DOMINATING SETS IN PERFECT GRAPHS

Derek G. CORNEIL

Department of Computer Science, University of Toronto, Toronto, Ont., Canada M5S 1A1

Lorna K. STEWART

Department of Computing Science, University of Alberta, Edmonton, Alta., Canada T6G 2E7

Received 2 December 1988

In this paper, we review the complexity of the minimum cardinality dominating set problem and some of its variations on several families of perfect graphs. We describe the techniques which are used to attain these complexity results, with emphasis on the dynamic programming approach to the design of algorithms.

1. Introduction

Since their introduction by Claude Berge in the early 1960s [2], perfect graphs have attracted considerable attention, and many interesting families of graphs have been shown to be contained in the perfect graphs. Perfect graphs are graphs in which the maximum clique size is equal to the chromatic number for every induced subgraph. One of the problems which has been widely studied in relation to these graphs is that of finding a minimum dominating set.

In this paper, we review the results in this area and attempt to give the reader an understanding of the techniques which have been used. Many of these methods involve dynamic programming; we illustrate this approach by developing solutions to the dominating set problem for the family of perfect graphs known as 1-CUBs, and the total dominating set problem for permutation graphs. These algorithms are described in a ‘tutorial’ manner in an attempt to give the reader a detailed understanding of the problem solving techniques.

2. Complexity summary

For a graph $G(V, E)$, a dominating set $S$ is a subset of the vertices such that every vertex in $V - S$ is adjacent to some vertex in $S$. Throughout this paper, $d(G)$ will denote the size of a minimum dominating set in a graph $G$. A dominating set $S$ is independent if the vertices of $S$ are pairwise non-adjacent, total if the subgraph induced by $S$ has no isolated vertices, connected if the vertices of $S$ induce a connected subgraph, a dominating cycle if the subgraph
Table 1. Abbreviations: N = NP-hard, P = Polynomial, * = this paper, e = easy to see

<table>
<thead>
<tr>
<th>Graph family</th>
<th>Domination</th>
<th>Independent</th>
<th>Connected</th>
<th>Total</th>
<th>Cycle</th>
<th>Clique</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-CUBs</td>
<td>N[*]</td>
<td>N[*]</td>
<td>N[*]</td>
<td>N[*]</td>
<td>N[*]</td>
<td>N[*]</td>
</tr>
<tr>
<td>1-CUBs</td>
<td>P[*]</td>
<td>P[*]</td>
<td>P[*]</td>
<td>P[*]</td>
<td>P[*]</td>
<td>P[*]</td>
</tr>
</tbody>
</table>

induced by S has a Hamiltonian cycle, and a dominating clique if the subgraph induced by S is a complete graph. The minimum dominating set problem is that of finding a minimum cardinality dominating set. Variations of this problem which have been studied include finding the minimum cardinality dominating set which is independent, total, connected, a cycle or a clique. We note in passing that considerable work has been done on the weighted cases of these problems as well.

Since all of these problems are known to be NP-hard for general graphs [27], much research has been focussed on finding polynomial time solutions for certain families of graphs. Table 1 summarizes the complexity status of the six minimum cardinality dominating set problems for several families of perfect graphs. The definitions of the various families referred to can be found in the glossary at the end of this paper and in many of the references. The graph theory terminology used is standard and may be found in [28].

The relationships among these and other families of perfect graphs are described in many sources, including [28] and [30]. Some of the straightforward containment relations are as follows:

cographs c permutation c comparability c perfectly orderable,
perfectly orderable c strongly perfect c perfect,
trees c bipartite c comparability,
bipartite c parity c Meyniel c strongly perfect,
split c chordal c weakly chordal,
trees c k-trees (fixed k) c k-trees (variable k) c chordal,
interval c directed path c undirected path c chordal,
interval c directed path c strongly chordal c chordal,
cographs c 1-CUBs c 2-CUBs c CUBs,
chordal c CUBs,
chordal c Meyniel.
Notice that the results of Table 1 imply a number of other polynomial and NP-hardness results because of the containment relationships among the various perfect graphs families. For example, we see that the dominating set problem is NP-hard for perfect graphs as well as for perfectly orderable graphs and strongly perfect graphs, since the comparability graphs are contained in each of these families [3, 14]. The problem can also be seen to be NP-hard for Meyniel graphs and parity graphs since bipartite graphs are contained in these families [13, 37]. In addition, the NP-hardness of the dominating set problem on weakly chordal graphs follows from the result on chordal graphs [29]. Finally, we see that a polynomial time algorithm for domination on interval graphs follows from the algorithm for strongly chordal graphs and from the algorithm for directed path graphs. since interval graphs are contained in both of these classes of graphs [9, 25].

3. Techniques

In this section we examine the techniques most commonly used for establishing the complexity status of various domination problems on perfect graphs. Polynomial algorithms will be demonstrated for the domination problem on \( 1\)-CUBs and the total domination problem on permutation graphs. First we show that the domination problem (i.e. given a graph \( G \) and integer \( k \) is \( d(G) \leq k \)) is NP-complete for \( k \)-CUBs \((k \geq 2)\).

3.1. NP-completeness techniques

The most commonly used transformation is from the \( h \)-vertex cover problem namely given a graph \( G(V, E) \) and a positive integer \( h \leq |V| \) does there exist a subset \( V' \subseteq V \) s.t. \( |V'| \leq h \) and each edge in \( E \) has at least one endpoint in \( V' \)? This problem was shown to be NP-complete by Karp [31]. As an illustration of a transformation from the \( h \)-vertex cover problem to the domination problem, consider the class of \( k \)-CUBs \((k \geq 2)\) (see [19]). This proof is identical to that used to establish the NP-completeness of the domination problem on split graphs [20].

