An O(n)-time Algorithm for the Paired-Domination Problem on Permutation Graphs

Evaggelos Lappas Stavros D. Nikolopoulos Leonidas Palios
Department of Computer Science, University of Ioannina
P.O.Box 1186, GR-45110 Ioannina, Greece
{elappas, stavros, palios}@cs.uoi.gr

Abstract

A vertex subset D of a graph G is a dominating set if every vertex of G is either in D or is adjacent to a vertex in D. The paired-domination problem on G asks for a minimum-cardinality dominating set S of G such that the subgraph induced by S contains a perfect matching; motivation for this problem comes from the interest in finding a small number of locations to place pairs of mutually visible guards so that the entire set of guards monitors a given area. The paired-domination problem on general graphs is known to be NP-complete.

In this paper, we consider the paired-domination problem on permutation graphs. We define an embedding of permutation graphs in the plane which enables us to obtain an equivalent version of the problem involving points in the plane, and we describe a sweeping algorithm for this problem; given the permutation over the set $N_n = \{1, 2, \ldots, n\}$ defining a permutation graph on n vertices, our algorithm computes a paired-dominating set of the graph in $O(n)$ time, and is therefore optimal.

Keywords: permutation graphs, paired-domination, domination, algorithms, complexity.

1 Introduction

A subset D of vertices of a graph G is a *dominating set* if every vertex of G either belongs to D or is adjacent to a vertex in D; the minimum cardinality of a dominating set of G is called the *domination number* of G and is denoted by $\gamma(G)$. The problem of computing the domination number of a graph has received and keeps receiving considerable attention by many researchers (see [10] for a long bibliography on domination). The problem finds many applications, most notably in relation to area monitoring problems by the minimum number of guards: the potential guard locations are vertices of a graph in which two locations are adjacent if a guard in one of them monitors the other; then, the minimum dominating set of the graph determines the locations to place the guards.

The domination problem admits many variants; the most basic ones include: domination, edge domination, weighted domination, independent domination, connected domination, total/open domination, locating domination, and paired-domination [10, 11, 12, 13, 17, 29]. Among these, we will focus on paired-domination: a vertex subset S of a graph G is a *paired-dominating set* if it is a dominating set and the subgraph induced by the set S has a perfect matching; the minimum cardinality of a paired-dominating set in G is called the *paired-domination number* and is denoted by $\gamma_p(G)$. Paired-domination was introduced by Haynes and Slater [12]; their motivation came from the variant of the area monitoring problem in which each guard has another guard as a backup (i.e., we have pairs of guards protecting each other). Haynes and Slater noted that every graph with no isolated vertices has a paired-dominating set (on the other hand, it easily follows from the definition that a graph with isolated vertices does not have a paired-dominating set). Additionally, they showed that the

paired-domination problem is NP-complete on arbitrary graphs; thus, it is of theoretical and practical importance to find classes of graphs for which this problem can be solved in polynomial time and to describe efficient algorithms for its solution.

Trees have been one of the first targets of researchers working on paired-domination: Qiao *et al.* [21] presented a linear-time algorithm for computing the paired-domination number of a tree and characterized the trees with equal domination and paired-domination number; Henning and Plummer [15] characterized the set of vertices of a tree that are contained in all, or in no minimum paired-dominating sets of the tree. Kang *et al.* [16] considered "inflated" graphs (for a graph G, its inflated version is obtained from G by replacing each vertex of degree d in G by a clique on d vertices), gave an upper and a lower bound for the paired-domination number of the inflated version of a graph, and described a linear-time algorithm for computing a minimum paired-dominating set of an inflated tree. Bounds for the paired-domination number have been established also for claw-free cubic graphs [8], for cartesian products of graphs [3], and for generalized claw-free graphs [6]. Very recently, Cheng *et al.* [5] gave an $O(n+m)$-time algorithm for computing a paired-dominating set of an interval graph on n vertices and m edges, when an interval model for the graph with endpoints sorted is available; they also extended their result to circular-arc graphs giving an algorithm running in $O(m(m+n))$ time in this case.

We consider the class of permutation graphs, a well-known subclass of perfect graphs. Given a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ over the set $N_n = \{1, 2, \ldots, n\}$, we define the n-vertex graph $G[\pi]$ with vertex set $V(G[\pi]) = N_n$ and edge set $E(G[\pi])$ such that $ij \in E(G[\pi])$ if and only if $(i-j)(\pi_i^{-1} - \pi_j^{-1}) < 0$, for all $i, j \in V(G[\pi])$, where π_i^{-1} is the index of the element i in π. A graph G on n vertices is a *permutation graph* if there exists a permutation π on N_n such that G is isomorphic to $G[\pi]$ (the graph $G[\pi]$ is also known as the inversion graph of G [9]). We, therefore, assume in this paper that a permutation graph $G[\pi]$ is represented by the corresponding permutation π; see [9].

A lot of research work has been devoted to the study of permutation graphs, and several algorithms have been proposed for recognizing permutation graphs and for solving combinatorial and optimization problems on them. Pnueli *et al.* [20] gave an $O(n^3)$-time algorithm for recognizing permutation graphs using the transitive orientable graph test, where n is the number of vertices of the given graph. Later, Spinrad [24] improved their results by designing an $O(n^2)$-time algorithm for the same problem. In the same paper, Spinrad also proposed an algorithm that determines whether two permutation graphs are isomorphic in $O(n^2)$ time. In [25], Spinrad *et al.* proved that a bipartite permutation graph can be recognized in linear time by using some nice algorithmic properties of such a graph; they also studied other combinatorial and optimization problems on permutation graphs. Supowit [27] solved the coloring problem, the maximum clique problem, the clique cover problem, and the maximum independent set problem, all in $O(n \log n)$ time. Nikolopoulos *et al.* [19] studied the behavior of the on-line coloring algorithm First-Fit (FF) on the class of permutation graphs and proved that this class of graphs is not FF-bounded. We also note that many parallel algorithms have also been proposed for the recognition problem and various optimization problems on permutation graphs; see [14, 18, 22].

Several variants of the domination problem have been considered on permutation graphs. Farber and Keil [7] solved the weighted domination problem and the weighted independent domination problem in $O(n^3)$ time, using dynamic programming techniques. Later, Brandstadt and Kratsch [2] published an $O(n^2)$-time algorithm for the weighted independent domination problem. Atallah *et al.* [1] solved the independent domination problem in $O(n \log^2 n)$ time, while Tsai and Hsu [28] solved the domination problem and the weighted domination problem in $O(n \log \log n)$ time and $O(n^2 \log^2 n)$ time, respectively. Rhee *et al.* [23] described an $O(n+m)$-time algorithm for the minimum-weight domination problem, where m is the number of edges of the given graph. Finally, Chao *et al.* [4] gave an $O(n)$-time algorithm for the minimum cardinality domination problem. On bipartite permutation graphs, Srinivasan *et al.* [26] described and $O(mn+n^2)$-time algorithm for the edge domination problem.

