of Science and Technology, Taipei, Taiwan, in 1992, 1994, and 1998, respectively. After the two year military service, he was an assistant professor in National Kaohsiung Institute of Marine Technology, Kaohsiung, Taiwan, from 2000 to 2002. Since 2002, he has been an assistant professor in the Department of Information Management, National Chin-Yi Institute of Technology, Taichung, Taiwan. His research interests include algorithms, parallel processing, and graph theory.

Shin-Huei Wu received the BS degree in Applied Mathematic from National Chung Hsing University, Taichung, Taiwan, in 1997 and the MS degree in Computer Science from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2002. He currently works in a software company in charge of firmware programming.

Chang-Biau Yang received the BS degree in Electronic Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1982 and the MS degree in Computer Science from National Tsing Hua University, Hsinchu, Taiwan, in 1984. Then, he obtained his PhD degree in Computer Science from National Tsing Hua University, Hsinchu, Taiwan, in 1988. He is currently a professor and the chairman of the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan. His research interests include computer algorithms, interconnection networks, data compression and bioinformatics.