Linear Structure of Bipartite Permutation Graphs and the Longest Path Problem

Ryuhei Uehara *

School of Information Science, JAIST, Ishikawa 923-1292, Japan

Gabriel Valiente [1]

Department of Software, Technical University of Catalonia, E-08034 Barcelona

Abstract

The class of bipartite permutation graphs is the intersection of two well known graph classes: bipartite graphs and permutation graphs. A complete bipartite decomposition of a bipartite permutation graph is proposed in this note. The decomposition gives a linear structure of bipartite permutation graphs, and it can be obtained in $O(n)$ time, where n is the number of vertices. As an application of the decomposition, we show an $O(n)$ time and space algorithm for finding a longest path in a bipartite permutation graph.

Key words: Bipartite permutation graphs; Graph decomposition; Linear time algorithms.

1 Introduction

A large number of graph classes have been recently proposed, and the complexity of hard problems over these graph classes has been thoroughly investigated [2,11], motivated by the fact that some hard problems become efficiently

* Corresponding author. Fax: +81-761-51-1149.
Email addresses: uehara@jaist.ac.jp (Ryuhei Uehara), valiente@lsi.upc.edu (Gabriel Valiente).

[1] Partially supported by Spanish CICYT project GRAMMARS (TIN2004-07925-C03-01) and by the Japan Society for the Promotion of Science through Long-term Invitation Fellowship L05511 for visiting JAIST (Japan Advanced Institute of Science and Technology).

solvable when restricted to particular graph classes. This is a very common approach to solve realistic hard problems; for example, the class of interval graphs was first studied by a molecular biologist, and it turned out that many hard problems can be solved efficiently on these graphs [5, Chapter 8]. On the other hand, from the graph theoretic point of view, these algorithms reveal and make use of graph theoretical properties of the graph classes.

In this note, we focus on the class of bipartite permutation graphs. A graph is a bipartite permutation graph if it is a bipartite graph and a permutation graph. This class was investigated by Spinrad, Brandstädt, and Stewart [12], and some applications were also investigated by Lai and Wei [8].

We first introduce a new notion of complete bipartite decomposition of a bipartite permutation graph. Intuitively, a bipartite permutation graph can be decomposed into an alternating sequence of complete bipartite graphs and independent sets. In the decomposition, two complete bipartite graphs are joined only if they are consecutive. Hence this decomposition allows us to apply dynamic programming techniques to problems on bipartite permutation graphs. The decomposition can be computed in $O(n)$ time for any given bipartite permutation graph, where n is the number of vertices. (We note that a bipartite permutation graph has a simple representation taking only $O(n)$ space, and hence we assume that it is given in such a compact representation.)

In [3], Brandstädt and Lozin gave a similar characterization of bipartite permutation graphs, which from a graph theoretic point of view is a sequence of chain graphs. Each chain graph can be obtained by merging two consecutive graphs (a complete bipartite graph and an independent set) in our decomposition. Hence our result also gives a linear time algorithm for their characterization.

As an application of the decomposition, we give an $O(n)$ time and space algorithm for finding a longest path in a bipartite permutation graph. The longest path problem is one of the most basic problems, and it seems to be harder than the Hamiltonian path problem. The main difficulty of the longest path problem compared to the Hamiltonian path problem is the difference between "all vertices" and "the maximum number of vertices." For the Hamiltonian path problem, we try to join all vertices, and halt if it is impossible. On the other hand, for the longest path problem, we have to choose the best vertex in each case. The computation of the best choice is hard in general; even if we know that a given graph has a Hamiltonian path, it is impossible to find a path of length $n - n^{\epsilon}$ for any $\epsilon > 0$ in polynomial time unless $\mathsf{P} = \mathsf{NP}$ [6]. It is well known that the Hamiltonian path problem is already hard; it is NP-complete for a chordal bipartite graph [10]. Hence there are few polynomial time algorithms for the longest path problem except on trees and some small graph classes [13]. For example, the complexity of the longest path problem for the class of biconvex graphs, which properly contains the class of bipar-

tite permutation graphs, is still open [13]. We show an $O(n)$ time and space algorithm for finding a longest path in a bipartite permutation graph, which improves the previously known $O(n+m)$ time and space algorithm [13].

2 Preliminaries

The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$, and the *degree* of a vertex v is $|N_G(v)|$ and it is denoted by $\deg_G(v)$. For a vertex subset U of V, we denote by $N_G(U)$ the set $\{v \in V \mid v \in N(u) \text{ for some } u \in U\}$. If no confusion can arise we will omit the index G. A vertex set I is called *independent set* if G contains no edges between any pair of vertices in I.

