The Dominating Set Problem on Shiftable Interval Graphs

Federico Malucelli
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Via Ponzio 34/5
20133 Milano, Italy
malucell@elet.polimi.it

Sara Nicoloso
IASI–CNR
Viale Manzoni, 30
00185 Roma, Italy
nicoloso@iasi.rm.cnr.it

Paola Bonfiglio
Università di Roma
"La Sapienza"
Dipartimento di Informatica e Sistemistica

Abstract: In this paper the problem of computing a minimum dominating set of a Shiftable Interval Graph (SIG) is studied. SIG's can be considered a generalization of interval graphs, in the sense that a SIG identifies a whole family of interval graphs. An optimization problem defined on a SIG consists in determining an interval graph of the family which optimizes the value of the chosen measure. In this paper, in particular, we studied the problem of minimizing the cardinality $d(S)$ of a dominating set. The problem is formally stated, its strong NP-completeness is proved, and upper and lower bounds for $d(S)$ are discussed. Special cases solvable at optimality are characterized. Several algorithms are proposed to solve the problem on arbitrary SIG's, and are tested on 220 randomly chosen problem and on five ad hoc designed examples which emphasize the differences among the proposed algorithms.

1. Introduction

In this paper the problem of computing the dominating set of a Shiftable Interval Graph (SIG, for short) is studied. A SIG S is a set of n triples $t_i = <l_i,r_i,\lambda_i>$ of non-negative integer numbers satisfying $0 < \lambda_i \le r_i - l_i$, i.e. $S = \{t_i = <l_i,r_i,\lambda_i> \in Z^3_+ : 0 < \lambda_i \le r_i - l_i$, for $i = 1, \ldots, n\}$. The pair $[l_i,r_i]$ will be called *window* w_i, and λ_i will be called the *length of the interval i* associated with window w_i. In fact, it is easy to think of a SIG as a set of intervals each of which is to be placed within the corresponding window, i.e. such that the left (right, respectively) endpoint of the ith interval does not lay on the left of l_i (on the right of r_i, respectively). The exact position of each interval within its window is easily described by means of a *placement vector* $\varphi = [\varphi_1, \varphi_2, \ldots, \varphi_n]$, whose jth component represents the distance between the left endpoint of the jth interval and the left endpoint of the corresponding window w_j. φ is *feasible* if $0 \le \varphi_j \le r_j - l_j - \lambda_j$ for all j. Thus, once φ_j has been fixed to assume value $\bar{\varphi}_j$, the coordinate of the left and right endpoints of the jth interval are given by $l_j + \bar{\varphi}_j$ and $l_j + \bar{\varphi}_j + \lambda_j$, respectively. Unless stated otherwise, we shall limit ourselves to feasible φ's only.

For the sake of shortness, in what follows, the pair $(t_i,\bar{\varphi}_i)$ will represent the interval
$[l_i + \bar{\varphi}_i, l_i + \bar{\varphi}_i + \lambda_i]$. By *interval model* $M(\bar{\varphi})$ we shall indicate the set $M(\bar{\varphi}) = \{(t_1,\bar{\varphi}_1), (t_2,\bar{\varphi}_2), \ldots,(t_n,\bar{\varphi}_n)\}$, which is, in fact, a set of intervals of the real line.

The intersection graph $G(\varphi)$ of the intervals in $M(\varphi)$ is, clearly, an interval graph, thus a perfect graph. The set of all interval graphs $G(\varphi)$ obtained by varying φ in all possible (feasible) ways is called the *family* F_S associated with the given SIG S. Notice that different values of φ, hence different interval models, may give rise to identical interval graphs.

A minimization (maximization, respectively) problem on a SIG S is defined as follows:

Given:	a SIG $S = \{t_i = <l_i,r_i,\lambda_i> \in \mathrm{Z}^3_+ : r_i - l_i \geq \lambda_i > 0$, and $i = 1, \ldots, n\}$ and a function f: $\mathrm{F}_S \rightarrow \mathrm{Z}_+$,
Find:	a graph $G(\varphi) \in \mathrm{F}_{S,}$
Such That:	$f(G(\varphi))$ is minimum (maximum, resp.).

In other words, an optimization problem on a SIG S consists in individuating a placement φ such that $f(G(\varphi))$ attains its optimum value on the corresponding interval graph $G(\varphi) \in \mathrm{F}_S$.