**Theorem 3.1.** The domination problem is NP-complete for \( k \)-CUBs \((k \geq 2)\).

**Proof.** Clearly the problem belongs to NP. We reduce the \( h \)-vertex cover problem to this restricted domination problem as follows. Given graph \( G(V, E) \) and integer \( h \) we construct a 2-CUB \( G'(V', E') \) by bonding using a \( K_{h+1} \) and \( |E| \) copies of \( K_3 \). (Clearly these complete graphs are 2-CUBs.) For each edge \( (i, j) \in E \) a new \( K_3 \) is 2-bonded to the edge \( (i, j) \) in \( K_{h+1} \). We will show that \( G \) has a vertex cover of \( h \) vertices if and only if \( G' \) has a dominating set of \( h \) vertices. Let \( A \) be a vertex cover of \( G \). The same set of vertices chosen in the \( K_{h+1} \) in \( G' \) is
clearly a dominating set in $G'$. Conversely let $B$ be a dominating set in $G'$. Any vertex which does not belong to $K_{|V_1|}$ can be replaced by a neighbouring vertex in $K_{|V_1|}$ without destroying the domination property. This new dominating set $B' \subseteq K_{|V_1|}$ also forms a vertex cover of $G$. □

As an immediate corollary of this theorem we have:

**Corollary 3.2.** The connected, total, cycle and clique domination problems on $k$-CUBs ($k \geq 2$) are NP-complete.

### 3.2. Polynomial techniques

We now turn to the techniques used to show that various classes of perfect graphs have polynomial time domination algorithms. Many classes of perfect graphs have tree representations (sometimes unique). Dynamic programming on such a tree is then used to solve the domination problem. In these algorithms a minimum dominating set for the subgraph $G_x$ represented by node $x$ in the tree is determined from minimum dominating sets for the subgraphs represented by the children of $x$. For some families of graphs, this straightforward dynamic programming is not sufficient and a more sophisticated variation is needed. This could involve the storage of all minimum dominating sets for $G_x$ or other information such as a minimum dominating set with specific properties (such as the inclusion or exclusion of a particular node). Furthermore in some cases there could be an exponential number of minimum dominating sets for $G_x$. Sometimes a polynomial sized representative set suffices. An example of this is presented in [21].

We now illustrate these techniques by solving two new dominating set problems, namely the dominating set problem on 1-CUBs and the total domination problem on permutation graphs. In these presentations, we follow a tutorial approach and illustrate both the false attempts and the solution.

#### 3.2.1. Polynomial algorithm for the domination problem on 1-CUBs

1-CUBs are an extension of cographs and thus it seems possible that an algorithm similar to one for cographs will work for 1-CUBs. The essential difference arises in the treatment of 1-bonding. A parse tree for a CUB is a tree which illustrates a possible composition of the graph using the operations of complement, union and bonding. Assuming that there is a 1-CUB recognition algorithm which will produce a parse tree for a 1-CUB (such an algorithm will be presented later), we focus on the part of the domination algorithm dealing with graph $G$ where $G$ is the 1-bonding of graphs $G_1$ and $G_2$ at vertex $x$. For this analysis we will try to determine the domination number, $d(G)$ from information which could have been recursively calculated for $G_1$ and $G_2$. (In a subsequent algorithm an actual minimum dominating set will be produced.) The first
observation is that either \( d(G) = d(G_1) + d(G_2) \) or \( d(G) = d(G_1) + d(G_2) - 1 \). An attempt to determine which case holds leads to the following lemma. In this lemma \( d(G, x) \) refers to the smallest domination set of \( G \) where \( x \) must be in the domination set.

**Lemma 3.3.** If \( G \) is formed by the 1-bonding of graphs \( G_1 \) and \( G_2 \) at vertex \( x \), then

\[
\begin{align*}
d(G) &= \begin{cases} 
   d(G_1) + d(G_2) - 1 & \text{if } d(G_1) = d(G_1, x) \land d(G_2) = d(G_2, x), \\
   d(G_1) + \min(d(G_2), d(G_2 - x)) & \text{if } d(G_1) \neq d(G_1, x) \land d(G_2) = d(G_2, x), \\
   d(G_2) + \min(d(G_1), d(G_1 - x)) & \text{if } d(G_1) = d(G_1, x) \land d(G_2) \neq d(G_2, x), \\
   d(G_1) + d(G_2) & \text{if } d(G_1) \neq d(G_1, x) \land d(G_2) \neq d(G_2, x).
\end{cases}
\end{align*}
\]

In order for this lemma to be useful in a polynomial time algorithm for calculating the domination number for a 1-CUB we have to calculate the following information for each subgraph \( H \) determined by a vertex in the parsing tree of the given 1-CUB:

\[
d(H), \quad d(H, y), \quad \forall y \in H; \quad d(H - y), \quad \forall y \in H.
\]

Although it may be possible to calculate \( d(H, y) \) efficiently there appear to be difficulties with \( d(H - y) \). It seems that one may need to calculate \( d(H - y - z) \forall z \in H, \ z \neq y \). This in turn would require the exclusion of all possible triples etc. Thus we see that an exponential amount of work may be required.

The obvious difficulty with this general approach is that we have not exploited the structure of 1-CUBs. We now present a polynomial time algorithm which does exploit this structure. This algorithm will use a tree representation of the 1-CUB which is constructed in the following polynomial time 1-CUB recognition algorithm.

**Algorithm 3.1.** 1-CUB \( (G) \) (1-CUB recognition algorithm).

Input: \( G \).
Output: a 1-CUB parse tree for \( G \) if \( G \) is a 1-CUB, “NO” otherwise.