In this paper, we study the paired-domination problem on permutation graphs. We define an em-

PSfrag replacements PSfrag replacements

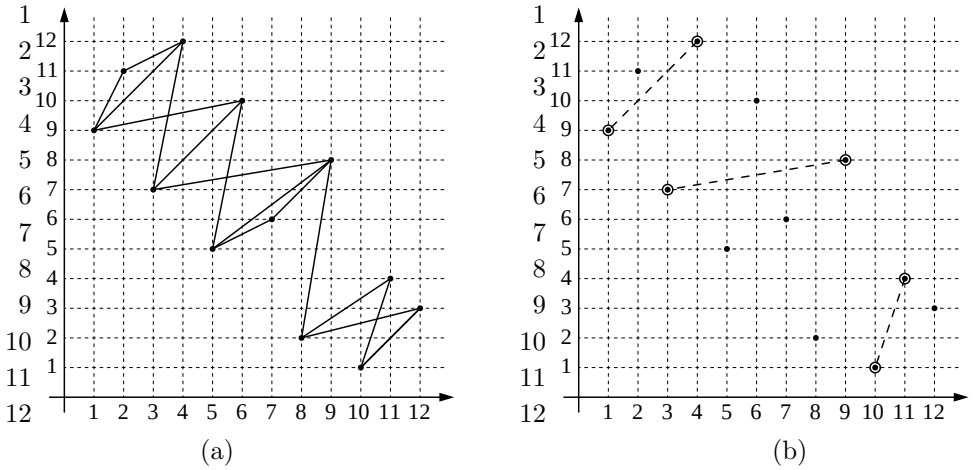


Figure 1: (a) The embedding of the permutation graph corresponding to the permutation $(4, 2, 6, 1, 9, 3, 7, 5, 12, 11, 8, 10)$; (b) A minimum paired-dominating set.

bedding of permutation graphs in the plane and show that every permutation graph G with no isolated vertices admits a minimum-cardinality paired-dominating set of a particular form in the embedding of G. We take advantage of this property to describe an algorithm which "sweeps" the vertices of the embedding from left to right and computes a minimum-cardinality paired-dominating set if such a set exists ("sweeping" is a well-known technique of computational geometry); given the permutation over the set $N_n = \{1, 2, \ldots, n\}$ defining a permutation graph on n vertices, our algorithm runs in $O(n)$ time using $O(n)$ space, and is therefore optimal.

2 Theoretical Framework

We consider finite undirected graphs with no loops or multiple edges; for a graph G, we denote its vertex and edge set by $V(G)$ and $E(G)$, respectively.

Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a permutation over the set $N_n = \{1, 2, \ldots, n\}$. A *subsequence* of π is a sequence $\alpha = (\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k})$ such that $i_1 < i_2 < \cdots < i_k$. If, in addition, $\pi_{i_1} < \pi_{i_2} < \cdots < \pi_{i_k}$, then we say that α is an *increasing subsequence* of π.

A *left-to-right maximum* of π is an element π_i, $1 \leq i \leq n$, such that $\pi_i > \pi_j$ for all $j < i$. The first element in every permutation is a left-to-right maximum. If the largest element is the first, then it is the only left-to-right maximum; otherwise there are at least two (the first and the largest). The increasing subsequence $\alpha = (\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k})$ is called a *left-to-right maxima subsequence* if it consists of all the left-to-right maxima of π; clearly, $\pi_{i_1} = \pi_1$. For example, the left-to-right maxima subsequence of the permutation $(4, 2, 6, 1, 9, 3, 7, 5, 12, 11, 8, 10)$ is $(4, 6, 9, 12)$.

The *right-to-left minima subsequence* of π is defined analogously: $\alpha' = (\pi_{j_1}, \pi_{j_2}, \ldots, \pi_{j_{k'}})$ is called a right-to-left minima subsequence if it is an increasing subsequence and consists of all the right-to-left minima of π, where an element π_i, $1 \leq i \leq n$, is a *right-to-left minimum* if $\pi_i < \pi_j$ for all $j > i$. The last element in every permutation is a right-to-left minimum, and thus $\pi_{j_{k'}} = \pi_n$. For the permutation $(4, 2, 6, 1, 9, 3, 7, 5, 12, 11, 8, 10)$, the right-to-left minima subsequence is $(1, 3, 5, 8, 10)$.

We will also be considering points in the plane. For such a point p, we denote by $x(p)$ and $y(p)$ the x- and y-coordinate of p, respectively.

An embedding of permutation graphs. Given a permutation π over the set $N_n = \{1, 2, \ldots, n\}$, we define and use an embedding of the vertices of the permutation graph $G[\pi]$ in the plane based on the mapping:

$$\text{vertex corresponding to the integer } i \quad \longrightarrow \quad \text{point } p_i = (i,\, n + 1 - \pi_i^{-1}); \tag{1}$$

the x-coordinate is identical to the integer corresponding to the vertex of the graph, while an appropriate distinct y-coordinate is added by the mapping of Eq. (1). Because of this, all the points p_i, $1 \leq i \leq n$, are located in the first quadrant of the cartesian coordinate system and no two such points have the same x- or the same y-coordinate (see Figure 1(a)). Let $P_\pi = \{p_1, p_2, \ldots, p_n\}$. (Similar representations have been used by other authors as well; see [1, 19].) The adjacency condition $ij \in E(G[\pi])$ iff $(i-j)(\pi_i^{-1} - \pi_j^{-1}) < 0$ (for all $i, j \in N_n$) for the permutation graph $G[\pi]$ implies that two points p_i and p_j are adjacent iff $\big(x(p_i) - x(p_j)\big) \cdot \big(y(p_i) - y(p_j)\big) > 0$, i.e., the one of the points is below and to the left of the other. Thus, all the edges have a down-left to up-right direction (Figure 1(a)).

Due to the bijection between the vertices of the permutation graph and the points p_i, with a slight abuse of notation, in the following, we will regard *the points p_i as the vertices of the permutation graph.*

In terms of the above embedding, the notions of vertex domination, left-to-right-maxima, and right-to-left minima become as follows:

- A vertex p_i dominates all vertices $p \in P_\pi$ such that $\big(x(p) - x(p_i)\big) \cdot \big(y(p) - y(p_i)\big) \geq 0$, i.e., p is either below and to the left or above and to the right of p_i (the shaded area in Figure 2 (left)). Then, a pair of adjacent vertices p_i, p_j dominate all the vertices in the portion of the plane

$$\{\, q \in \mathbb{R}^2 \mid \big(x(q) - x(p_i)\big) \cdot \big(y(q) - y(p_i)\big) \geq 0 \text{ or } \big(x(q) - x(p_j)\big) \cdot \big(y(q) - y(p_j)\big) \geq 0 \,\}$$

  (this is the shaded area in Figure 2 (right)); if the edge connecting p_i, p_j is e, we will say that this portion of the plane *is covered* by e, and we will denote it by $C(e)$. The part of the plane not covered by e consists of two disjoint open quadrants, one occupying the upper left corner and the other the bottom right corner; the latter will be important for our algorithm and we will denote it by $Q(e)$.

- A left-to-right maximum of a permutation π defining a permutation graph is mapped to a vertex $p \in P_\pi$ that is a vertex of the upper envelope of the pointset P_π (i.e., there does not exist a point $q \in P_\pi - \{p\}$ for which $x(p) \leq x(q)$ and $y(p) \leq y(q)$[1]. For example, the 4 left-to-right maxima of the permutation defining the graph of Figure 1(a) correspond to the vertices $(4, 12)$, $(6, 10)$, $(9, 8)$, and $(12, 4)$. Similarly, a right-to-left minimum is mapped to a vertex $p \in P_\pi$ that is a vertex of the lower envelope of the pointset P_π (i.e., there does not exist a point $q \in P_\pi - \{p\}$ for which $x(p) \geq x(q)$ and $y(p) \geq y(q)$); the 5 right-to-left minima of the graph of Figure 1(a) correspond to the vertices $(1, 9)$, $(3, 7)$, $(5, 5)$, $(8, 2)$, and $(10, 1)$ of the lower envelope of P_π. For convenience, *each vertex in P_π corresponding to a left-to-right-maximum (right-to-left minimum, resp.) of a permutation π will be called left-to-right-maximum (right-to-left minimum, resp.) as well.*

Finally, the following result helps us focus on solutions to the paired-domination problem on permutation graphs which are of a particular form, thus enabling us to obtain an efficient algorithm.

