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; if the permutation over the set $N_n = \{1, 2, \ldots, n\}$ defining a permutation graph G on n vertices is given, our algorithm computes a paired-dominating set of G 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 [11] 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 [11, 12, 13, 14, 18, 31]. 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 [13]; 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.

Classes of graphs	Complexity	Authors
General graphs	NP-complete	Haynes and Slater [13]
Trees	$O(n)$	Qiao, Kang, Gardei, and Du [23]
Inflated trees	$O(n)$	Kang, Sohn, and Cheng [17]
Interval graphs	$O(n+m)$	Cheng, Kang, and Ng [5]
Circular-arc graphs	$O(m(n+m))$	Cheng, Kang, and Ng [5]
Permutation graphs	$O(mn)$	Cheng, Kang, and Shan [6]
Permutation graphs	$O(n)$	Nikolopoulos and Palios [this paper]

Table 1: The complexity status of the paired-domination problem on several classes of graphs.

Trees have been one of the first targets of researchers working on paired-domination: Qiao *et al.* [23] 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 [16] 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.* [17] 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 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 [9], for Cartesian products of graphs [3], and for generalized claw-free graphs [7]; we call $K_{1,3}$ a claw and $K_{1,a}$ a generalized claw, where $a \geq 3$, and thus a graph G is called claw-free (generalized claw-free, resp.) graph if G does not contain $K_{1,3}$ ($K_{1,a}$, resp.) as an induced subgraph. 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 has been given by Cheng *et al.* [5]; they also extended their result to circular-arc graphs giving an algorithm running in $O(m(m+n))$ time in this case. Very recently, Cheng *et al.* [6] gave an $O(mn)$-time algorithm for the paired domination problem on permutation graphs working on the permutation defining the input graph; they use dynamic programming on subsequences of contiguous elements of the permutation.

We too consider the paired domination problem on 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 [10]). Therefore, in this paper, we assume that a permutation graph $G[\pi]$ is represented by the corresponding permutation π.

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.* [22] 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 [26] improved their results by designing an $O(n^2)$-time algorithm for the same problem. In [19], McConnell and Spinrad proposed an $O(n+m)$-time algorithm for modular decomposition and transitive orientation, where n and m are the number of vertices and edges of the graph, which gave a linear time bound for recognizing permutation graphs; a graph G is a permutation graph if and only if both G and its complement are comparability graphs [10]. In [27], 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 [29] 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.* [21] 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 [15, 20, 24].

In addition to the result of Cheng *et al.* [6] on paired domination, several variants of the domination problem have been considered on permutation graphs. Farber and Keil [8] 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 [30] 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.* [25] 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.* [28] 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 following an approach different from that of Cheng *et al.* [6]. We define an embedding 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); if the permutation over the set $N_n = \{1, 2, \ldots, n\}$ defining a permutation graph on n vertices is given, our algorithm runs in $O(n)$ time using $O(n)$ space, and is therefore optimal (recall that the permutation corresponding to a permutation graph G can be computed in time linear in the size of G by producing transitive orientations of G and its complement in $O(n+m)$ time [19]).

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:

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

We note that similar representations have been used by other authors as well; see [1, 21]. In our representation, 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 $P_\pi = \{p_1, p_2, \ldots, p_n\}$.

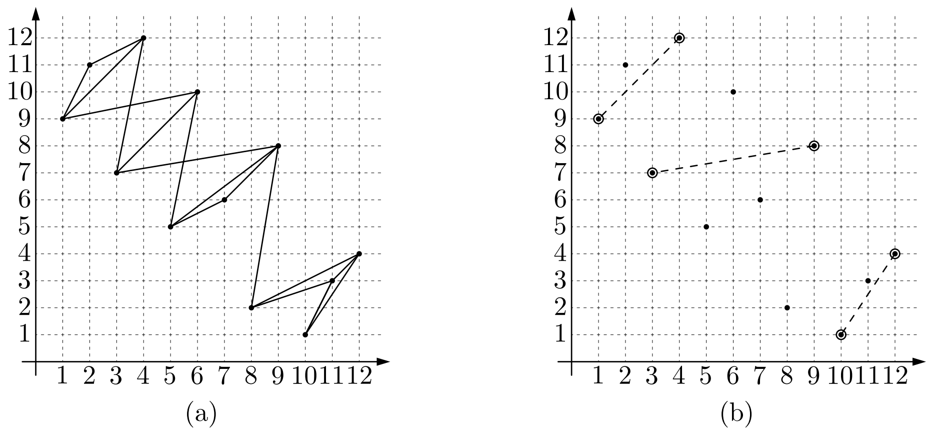
Figure 1: (a) The embedding of the permutation graph corresponding to the permutation $(4, 2, 6, 1, 9, 3, 7, 5, 12, 11, 8, 10)$; (b) A minimum paired-dominating set.