1. If \( G = \{x\} \), then output \[ x. \]
2. If \( G = A_1 \cup A_2 \), then output \[ U \]

\[ 1\text{-CUB}(A_1) \quad 1\text{-CUB}(A_2). \]
(3) If $\mathcal{G} = A_1 \cup A_2$, then output

$$
\begin{array}{c}
\emptyset \\
\text{1-CUB}(A_1) \quad \text{1-CUB}(A_2)
\end{array}
$$

(4) If $\mathcal{G}$ is formed by the 1-bonding of $A_1$ and $A_2$ at vertex $x$, then output

$$
\begin{array}{c}
\mathcal{B}_x \\
\text{1-CUB}(A_1) \quad \text{1-CUB}(A_2)
\end{array}
$$

(5) If $G$ has no cutpoints, output 'NO' and halt. Otherwise let $A_1, A_2, \ldots, A_k$ be the blocks of $G$, that is, the induced subgraphs of $G$ which have no cutpoints and which are maximal with respect to this property. Output

$$
\begin{array}{c}
T \\
\text{1-CUB}(A_1) \quad \text{1-CUB}(A_2) \quad \text{1-CUB}(A_k)
\end{array}
$$

where $T$ is the rooted tree whose nodes are $A_i$ (1 $\leq$ $i$ $\leq$ $k$) and $T$ represents the tree structure among the blocks of $G$.

As an example of the above algorithm consider the graph in Fig. 1. The 1-CUB tree produced by the algorithm is presented in Fig. 2.

Examination of Algorithm 3.1 yields the following facts about the 1-CUB tree $T_G$.

**Fact 3.4.** If the root of $T_G$ is a $\mathcal{B}_x$ node with children $A_1$ and $A_2$ then:

(i) there exists a vertex $y \in A_1$ s.t. $(x, y) \notin E_G$,
(ii) there exists a vertex $z \in A_2$ s.t. $(x, z) \notin E_G$,
(iii) there exists a vertex $w \in A_1$ or $A_2$ s.t. $(x, w) \in E_G$.

**Fact 3.5.** If the root of $T_G$ is a $B^T$ node with children $A_1, \ldots, A_k$ then each $A_i$ is rooted at a $\mathcal{U}$ node or a $\mathcal{B}_x$ node.

![Fig. 1. G.](image-url)
We now present Algorithm 3.2 which will calculate a minimum dominating set of 1-CUB $G$.

**Algorithm 3.2.** Domination (1-CUB domination algorithm).

Output $D$ a minimum dominating set of $G$.

1. If $G = \{x\}$ set $D$ to $\{x\}$ and halt.
2. If $T_G$ is rooted at a $\cup$ node with children $A_1$ and $A_2$, use Algorithm 3.2 to calculate $D_1$, a minimum dominating set of $A_1$ and $D_2$, a minimum dominating set of $A_2$. Set $D$ to $D_1 \cup D_2$ and halt.
3. If $T_G$ is rooted at a $\cup$ node with children $A_1$ and $A_2$ then look for $x$, a universal vertex in $G$, that is, a vertex which is adjacent to all other vertices of $G$. If such an $x$ exists set $D$ to $\{x\}$ and halt. Otherwise choose $x \in A_1$, $y \in A_2$, set $D$ to $\{x, y\}$ and halt.
4. If $T_G$ is rooted at a $\bar{B}_x$ node with children $A_1$ and $A_2$ then look for $y \neq x$ a universal vertex in $G$ in which case set $D = \{y\}$ and halt. Otherwise choose $y \in A_1$ and $z \in A_2$, set $D$ to $\{y, z\}$ and halt. ($y$ and $z$ are chosen so that either $y$ or $z$ is adjacent to $x$.)
5. If $T_G$ is rooted at a $B^T$ node with children $A_1, \ldots, A_k$, where these children are in a pruning order of $T$ (i.e. $A_k$ is the root of $T$) and $x_i$ is the cutnode between $A_i$ and the path in $T$ leading to the root then perform the following algorithm. Note that each $A_i$ is a $\cup$ or $\bar{B}_x$ node.
Set $D$ to $\emptyset$

/*do a pruning order scan of $A_1, \ldots, A_{k-1}$/

for $i := 1$ to $k - 1$:

if $D$ dominates $A_i \setminus \{x_i\}$ then continue

else if $D \cup \{x_i\}$ dominates $A_i$ then add $x_i$ to $D$ and continue

else if there exists $z \in A_i \setminus \{x_i\}$ such that $D \cup \{z\}$ dominates $A_i \setminus \{x_i\}$

then add $z$ to $D$ and continue

else if there exists $z \in A_i \setminus \{x\}$ such that $D \cup \{x\} \cup \{z\}$ dominates $A_i$

then add $z$ and $x_i$ to $D$ and continue

else find $y, z \in A_i \setminus \{x_i\}$ such that $D \cup \{y\} \cup \{z\}$ dominates $A_i$,

add $y$ and $z$ to $D$ and continue
end

/* now handle the root $A_k$/

if $D$ dominates $G$ then halt

else if there exists $x \in A_k$ such that $D \cup \{x\}$ dominates $G$

then add $x$ to $D$ and halt

else find $x, y \in A_k$ such that $D \cup \{x\} \cup \{y\}$ dominates $G$,

add $x$ and $y$ to $D$ and halt

As an example of this algorithm, consider the graph in Fig. 1. Let the pruning order be $A_1 = \{1, 2, 3, 9, 10\}$, $A_2 = \{5, 6, 7\}$ and $A_3 = \{3, 4, 5, 8\}$, where $x_1 = 3$ and $x_2 = 5$. When $i = 1$, $D = \emptyset$ initially, and $x_1 = 3$ does not dominate $A_1$. However vertex 1 dominates $A_1 \setminus \{3\}$. Thus $D = \{1\}$. When $i = 2$, $D = \{1\}$ initially, which does not dominate $A_2$. The addition of $x_2 = \{5\}$ to $D$ does dominate $A_2$, so $D$ now is $\{1, 5\}$. When $i = 3$, since $\{1, 5\}$ does not dominate $G$, another node (for example 4) is added, yielding a minimum dominating set $\{1, 4, 5\}$.