For a graph $G = (V, E)$, a sequence of distinct vertices $v_1, v_2, \ldots, v_\ell$ is a *path*, denoted by $(v_1, v_2, \ldots, v_\ell)$, if $\{v_j, v_{j+1}\} \in E$ for each $0 < j < \ell$. The *length* of a path P is the number of vertices on the path, and it is denoted by $|P|$. In this note, we sometimes deal with a path P as a vertex set; hence we define the length of a path by the number of vertices, and $|P|$ always denotes a number of vertices. The longest path problem is to find a path of maximum length.

A graph $G = (V, E)$ is *bipartite* if V can be partitioned into two sets X and Y such that every edge joins a vertex in X and another vertex in Y. A bipartite graph $G = (X, Y, E)$ is said to be *complete* if each vertex in X is adjacent to all vertices in Y.

A graph $G = (V, E)$ with $|V| = \{v_1, v_2, \ldots, v_n\}$ is said to be a *permutation graph* if there is a permutation σ over $\{1, 2, \ldots, n\}$ such that $\{v_i, v_j\} \in E$ if and only if $(i - j)(\sigma^{-1}(i) - \sigma^{-1}(j)) < 0$. Intuitively, each vertex v in a permutation graph corresponds to a line ℓ_v joining two points on two parallel lines L_1 and L_2, which is called *line representation*. Then, two vertices v and u are adjacent if and only if the corresponding lines ℓ_v and ℓ_u are crossing. Vertex indices give the ordering of the points on L_1, and the permutation of the indices gives the ordering of the points on L_2. When a permutation graph is bipartite, it is said to be a *bipartite permutation graph*.

Hereafter, we sometimes identify the vertex v and the corresponding line ℓ_v and denote it by v in the line representation. In this paper, a given bipartite graph is denoted by $G = (X, Y, E)$, with $n_x = |X|$, $n_y = |Y|$, and $n = n_x + n_y$.

Let $G = (X, Y, E)$ be a bipartite permutation graph. Then, no line x in X intersects any other line x' in X. Hence, we order vertices $x_1, x_2, \ldots, x_{n_x}$ from left to right. We also order vertices $y_1, y_2, \ldots, y_{n_y}$ from left to right. In this paper, we use the standard random-access machine (RAM) model of computation

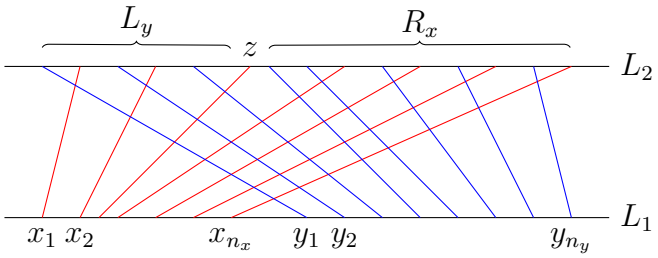

Fig. 1. A chain graph

(see, e.g., [4]); each simple operation takes 1 time step, the time complexity is measured by the number of operations, and the space complexity is measured by the number of required memory cells. Especially, it stores and reads each datum in a memory cell in $O(1)$ time and space, even if it requires $O(\log n)$ bits. Under the RAM model, a bipartite permutation graph $G = (X, Y, E)$ can be represented in $O(n)$ space by storing the vertex orderings on L_1 and L_2. We use two arrays L_1 and L_2, which correspond to the lines L_1 and L_2. For each $i = 1, 2$ and $j = 1, 2, \ldots, n$, $L_i(j)$ stores

(1) a flag that indicates if the endpoint belongs to X or to Y,
(2) the index of the element in X or Y, and
(3) the pointer to the other endpoint on L_{3-i}.

We sometimes say $u < v$ *on* L_1 if the endpoint of u on L_1 is to the left of the endpoint of v. For a point p (not necessarily endpoint) on L_1, we also make an abuse of notation and denote by $u < p$ on L_1 that the endpoint of u on L_1 is to the left of point p.

A bipartite graph $G = (X, Y, E)$ is called *chain graph* if the vertices can be linearly ordered by inclusion; namely, $N(x_1) \subseteq N(x_2) \subseteq \cdots \subseteq N(x_{n_x})$ and $N(y_{n_y}) \subseteq \cdots \subseteq N(y_2) \subseteq N(y_1)$ [7,14]. It is known that any chain graph is a bipartite permutation graph, and its linear orderings over X and Y can be computed in $O(n)$ time [13, Lemma 7] [2].