Lemma 2.1 *Let G be an embedded permutation graph with no isolated vertices whose vertex set is $P_\pi = \{p_1, p_2, \ldots, p_n\}$ (determined by the mapping in Eq. (1)), and let $u_1, u_2, \ldots, u_\ell$ ($v_1, v_2, \ldots, v_{\ell'}$, resp.) be the left-to-right maxima (right-to-left minima, resp.) in P_π in order from left to right. Then, for any set A of edges of G whose endpoints dominate the entire vertex set P_π, there exists a matching M of edges of G such that*

- *the endpoints of the edges in M dominate the entire P_π,*
- *$|M| \leq |A|$, and*
- *$M = \{v_{s_1}u_{t_1}, v_{s_2}u_{t_2}, \ldots, v_{s_{|M|}}u_{t_{|M|}}\}$ where $s_1 < s_2 < \ldots < s_{|M|} \leq \ell'$ and $t_1 < t_2 < \ldots < t_{|M|} \leq \ell$ (i.e., M is a matching which dominates P_π and consists of at most $|A|$ non-crossing edges each of which connects a left-to-right maximum to a right-to-left minimum of P_π).*

[1] When such inequalities hold for the coordinates of two points p and q, it is often said that q *dominates* p; however, we will avoid using this term so that there is no confusion with the notion of *vertex domination* which is central to our work.

PSfrag replacements

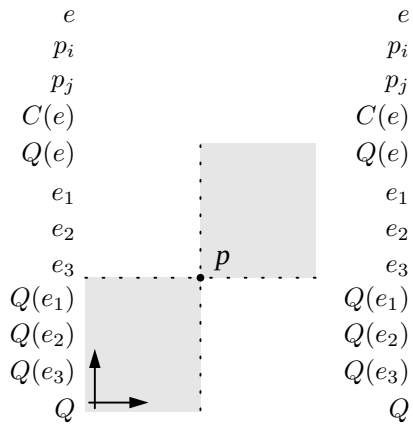
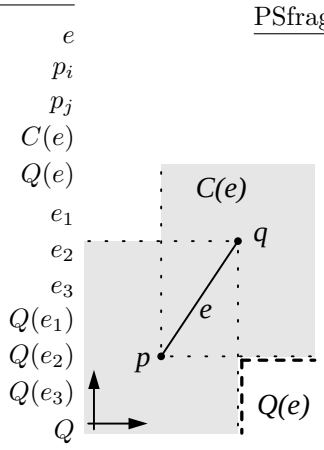


Figure 2:

PSfrag replacements



PSfrag replacements

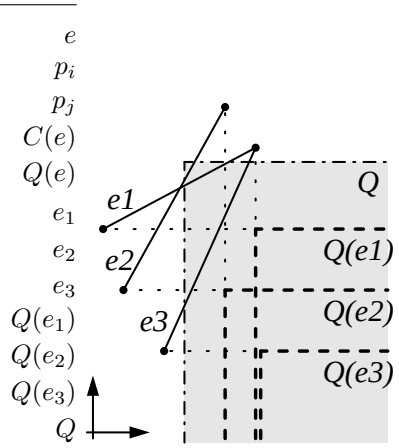


Figure 3:

(*The proof of the lemma can be found in the Appendix.*) Lemma 2.1 readily implies the following corollary.

Corollary 2.1 *Let G be an embedded permutation graph with no isolated vertices whose vertex set is $P_\pi = \{p_1, p_2, \ldots, p_n\}$. Then, G has a paired-dominating set of minimum cardinality whose induced subgraph admits a perfect matching consisting of non-crossing edges of G each of which connects a left-to-right maximum to a right-to-left minimum.*

Such a matching is of the form shown in Figure 1(b). As the edges in such a matching do not cross, they exhibit an ordering from up-left to bottom-right. The following observation pertaining to two non-crossing edges will be very useful:

Observation 2.1 *Let G be an embedded permutation graph with vertex set $P_\pi = \{p_1, p_2, \ldots, p_n\}$, and let e and e' be two edges which are incident on a left-to-right maximum and a right-to-left minimum, and do not cross in the embedding of G (see Figure 1(b)). If e is to the left of e', then for every vertex $p_i \in P_\pi$ for which $p_i \notin C(e) \cup Q(e)$, it holds that $p_i \notin C(e') \cup Q(e')$.*

3 The Algorithm

As mentioned, Corollary 2.1 implies that for every permutation graph with no isolated vertices there exists a minimum-cardinality paired-dominating set whose induced embedded subgraph admits a perfect matching of the form shown in Figure 1(b); for any given permutation graph G, our algorithm precisely computes a minimum matching M of (the embedded) G of this form whose endpoints dominate all the vertices of G. As the edges in such a matching exhibit an ordering from left to right, our algorithm works by identifying candidates for each edge in M in order from left to right.

In particular, regarding the leftmost edge in M, Observation 2.1 implies that

- for each candidate e for the leftmost edge, every vertex in P_π not dominated by the endpoints of e has to lie in the bottom-right non-covered quadrant $Q(e)$ of e, i.e,

$$\big(\mathbb{R}^2 - C(e)\big) \cap P_\pi \;=\; Q(e) \cap P_\pi. \tag{2}$$

Furthermore, in order to obtain a minimum-size set M, we additionally require that

- the non-covered part $Q(e)$ of the plane be minimized.

In order to formalize the latter condition, we give the following definition of redundant edges.

Definition 3.1 *Let G be an embedded permutation graph, Q an open quadrant (bounded only from above and left) which we wish to cover, and $X = \{\, e \in E(G) \mid \big(Q - C(e)\big) \cap P_\pi = Q(e) \cap P_\pi \,\}$. Then, we say that an edge $d \in X$ is* redundant *if there exists another edge $d' \in X$ such that $Q(d) \subset Q(d')$.*

For example, in Figure 3, the edges e_1 and e_2 are redundant in light of e_3.

We note that we are interested in minimizing the non-covered part of the plane rather than minimizing the number of points that are not dominated. In light of Definition 3.1, the fact that we are interested in edges e that minimize the non-covered part $Q(e)$ of the plane is rephrased into that *we are interested in edges e that are not redundant*. The following lemma enables us to identify redundant edges among edges incident on a left-to-right maximum and a right-to-left minimum (see Figure 3):

Lemma 3.1 *Let G be an embedded permutation graph and let $u_1, u_2, \ldots, u_\ell$ ($v_1, v_2, \ldots, v_{\ell'}$, resp.) be the left-to-right maxima (right-to-left minima, resp.) among the vertices of G in order from left to right. Moreover, let A be a subset of edges of G which cover the plane except for an open quadrant Q (bounded only from above and left), and $X = \{\, e \in E(G) - A \mid \big(Q - C(e)\big) \cap P_\pi = Q(e) \cap P_\pi \,\}$. Then, if X contains an edge $d = v_i u_j$, any edge $v_{i'} u_{j'} \in X - \{d\}$ such that $i' \leq i$ and $j' \leq j$ is redundant.*

Lemma 3.1 implies that for two edges $v_i u_j$ and $v_{i'} u_{j'}$ to be non-redundant, it has to be the case that $(i' - i) \cdot (j' - j) < 0$, that is, the edges form a *crossing pattern* like the one shown in Figure 4.

We give next an outline of our algorithm for computing a minimum matching M such that the edges in M are of the form shown in Figure 1(b) and their endpoints dominate all the vertices of the given permutation graph G. The algorithm identifies the non-redundant candidates for the leftmost edge of M and constructs a set $E_1 = \{e_{1,1}, e_{1,2}, \ldots, e_{1,h_1}\}$ of all these candidates. In the general step, we have a set $E_i = \{e_{i,1}, e_{i,2}, \ldots, e_{i,h_i}\}$ of candidates for the i-th edge of the matching M. Then, the algorithm constructs the set E_{i+1} of candidates for the $(i+1)$-st edge by selecting among the edges in $\{\, e \in \big(E(G) - \bigcup_{r=1}^{i} E_r\big) \mid \exists\, j$ such that $\big(Q(e_{i,j}) - C(e)\big) \cap P_\pi = Q(e) \cap P_\pi \,\}$ those that are non-redundant. The algorithm uses two arrays $A_x[\,]$ and $A_y[\,]$ in which it stores the elements of the set P_π by increasing x-coordinate and by decreasing y-coordinate, respectively, and two arrays $lrmax\_above[\,]$ and $rlmin\_left[\,]$ such that for a vertex $p \in P_\pi$, $lrmax\_above[p]$ ($rlmin\_left[p]$, resp.) stores the lowest left-to-right maximum (rightmost right-to-left minimum, resp.) above (to the left, resp.) of p. Additionally, each collected candidate edge $e \in E_{i+1}$ $(i > 1)$ has a pointer *back* which points to an edge $e' \in E_i$ such that $\big(Q(e') - C(e)\big) \cap P_\pi = Q(e) \cap P_\pi$; these *back*-pointers help us collect the matching M that we seek.