We now sketch a proof that Algorithm 3.2 does in fact find a minimum dominating set for a 1-CUB in polynomial time.

Lemma 3.6. Algorithm 3.2 runs in polynomial time.

Proof. Since the operations of complementation, connected components and search for cut points all may be performed in polynomial time, a 1-CUB tree $T_G$ can be constructed in polynomial time. Steps 1, 3 and 4 of Algorithm 3.2 clearly may be performed in polynomial time. In step 5 for each $i$ at worst all pairs of vertices have to be checked to see if they dominate $A_i$ or $A_i \setminus \{x_i\}$. A simple induction argument establishes the polynomial time requirement of step 2.

Theorem 3.7. The $D$ calculated by Algorithm 3.2 is a minimum cardinality dominating set of the 1-CUB $G$.

Proof. The theorem is trivially true for the cases where the root of $T_G$ is not a $B^r$ node. Thus we assume that the root is a $B^r$ node with children $A_1, \ldots, A_k$ with
this ordering being a 'pruning' order of \( T \) which is rooted at \( A_k \). From Fact 3.4 we know that each \( A_i \) \((1 \leq i \leq k)\) is rooted at a \( U \) or a \( B_x \) node. We let \( G_i^* \) denote the subgraph of \( G \) induced by the subtree of \( T \) rooted at \( A_i \). The bonding node joining \( G_i^* \) to the rest of \( G \) is \( x_i \) (by convention \( x_k = \emptyset \)). Note that \( x_i \) may equal \( x_j \) for \( i \neq j \). Finally we let \( D_i \) denote the value of \( D \) after \( A_i \) has been processed. Thus \( D_{i-1} \) is the value of \( D \) on input to the processing of \( A_i \) (by convention \( D_0 = \emptyset \) denotes the value of \( D \) on input to the processing of \( A_1 \)). \( D_i^* \) is defined to be \( D_i \cap G_i^* \). In order to prove Theorem 3.7 we use the following lemma:

**Lemma 3.8.** \( D_i^* \) is a minimum dominating set of \( G_i^* \setminus \{x_i\} \) \((1 \leq i \leq k)\) where \( D_i^* \) contains \( x_i \) if possible.

**Proof.** (By induction). Assume \( A_i \) is a leaf of \( T \) (note \( G_i^* = A_i \)). If \( i = 1 \) (i.e. \( D_0 = \emptyset \)) then clearly \( D_1 = D_1^* \) is a minimum dominating set of \( A_1 \setminus \{x_1\} \) which will contain \( x_1 \) if possible. If \( i > 1 \) and \( A_i \) is a leaf of \( T \) \( D_{i-1} \cap G_i^* \) is either \( \emptyset \) or \( \{x_i\} \). In both cases it is clear that \( D_i^* \) is a minimum dominating set of \( G_i^* \setminus \{x_i\} \) and will include \( x_i \) if possible.

For \( i > 1 \) we now assume that for all \( j \leq i - 1 \), \( D_j^* \) is a minimum dominating set of \( G_j^* \setminus \{x_j\} \) which contains \( x_j \) if possible. Since \( D_i^* \) contains \( x_j \) if possible, \( D_{i-1} \) dominates as much of \( A_i \) as possible without losing minimality. Note that the only nodes of \( A_i \) which may be dominated by \( D_{i-1} \) are \( x_j \)'s \((1 \leq j \leq i - 1)\) which are in \( A_i \) or vertices adjacent to such an \( x_j \). Since \( A_i \) is a \( U \) or \( B_x \) node, clearly the algorithm will add the minimum number of nodes to \( D_{i-1} \) in order to complete the domination of \( A_i \setminus \{x_i\} \) and will include \( x_i \) if possible. \( \square \)

We now return to the proof of Theorem 3.7. \( D_k^* = D_k \) (the final value of \( D \)) is a minimum dominating set of \( G_k^* \setminus \{x_k\} = G \) as required \( \square \)

We note that Algorithm 3.2 can be modified to produce a minimum cardinality connected or total dominating set for 1-CUBs. The dominating cycle and dominating clique problems can also be solved with a similar algorithm. Some of the key observations are the following, where \( x \) is a \( B^r \) node. Any connected dominating set for \( G_x \) must contain all of the cutpoints. In calculating a total dominating set, we must be careful not to leave any isolated vertices in the set which cannot subsequently have a neighbour added to the set. Any dominating cycle must be entirely contained within a single 2-connected component, and any dominating clique must be contained within a block. We now turn to the problem of total domination on permutation graphs.

### 3.2.2. Total domination of permutation graphs

In this section we develop a different type of dynamic programming algorithm which finds a minimum cardinality total dominating set in a permutation graph. The preceding domination algorithm relies on a decomposition of a 1-CUB,
whereas the following algorithm relies on a representation for the entire permutation graph and a theorem describing the structure of the particular sets that we are looking for. The algorithm of this section is similar to the other known domination algorithms for permutation graphs [10, 12, 16, 18, 26]. After this paper was written, it was brought to our attention that a different, independently discovered, algorithm for total domination of permutation graphs appears in [10, 11].

A permutation graph is a graph $G(V, E)$ for which there exists a labelling $\{v_1, v_2, \ldots, v_n\}$ of $V$ and a permutation $\pi$ of $\{1, 2, \ldots, n\}$ such that $i$ appears before $j$ in exactly one of $\{1, 2, \ldots, n\}$ and $\pi$ if and only if $(v_i, v_j) \in E$ [39]. A widely used representation for a permutation graph is the permutation diagram [23]. The permutation diagram for a graph $G$ with $n$ vertices is formed by writing in a column the integers $\{1, 2, \ldots, n\}$ in order and, to the right, another column containing the integers $\{1, 2, \ldots, n\}$ in the order in which they appear in $\pi$. $\pi$ refers to a permutation which gives rise to this permutation graph, as described in the definition. We then add lines joining $i$ in the left column with $i$ in the right column for all $1 \leq i \leq n$. We are left with a set of $n$ line segments, each of which corresponds to a vertex of the graph, and two line segments cross if and only if the corresponding vertices are adjacent in the graph. There may be an exponential number of permutation diagrams for a particular permutation graph, but for our purpose, any one will suffice. Given a graph, the $O(n^2)$ algorithm of Spinrad [41] will determine whether or not it is a permutation graph and, if so, will produce a permutation diagram for the graph.