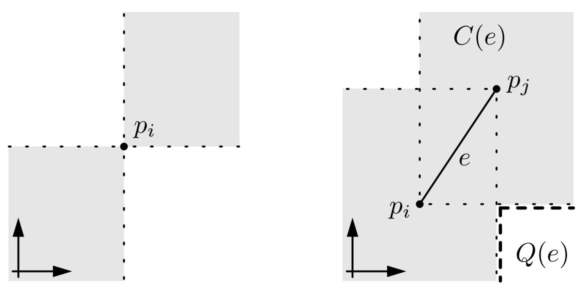
Figure 2:

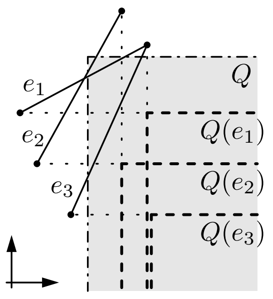
Figure 3:

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, a point p_i dominates all points $p \in P_\pi$ 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,

Definition 2.1 *For any edge $e = p_i p_j$, where $p_i, p_j \in P_\pi$, the portion of the plane* covered by *e is the portion of the plane*

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

dominated by p_i or p_j.

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. For simplicity, we introduce the following notation (see Figure 2 (right)):

Notation 2.1 *For an edge e, we denote by*

$C(e)$ *the portion of the plane covered by e, and*

$Q(e)$ *the bottom right quadrant not covered by e.*

Moreover, a left-to-right maximum of a permutation π defining a permutation graph is mapped to a point $p \in P_\pi$ that is a vertex of the upper envelope of the point set 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 points $(4, 12)$, $(6, 10)$, $(9, 8)$, and $(12, 4)$. Similarly, a right-to-left minimum is mapped to a point $p \in P_\pi$ that is a vertex of the lower envelope of the point set 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 points $(1, 9)$, $(3, 7)$, $(5, 5)$, $(8, 2)$, and $(10, 1)$ of the lower envelope of P_π. For convenience, *each point in P_π corresponding to a left-to-right maximum (right-to-left minimum, resp.) of a permutation π will be called a 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, $P_\pi = \{p_1, p_2, \ldots, p_n\}$ the corresponding point set (determined by the mapping in Eq. (1)), and $u_1, u_2, \ldots, u_\ell$ ($v_1, v_2, \ldots, v_{\ell'}$, resp.) be the left-to-right maxima (right-to-left minima, resp.) of P_π in order from left to right. Then, for any set A of edges of G whose endpoints dominate the entire point 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 $p_i p_j \in A$ $(i < j)$ 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. Since $i < j$, p_i is below and to the left of p_j: if p_i is a right-to-left minimum, then $p_{i'} = p_i$, otherwise there exists a right-to-left minimum $p_{i'}$ below and to the left of p_i; similarly, if p_j is a left-to-right maximum, then $p_{j'} = p_j$, otherwise there exists a left-to-right maximum $p_{j'}$ above and to the right of p_j. In any case, $p_{i'} p_{j'}$ is an edge of G connecting a left-to-right maximum to a right-to-left minimum of P_π. 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.

[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.

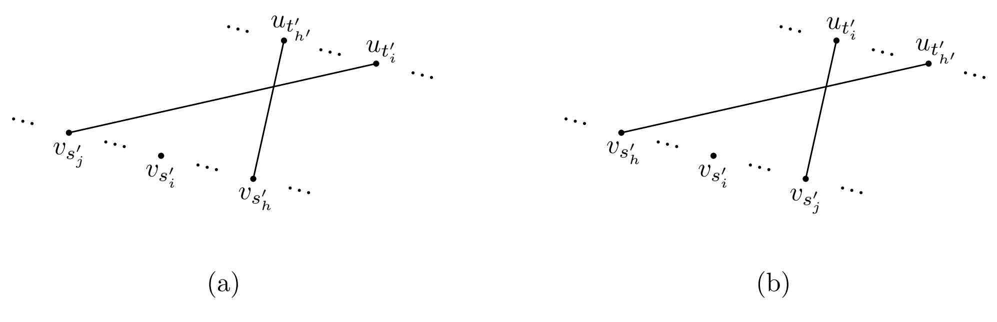

Figure 4: For the proof of Lemma 2.1: (a) the case for $j < i$; (b) the case for $j > i$.