Lemma 1 *Let $G = (X, Y, E)$ be a connected chain graph with $N(x_1) \subseteq N(x_2) \subseteq \cdots \subseteq N(x_{n_x})$ and $N(y_{n_y}) \subseteq \cdots \subseteq N(y_2) \subseteq N(y_1)$. Then, it has a line representation such that (Fig. 1); (1) $x_1 < x_2 < \cdots < x_{n_x} < y_1 < y_2 < \cdots < y_{n_y}$ on L_1, and (2) $y_1 < x_1$ and $y_{n_y} < x_{n_x}$ on L_2. Conversely, if a graph G has a line representation satisfying conditions (1) and (2), then G is a connected chain graph.*

PROOF. From the linear orderings over X and Y, we have that $N(x_1) = \{y_1, y_2, \ldots, y_{i_1}\}$, $N(x_2) = \{y_1, y_2, \ldots, y_{i_2}\}$, $\ldots$, $N(x_{n_x}) = \{y_1, y_2, \ldots, y_{i_{n_x}}\}$ for

[2] In [13], a bipartite permutation graph is called proper biconvex graph, and a chain graph is called linearly included biconvex graph.

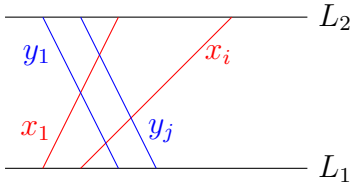

Fig. 2. Induced complete bipartite graph

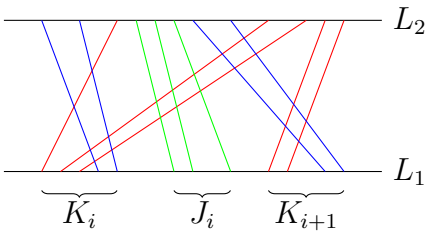

Fig. 3. Isolated vertices J_i between K_i and K_{i+1}

some $i_1 \leqslant i_2 \leqslant \cdots \leqslant i_{n_x}$. A symmetric relationship can be obtained for Y. Then, we can construct the line representation with the condition (1). The connectedness of G implies (2). The converse direction is easy. □

3 Complete Bipartite Decomposition

In this section, we introduce a new decomposition of a connected bipartite permutation graph $G = (X, Y, E)$. We first observe that $\{x_1, y_1\} \in E$, since G is connected. We have the following lemma.

Lemma 2 *For a bipartite permutation graph $G = (X, Y, E)$, $N(x_1) \cup N(y_1)$ induces a complete bipartite graph.*

PROOF. Let y_j be any vertex in $N(x_1)$ and x_i be any vertex in $N(y_1)$. Since x_i and y_j are located on the right side of x_1 and y_1, respectively, the only possible arrangement of these vertices on L_1 is x_1, x_i, y_1, y_j, and the arrangement of the vertices on L_2 is y_1, y_j, x_1, x_i. (See Fig. 2.) Hence, we have $\{x_i, y_j\} \in E$, which completes the proof. □

Let K_1 be the complete bipartite graph induced by $N(x_1) \cup N(y_1)$. Assume that we remove all vertices in K_1 (and incident edges) from G, and the resulting graph is not connected. In this case, if two connected components $G_1 = (X_1, Y_1, E_1)$ and $G_2 = (X_2, Y_2, E_2)$ are such that $X_1 \neq \emptyset$, $Y_1 \neq \emptyset$, $X_2 \neq \emptyset$, and $Y_2 \neq \emptyset$, we have a contradiction; it is easy to see that K_1 cannot connect G_1 and G_2 if G is a bipartite permutation graph. Hence, the resulting graph can be disconnected only if either $X_1 = \emptyset$ or $Y_1 = \emptyset$. (See Fig. 3). In this case, we call the (maximal) set of independent vertices which appears leftmost in the line representation *isolated vertices*.

Here, we define the *complete bipartite decomposition* of a bipartite permutation graph as follows: First, we define K_1 as the complete bipartite graph induced by $N(x_1) \cup N(y_1)$. Then, we remove K_1 from G. We next let J_1 be the set

of isolated vertices, and remove it from G. Finally, we repeat this process until G becomes empty. (Thus, the decomposition ends with either a complete bipartite graph or some isolated vertices.) The sequence of complete bipartite graphs $K_1, K_2, \ldots$ with sets $J_1, J_2, \ldots$ of isolated vertices is called *complete bipartite decomposition* of G. We note that we will not deal with an isolated vertex as a complete bipartite graph. Hence, each complete bipartite graph K_i satisfies $K_i \cap X \neq \emptyset$ and $K_i \cap Y \neq \emptyset$ [3].