The following result tells us that any permutation graph which has a total dominating set must have a minimum cardinality total dominating set with a very specific structure.

**Theorem 3.9.** Let $G$ be a permutation graph for which there exists a total dominating set. Then there exists a minimum cardinality total dominating set (mctds) of $G$ which consists of the union of disjoint non-trivial paths (simple paths, each of size two or more).

**Proof.** Let $T$ be any total dominating set for a permutation graph $G$. Let $G(T)$ be the subgraph of $G$ induced by the vertices of $T$ and let $H$ be any connected component of $G(T)$. $H$ must have two or more vertices since $T$ is a total dominating set.

**Claim 1.** $H$ contains no vertex of degree $> 2$.

**Proof of Claim 1.** Suppose $H$ had a vertex $v$ of degree $\geq 3$. Let us examine $v$ together with any three of its neighbours in $H$. All possible permutation diagrams (up to symmetry) for $v$ and these three adjacent vertices are shown in Fig. 3. In each case, we can identify at least one line which corresponds to a vertex which is redundant in $T$. All such lines are labelled $r$ in the figure. The existence of these redundant lines contradicts the minimality of $T$. Thus Claim 1 is proved.
From Claim 1, we conclude that $H$ must consist of a simple cycle or a simple path. Since $G$ is a permutation graph, we know that any induced cycle has 3 or 4 vertices. By examination of the permutation diagram, it can be seen that $H$ will not be a $C_3$, because this contains a vertex which would be redundant in $T$. Thus $H$ is a simple path or $H$ is isomorphic to $C_4$.

**Claim 2.** Any connected component of $G(T)$ which is a $C_4$ can be replaced in $T$ by a $P_4$, resulting in a total dominating set $T'$ with the same cardinality as $T$.

**Proof of Claim 2.** Let $H$ be a connected component of $G(T)$ which is isomorphic to $C_4$. Let $v$ be any vertex of $H$ and let $w$ be a vertex in $V - T$ which is adjacent to $v$ but to no other vertex of $T$. Such a vertex must exist since otherwise $v$ would be redundant in $T$, contradicting the minimality of $T$. The permutation diagram for $H \cup \{w\}$ must be symmetric to that of Fig. 4. But from the diagram, we can see that $H' = H - \{x\} \cup \{w\}$ dominates all vertices that $H$ dominates, and that $H' \cong P_4$. Furthermore, $|T'| = |T|$, where $T' = T - \{x\} \cup \{w\}$. Thus Claim 2 is proved.

From the proof of Claim 2, we can also conclude that $H'$ is guaranteed to be a connected component of $T' = T - \{x\} \cup \{w\}$ since $w$ is not adjacent to any vertex of $T$ except $v$. Let $T''$ be the set of lines which results from $T$ by replacing each $C_4$ of $G(T)$ by a $P_4$, as described. Now $T''$ is a mctds for $G$ which consists of the union of disjoint non-trivial paths. $\square$
From the theorem we see that for a permutation graph $G$, a dominating set consisting of a collection of disjoint nontrivial paths, with minimum cardinality over all such sets, is a mctds for $G$. Thus, our algorithm need only construct such a minimum cardinality collection of paths. We do this by identifying a set of lines in the permutation diagram which corresponds to such a dominating set of disjoint paths in the graph.

Some of the notation concerning the permutation diagram follows. A dominating set of lines in a permutation diagram is a set of lines $L$ such that every line not in $L$ crosses at least one line of $L$. There is a one-to-one correspondence between dominating sets of vertices in a permutation graph $G$ and dominating sets of lines in a permutation diagram representing $G$. For a line $x$ in a diagram $D$, $\text{left}(x)$ refers to the position of $x$ on the left side of $D$ and $\text{right}(x)$ is the position of $x$ on the right side, where the top position on both sides is 1 and the bottom position is $n$, the total number of lines in $D$. The first line of a set of lines $Y$ in $D$ is the line $x \in Y$ such that $\text{left}(x) \leq \text{left}(y)$ for all lines $y \in Y$.

**Lemma 3.10.** For a permutation diagram $D$, let $\text{DS}$ be a dominating set of lines in $D$. Let $e$ be the first line of $\text{DS}$ and let $l$ be the line of $D$ with $\text{right}(l) = 1$. Then $\text{left}(e) \leq \text{left}(l)$.

**Proof.** Any line $x$ in $D$ crosses $l$ if and only if $\text{left}(x) < \text{left}(l)$. This is because all lines have right endpoints greater than $\text{right}(l)$. Since $l$ must be dominated, there must be some line in $\text{DS}$ with $\text{left}(\cdot) \leq \text{left}(l)$; hence, we must have $\text{left}(e) \leq \text{left}(l)$. □

In light of this lemma and the previous theorem, we might use the following approach to calculate a mctds in a given permutation diagram. We begin by finding all potential first lines for a mctds. These are the lines with left endpoints $\leq \text{left}(l)$, where $l$ is the line with $\text{right}(l) = 1$. Then for each line $e$ with
left(e) \leq \text{left}(l), we calculate a mctds with e as the first line, and finally choose the minimum cardinality set obtained over all choices of e.