Algorithm PERMUT_PAIRED-DOMINATION
Input : a permutation π over the set $N_n = \{1, 2, \ldots, n\}$ defining a permutation graph G
Output : a solution to the paired-domination problem on G

---

1. Compute the set P_π of points corresponding to the vertices of the graph G based on the mapping in Eq. (1);
   for $i = 1, 2, \ldots, n$ **do**
   $A_x[i] \leftarrow p_i$; $\{A_x$ *stores* P_π *sorted by increasing x-coordinate*$\}$
   $A_y[\pi^{-1}(i)] \leftarrow p_i$; $\{A_y$ *stores* P_π *sorted by decreasing y-coordinate*$\}$
   using the array $A_y[\,]$, compute the left-to-right maxima $(u_1, u_2, \ldots, u_\ell)$ of P_π as well as the contents of the array $lrmax\_above[\,]$; similarly, using the array $A_x[\,]$, compute the right-to-left minima $(v_1, v_2, \ldots, v_{\ell'})$ of P_π and the contents of the array $rlmin\_left[\,]$;
   if there exists a point which is both a left-to-right maximum and a right-to-left minimum
   then print("There exist isolated vertices; the graph has no paired-dominating set");
   exit;

2. Compute the set $E_1 = \{e_{1,1}, e_{1,2}, \ldots, e_{1,h_1}\}$ of candidates for the leftmost edge in a minimum matching with endpoints that dominate all the vertices of the graph G;

3. $i \leftarrow 1$;
   while each of the (non-covered) quadrants $Q(e_{i,1}), Q(e_{i,2}), \ldots, Q(e_{i,h_i})$ of the edges $e_{i,1}, e_{i,2}, \ldots, e_{i,h_i}$ contains at least one point of P_π **do**

3.1. compute the set $E_{i+1} = \{e_{i+1,1}, e_{i+1,2}, \ldots, e_{i+1,h_{i+1}}\}$ of candidates for the $(i+1)$-st edge in a minimum matching (with endpoints dominating all the vertices of G), where each edge $e_{i+1,j}$ $(1 \leq j \leq h_{i+1})$ points to one edge in E_i (by means of a pointer *back*);
$i \leftarrow i+1$;

4. let e_{i,j_i} be the element of E_i such that the quadrant $Q(e_{i,j_i})$ is empty;
$M \leftarrow \{e_{i,j_i}\}$;
for $t = i, i-1, \ldots, 2$ **do**
$e_{t-1,j_{t-1}} \leftarrow$ the edge in E_{t-1} pointed to by the *back* pointer of e_{t,j_t};
include $e_{t-1,j_{t-1}}$ in the set M;

5. Report the x-coordinates of the endpoints of the edges in M as a solution to the paired-domination problem on the graph G.

The correctness of Algorithm PERMUT_PAIRED-DOMINATION follows from the correctness of Procedures Compute_E$_1$ and Compute_E$_{i+1}$ and is established by induction on the size of any solution to the paired-domination problem on the input permutation graph G.

We give below the description of Procedure Compute_LRMaxima for computing the left-to-right maxima and for updating the contents of array $lrmax\_above[\,]$; the procedure for computing the right-to-left minima and for updating the contents of array $rlmin\_left[\,]$ is similar. In the next paragraphs, we describe how we execute Steps 2 and 3.1.

Procedure Compute_LRMaxima

1. $p \leftarrow A_y[1]$; $\{A_y$ *stores* P_π *sorted by decreasing* y*-coordinate*$\}$
$W \leftarrow$ list containing a single node storing p; $\{W$ *will store the left-to-right maxima*$\}$
$lrmax\_above[p] \leftarrow$ **NIL**; $\{$*indicates that* p *is a left-to-right maximum*$\}$
$x\_max \leftarrow x(p)$;
$lowest\_lrmaximum \leftarrow p$; $\{$*lowest left-to-right maximum seen so far*$\}$
for $i = 2, 3, \ldots, n$ **do**
$p \leftarrow A_y[i]$;
if $x(p) > x\_max$
then $\{$*point* p *is a left-to-right maximum*$\}$
insert point p at the end of the list W;
$lrmax\_above[p] \leftarrow$ **NIL**;
$x\_max \leftarrow x(p)$;
$lowest\_lrmaximum \leftarrow p$; $\{$*update lowest left-to-right maximum seen so far*$\}$
else $lrmax\_above[p] \leftarrow lowest\_lrmaximum$;

3.1 Computing the set E_1

The goal in the construction of the set E_1 is that each edge $e \in E_1$ is incident on a left-to-right maximum and a right-to-left minimum, is not redundant, and satisfies Eq. (2). For the construction of E_1, we take advantage of the following lemma:

Lemma 3.2 *Let G be an embedded permutation graph with no isolated vertices whose vertex set is $P_\pi = \{p_1, p_2, \ldots, p_n\}$, and let $u_1, u_2, \ldots, u_\ell$ ($v_1, v_2, \ldots, v_{\ell'}$, resp.) be the left-to-right maxima (right-to-left minima, resp.) in P_π in order from left to right. If $rlmin\_left[u_1] = v_r$, we have:*

(i) Consider v_i, where $i = 1, 2, \ldots, r$, and let $p(v_i)$ be the highest among the points in P_π with x-coordinate $\leq x(v_i)$, and $u_{q_i} = lrmax\_above[p(v_i)]$. Then, for any edge $e_q = v_i u_q$ with $1 \leq q \leq q_i$, it holds that $\big(\mathbb{R}^2 - C(e_q)\big) \cap P_\pi = Q(e_q) \cap P_\pi$; this equality does not hold for any edge $e_q = v_i u_q$ with $q > q_i$.

(ii) Among the edges referred to in the statement (i) of the lemma, the edges $v_i u_q$ (where $1 \leq q < q_i$) are all redundant in light of the existence of the edge $v_i u_{q_i}$.

(iii) No edge e incident on a right-to-left minimum to the right of v_r satisfies Eq. (2).

In Figure 1(a), $v_1 = (1, 9)$, $v_2 = (3, 7)$, and $v_r = v_2$; so, the edges considered are v_1u_1, v_1u_2, v_2u_1 (where $u_1 = (4, 12)$ and $u_2 = (6, 10)$), among which v_1u_1 is redundant. We give below the outline of this procedure: in Step 1, we use Lemma 3.2 to construct a list L of edges satisfying Eq. (2) where L contains exactly one edge incident on each right-to-left minimum to the left of u_1; in Step 2, we obtain the desired set E_1 by removing all the redundant edges from L. For the correctness of Step 2, it is important to note that because the y-coordinate of point *highest_p* never decreases during the execution of Step 1, the edges $v_{s_i}u_{t_i}$ and $v_{s_j}u_{t_j}$ located in the i-th and j-th node of the list L (for any $i < j$) have $s_i < s_j$ and $t_i \geq t_j$.