In each complete bipartite graph, the first chosen pair of vertices in X and Y is called the pair of *leftmost* vertices. That is, the leftmost vertices in K_i are the vertices with the smallest indices in X and Y, respectively. Similarly, the vertices with the largest indices in K_i are called *rightmost* vertices.

Theorem 3 *Given a bipartite permutation graph $G = (X, Y, E)$, the complete bipartite decomposition of G can be obtained in $O(n)$ time.*

PROOF. We first find K_1 in $O(1)$ time. Let $x_i = \max\{N(y_1)\}$ and $y_j = \max\{N(x_1)\}$. We note that $\max\{N(x_i)\} \geqslant y_j$ and $\max\{N(y_j)\} \geqslant x_i$. Then, there are no isolated vertices between K_1 and K_2 if $\max\{N(x_i)\} > y_j$ and $\max\{N(y_j)\} > x_i$. On the other hand, for example, if $\max\{N(x_i)\} = y_j$ and $\max\{N(y_j)\} > x_i$, we have a sequence of isolated vertices $x_{i+1}, \ldots, x_{k-1}$, where $x_k = \min\{N(y_{j+1})\}$. (We note that even in this case, there are no isolated vertices when $x_k = x_{i+1}$.) When $\max\{N(x_i)\} = y_j$ and $\max\{N(y_j)\} = x_i$, we have $i = n_x$ and $j = n_y$. Hence, we can find all isolated vertices between K_1 and K_2 in $O(1)$ time. Repeating this process, we can obtain the complete bipartite decomposition of G. Each vertex is touched $O(1)$ times and hence, the algorithm runs in $O(n)$ time. □

In a complete bipartite decomposition of G, we say two graphs K and K' (which may be isolated vertex sets) *overlap* if there is an edge in G that joins one vertex in K and another vertex in K'.

Lemma 4 *Let $K_1, K_2, \ldots, K_k$ with $J_1, J_2, \ldots, J_k$ be the complete bipartite decomposition of a connected bipartite permutation graph $G = (X, Y, E)$. Then (1) K_i and K_{i+1} overlap for $1 \leqslant i < k$, and (2) K_i does not overlap with J_{i+1}, and hence K_{i+2}, for $1 \leqslant i < k - 1$.*

PROOF. (1) Suppose K_i and K_{i+1} do not overlap for some i. Since G is connected, there exists some graph that joins K_i and K_{i+1}. First suppose K_j joins them. Then, we have $j < i$. Without loss of generality, we assume that the rightmost vertex y in $K_j \cap Y$ joins K_i and K_{i+1}. In other words, y is

[3] We sometimes deal with a graph as a vertex set if no confusion can arise.

adjacent to the rightmost vertex x in $K_i \cap X$ and the leftmost vertex x' in $K_{i+1} \cap X$. By definition, $K_i \cap Y \neq \emptyset$. Hence, there is a vertex y' in $K_i \cap Y$. Since $j < i$, it must be $y < y'$. However, in this case, vertex y' is also adjacent to x and x', which is a contradiction. Next, suppose J_j joins K_i and K_{i+1}. Without loss of generality, assume $J_j \cap X = \emptyset$. If $j \neq i$, J_j cannot join K_i and K_{i+1}. Hence, $j = i$. Let y be the rightmost vertex in J_j adjacent to $K_i \cap X$. Then, y has to intersect the leftmost vertex in $K_{i+1} \cap X$, since J_j joins K_i and K_{i+1}. However, by

(2) Suppose K_i overlaps with J_{i+1}. Without loss of generality, we assume that the rightmost vertex y in $K_i \cap Y$ is adjacent to the leftmost vertex x in $J_{i+1} \cap X$. Let y' be the leftmost vertex in K_{i+1}. Then, y' is adjacent to x. By definition, this implies x is in $K_{i+1} \cap X$, which is a contradiction. □

Hereafter, we assume that each vertex v knows if it is in either K_i or J_i, and the index i. We also denote by $G_i = (X_i, Y_i, E_i)$ the subgraph of G (vertex) induced by $K_i \cup J_i$.