- $j < i$: 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 4(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.
- $j > i$: 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 4(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. ∎

Lemma 2.1 readily implies the following corollary.

Corollary 2.1 *Let G be an embedded permutation graph with no isolated vertices, and $P_\pi = \{p_1, p_2, \ldots, p_n\}$ the corresponding point set. 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 a permutation graph, $P_\pi = \{p_1, p_2, \ldots, p_n\}$ the corresponding point set, 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 point $p_i \in P_\pi$ for which $p_i \notin C(e) \cup Q(e)$, it holds that $p_i \notin C(e') \cup Q(e')$.*

This observation readily implies the following properties for the edges $\{e_1, e_2, ..., \ldots\}$ (in order from left to right) of a matching as described in Corollary 2.1:

- for the *leftmost* edge e_1, it holds that

$$P_\pi \;=\; \big(C(e_1) \cup Q(e_1)\big) \cap P_\pi \tag{2}$$

  i.e., every point in P_π not dominated by the endpoints of e_1 has to lie in its bottom-right non-covered quadrant $Q(e_1)$;

- for *each pair* of consecutive edges e_i, e_{i+1} ($i \geq 1$), it holds that

$$Q(e_i) \cap P_\pi \;=\; \big(C(e_{i+1}) \cup Q(e_{i+1})\big) \cap P_\pi \tag{3}$$

  i.e., each point of P_π in the bottom-right non-covered quadrant $Q(e_i)$ of e_i that is not dominated by the endpoints of e_{i+1}, has to lie or in its bottom-right non-covered quadrant $Q(e_{i+1})$ $\big($in other words, no point is left "between" the covered regions $C(e_i)$ and $C(e_{i+1})\big)$.

3 The Algorithm

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 a 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.

Additionally, in order to obtain a minimum-size set M,

- we maintain only the "usefull" candidates for each edge of M.

In order to formalize this 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*

$$X = \{\, e \in E(G) \mid Q \cap P_\pi = \big(C(e) \cup Q(e)\big) \cap P_\pi \,\}$$

(i.e., all the points of P_π belonging to Q lie either in $C(e)$ or in $Q(e)$). 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 observation enables us to easily identify redundant edges among edges incident on a left-to-right maximum and a right-to-left minimum (see Figure 3):

Observation 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.) 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 $X = \{\, e \in E(G) - A \mid Q \cap P_\pi = \big(C(e) \cup Q(e)\big) \cap P_\pi \,\}$. 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.*

Observation 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 5.

Next, we give 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 $Q(e_{i,j}) \cap P_\pi = \big(C(e) \cup Q(e)\big) \cap P_\pi \,\}$ those that are non-redundant.

To simplify our description, we use the following notation:

Notation 3.1 *For a point $p \in P_\pi$, we denote by*
$lrmax\_above[p]$ the lowest left-to-right maximum above p and
$rlmin\_left[p]$ the rightmost right-to-left minimum to the left of p.

In fact, in the algorithm we use two arrays bearing these names and storing this information; the arrays can be easily filled in $O(n)$ time, the former by processing the points in P_π by decreasing y-coordinate, the latter by processing them by increasing x-coordinate.