Procedure Compute_E$_1$

1. $p \leftarrow A_x[1]$; {*the leftmost point*}
   $L \leftarrow$ a list containing a single node storing the edge connecting p to $lrmax\_above[p]$;
   $highest\_p \leftarrow p$; {*the highest point seen so far*}
   $i \leftarrow 2$;
   while $A_x[i]$ does not coincide with the leftmost left-to-right maximum u_1 **do**
   $p \leftarrow A_x[i]$;
   if p is a right-to-left minimum
   then insert at the end of L the edge connecting p to $lrmax\_above[highest\_p]$;
   if $y(p) > y(highest\_p)$
   then $highest\_p \leftarrow p$; {*update highest point seen so far*}
   $i \leftarrow i + 1$;

2. $E_1 \leftarrow \emptyset$;
   let the list L contain the edges $e_1, e_2, \ldots, e_{|L|}$ in order and suppose that $e_i = v_{s_i}u_{t_i}$, where v_{s_i} is a right-to-left minimum and u_{t_i} is a left-to-right maximum;
   $i \leftarrow 1$; {i *indicates the position in L of edge checked for inclusion in* E_1}
   while $i < |L|$ **do**
   $j \leftarrow i + 1$;
   while $j <= |L|$ **and** $u_{t_j} = u_{t_i}$ **do**
   $j \leftarrow j + 1$; {*ignore all edges incident on the same left-to-right maximum except...*}
   add the edge e_{j-1} in E_1 with its *back*-pointer pointing to **NIL**; {*...for the last one*}
   $i \leftarrow j$;
   if $i = |L|$ {$i = |L| \iff e_{|L|-1}$ *is the last edge included in* E_1 *and* $u_{t_{|L|-1}} \neq u_{t_{|L|}}$}
   then add the edge $e_{|L|}$ in E_1 with its *back*-pointer pointing to **NIL**;

The correctness of Step 1 follows from Lemma 3.2; in accordance with statement (ii), for each v_i, we consider only the edge $v_iu_{q_i}$ where $u_{q_i} = lrmax\_above[p(v_i)]$. The correctness of Step 2 follows from Lemma 3.1; the edges in the resulting set E_1 form a crossing pattern like the one shown in Figure 4.

3.2 Computing the set E_{i+1} from E_i

Let $E_i = \{e_{i,1}, e_{i,2}, \ldots, e_{i,h}\}$ be the set of candidate edges for the i-th edge in a minimum matching M such that the edges in M are of the form shown in Figure 1(b) and their endpoints dominate all the vertices of the given permutation graph G. For the construction of E_{i+1}, we are interested in non-redundant edges e incident on a left-to-right maximum and on a right-to-left minimum such that there exists $e_{i,j} \in E_i$ for which

$$\big(Q(e_{i,j}) - C(e)\big) \cap P_\pi \;=\; Q(e) \cap P_\pi. \tag{3}$$

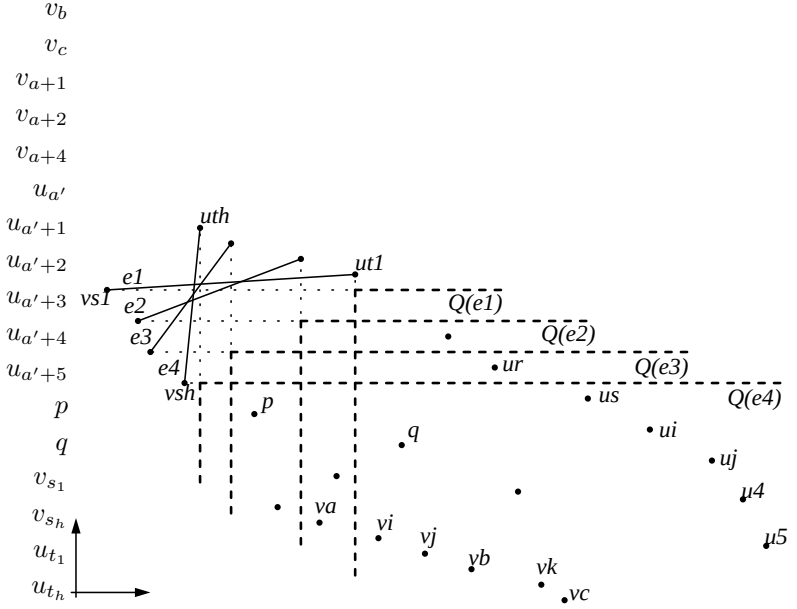


Figure 4: $E_{i+1} = \{v_{a+1}u_{a'+4}, v_{a+2}u_{a'+2}, v_c u_{a'+1}\}$; redundant candidates: $v_a u_{a'+4}, v_b u_{a'+1}, v_{a+4}u_{a'+1}$.

(Eq. (3) follows from Observation 2.1 as no edge to the right of e can dominate a vertex not in $C(e) \cup Q(e)$.) In order to find such edges, we take advantage of the following lemma:

Lemma 3.3 *Let G be an embedded permutation graph with no isolated vertices whose vertex set is $P_\pi = \{p_1, p_2, \ldots, p_n\}$, and let $u_1, u_2, \ldots, u_\ell$ ($v_1, v_2, \ldots, v_{\ell'}$, resp.) be the left-to-right maxima (right-to-left minima, resp.) in P_π in order from left to right. Suppose further that the set E_i contains the edges $e_{i,1}, e_{i,2}, \ldots, e_{i,h}$ where $e_{i,j} = v_{s_j}u_{t_j}$. If $rlmin\_left[u_{t_1}] = v_a$, $lrmax\_above[v_{s_h}] = u_{a'}$, $rlmin\_left[u_{a'}] = v_b$, and $rlmin\_left[u_{a'+1}] = v_c$ (see Figure 4), we have:*

(i) The edge connecting v_a to $lrmax\_above[v_a]$ satisfies Eq. (3) for $j = 1$.

(ii) Consider v_k, where $k = a+1, a+2, \ldots, b$. Let $Q(e_{i,r})$ be a quadrant that contains a point $p \in P_\pi$ such that $x(p) < x(v_k)$ and $y(p) > y(v_{s_h})$. If Eq. (3) is satisfied for $Q(e_{i,j}) = Q(e_{i,r})$ and an edge e incident on v_k, then Eq. (3) is also satisfied for $Q(e_{i,j}) = Q(e_{i,h})$ and that edge e.

(iii) Consider v_k, where $k = a+1, a+2, \ldots, b$. Suppose that there exist quadrants $Q(\ )$ of edges in E_i that do not contain points $p \in P_\pi$ such that $x(p) < x(v_k)$ and $y(p) > y(v_{s_h})$; let $Q(e_{i,r})$ be the rightmost among these quadrants (i.e., its left side is to the right of the left sides of the other quadrants), and let $u_{q_k} = lrmax\_above[p(v_k)]$ where $p(v_k)$ is the highest point in P_π which belongs to $Q(e_{i,r})$ and is not to the right of v_k. Then, Eq. (3) is satisfied for $Q(e_{i,j}) = Q(e_{i,r})$ and the edge $e = v_k u_{q_k}$; this does not hold for any edge $e = v_k u_q$ with $q > q_k$.

(iv) Consider v_k, where $k = b+1, b+2, \ldots, c$. Suppose that there exists a quadrant $Q(e_{i,r})$ that contains no points $p \in P_\pi$ such that $y(p) > y(u_{a'+1})$, and let $u_{q_k} = lrmax\_above[p(v_k)]$ where $p(v_k)$ is the highest point in P_π which belongs to $Q(e_{i,r})$ and is not to the right of v_k. Then, Eq. (3) is satisfied for $Q(e_{i,j}) = Q(e_{i,r})$ and the edge $e = v_k u_{q_k}$; this does not hold for any edge $e_q = v_k u_q$ with $q > q_k$.

(v) Each edge incident on a right-to-left minimum to the left of v_a is redundant. Moreover, if every quadrant $Q(\ )$ contains points $p \in P_\pi$ such that $y(p) > y(u_{a'+1})$, then for any edge e incident on a right-to-left minimum to the right of v_b, there does not exist $e_{i,r} \in E_i$ such that e satisfies Eq. (3) for $Q(e_{i,j}) = Q(e_{i,r})$; if there exists a quadrant $Q(\ )$ containing no points $p \in P_\pi$ such that $y(p) > y(u_{a'+1})$, then for any edge e incident on a right-to-left minimum to the right of v_c, there does not exist $e_{i,r} \in E_i$ such that e satisfies Eq. (3) for $Q(e_{i,j}) = Q(e_{i,r})$.