Theorem 5 *Graph G_i is a chain graph for each i.*

PROOF. If $J_i = \emptyset$, $G_i = K_i$ is a complete bipartite graph, and is thus a chain graph. Without loss of generality, we assume that $i = 1$, $X_1 = \{x_1, x_2, \ldots, x_a\}$, $Y_1 = \{y_1, y_2, \ldots, y_b\}$, and $J_1 = \{y_{b+1}, y_{b+2}, \ldots, y_{b+c}\}$ for some $a, b, c > 0$. Then, it is easy to see that $x_1 < x_2 < \cdots < x_a < y_1 < y_2 < \cdots < y_b < y_{b+1} < y_{b+2} < \cdots < y_{b+c}$ on L_2, and $y_1 < x_1$ and $y_{b+c} < x_a$ on L_1. By Lemma 1, the theorem holds. □

We note that Theorem 5 immediately gives another proof of the characterization by chain graphs by Brandstädt and Lozin [3, Theorem 1]; their independent sets $D_0, D_1, \ldots, D_q$ correspond to $X_1, Y_1, X_2, Y_2, \ldots$ in our notation.

4 Longest Path in a Bipartite Permutation Graph

The main result in this section is the following.

Theorem 6 *A longest path in a connected bipartite permutation graph $G = (X, Y, E)$ can be found in $O(n)$ time.*

We remind that $X = \{x_1, x_2, \ldots, x_{n_x}\}$ and $Y = \{y_1, y_2, \ldots, y_{n_y}\}$ are ordered from left to right. The following lemma allows us to use dynamic programming techniques for finding a longest path in a bipartite permutation graph.

Lemma 7 *[13] There is a longest path P of a bipartite permutation graph G such that the vertices on P are ordered according to the orderings over X and Y. That is, if P contains vertices $x_{i_1}, x_{i_2}, x_{i_3}, \ldots$ in X in this order, we have $i_1 < i_2 < i_3 < \cdots$, and similarly for Y.*

Hereafter, we consider four candidates for a longest path P, which starts from a vertex in X or Y, and ends at a vertex in X or Y. We denote them by P^{XX}, P^{XY}, P^{YX}, and P^{YY}; that is, P^{ST} is a longest path among the set of paths starting from a vertex in S and ending at a vertex in T, and the vertices are ordered according to the ordering over X and Y. Clearly, a longest path is given by the longest one among all these paths. We first consider chain graphs, before dealing with general bipartite permutation graphs.

4.1 Longest Path in a Chain Graph

We consider a linear time algorithm for finding a longest path in a chain graph. Let $G = (X, Y, E)$ be a connected chain graph with the line representation of Lemma 1. We moreover assume that each endpoint on L_2 corresponds to a distinct integer point in $[1..n]$. Let z be any integer point on L_2. Then, for each z, we define a 4-tuple $f(z) = (L_x, L_y, R_x, R_y)$ as follows: L_x is the number of vertices x in X with $x \leqslant z$ on L_2, L_y is the number of vertices y in Y with $y \leqslant z$ on L_2, R_x is the number of vertices x in X with $z < x$ on L_2, and R_y is the number of vertices y in Y with $z < y$ on L_2. (See Fig. 1.) Clearly, we have $L_x + R_x = n_x$ and $L_y + R_y = n_y$. We also have the following remark.

Remark 8 *(1) $f(0) = (0, 0, n_x, n_y)$ and $f(n_x + n_y) = (n_x, n_y, 0, 0)$. (2) Let $f(i) = (a, b, c, d)$. Then, $f(i+1) = (a+1, b, c-1, d)$ if $i+1$ is the endpoint of a vertex x in X, and $f(i+1) = (a, b+1, c, d-1)$ if $i+1$ is the endpoint of a vertex y in Y.*

We here show a linear time algorithm for finding a longest path P^{XY} in G. Let P be a longest path starting from a vertex in X and ending at a vertex in Y. Assume that the second vertex in P is y with $y_1 < y$. Then, since $N(y) \subseteq N(y_1)$, we can replace y by y_1. Repeating this process, the vertices in $P \cap Y$ are $y_1, y_2, \ldots$ in this order. Similarly, the vertices in $P \cap X$ are $\ldots, x_{n_x-1}, x_{n_x}$ in this order. Hence, letting w be the maximum value such that y_w intersects x_{n_x-w}, we have $\left|P^{XY}\right| = 2w$. Now, let $z_{\max}$ be the endpoint of y_w on L_2. Then, we have $f(z_{\max}) = (n_x - w, w, w, n_y - w)$. Moreover, we can see that for any z with $f(z) = (a, b, c, d)$, $w \geqslant \min\{b, c\}$. Hence, by Remark 8, we have the linear time procedure in Algorithm 1 for computing the length of a longest path in a chain graph.