For a particular choice of first line e, we proceed as follows. Let x be the line of minimum right endpoint such that left(x) < left(e). If right(x) < right(e) then x is the line highest on the right side of the diagram which is not dominated by e. Let f be the line of maximum left endpoint such that right(f) < min[right(x), right(e)]. If no such line exist or if e and f do not cross then e cannot be the first line of a total dominating set for D. Otherwise, the lines e and f cross.

**Lemma 3.11.** If there exists a minumum cardinality collection of disjoint non-trivial paths which dominates D and which has e as the first line, then there exists a minimum cardinality collection of disjoint non-trivial paths which dominates D, has e as the first line, and contains f.

**Proof.** Let T be a minimum cardinality collection of disjoint nontrivial paths which dominates D. Suppose T has e as the first line and suppose that f \notin T. Now T must contain some line y with right(y) < min[right(x), right(e)] and left(e) < left(y) < left(f). The set \{e, f\} clearly dominates all lines which are dominated by \{e, y\} and possibly more. Thus, T' = T - \{y\} \cup \{f\} is a mctds in D. All components of T' are the same as those of T, except for the component C' of T' which contains f. Let C be the component of T which contains y.

From Claim 1 of Theorem 3.9, we know that each line of T (T') crosses at most two other lines of T (T').

If y crosses two lines in T then f crosses the same two lines in T', since f crosses at least as many lines as y, but cannot cross more than two lines of T'. Thus C' has the same configuration as C, that is, a path.

If y crosses only e in T then either C = \{e, y\} or C is a path of length greater than 2, \{y, e, . . . \}.

Suppose C = \{e, y\}, and let w be the first line of T - C. If w crosses two lines of T, then f cannot cross w, because then w would have three neighbours in T'. In this case, C' has the same configuration as C, namely, a path. If w crosses only one line of T, then f may or may not cross w. If f crosses w then C' consists of the first two components of T combined into a path of length \geq 4. If f does not cross w then C' = \{e, f\} and T' - C' = T - C.

If C is a path of length 3, \{y, e, v\}, then f cannot cross v as this would render v redundant. Thus C' is the same as C, a path.

If C is a path of length 4, \{y, e, v, z\}, then if f does not cross z we have a path in C'. The line f cannot cross v since this would give f three neighbours in T'. If f does cross z, we have a C_4. But then z must dominate some line w in D - T which is not dominated by any other line of T, for otherwise, z would be redundant in T. Now, T' - \{v\} \cup \{w\} is a minimum cardinality collection of nontrivial paths which dominates D, has e as its first line, and contains f.
If \( C \) is a path of length \( >4 \), \( \{ y, e, v, z, \ldots \} \), then \( f \) cannot cross \( z \) since this would give \( z \) three neighbours in \( T' \) \( \Box \)

For a particular first line \( e \), we can calculate the line \( f \), resulting in a path of length 2, \( \{ e, f \} \), which we know is part of a mctds for \( D \) if any mctds exists for \( D \) having \( e \) as its first line. We wish to extend this \( P_2 \) to a total dominating set for \( D \) by adding the minimum number of lines. We now consider a general incremental or recursive formula for constructing a mctds in a permutation diagram by extending a partial total dominating set to a complete one by adding the minimum number of lines.

Let \( TD(l_m, r_m, l_s, r_s, \text{type}) \) be the minimum number of lines which must be added to \( TD_p \) to form a total dominating set for \( D \), where \( TD_p \) is any subset of the lines of \( D \) with the following properties.

(i) \( TD_p \) consists of a collection of disjoint nontrivial paths.
(ii) \( TD_p \) dominates exactly those lines of \( D \) with \( left \leq l_m \) or \( right \leq r_m \) or both.
(iii) The maximum left endpoint of a line in \( TD_p \) is \( l_m \), the maximum right endpoint of a line in \( TD_p \) is \( r_m \), the second maximum left endpoint in \( TD_p \) is \( l_s \), the second maximum right endpoint in \( TD_p \) is \( r_s \).
(iv) The structure of the path in the connected component of \( TD_p \) which is lowest in \( D \) is given by \( \text{type} \) as follows:

\[
\text{type} = \begin{cases} 
\emptyset & \text{if } TD_p = \emptyset, \\
2 & \text{if the lowest component of } TD_p \text{ has size } 2, \\
l & \text{if the lowest component has size } >2 \text{ and the line with } \text{left} = l_m \text{ is an endpoint of the path}, \\
r & \text{if the lowest component has size } >2 \text{ and the line with } \text{right} = r_m \text{ is an endpoint of the path}.
\end{cases}
\]

We now state a formula for \( TD(l_m, r_m, l_s, r_s, \text{type}) \) in terms of \( TD(l'_m, r'_m, l'_s, r'_s, \text{type}') \) where \( l'_m \geq l_m, r'_m \geq r_m \) and \( l'_m + r'_m > l_m + r_m \). For a particular set of endpoints \( l_m, r_m, l_s, \) and \( r_s \), we define the lines \( g, h, e_i, f_i, \text{lex} \) and \( \text{rex} \), with respect to these endpoints as follows. Let \( ND = \{ x \mid \text{left}(x) > l_m \text{ and } \text{right}(x) > r_m \} \). Now \( g \) is the line in \( ND \) with minimum left endpoint and \( h \) is the line in \( ND \) with minimum right endpoint. For any \( i \) such that \( \text{left}(g) \leq i \leq \text{left}(h) \), we let \( e_i \) refer to the line having left endpoint equal to \( i \), and let \( f_i \) refer to the line with maximum left endpoint such that \( f_i \) crosses all lines \( x \) with \( \text{left}(g) \leq \text{left}(x) \leq i \) and \( \text{right}(h) \leq \text{right}(x) \leq \text{right}(e_i) \). The line \( \text{lex} \) is the line of maximum right endpoint under the constant that \( l_s < \text{left}(\text{lex}) < l_m \text{ and } \text{right}(\text{lex}) > r_m \). Similarly, \( \text{rex} \) is the line of maximum left endpoint such that \( r_s < \text{right}(\text{rex}) < r_m \text{ and } \text{left}(\text{rex}) > l_m \).