Statement (ii) follows from the fact that if Eq. (3) is satisfied for $Q(e_{i,j}) = Q(e_{i,r})$ and an edge $e = v_k u_{k'}$, then $y(u_{k'}) > y(v_{s_h})$. As an example for statement (iii), consider $v_k = v_{a+2}$ in Figure 4: then all 4 quadrants $Q(e_{i,1}), \ldots, Q(e_{i,4})$ contain no points $p \in P_\pi$ such that $x(p) < x(v_k)$ and $y(p) > y(v_{s_h})$; the

rightmost quadrant $Q(e_{i,r})$ is $Q(e_{i,1})$, $p(v_k) = q$, and $u_{q_k} = u_{a'+2}$. As an example for statement (iv), we may consider $v_k = v_{a+4}$ or v_c in Figure 4: in either case, $Q(e_{i,r}) = Q(e_{i,4})$, $p(v_k) = p$, and $u_{q_k} = u_{a'+1}$.

Our procedure for computing E_{i+1} takes advantage of Lemma 3.3. In Step 1, it constructs a list L containing at most one edge incident on each of the right-to-left minima from v_a (inclusive) to v_b (inclusive) and potentially to v_c (inclusive) depending on whether the conditions of statement (iv) of the lemma hold. The procedure processes the points in P_π from left to right, and maintains among the quadrants whose left side is not to the right of the point currently being processed (which we will call *active* quadrants) only those that do not contain any point above the line $y = y(v_{s_h})$; in light of statement (ii), the quadrants containing points above the line $y = y(v_{s_h})$ do not provide solutions in addition to those from $Q(e_{i,h})$. Note that if the currently processed point belongs to a quadrant $Q(\ )$, it also belongs to the active quadrants whose left side is to the right of the left side of $Q(\ )$. Thus, the procedure stores the active quadrants in a stack S in order from left to right (the rightmost is at the top of the stack); for each such quadrant $Q(e_{i,j})$, the stack stores $e_{i,j}$ (field *edge*), the y-coordinate of the line bounding $Q(e_{i,j})$ from above (field *top_edge_y*), and the highest point in $Q(e_{i,j}) - R$ (field *highest_p*) where R either is the quadrant stored in the stack record immediately above the record storing $Q(e_{i,j})$ or is the halfplane to the right of the point that is currently being processed if $Q(e_{i,j})$ is at the top of S. The initialization of the list L implements statement (i) of Lemma 3.3, Step 1.2 implements statement (ii) (quadrants containing points above the line $y = y(v_{s_h})$ are popped from the stack), Step 1.4 implements statement (iii), and Step 1.5 implements statement (iv). Step 2 removes any redundant edges. (By $top(S)$ we denote the record at the top of the stack S and by $top(S).edge$, $top(S).highest\_p$, and $top(S).top\_edge\_y$ the values of its fields *edge*, *highest_p*, and *top_edge_y*.)

Procedure Compute_E$_{i+1}$

1. let the set E_i contain the edges $e_{i,1}, e_{i,2}, \ldots, e_{i,h}$ where $e_{i,j} = v_{s_j}u_{t_j}$ with v_{s_j} (u_{t_j}, resp.) being right-to-left minima (left-to-right maxima, resp.) (note that $\forall\, j < j'$, $s_j < s_{j'}$ and $t_j > t_{j'}$);
   let the right-to-left minimum $rlmin\_left[u_{t_1}]$ be v_a (i.e., $v_a = rlmin\_left[u_{t_1}]$);
   $L \leftarrow$ a list containing a single node storing the edge connecting v_a to $lrmax\_above[v_a]$ with its *back*-pointer pointing to the edge $e_{i,1}$;
   $S \leftarrow$ empty stack;
   for each point $p \in P_\pi$ from u_{t_h} to $lrmax\_above[v_{s_h}]$ in order of increasing x-coordinate **do**
      if $p = u_{t_j}$ for an edge $e_{i,j} = v_{s_j}u_{t_j} \in E_i$
   1.1 **then** create a stack record storing $edge \leftarrow e_{i,j}$, $highest\_p \leftarrow$ **NIL**, and $top\_edge\_y \leftarrow y(v_{s_j})$;
            push the record in the stack S;
      else if $y(p) > y(v_{s_h})$
   1.2 **then** $q \leftarrow top(S).highest\_p$;
            while $y(p) < top(S).top\_edge\_y$ **do**
               if $q =$ **NIL or** $\big(top(S).highest\_p \neq$ **NIL and** $y(top(S).highest\_p) > y(q)\big)$
               then $q \leftarrow top(S).highest\_p$; {*q: highest highest_p of popped quadrants*}
               pop the record at the top of S;
            $top(S).highest\_p \leftarrow q$;
   1.3 **else if** $top(S).highest\_p =$ **NIL or** $y(p) > y(top(S).highest\_p)$
            then $top(S).highest\_p \leftarrow p$; {*update highest point*}
            if p is a right-to-left minimum to the right of v_a
   1.4 **then** add the edge connecting p to $lrmax\_above[top(S).highest\_p]$ at the end of the list L with its *back*-pointer pointing to the edge $top(S).edge \in E_i$;
   if $top(S).highest\_p =$ **NIL or** $y(top(S).highest\_p) < y(u_{a'+1})$ where $u_{a'} = lrmax\_above[v_{s_h}]$
   then for each $p \in P_\pi : x(u_{a'}) < x(p) \leq x(u_{a'+1})$ in order of increasing x-coordinate **do**
   1.5 **if** $top(S).highest\_p =$ **NIL or** $y(p) > y(top(S).highest\_p)$
      then $top(S).highest\_p \leftarrow p$; {*update highest point*}
      if p is a right-to-left minimum

then add the edge connecting p to $lrmax\_above[top(S).highest\_p]$ at the end of the list L with its *back*-pointer pointing to the edge $top(S).edge \in E_i$;

2. perform Step 2 of procedure Compute_E_1 to remove redundant edges from the list L;

Note that after a record has been pushed in the stack S (which happens in the first iteration of the for-loop), the stack is never empty because the record corresponding to the edge $e_{i,h}$ is never popped during Step 1.2. For the case shown in Figure 4, at the end of Step 1, the list L contains the edges $v_a u_{a'+4}$, $v_{a+1} u_{a'+4}$, $v_{a+2} u_{a'+2}$, $v_b u_{a'+1}$, $v_{a+4} u_{a'+1}$, and $v_c u_{a'+1}$.

3.3 Complexity of Algorithm Permut_Paired-Domination

Regarding the time and space complexity of the Algorithm Permut_Paired-Domination (*the analysis is given in the Appendix*), the following theorem holds:

Theorem 3.1 *Let G be a permutation graph with no isolated vertices determined by a permutation π over the set N_n. Then, given π, Algorithm* Permut_Paired-Domination *computes a minimum-cardinality paired-dominating set of G in $O(n)$ time using $O(n)$ space.*

References

[1] M.J. Atallah, G.K. Manacher, and J. Urrutia, Finding a minimum independent dominating set in a permutation graph, Discrete Appl. Math. 21 (1988) 177–183.

[2] A. Brandstadt and D. Kratsch, On domination problems for permutation and other graphs, Theoret. Comput. Sci. 54 (1987) 181–198.

[3] B. Brešar, M.A. Henning, and D.F. Rall, Paired-domination of Cartesian products of graphs and rainbow domination, Electr. Notes in Discrete Math. 22 (2005) 233–237.

[4] H.S. Chao, F.R. Hsu, and R.C.T. Lee, An optimal algorithm for finding the minimum cardinality dominating set on permutation graphs, Discrete Appl. Math. 102 (2000) 159–173.