Additionally, each collected candidate edge $e \in E_{i+1}$ $(i > 1)$ has a pointer *back* pointing to an edge $e' \in E_i$ so that the edges e and e' satisfy the property in Eq. (3) for $e_{i+1} = e$ and $e_i = e'$; these *back*-pointers help us collect the matching M that we seek.

```
Algorithm PERMUT_PAIRED-DOMINATION
Input  : a permutation $\pi$ 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 $P_\pi$ of points corresponding to the vertices of the graph $G$ based on the mapping
   in Eq. (1);
   compute the left-to-right maxima of the permutation $\pi$ and from these, the left-to-right maxima
   $(u_1, u_2, \ldots, u_\ell)$ of $P_\pi$ as well as the contents of the array lrmax_above[ ]; similarly, compute the
   right-to-left minima $(v_1, v_2, \ldots, v_{\ell'})$ of $P_\pi$ and the contents of the array rlmin_left[ ];
   if there exists a point in $P_\pi$ that 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 non-redundant 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 $P_\pi$ 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 non-redundant candidates for the
          $(i+1)$-st edge in a minimum matching (with endpoints dominating all the vertices of $G$),
          where each edge $e_{i+1,j}$ $(1 \le j \le h_{i+1})$ points to one edge in $E_i$ (by means of a pointer back);
     3.2. $i \leftarrow i + 1$;

4. {collect a solution}
   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$_{i+1}$ (to be presented below) while the minimality follows from the selection of non-redundant candidate edges.

3.1 Computing the set E_1

The goal in the construction of the set E_1 of candidates for the leftmost edge of a solution is that each edge in E_1 is incident on a right-to-left minimum and a left-to-right maximum, is not redundant, and satisfies Eq. (2).

Let e be an edge in E_1 and let it be incident on the right-to-left minimum v_i and the left-to-right maximum u_j (note that v_i is to the left of the leftmost left-to-right maximum u_1; otherwise, u_1 would not be dominated by v_i or by u_j (then, $j > 1$), nor would it belong to $Q(e)$, in contradiction to Eq. (2)). The endpoint u_j of e, in addition to being adjacent to v_i, has to be adjacent to all the points in P_π to the left of v_i (which are not dominated by v_i); therefore, it needs to be above and to the right of the highest point, say, p, among the points with x-coordinate less than or equal to the x-coordinate of v_i. If u_{q_i} is the *lowest* left-to-right maximum above p, then u_j could be any of the left-to-right maxima $u_1, \ldots, u_{q_i}$, and none other. Yet, among the edges $v_i u_1, \ldots, v_i u_{q_i}$, all but the last one are redundant.

More formally, the above discussion is summarized in the following lemma:

Lemma 3.1 *Let G be an embedded permutation graph with no isolated vertices, $P_\pi = \{p_1, p_2, \ldots, p_n\}$ the corresponding point set (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. If $v_r = rlmin\_left[u_1]$, we have:*

(i) For each v_i, $i = 1, 2, \ldots, r$, let $highest\_p_i$ be the highest among the points in P_π with x-coordinate less than or equal to the x-coordinate of v_i, and let $u_{q_i} = lrmax\_above[highest\_p_i]$. Then, Eq. (2) holds for the edge $e_1 = v_i u_q$ for each q such that $1 \le q \le q_i$; Eq. (2) does not hold for any edge $v_i u_q$ with $q > q_i$.

(ii) Among the edges referred to in the statement (i) of the lemma, the edges $v_i u_q$ (where $1 \le q < q_i$) 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 3.1 to construct a list L of edges satisfying Eq. (2) where L contains exactly the single non-redundant edge incident on each right-to-left minimum to the left of u_1 (see statement (ii) of Lemma 3.1); in Step 2, we obtain the final set E_1 by removing any 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 positions in the list L (for any $i < j$) have $s_i < s_j$ and $t_i \ge t_j$.

Procedure Compute_E$_1$

```
1. $highest\_p \leftarrow p_1$;     {the highest point seen so far is the leftmost point}
   $L \leftarrow$ a list containing only the edge connecting $p_1$ to $lrmax\_above[p_1]$;
   $i \leftarrow 2$;     {process the points $p_2, p_3, \ldots$ in order, i.e., by increasing $x$-coordinate}
   **while** $p_i$ does not coincide with the leftmost left-to-right maximum $u_1$ **do**
       **if** $y(p_i) > y(highest\_p)$
       **then** $highest\_p \leftarrow p_i$;     {update highest point seen so far}
       **if** $p_i$ is a right-to-left minimum
       **then** insert at the end of $L$ the edge connecting $p_i$ to $lrmax\_above[highest\_p]$;
       $i \leftarrow i + 1$;