If f is defined as a max–type function itself, a minimization problem on a SIG S turns out to be a min–max problem. By similar reasonings we obtain min–min, max–min, and max–max problems. This happens, for example, when f is defined as the clique number or the stability number of G. In what follows, we shall refer to these objective functions as $\omega(S)$ and $\alpha(S)$, respectively.

SIG's have been introduced very recently. In [10] the close relation among SIG's and the Scheduling problems are discussed. The state of the art of optimization problems like min $\omega(S)$, max $\omega(S)$, min $\alpha(S)$, max α (S) is broadly dealt with.

In this paper we study the minimization problem obtained defining f as the size $|D_{G(\varphi)}|$ of a minimum dominating set $D_{G(\varphi)}$ of $G(\varphi) \in \mathrm{F}_S$. In other words, we want to find a placement vector φ^* such that $G(\varphi^*)$ verifies $|D_{G(\varphi^*)}| = \min_{G \in \mathrm{F}_S} \{|D_{G(\varphi)}|\}$.

Solving the minimum dominating set problem is an easy task for interval graphs in the unweighted case (i.e. minimum cardinality) as well as in the weighted case (i.e. minimum node weight sum). It takes O(n+m) time, where n and m are the number of nodes and edges of the given interval graph, respectively [1,5]. The time complexity of finding the minimum independent dominating set, the minimum connected dominating set, and the minimum total dominating set takes O($n+m$) time as well, both in node-weighted or unweighted interval graphs [1,4,9,12,13].

The paper is organized as follows: in Section 2 some preliminary definitions and observations are dealt with; in Section 3 the problem is formally stated; in Section 4 its computational complexity is studied. Section 5 is devoted to study lower and upper bounds on the objective function value. The remaining of the paper deals with algorithms and special cases. In particular, Section 6 characterizes special cases which can be solved at optimality by a simple greedy algorithm, in Section 7 several algorithms for the general case are proposed, and in Section 8 the experimental results obtained by running such algorithms on 220 randomly generated problems are reported.

In the remaining of the paper we shall indifferently use the graph theoretic terminology and the terminology related to the geometric aspect of the problem.

2. Preliminaries

This Section is devoted to discuss into details the relationship between the given SIG S, the graphs of the family F_S, the interval models which can arise, and the *derived* SIG which we will be introduce in a while.

It is convenient to define the intersection graph $H = (V,E_H)$ of the set of windows $[l_i,r_i]$ for all $t_i \in S$: the nodes of V are in one–to–one correspondence with the windows of the given SIG, and an egde connects two nodes u,v iff the windows of the corresponding triples do have non–empty intersection. Clearly, H is an interval graph [8]. To this regard, it is important to recall the fundamental consecutive clique arrangement property of interval graphs. Throughout the paper, *clique* will denote a set of nodes inducing a complete subgraph, maximal w.r.t. node addition:

Theorem 1: [7] A graph G is an interval graph if and only if the cliques of G can be linearly ordered such that for every vertex x the cliques containing x occur consecutively.

We now observe the following:

Observation 2: Any (interval) graph $G = (V,E_G) \in F_S$ is a partial subgraph of $H = (V,E_H)$, in the sense that $E_G \subseteq E_H$.

The following Figure shows that not all partial subgraphs of H are interval graphs, nor that any partial interval subgraph of H belongs to F_S, being $S = \{t_1=<1,7,4>, t_2=<3,13,3>, t_3=<2,7,1>, t_4=<4,6,1>, t_5=<8,12,2>\}$.

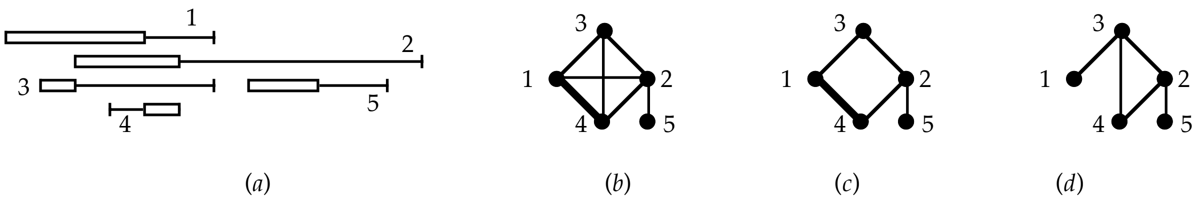


Fig. 1 — (*a*) a SIG and a feasible placement; (*b*) the graph H; (*c*) a partial subgraph of H which is not an interval graph; (*d*) a partial subgraph of H which is an interval graph but does not belong to F_S.