[5] T.C.E. Cheng, L.Y. Kang, and C.T. Ng, Paired domination on interval and circular-arc graphs, Discrete Appl. Math. 155 (2007) 2077–2086.

[6] P. Dorbec, S. Gravier, and M.A. Henning, Paired-domination in generalized claw-free graphs, J. Comb. Optim. 14 (2007) 1–7.

[7] M. Farber and J.M. Keil, Domination in permutation graphs, J. Algorithms 6 (1985) 309–321.

[8] O. Favaron and M.A. Henning, Paired-domination in claw-free cubic graphs, Graphs and Combinatorics 20 (2004) 447–456.

[9] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, Inc., New York, 1980.

[10] T.W. Haynes, S.T. Hedetniemi, and P.J. Slater, Fundamentals of Domination in Graphs, Marcel Dekker, New York (1998).

[11] T.W. Haynes, S.T. Hedetniemi, and P.J. Slater, Domination in Graphs: Advanced Topics, Marcel Dekker, New York (1998).

[12] T.W. Haynes and P.J. Slater, Paired-domination in graphs, Networks 32 (1998) 199–206.

[13] S.T. Hedetniemi and R. Laskar (eds.), Topics on Domination, Ann. Discrete Math. 48, North-Holland, Amsterdam (1991).

[14] D. Helmbold and E.W. Mayr, Applications of parallel algorithms to families of perfect graphs, Computing 7 (1990) 93–107.

[15] M.A. Henning and M.D. Plummer, Vertices contained in all or in no minimum paired-dominating set of a tree, J. Comb. Optim. 10 (2005) 283–294.

[16] L. Kang, M.Y. Sohn, and T.C.E. Cheng, Paired-domination in inflated graphs, Theor. Comput. Sci. 320 (2004) 485–494.

[17] C.L. Lu, M-T. Ko, and C.Y. Tang, Perfect edge domination and efficient edge domination in graphs, Discrete Appl. Math. 119 (2002) 227–250.

[18] S.D. Nikolopoulos, Coloring permutation graphs in parallel, Discrete Appl. Math. 120 (2002) 165–195.

[19] S.D. Nikolopoulos and Ch. Papadopoulos, On the performance of the First-Fit coloring algorithm on permutation graphs, Inform. Process. Lett. 75 (2000) 265–273.

[20] A. Pnueli, A. Lempel, and S. Even, Transitive orientation of graphs and identification of permutation graphs, Canadian J. Math. 23 (1971) 160–175.

[21] H. Qiao, L. Kang, M. Gardei, and D.-Z. Du, Paired-domination of trees, J. of Global Optimization 25 (2003) 43–54.

[22] J. Reif (Ed.), Synthesis of Parallel Algorithms, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[23] C. Rhee, Y.D. Liang, S.K. Dhall, and S. Lakshmivarahan, An $O(n + m)$-time algorithm for finding a minimum-weight dominating set in a permutation graph, SIAM J. Comput. 25 (1996) 404–419.

[24] J. Spinrad, On comparability and permutation graphs, SIAM J. Comput. 14 (1985) 658–670.

[25] J. Spinrad, A. Brandstadt, and L. Stewart, Bipartite permutation graphs, Discrete Appl. Math. 18 (1987) 279–292.

[26] A. Srinivasan, K. Madhukar, P. Nagavamsi, C.P. Rangan, and M.-S. Chang, Edge domination on bipartite permutation graphs and cotriangulated graphs, Inform. Process. Lett. 56 (1995) 165–171.

[27] K.J. Supowit, Decomposing a set of points into chains, with applications to permutation and circle graphs, Inform. Process. Lett. 21 (1985) 249–252.

[28] K.H. Tsai and W.L. Hsu, Fast algorithms for the dominating set problem on permutation graphs, Algorithmica 9 (1993) 601–614.

[29] M. Yannakakis and F. Gavril, Edge domination sets in graphs, SIAM J. Appl. Math. 38 (1980) 364–372.

APPENDIX

Proof of Lemma 2.1

Lemma 2.1 *Let G be an embedded permutation graph with no isolated vertices whose vertex set is $P_\pi = \{p_1, p_2, \ldots, p_n\}$ (determined by the mapping in Eq. (1)), and let $u_1, u_2, \ldots, u_\ell$ ($v_1, v_2, \ldots, v_{\ell'}$, resp.) be the left-to-right maxima (right-to-left minima, resp.) in P_π in order from left to right. Then, for any set A of edges of G whose endpoints dominate the entire vertex set P_π, there exists a matching M of edges of G such that*

- *the endpoints of the edges in M dominate the entire P_π,*
- *$|M| \leq |A|$, and*
- *$M = \{v_{s_1}u_{t_1}, v_{s_2}u_{t_2}, \ldots, v_{s_{|M|}}u_{t_{|M|}}\}$ where $s_1 < s_2 < \ldots < s_{|M|} \leq \ell'$ and $t_1 < t_2 < \ldots < t_{|M|} \leq \ell$*

(i.e., M is a matching which dominates P_π and consists of at most $|A|$ non-crossing *edges each of which connects a left-to-right maximum to a right-to-left minimum of P_π).*

Proof: First, we replace each edge $p_ip_j \in A$ ($i < j$) that does not connect a left-to-right maximum to a right-to-left minimum of P_π by an edge $p_{i'}p_{j'}$ ($i' < j'$) that does so: if p_i is a right-to-left minimum, then $p_{i'} = p_i$, otherwise $p_{i'}$ is a right-to-left minimum that is adjacent to p_i (such a point always exists); similarly, if p_j is a left-to-right maximum, then $p_{j'} = p_j$, otherwise $p_{j'}$ is a left-to-right maximum that is adjacent to p_j. The replacement of edges in A yields an equal-cardinality set A' of edges which are incident on a left-to-right maximum and a right-to-left minimum, and whose endpoints dominate G; yet, the edges in A' do not necessarily form a matching.

Next, let us collect the endpoints of the edges in A' which are right-to-left minima and let $\mathcal{V} = (v_{s'_1}, v_{s'_2}, \ldots, v_{s'_{|A|}})$ be the ordering of these endpoints from left to right (i.e., $s'_1 \leq s'_2 \leq \ldots \leq s'_{|A|}$). We work similarly with the endpoints which are left-to-right maxima and let their ordering from left to right be $\mathcal{U} = (u_{t'_1}, u_{t'_2}, \ldots, u_{t'_{|A|}})$. We show that for each $i = 1, 2, \ldots, |A|$, the points $v_{s'_i}$ and $u_{t'_i}$ are adjacent. Let $v_{s'_j}u_{t'_i}$ be the edge in the set A' that contributed the endpoint $u_{t'_i}$; we distinguish the following cases:

- $j = i$: Trivially true.
- $j < i$: Then, $y(v_{s'_i}) \leq y(v_{s'_j}) < y(u_{t'_i})$. Moreover, since there exist $i-1$ edge endpoints to the left of $u_{t'_i}$ in the ordering $\mathcal{U}$ and $i-2$ edge endpoints, other than $v_{s'_j}$ to the left of $v_{s'_i}$ in the ordering $\mathcal{V}$, there exists an edge $v_{s'_h}u_{t'_{h'}}$ in A' with $h' < i < h$ (Figure 5(a)). This implies that $x(v_{s'_i}) \leq x(v_{s'_h}) < x(u_{t'_{h'}}) \leq x(u_{t'_i})$. The two inequalities relating the coordinates of the points $v_{s'_i}$ and $u_{t'_i}$ imply that these two points are adjacent.