2. $E_1 \leftarrow \emptyset$;     {initially empty set}
   $e \leftarrow$ first edge in the list $L$;
   **while** $e$ is not the last edge in $L$ **do**
     {ignore all edges incident on the same left-to-right maximum as $e$ except for the last one}
       $d \leftarrow e$;
       **while** $d$ is not the last edge in $L$ **and**
               the edge after $d$ in $L$ is adjacent to the same left-to-right maximum as $e$ **do**
           $d \leftarrow$ the edge after $d$ in $L$;
       add the edge $d$ in $E_1$ with its back-pointer pointing to **NIL**;
       $e \leftarrow$ the edge after $d$ in $L$;
   **if** $e$ is the last edge in $L$
   **then** add the edge $e$ in $E_1$ with its back-pointer pointing to **NIL**;
```

The correctness of Step 1 follows from Lemma 3.1; in accordance with statement (ii), for each right-to-left minimum v_i to the left of u_1, we consider only the edge $v_i u_{q_i}$ where $u_{q_i} = lrmax\_above[highest\_p]$ and $highest\_p$ is the highest point among the points with x-coordinate less than or equal to the x-coordinate of v_i. The correctness of Step 2 follows from Observation 3.1; the edges in the resulting set E_1 form a crossing pattern like the one shown in Figure 5.

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 non-redundant 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; we are interested in constructing a set of non-redundant candidate edges E_{i+1} for the $(i+1)$-st edge of M. This case is a generalization of the case for E_1; this time, however, we are dealing with a number of quadrants $Q(e_{i,j})$ which are as shown in Figure 5 due to the crossing pattern of the edges in E_i. Due to the many quadrants, for an edge $e \in E_{i+1}$, Eq. (3) becomes

$$\text{there exists } e_{i,j} \in E_i \ \text{ such that } \ Q(e_{i,j}) \cap P_\pi \;=\; \big(C(e) \cup Q(e)\big) \cap P_\pi. \qquad (4)$$

Therefore, for the construction of E_{i+1}, we are interested in non-redundant edges e incident on a right-to-left minimum and on a left-to-right maximum that satisfy Eq. (4).

The non-redundancy of the elements of E_{i+1} and Eq. (4) imply the following lemma:

Lemma 3.2 *Let G be an embedded permutation graph with no isolated vertices, $P_\pi = \{p_1, p_2, \ldots, p_n\}$ the corresponding point set, 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 $v_a = rlmin\_left[u_{t_1}]$, $u_{a'} = lrmax\_above[v_{s_h}]$, $v_b = rlmin\_left[u_{a'}]$, and $v_c = rlmin\_left[u_{a'+1}]$ (see Figure 5), we have:*

(i) The edge connecting v_a to $lrmax\_above[v_a]$ is a candidate for E_{i+1} (note that this edge may prove to be redundant).

(ii) Each edge incident on a right-to-left minimum to the left of v_a is redundant.

(iii) (a) For no *edge e incident on a right-to-left minimum to* the right *of v_c, there exists $e_{i,j} \in E_i$ that satisfies Eq. (4) with e.*
(b) In particular, if every quadrant $Q(e_{i,j})$, $1 \le j \le h$, contains points $p \in P_\pi$ such that $y(p) > y(u_{a'+1})$, then for no *edge e incident on a right-to-left minimum to* the right *of v_b, there exists $e_{i,j} \in E_i$ that satisfies Eq. (4) with e.*

Proof: (i) Immediate, since this edge is incident on a right-to-left minimum and on a left-to-right maximum and satisfies Eq. (4) for $e_{i,j} = e_{i,1}$.
(ii) Let us consider an edge e incident on a right-to-left minimum v_t, where $t < a$. At best, e would be incident to the left-to-right maximum $lrmax\_above[v_t]$ which is above and to the left of or coincides with $lrmax\_above[v_a]$. Therefore, in accordance with Observation 3.1, e is redundant in light of the edge in statement (i).
(iii) (a) Easy to see, since the endpoints of any edge incident on a right-to-left minimum v_t, where $t > c$, cannot dominate the left-to-right maximum $u_{a'+1}$.
(b) Let us consider an edge e incident on a right-to-left minimum to the right of v_b. This edge cannot satisfy Eq. (4) for any $e_{i,j}$, as every $Q(e_{i,j})$ contains points above the horizontal line $y = y(u_{a'+1})$, and these points cannot be dominated by the endpoints of e. ∎

As an example for statement (i) of the lemma, in Figure 5, the right-to-left minimum v_a contributes the edge $v_a u_{a'+4}$. In the same figure, we note that the quadrant $Q(e_{i,h})$ contains no points above the line $y = y(u_{a'+1})$ and therefore the right-to-left minima v_{b+1} and v_c contribute candidate edges for E_{i+1}.

Therefore, in order to collect candidates (not necessarily non-redundant) for the edges in E_{i+1}, we need to take into account the edge in statement (i) of the above lemma, as well as *one* appropiate edge incident on *each* of the right-to-left minima v_{a+1} to v_b, or to v_c if there exists a quadrant $Q(e_{i,j})$ containing no points with y-coordinate larger than $y(u_{a'+1})$. The obvious way to compute the appropriate edge incident on a right-to-left minimum v_k, is to take each element $e_{i,j}$ of E_i in turn and find the edge incident on v_k and a left-to-right maximum that satisfies Eq. (4) for that $e_{i,j}$ in a way similar to that we used to compute the set E_1 in the previous paragraph (note that each v_k to the right of v_a belongs to the quadrants of all the elements of E_i); then, the sought edge is the non-redundant edge among these

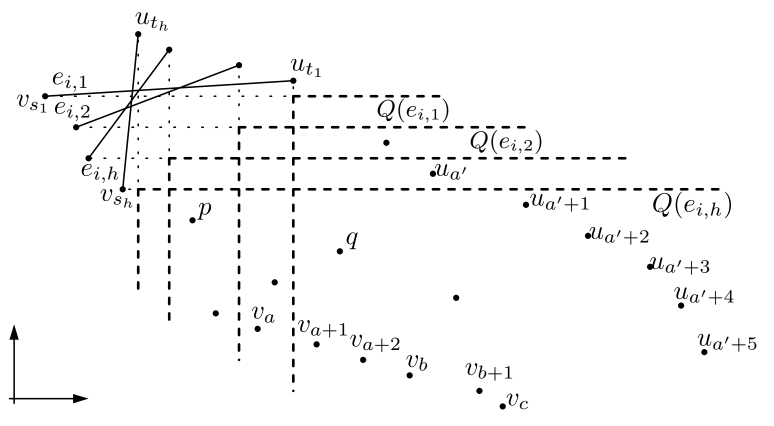