Hence, we immediately have the following result.

Algorithm 1: Longest path in a chain graph

Input : A chain graph $G = (X, Y, E)$
Output: A longest path P^{XY}
initialize $z := 0$, $a := 0$, $b := 0$, $c := n_x$, $d := 1$, $w = 0$
for $z = 1, 2, \ldots, n$ **do**
  if *z is the endpoint of a vertex $x \in X$* **then**
    $a := a + 1$, $c := c - 1$
  else
    $b := b + 1$, $d := d - 1$
  end
  if $w < \min\{b, c\}$ **then** $w := \min\{b, c\}$
end
return $P^{XY} := (x_{n_x-w}, y_1, x_{n_x-w+1}, y_2, \ldots, x_{n_x-1}, y_{w-1}, x_{n_x}, y_w)$

Theorem 9 *A longest path in a chain graph can be found in $O(n)$ time and space.*

Corollary 10 *A longest path in a chain graph that contains $x_1, x_{n_x}, y_1, y_{n_y}$ and with all vertices ordered according to the orderings over X and Y, can be found in $O(n)$ time and space.*

PROOF. Assume that a longest path P (ordered according to the orderings over X and Y) found by Algorithm 1 starts from a vertex $x \in X$ with $x_1 < x$. Then, since $x \in N(y_1) = X$, we can replace x by x_1. Similarly, we can replace y by y_{n_y}, and path P starts from x_1 and ends at y_{n_y}. □

For the path P constructed in Corollary 10, there are indices i and j such that P consists of $\{x_1, x_i, x_{i+1}, \ldots, x_{n_x}\}$ and $\{y_1, y_2, \ldots, y_{j-1}, y_j, y_{n_y}\}$. Intuitively, rightmost vertices in Y (except y_{n_y}) are not used for building the path P. We also pack the vertices in X as much to the left as possible. More precisely, we can achieve that with the procedure in Algorithm 2, which assumes that P is P^{XY}.

Algorithm 2: Pack the vertices in X to the left

Input : A chain graph $G = (X, Y, E)$ and a longest path
  $P = (x_1, y_1, x_i, y_2, x_{i+1}, \ldots, y_j, x_{n_x}, y_{n_y})$ stated in Corollary 10;
Output: The lexicographically first longest path P';
initialize $i := 1$
for $j' = 1, 2, \ldots, j$ **do**
  let i' be the minimum index with $i' > i$ and $x_{i'}$ intersecting $y_{j'}$ and $y_{j'+1}$
  replace $x_{i-j'+2}$ by $x_{i'}$
  set $i := i'$
end
return P

It is easy to see that the path P obtained with Algorithm 2 is the lexicographically first (lex-first, for short) path among the longest paths satisfying Corollary 10. Roughly speaking, unused vertices of the lex-first longest path are collected as much to the right as possible. The lexicographically last (lex-last, for short) longest path is defined in a similar way, and we have the following, immediate result.

Corollary 11 *The lex-first and lex-last longest paths in a chain graph among the longest paths that contain $x_1, x_{n_x}, y_1, y_{n_y}$ and with all vertices ordered according to the orderings over X and Y, can be found in $O(n)$ time and space.*

The following result will be used in the next section.

Lemma 12 *Let P be the lex-first longest path in a chain graph $G = (X, Y, E)$ stated in Corollary 11. Let X' and Y' denote the vertices not used in P; that is, $X' := X \setminus P$ and $Y' := Y \setminus P$. Then, $X' \cup Y'$ is an independent set.*

PROOF. To derive a contradiction, we assume that $x_i \in X'$ intersects $y_j \in Y'$. Then, since P contains $x_1, x_{n_x}, y_1, y_{n_y}$, all vertices in P are ordered from left to right, and since P is connected, it contains four vertices x, x', y, y' such that $x < x_i < x'$, $y < y_j < y'$, and it also contains one of the subpaths (x, y, x', y') and (y, x, y', x'). In the former case, we can extend P by replacing the subpath by (x, y, x_i, y_j, x', y'). In the latter case, we can extend P by replacing the subpath by (y, x, y_j, x_i, y', x'). Both cases contradict the assumption that P is a longest path. □

4.2 Longest Path in a Bipartite Permutation Graph

The outline of a linear time algorithm for finding a longest path in a bipartite permutation graph $G = (X, Y, E)$ is the following.