**Lemma 3.12.** The following formula holds:
TD$(l_m, r_m, l_s, r_s, \text{type})$

\[
= \begin{cases} 
0 & \text{if neither } g \text{ nor } h \text{ exists}, \\
1 + \text{TD}(l_m, \text{left}(\text{lex}), \text{left}(\text{lex}), r_m, r) & \text{if } g, h \text{ and } \text{lex exist and type } \in \{2, l\}, \\
\infty & \text{otherwise}, \\
1 + \text{TD}(\text{left}(\text{rex}), r_m, l_m, \text{right}(\text{rex}), l) & \text{if } g, h \text{ and rex exist and type } \in \{2, r\}, \\
\infty & \text{otherwise}, \\
\text{MIN}[2 + \text{TD}(\text{left}(f_i), \text{right}(e_i), i, \text{right}(f_i), 2)], & i = \text{left}(g), \ldots, \text{left}(h), \text{where right}(e_i) \geq \text{right}(h), \\
\infty & \text{otherwise,}
\end{cases}
\]

**Proof.** In any case, if neither $g$ nor $h$ exists then all lines are dominated by $TD_p$ and thus no additional lines are required. Otherwise, the two possibilities are to extend the path of $TD_p$ whose lines have maximum endpoints in $D$ or to start a new path.

*Case (i):* type = $\Phi$. This corresponds to $TD_p = \emptyset$. We must have $l_m = r_m = l_s = r_s = 0$ in this case, for otherwise, these endpoints would indicate an impossible configuration. If $g$ and $h$ exist and $g = h$ then the top line in $D$ corresponds to an isolated vertex in the graph and no total dominating set exists. We indicate this by assigning the artificial value $\infty$. If $g$ and $h$ exist and $g \neq h$ then $g$ must be the top left line of $D$ and $h$ is the top right line. By Lemma 3.10 and 3.11, the formula is correct. There is no possibility of extending an existing path, here, since no paths exist in $TD_p$.

*Case (ii):* type = $2$. Here we have two choices: we can extend the lowest path of $TD_p$ or start a new component. We will choose the minimum result of these two approaches.

Examination of the permutation diagram for $P_2$ shows that there are two different ways to extend such a path. We can add a line $x$ with $l_x < \text{left}(x) < l_m$ and $\text{right}(x) > r_m$, thereby extending on the left, or we can add a line $y$ with $r_y < \text{right}(y) < r_m$ and $\text{left}(y) > l_m$, which is called extending on the right. Of course, we can only extend on the left (right) if a line such as $x$ ($y$) exists. If such a line does exist then choosing $x$ ($y$) to be the line which reaches furthest down on the right (left) in the diagram guarantees that as many lines as possible are dominated, and that any total dominating set which is eventually found will have the minimum possible cardinality. The fact that the line which reaches furthest down in the diagram can be in a minimum cardinality collection of dominating non-trivial paths follows from arguments similar to those of the proof of Lemma
3.11. Thus, the correct value for $TD(l_m, r_m, l, r, 2)$ in this case is one plus the minimum number of lines needed to complete $TD_p \cup \{x\}$ or $TD_p \cup \{y\}$.

Whether or not $g = h$ has no bearing on path extension. However, suppose $g = h$ and we wish to start a new path. This means that all lines with left < left$(g)$ and all lines with right < right$(g)$ are dominated by $TD_p$. Thus, the vertex corresponding to $g = h$ is an isolated vertex in the graph, in which case no total dominating set exists, or else every line which crosses $g = h$ also crosses a line of $TD_p$ in which case we are forced to extend the path rather than start a new one.

However, if $g \neq h$, then we may begin a new path. The new path is precisely the best first path in a mcts for the diagram $D'$ consisting of all lines $x$ of $D$ having left$(x) \geq$ left$(g)$ and right$(x) \geq$ right$(h)$. Thus, by Lemma 3.10, the possible first lines in the path are those lines $x'$ with left$(g) \leq$ left$(x') \leq$ left$(h)$ and right$(x') \geq$ right$(h)$. For a particular choice of first line $e'$ in this path, a unique best second line $f'$ can be calculated as that line which dominates all lines of $D'$ with both endpoints less than or equal to the endpoints of $e'$, and which reaches furthest down on the left. By Lemma 3.11, we know that $f'$ is the best choice here. Thus, we take the minimum over all possible choices of $e'$, of 2 plus the minimum number of lines required to complete $TD_p \cup \{e', f'\}$.

Case (iii): $type = l$ or $type = r$. This case is similar to Case (ii); the major difference is that paths of length three or more can only be extended on the left or on the right. □

From the formula of Lemma 3.12, it appears that an exponential amount of work may be required to calculate $TD(0, 0, 0, 0, \Phi)$, the size of a mcts for a permutation diagram $D$. This is because there will be potentially $O(n)$ applications of the formula, and within each application, there are at least two choices which must be considered. This rough analysis seems to indicate an $O(2^n)$ algorithm. However, using this top-down approach, we are likely to be calculating the same factor many times. This is precisely the type of situation in which dynamic programming can be used to great advantage. By performing the calculations in a bottom-up manner, we avoid all such duplication and arrive at the following polynomial algorithm.

**Algorithm 3.3.** Total Domination (of permutation graphs).

*Input*: $D$, a permutation diagram representing graph $G$.