Figure 5: $E_{i+1} = \{v_{a+1}u_{a'+4}, v_{a+2}u_{a'+2}, v_c u_{a'+1}\}$; redundant candidates: $v_a u_{a'+4}, v_b u_{a'+1}, v_{b+1}u_{a'+1}$.

edges. It turns out, however, that for each v_k, we need process only one of the quadrants, as the following lemma establishes.

Lemma 3.3 *Let $E_i = \{e_{i,1}, e_{i,2}, \ldots, e_{i,h}\}$ where $e_{i,j} = v_{s_j}u_{t_j}$ and for $1 \leq j < j' \leq h$, u_{t_j} is* below and to the right *of $u_{t_{j'}}$ (see Figure 5). Consider a right-to-left minimum v_k to the right of v_a. For the computation of the candidate edge incident on v_k to be considered for E_{i+1}:*

(i) there is no need to consider quadrants $Q(e_{i,j})$ that contain points which are above the line $y = y(v_{s_h})$ and to the left of v_k;

(ii) among the quadrants[2] *that do not contain points which are above the line $y = y(v_{s_h})$ and to the left of v_k, consider only the* rightmost *one (i.e., its left side is to the right of the left sides of the other quadrants).*

Proof: (i) Let p be the highest point above the line $y = y(v_{s_h})$ and to the left of v_k in the quadrant $Q(e_{i,t})$ of some edge $e_{i,t} \in E_i$. If u is the leftmost left-to-right maximum incident on v_k, then the edges connecting v_k to each of the left-to-right maxima $u, \ldots, lrmax\_above[p]$ satisfy Eq. (4) for $e_{i,t}$. The non-redundant edge among them is the last one and $lrmax\_above[p]$ coincides with or is to the left of $u_{a'}$. The statement follows by considering the quadrant $Q(e_{i,h})$, which trivially does not contain any points above the line $y = y(v_{s_h})$; an edge incident on v_k that satisfies Eq. (4) for $e_{i,h}$ is the edge connecting v_k to $u_{a'}$ and hence the non-redundant edge for $Q(e_{i,h})$ is certainly no worse than the non-redundant edge for $Q(e_{i,t})$.
(ii) Since these quadrants do not contain points above the line $y = y(v_{s_h})$ and to the left of v_k, each point to the left of v_k that belongs to the rightmost of these quadrans also belongs to the other quadrants, and thus the y-coordinate of the highest point of the rightmost quadrant will not be larger than the y-coordinate of the highest point of any of the remaining quadrants. ∎

For example, in Figure 5, the quadrant that matters for v_{a+1} and v_{a+2} is $Q(e_{i,1})$ and the contributed edges are $v_{a+1}u_{a'+4}$ and $v_{a+2}u_{a'+2}$, respectively; in the former case, the highest point is v_{a+1}, in the latter, it is q. For v_b, the quadrants $Q(e_{i,1})$ and $Q(e_{i,2})$ are not important as they contain a point above the line $y = y(v_{s_h})$ and to the left of v_b; the contributed edge is $v_b u_{a'+1}$ because the highest point is p.

Lemma 3.3 implies that for each right-to-left minimum v_k, it suffices that we consider only the rightmost quadrant that contains no points above the line $y = y(v_{s_h})$ and to the left of v_k; if *highest_p* is the highest point in the quadrant to the left of v_k, then the candidate edge incident on v_k is the edge connecting v_k to $lrmax\_above[highest\_p]$.

Our procedure Compute_E$_{i+1}$ for computing E_{i+1} takes advantage of Lemmas 3.2 and 3.3. Here is an outline of the procedure in general terms. We sweep the points in P_π from left to right starting from u_{t_h} (inclusive) up to v_b (not inclusive) and if there exist quadrants containing no points $p \in P_\pi$ with $y(p) > y(u_{a'+1})$ we continue to v_c (not inclusive). If we encounter the left edge of a new quadrant $Q(e_{i,j})$,

[2]There is at least one such quadrant since the quadrant $Q(e_{i,h})$ trivially contains no points above the line $y = y(v_{s_h})$.

then we save space for some useful information for that quadrant (.e.g., the highest point seen so far) and push the information at the top of a stack S (the use of the stack will be justified while discussing the next case); the stack stores information on only the quadrants that currently do not contain points above the line $y = y(v_{s_h})$ with the righmost among them at the top of the stack. If we encounter a point p' above the line $y = y(v_{s_h})$, we pop from the stack the records of all the quadrants containing it; these quadrants occupy consecutive positions at the top of the stack. At the same time, we keep track of the highest among the points (other than p') seen so far in the popped quadrants and we update the highest point of the quadrant at the top of the stack after the pop operations; note that these points of the popped quadrants also belong to that quadrant as well. If we encounter a point below the line $y = y(v_{s_h})$, we update, if needed, the highest point of the quadrant at the top of the stack, and if the point is a right-to-left minimum to the right of v_a, we add in a list L one edge incident to it and the lowest possible left-to-right maximum. After all the points have been processed, we select among the collected candidate edges those that are non-redundant.

Below, we give pseudocode for the procedure Compute_E$_{i+1}$. The initialization of the list L implements statement (i) of Lemma 3.2. Furthermore, for each quadrant $Q(e_{i,j})$, the stack S stores $e_{i,j}$ (field *edge*), the y-coordinate of the line bounding $Q(e_{i,j})$ from above (field *max_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. (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).*max_y* the values of its fields *edge*, *highest_p*, and *max_y*.)

Procedure Compute_E$_{i+1}$