(1) Compute the complete bipartite decomposition $K_1, J_1, K_2, J_2, \ldots, K_k, J_k$ and let G_i be the subgraph induced by $K_i \cup J_i$, for each i with $1 \leqslant i \leqslant k$.
(2) The algorithm computes four lex-first longest paths P_1^{XX}, P_1^{XY}, P_1^{YX}, and P_1^{YY} in G_1, using Algorithm 1 and Algorithm 2. For $i = 2, \ldots, k$, the algorithm also computes four lex-last longest paths P_i^{XX}, P_i^{XY}, P_i^{YX}, and P_i^{YY} in G_i. One of four candidates will be extended to a longest path in G.
(3) Let P_i^X and P_i^Y denote two longest paths ending at a vertex of $G_1 \cup G_2 \cup \cdots \cup G_i$ in X and Y, respectively. P_1^Y is initialized by the longer path of P_1^{XY} and P_1^{YY}, and P_1^X is the longer one of P_1^{XX} and P_1^{YX}. For $i = 2, 3, \ldots, k$, the algorithm updates two candidate paths P_i^X and

P_i^Y. Finally, the longer one of P_k^X and P_k^Y is a longest path in $G = G_1 \cup G_2 \cup \cdots \cup G_k$.

We note that in the third step, the update is not just connection of the candidates; we have to add extra vertices between them in some cases. The first and second steps can be done in $O(n)$ time and space, by Theorems 3, 5, and 9. Moreover, by Corollary 10, we can assume that each path is ordered, and it starts from the leftmost vertex and ends at the rightmost vertex. Now, we are ready to prove the main theorem in this section by showing the implementation and analysis of the third step.

Proof of Theorem 6 We first compute P_2^X from P_1^{XX}, P_1^{XY}, P_1^{YX}, P_1^{YY}, P_2^{XX}, P_2^{XY}, P_2^{YX}, and P_2^{YY}. (P_2^Y is symmetric and thus omitted here.) To obtain the resulting path P_2^X, we have four possible cases; combining (P_1^X or P_1^Y) and (P_2^{YX} or P_2^{XX}) with unused vertices between them. We here construct a path combining P_1^X and P_2^{YX}, which is one of the four candidates (the other cases are similar and omitted here). We remind that P_1^X is the lex-first, and P_2^{YX} is the lex-last. In other words, the unused vertices in G_1 are collected as much to the right as possible and the unused vertices in G_2 are collected as much to the left as possible. If P_1^X and P_2^{YX} are independent, P_1^X is a candidate of the final longest path of G, and P_2^{YX} is the candidate of P_2^X. Hence, we assume that the last vertex in P_1^X intersects the first vertex in P_2^{YX}.

Let $G_L = (X_L, Y_L, E_L)$ be the bipartite graph induced by $X_L = X_1 \setminus P_1^X$ and $Y_L = Y_1 \setminus P_1^X$, and let $G_R = (X_R, Y_R, E_L)$ be the bipartite graph induced by $X_R = X_2 \setminus P_2^{YX}$ and $Y_R = Y_2 \setminus P_2^{YX}$. That is, G_L and G_R are the subgraphs induced by unused vertices between G_1 and G_2. By Lemma 12, $X_L \cup Y_L$ and $X_R \cup Y_R$ are independent sets and hence, $E_L = E_R = \emptyset$. (We note that P_2^{XY} is not necessarily a longest path in G_2. However, the same argument of the proof of Lemma 12 with the maximality of P_2^{XY} implies that $X_R \cup Y_R$ is an independent set.) If $X_L \cup X_R \cup Y_L \cup Y_R$ is independent, there are no vertices that can extend the candidate of a longest path. Hence, we assume that either $X_L \cup Y_R$ or $X_R \cup Y_L$ is not independent. (In fact, at most one of them can induce a nonempty edge set.) Without loss of generality, we assume that the bipartite graph $G' = (X_L, Y_R, E')$ induced by $X_L \cup Y_R$ satisfies $E' \neq \emptyset$. Clearly, G' is a bipartite permutation graph. Hence, we can order $X_L = \{x_1, x_2, \ldots, x_\ell\}$ and $Y_R = \{y_1, y_2, \ldots, y_r\}$ from left to right (between L_1 and L_2). Here, we remove $x \in X_L$ that has no neighbor in Y_R, and $y \in Y_R$ that has no neighbor in X_L, since they cannot contribute to extend the candidate of a longest path. Thus, we assume G' is connected.