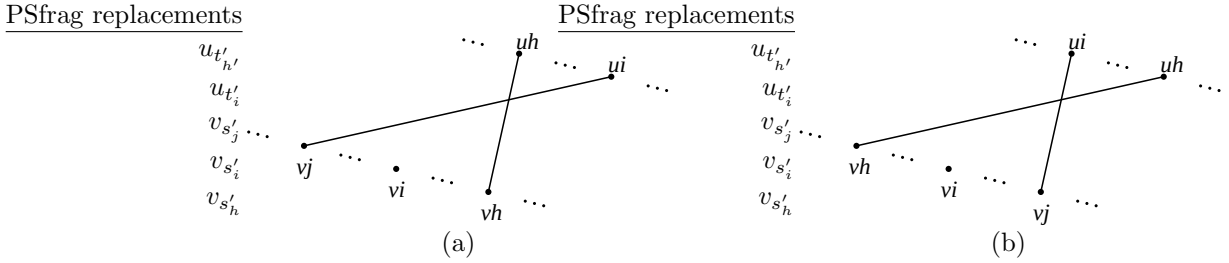


Figure 5: For the proof of Lemma 2.1: (a) the case for $j < i$; (b) the case for $j > i$.

- $j > i$: Then, $x(v_{s'_i}) \leq x(v_{s'_j}) < x(u_{t'_i})$. Moreover, since there exist $n - i$ edge endpoints to the right of $u_{t'_i}$ in the ordering $\mathcal{U}$ and $n - i - 1$ edge endpoints, other than $v_{s'_j}$ to the right of $v_{s'_i}$ in the ordering $\mathcal{V}$, there exists an edge $v_{s'_h}u_{t'_{h'}}$ in A' with $h < i < h'$ (Figure 5(b)). This implies that $y(v_{s'_i}) \leq y(v_{s'_h}) < y(u_{t'_{h'}}) \leq y(u_{t'_i})$. Again, the two inequalities relating the coordinates of the points $v_{s'_i}$ and $u_{t'_i}$ imply that these two points are adjacent.

Thus, we consider the set of edges $\{v_{s'_1}u_{t'_1}, v_{s'_2}u_{t'_2}, \ldots, v_{s'_{|A|}}u_{t'_{|A|}}\}$ whose endpoints dominate all the vertices of G; next, from this set, we remove any duplicate edges and let the resulting set of edges be $A'' = \{v_{s''_1}u_{t''_1}, v_{s''_2}u_{t''_2}, \ldots, v_{s''_{|A''|}}u_{t''_{|A''|}}\}$ where $|A''| \leq |A|$.

Finally, we construct the desired matching M (which initially is equal to the empty set) by processing the edges $v_{s''_i}u_{t''_i}$ of the set A'' for $i = 1, 2, \ldots, |A''|$ in order as follows: If $i = |A''|$ or both $v_{s''_i} \neq v_{s''_{i+1}}$ and $u_{t''_i} \neq u_{t''_{i+1}}$, then we include the edge $v_{s''_i}u_{t''_i}$ in M and proceed with the next edge, if one exists. If $v_{s''_i} = v_{s''_{i+1}}$, let $j_i \geq i + 1$ be such that $v_{s''_i} = v_{s''_{i+1}} = \ldots = v_{s''_{j_i}}$ and either $j_i = |A''|$ or $v_{s''_{j_i+1}} \neq v''_{s_i}$ (i.e., $v_{s''_i}$, $v_{s''_{i+1}}$, $\ldots$, $v_{s''_{j_i}}$ is a maximal subsequence of values equal to $v_{s''_i}$ in the ordered sequence of the endpoints of the edges in A'' that are right-to-left minima); note that because there are no duplicate edges in A'', it holds that $u_{t''_i} < u_{t''_{i+1}} < \ldots < u_{t''_{j_i}}$. Then, we include the edge $v_{s''_i}u_{t''_i}$ in M, and if $j_i \neq |A''|$ we replace the edge $v_{s''_{j_i}}u_{t''_{j_i}}$ by $v_{s''_{j_i+1}}u_{t''_{j_i}}$ and continue by processing this edge, or if $j_i = |A''|$ and $v_{s_{j_i}} < v_{\ell'}$ we include in M the edge $v_{\ell'}u_{t''_{j_i}}$ and stop, whereas if $j_i = |A''|$ and $v_{s_{j_i}} = v_{\ell'}$ we stop without including any edge in M. It is not difficult to see that the resulting set M indeed meets the requirements in the statement of the lemma. ∎

Algorithm PERMUT_PAIRED-DOMINATION: Time and Space Complexity

It is not difficult to see that Procedure Compute_LRMaxima takes $O(n)$ time, and similarly, computing the right-to-left minima and filling the array $rlmin\_left[\,]$ can be done within the same time complexity. Then, Step 1 of Algorithm PERMUT_PAIRED-DOMINATION takes $O(n)$ time and space.

From its description, it follows that Procedure Compute_E_1 also takes $O(n)$ time and space: Step 1 processes the points from the leftmost up to the point preceding u_1 and spends constant time for each one of them, collecting (in a list L) one edge for each right-to-left minimum among these points (thus, the time spent is $O(n)$ and $|L| = O(n)$); Step 2 spends constant time for each of the edges collected in the list L (note that the assignment "$i \leftarrow j$" implies that the two nested while-loops help traverse the list L exactly once). Thus, the entire Procedure Compute_E_1 and hence Step 2 of Algorithm PERMUT_PAIRED-DOMINATION takes $O(n)$ time and space.

In Step 3, we need to be able to determine whether a quadrant contains at least one point of the set P_π. We can efficiently do this check by using an auxiliary array $lowest\_at\_right[1..n-1]$ where

$$lowest\_at\_right[i] = \text{the lowest element of } P_\pi \text{ to the right of point } p_i, \quad 1 \leq i < n.$$

By processing the points in P_π by decreasing value of their x-coordinate (i.e., by traversing the array $A_x[\,]$ from its end to its start), we can fill the array $lowest\_at\_right[\,]$ in $O(n)$ time. Then, for a quadrant Q bounded from left by the line $x = x_Q$ and from above by the line $y = y_Q$, we have that

$$Q \text{ contains a point in } P_\pi \text{ iff } x_Q < n \text{ and } y_Q > y(lowest\_at\_right[x_Q]);$$

this can be checked in $O(1)$ time.

Now let us compute the time and space complexity of Procedure Compute_E_{i+1}. Let n_{i+1} be the number of points $p \in P_\pi$ processed in both for-loops of Step 1 of the procedure. It is not difficult to see that, if we ignore the time taken by stack operations, the body of each of the for-loops can be executed in $O(1)$ time. Additionally, because the number of stack pops does not exceed the number of stack pushes, and at most one stack push is performed per point processed in the first for-loop performs, we conclude

that the total time for all stack operations is $O(n_{i+1})$ as well. In addition to the for-loops, Step 1 involves a constant number of constant-time operations; thus, Step 1 of Procedure Compute_E$_{i+1}$ takes $O(n_{i+1})$ time. Similarly to Procedure Compute_E$_1$, Step 2 of Procedure Compute_E$_{i+1}$ takes $O(|L|) = O(n_{i+1})$ time; thus, the procedure takes a total of $O(n_{i+1})$ time. In order to bound the total time taken by all the executions of Procedure Compute_E$_{i+1}$, we observe (i) that u_{t_h} is the highest endpoint of an edge in E_i and (ii) that the highest endpoint of the edges in E_{i+1} is a left-to-right maximum which either coincides with or is to the right of $u_{a'}$ if the 2nd for-loop is not executed, or coincides with or is to the right of $u_{a'+1}$ if the 2nd for-loop is executed. In light of the definition of n_{i+1}, this observation implies that $\sum_i (n_i - 1) \leq n$, from which we conclude that the total time taken by all the executions of Procedure Compute_E$_{i+1}$ is $O(n)$. The space required for the stack S is $O(|E_i|) = O(n)$. Therefore, Step 3 of Algorithm Permut_Paired-Domination takes $O(n)$ time and space.

Finally, Steps 4 and 5 take $O(|M|) = O(n)$ time and space; note that the set M contains at most one edge per right-to-left minimum.