```
0. 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 $v_a$ be the right-to-left minimum $rlmin\_left[u_{t_1}]$ (i.e., $v_a = rlmin\_left[u_{t_1}]$);
   $L \leftarrow$ a list containing only 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;

1. 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 $e_{i,j} = v_{s_j} u_{t_j} \in E_i$      {left edge of a new quadrant}
1.1    then create a stack record storing $edge \leftarrow e_{i,j}$, $highest\_p \leftarrow$ NIL, and $max\_y \leftarrow y(v_{s_j})$;
            push the record in the stack $S$;
       else if $y(p) > y(v_{s_h})$      {point $p$ above the line $y = y(v_{s_h})$}
1.2         then $q \leftarrow top(S).highest\_p$;
                 while $y(p) < top(S).max\_y$ do      {while $p$ belongs to quadrant at top of stack}
                     if $q =$ NIL or $\big(top(S).highest\_p \neq$ NIL and $y(top(S).highest\_p) > y(q)\big)$
                     then {$q$: highest point of popped quadrants in $Q(e_{i,h})$}
                          $q \leftarrow top(S).highest\_p$;
                     pop the record at the top of $S$;
                 if $top(S).highest\_p =$ NIL or $y(q) > y(top(S).highest\_p)$
                 then $top(S).highest\_p \leftarrow q$;      {new highest point; update}
            else      {point $p$ below the line $y = y(v_{s_h})$, i.e., $p \in Q(e_{i,h})$}
1.3              if $top(S).highest\_p =$ NIL or $y(p) > y(top(S).highest\_p)$
                 then $top(S).highest\_p \leftarrow p$;      {new highest point; update}
                 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 $top(S).edge \in E_i$;