We now remind that G_L is a subgraph of G_1, which is induced by $N(x_1) \cup N(y_1)$ and J_1. That is, all vertices in X_L are in $N(y)$, where y is the rightmost vertex

in Y_L. Since y_1 is the right side of y, we have $x_\ell < y_1$ on L_1. In other words, we have $x_1 < x_2 < \cdots < x_\ell < y_1 < y_2 < \cdots < y_r$ on L_1. Thus, by Lemma 1, G' is a connected chain graph. By Theorem 9, a longest path P' in G' can be found in $O(|X_L| + |Y_R|) = O(|X_1| + |Y_1| + |X_2| + |Y_2|)$ time and space.

Here, since the last vertex in P_1^X intersects the first vertex in P_2^{YX}, we can obtain the candidate of a longest path by connecting P_1^X, P', and P_2^{YX} in this order. Since P_1^X is lex-first and P_2^{YX} is lex-last, the obtained path is the longest possible one. Combining all possible cases, we can obtain two candidates P_2^X and P_2^Y, which are two longest paths in $G_1 \cup G_2$ ending at a vertex in X and Y, respectively.

Using a procedure similar to Algorithm 2, we can compute the lex-first candidates from P_2^X and P_2^Y. In this procedure, it is sufficient to move the rightmost vertices in X_1 and Y_1 and all vertices in X_2 and Y_2, since the other vertices in $X_1 \cup Y_1$ are already placed to the left as much as possible. Hence, its complexity can be bounded by $O(|X_2| + |Y_2|)$.

For the lex-first candidates in $G_1 \cup G_2$, we add the lex-last P_3^{XX}, P_3^{XY}, P_3^{YX} and P_3^{YY}, and take the lex-first candidates in $G_1 \cup G_2 \cup G_3$. By Lemma 4(2) and the previous discussion, this process can be done in $O(|X_2| + |Y_2| + |X_3| + |Y_3|)$ time and space. Repeating this process, we can compute a longest path in $G = G_1 \cup G_2 \cup \cdots \cup G_k$. Correctness of the algorithm follows from a simple induction with Lemma 7. The complexity of the algorithm is clearly $O(|X_1| + |Y_1| + |X_2| + |Y_2| + \cdots + |X_k| + |Y_k|) = O(n_x + n_y) = O(n)$. □

5 Concluding Remarks

Using a similar idea, we can find a maximum independent set (and hence a minimum vertex cover) of a bipartite permutation graph in $O(n)$ time and space. The result can be extended to the class of biconvex graphs. The extended result has an application to phylogenetic networks; it allows one to efficiently solve the constrained site consistency problem, which is referred to in [1, Thm. 10]. This was our original motivation for investigating the class of bipartite permutation graphs, but an $O(n)$ time algorithm for finding a maximum independent set in a biconvex graph has been already given by Lipski and Preparata [9] in a different way and thus, we have omitted it from this note.

References

[1] T. Asano, P. Evans, R. Uehara, and G. Valiente. Site consistency in phylogenetic networks with recombination. In C. S. Iliopoulos, K. Park, and K. Steinhöfel, editors, *Algorithms in Bioinformatics*, chapter 2, pages 15–26. College Publications, 2006.

[2] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.

[3] A. Brandstädt and V. V. Lozin. On the Linear Structure and Clique-Width of Bipartite Permutation Graphs. *Ars Combinatoria*, 67(1):273–281, 2003.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. Cambridge, 2nd edition, 2001.

[5] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*. Elsevier, 2nd edition, 2004.

[6] D. R. Karger, R. Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[7] T. Kloks, D. Kratsch, and H. Müller. Bandwidth of chain graphs. *Information Processing Letters*, 68(6):313–315, 1998.

[8] T.-H. Lai and S.-S. Wei. Bipartite permutation graphs with application to the minimum buffer size problem. *Discrete Mathematics*, 74(1):33–55, 1997.

[9] W. Lipski and F. P. Preparata. Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica*, 15(4):329–346, 1981.

[10] H. Müller. Hamiltonian Circuits in Chordal Bipartite Graphs. *Discrete Mathematics*, 156(1):291–298, 1996.

[11] J. P. Spinrad. *Efficient Graph Representations*. American Mathematical Society, 2003.

[12] J. P. Spinrad, A. Brandstädt, and L. Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987.

[13] R. Uehara and Y. Uno. Efficient algorithms for the longest path problem. In *Proc. 15th Annual International Symposium on Algorithms and Computation*, volume 3341 of *Lecture Notes in Computer Science*, pages 871–883. Springer-Verlag, 2004.

[14] M. Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, 10(2):310–327, 1981.