*Output*: $TD(0, 0, 0, 0, \Phi)$, the size of a mcts for $G$.

```plaintext
for $l_m = n, \ldots, 0$:
    for $r_m = n, \ldots, 0$:
        for $l_s = l_m - 1, \ldots, 0$:
            for $r_s = r_m - 1, \ldots, 0$:
                for type = $\Phi, 2, l, r$:
```
if $l_m, r_m, l_s, r_s,$ and type are consistent and reflect a possible $TD_p$ then
   Calculate $g, h, lex, rex.$
   Calculate $e_i, f_i$ for $i = \text{left}(g), \ldots, \text{left}(h).$
   Calculate $TD(l_m, r_m, l_s, r_s, \text{type})$ using the formula of Lemma 3.12.
end

The correctness of Algorithm 3.3 follows from Lemma 3.12.

**Theorem 3.13.** A mctds for a permutation graph can be calculated in polynomial time.

**Proof.** Notice that the $l_m$ and $r_m$ indices move from the bottom to the top of the permutation diagram. Thus, every time that a TD value $v$ is used in the calculation of another TD value, we know that $v$ has already been calculated and can be directly accessed in constant time. All steps within the five loops can certainly be performed in polynomial time, and these steps will be executed at most $O(n^4)$ times. Thus the algorithm is polynomial. □

We have made no attempt to produce an efficient algorithm; instead, simplicity was our concern. We note, however, that this algorithm can be improved by the following technique. Instead of generating all possible $(l_m, r_m, l_s, r_s, \text{type})$-tuples, and testing each one for consistency, we can actually calculate all consistent and valid tuples from the indices $l_m$ and $r_m.$ We find that the valid tuples for $l_m$ and $r_m$ are the following, where $i$ is the line with left($i$) = $l_m$ and $j$ is the line with right($j$) = $r_m$:

- $(l_m, r_m, 0, 0, \emptyset)$ if $l_m = r_m = 0,$
- $(l_m, r_m, \text{left}(j), \text{right}(i), 2)$ if left($j$) < $l_m$ and right($i$) < $r_m,$
- $(l_m, r_m, k, \text{right}(i), 1)$ if left($j$) < $l_m$ and right($i$) < $r_m,$ where $k = \text{left}(j), \ldots, l_m,$
- $(l_m, r_m, \text{left}(j), k, r)$ if left($j$) < $l_m$ and right($i$) < $r_m,$ where $k = \text{right}(i), \ldots, r_m.$

The pairs of lines corresponding to $l_m, r_m$ pairs with left($j$) < $l_m$ and right($i$) < $r_m$ are exactly the pairs of crossing lines. Each such pair corresponds to exactly one valid $l_m, r_m$ pair and to exactly one edge in the graph. And, as can be seen from above, each such pair has $O(n)$ tuples associated with it. Thus, this improvement leads to an $O(\max\{ne, n^2\})$ algorithm, where $e = |E|.$
4. Glossary

**Bipartite graph:** A graph such that the vertices can be partitioned into two independent sets.

**Chordal graph:** A graph in which every cycle of length four or more has a chord.

**Cograph:** A graph which can be constructed from single vertices using only the operations of complement and union.

**Comparability graph:** A graph for which there exists a transitive orientation.

**CUB:** A graph which can be constructed from single vertices using only the operations of complement, union and bonding, where bonding two graphs \( G_1 \) and \( G_2 \) is accomplished by identifying a clique of \( G_1 \) with a clique of \( G_2 \), where the two cliques are the same size.

**Directed path graph:** The intersection graph of directed paths in a directed tree.

**Interval graph:** The intersection graph of intervals on a line.

**k-CUB:** A CUB in which all bonding operations are required to identify cliques of cardinality less than or equal to \( k \).

**k-tree:** A graph which can be constructed from a \( k \)-clique by repeatedly adding a vertex adjacent to some \( k \)-clique.

**Meyniel graph:** A graph in which every odd cycle of length five or more has at least two chords.

**Parity graph:** A graph with the property that, for every pair of vertices \( u, v \), all of the minimal paths joining \( u \) and \( v \) have the same parity.

**Perfect graph:** A graph in which every induced subgraph has the property that the maximum clique size is equal to the chromatic number.

**Perfectly orderable graph:** A graph for which there exists a linear ordering of the vertices such that, for every induced subgraph with the same relative vertex ordering, the Grundy number equals the chromatic number. A Grundy numbering is obtained by scanning the vertices in order and assigning to each vertex the smallest positive integer which is not already assigned to one of its neighbours. The Grundy number of the graph is the largest integer so assigned.

**Permutation graph:** A graph for which there is a labelling \( \{v_1, v_2, \ldots, v_n\} \) of the vertices and a permutation \( \pi \) of \( \{1, 2, \ldots, n\} \) such that \((i-j)(\pi^{-1}(i) - \pi^{-1}(j)) < 0\) if and only if \((v_i, v_j)\) is an edge, where \( \pi^{-1}(i) \) can be read as ‘the position in \( \pi \) where \( i \) appears’.

**Split graph:** A graph in which the vertices can be partitioned into a clique and an independent set.

**Strongly chordal graph:** A graph \( G(V, E) \) for which there exists an ordering \( \{v_1, v_2, \ldots, v_n\} \) of \( V \) satisfying the following two conditions for all \( i, j, k, l \):

- if \( i < j < k \) and \((v_i, v_j), (v_i, v_k) \in E\), then \((v_j, v_k) \in E\);
- if \( i < j < k < l \) and \((v_i, v_k), (v_i, v_l), (v_j, v_k) \in E\), then \((v_j, v_l) \in E\).

**Strongly perfect graph:** A graph for which every induced subgraph \( H \) contains an independent set of vertices which intersects every maximal clique of \( H \).
**Undirected path graph:** The intersection graph of a set of undirected paths in a tree.

**Weakly chordal graph:** A graph with the property that neither the graph nor its complement contains an induced chordless cycle with five or more vertices.

**Acknowledgement**

The authors gratefully acknowledge financial assistance from the Natural Sciences and Engineering Research Council of Canada.

**References**