2. {conditionally, process points from $u_{a'} = lrmax\_above[v_{s_h}]$ to $u_{a'+1}$ (not inclusive)}
   if $top(S).highest\_p =$ NIL or $y(top(S).highest\_p) < y(u_{a'+1})$
   then repeat Step 1 for each point in $P_\pi$ from $u_{a'}$ (inclusive) to $u_{a'+1}$ (not inclusive);
```

3. perform Step 2 of procedure Compute_E$_1$ to remove redundant edges from the list L;

We 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. We also note that the edge in the list L incident on v_a needs to be handled separately in Step 3. The remaining edges have the property that for any two edges $v_{s_i}u_{t_i}$ and $v_{s_j}u_{t_j}$ located in the i-th and j-th positions in the list L (for any $i < j$) it holds that $s_i < s_j$ and $t_i \geq t_j$; this need not be true for the edge incident on v_a.

If Procedure Compute_E$_{i+1}$ is executed on the edges and points shown in Figure 5, at the end of Step 2, the list L contains the edges $v_a u_{a'+4}$, $v_{a+1}u_{a'+4}$, $v_{a+2}u_{a'+2}$, $v_b u_{a'+1}$, $v_{b+1}u_{a'+1}$, and $v_c u_{a'+1}$.

3.3 Complexity of Algorithm Permut_Paired-Domination

Let us now compute the time and space complexity of the Algorithm Permut_Paired-Domination. We assume that we are given a permutation π defining the permutation graph.
It is well known that the left-to-right maxima of a permutation can be computed by a single sweep in $O(n)$ time, and from these, the left-to-right maxima of the set P_π can be computed in additional $O(n)$ time by means of the mapping in Eq. (1). A single $O(n)$-time sweep is also needed for filling the array *lrmax_above*[ ]; we note that processing the points by decreasing y-coordinate corresponds to processing the points $(\pi_j,\, n+1-j)$ for $j = n, n-1, \ldots, 1$.

Similarly, computing the right-to-left minima and filling the array *rlmin_left*[ ] can be done within the same time complexity; processing the points in P_π by increasing x-coordinate corresponds to processing the points $(j,\, n+1-\pi_j^{-1})$ for $j = 1, 2, \ldots, n$. 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 "$e \leftarrow$ the edge after d in L" 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

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

(Clearly, no point is to the right of p_n.) By processing the points in P_π by decreasing value of their x-coordinate, 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

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

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

Now let us compute the time and space complexity of Procedure Compute_E$_{i+1}$. Step 0 involves a constant number of constant-time operations and consequently it takes $O(1)$ time. Let k_{i+1} be the number of points $p \in P_\pi$ processed in the for-loops in Step 1 and Step 2 of the procedure. It is not difficult to see that, if we ignore the time taken by stack operations, each execution of the body of the for-loop takes $O(1)$ 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, we conclude that the total time for all stack operations is $O(k_{i+1})$ as well. Hence, Steps 1 and 2 of Procedure Compute_E$_{i+1}$ take $O(k_{i+1})$ time. Similarly to Step 2 of Procedure Compute_E$_1$, Step 3 of Procedure Compute_E$_{i+1}$ takes $O(|L|) = O(k_{i+1})$ time; thus, the entire procedure takes a total of $O(k_{i+1})$ time. In order to bound the total time taken by all the executions of Procedure Compute_E$_{i+1}$, we observe that

(i) u_{t_h} is the highest endpoint of an edge in E_i;

(ii) if the condition at the beginning of Step 2 is false, then we process all the points from u_{t_h} (inclusive) to $u_{a'}$ (not inclusive), and the highest endpoint of the edges in E_{i+1} is the left-to-right maximum $u_{a'}$;

(iii) if the condition at the beginning of Step 2 is true, then we process all the points from u_{t_h} (inclusive) to $u_{a'+1}$ (not inclusive), and the highest endpoint of the edges in E_{i+1} is a left-to-right maximum u_t with $t \geq a' + 1$.

These observations imply that $\sum_{i=2}^{n} k_i \leq n$, 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.

Summarizing, we have the following theorem:

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.*

Since a permutation π corresponding to a permutation graph can be computed from the graph in time linear in its size [19], we conclude that, given a permutation graph G, we can compute a minimum-cardinality paired-dominating set of the graph G in $O(n + m)$ time, where n is the number of vertices and m is the number of edges of the graph.

4 Concluding Remarks

Summarizing, we studied the paired-domination problem on permutation graphs following an approach different from that of Cheng *et al.* [6], and we presented an optimal algorithm which, given a permutation over the set N_n, computes a minimum-cardinality paired-dominating set of the graph G determined by the permutation π in $O(n)$ time using $O(n)$ space.

An interesting open question would be to see whether the ideas and techniques presented in this paper can be efficiently used for optimally solving other related domination problems on permutation graphs such as the edge domination problem, the independent domination problem, the connected domination problem, the locating domination problem.

Additionally, it would be also interesting to see if similar ideas and techniques can be applied to find optimal solutions to paired-domination problem on other classes of graphs.

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] T.C.E. Cheng, L. Kang, and E. Shan, A polynomial-time algorithm for the paired-domination problem on permutation graphs, Discrete Appl. Math. 157 (2009) 262–271.

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

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

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

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

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

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

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

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

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

[16] 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.

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

[18] 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.

[19] R. McConnell and J. Spinrad, Modular decomposition and transitive orientation, Discrete Mathematics 201 (1999) 189–241.

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

[21] 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.

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

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

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

[25] 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.

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

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

[28] 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.

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

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

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