It is worth defining what a *strong* or a *weak* edge of H is.

Definition 3: An edge $(u,v) \in E_H$ is *strong* iff $(u,v) \in E_G$ for any $G \in F_S$, (thick in Figure 1) .

Definition 4: An edge $(u,v) \in E_H$ is *weak* iff there exists at least one graph $G \in F_S$ such that $(u,v) \notin E_G$.

Obviously, given any (weak) edge $(u,v) \in E_H$ there always exists at least one graph $G \in F_S$ such that $(u,v) \in E_G$.

From the above observations it follows that the family F_S has a finite cardinality, in fact $|F_S| \leq 2^{|E_H|} < 2^{n^2}$. This is a peculiar property of SIG's, for they allow to map an infinite number of interval models into the finite set F_S. In fact, there are ∞^n interval models, as each φ_i may assume one out of an infinite number of values. A different upper bound on $|F_S|$ can be proposed, which is $|F_S| \leq \prod_{i=1}^{n} (r_i - \lambda_i - l_i + 1)$. Throughout the rest of the paper, w.l.o.g., we shall always limit ourselves to integer φ: in fact, the family of the intersection graphs of all interval models $M(\varphi)$ for all possible $\varphi \in Z_+^n$ coincides with F_S, as l_i, r_i, and λ_i are integers.

We now introduce some definitions:

Definition 5: A SIG is called *degenerate* if all edges of H are strong.

Notice that in this case one necessarily has $|F_S| = 1$ and (the unique) $G \in F_S$ is isomorphic to H. In other words, in a degenerate SIG it does not matter where an interval i is placed within its window w_i, because it always intersects all the intervals whose window intersects window w_i.

Here are some examples of degenerate SIG's:

(1) – when $\lambda_i = r_i - l_i$ for all $t_i \in S$;

(2) – when given any two triples $t_u, t_v \in S$ one has that

(*i*) $l_u \le l_v \le r_u \le r_v$ implies $l_u + \lambda_u \ge r_v - \lambda_v$

(*ii*) $l_u \le l_v < r_v \le r_u$ implies both $l_u + \lambda_u \ge r_v - \lambda_v$ and $l_v + \lambda_v \ge r_u - \lambda_u$;

(3) – when $\lambda_i \ge r_i - l_i - k$, for any $t_i \in S$, for a fixed $k \ge 0$, and for any two triples $t_u, t_v \in S$ one has that:

(*i*) $l_u \le l_v \le r_u \le r_v$ implies $r_u - l_v \ge 2k$

(*ii*) $l_u \le l_v < r_v \le r_u$ implies both $r_u - l_v \ge 2k$ and $r_v - l_u \ge 2k$.

Proper interval graphs, also known as unit interval graphs [8], are the intersection graphs of a set of intervals none of which properly contains another one (a window w_i properly contains another window w_j if $l_i \le l_j$ and $r_j \le r_i$, and at least one of the two expressions holds as a strict inequality). Unit interval graphs are also characterized by the absence of $K_{1,3}$ as induced subgraph.

Definition 6: A SIG S is *proper* iff H is a proper interval graph.

Definition 7: A SIG S is a *(0,1)–SIG* iff $\lambda_i \ge r_i - l_i - 1$, for any $i = 1, \ldots, n$.

Notice that in a (0,1)–SIG every interval can assume either one of at most 2 positions, that is $\varphi_i \in \{0,1\}$ for all $i = 1, \ldots, n$. In this case most edges are strong, and in particular all edges are strong, that is, the (0,1)–SIG is degenerate, if every pair i,j of mutually intersecting windows satisfies $\min\{r_i, r_j\} - \max\{l_i, l_j\} \ge 2$.

A certain subSIG of the given SIG S plays a very important role in many situations. We define it as follows:

Definition 8: The *derived* SIG S_d associated to the given SIG S is obtained by removing from S all triples whose window properly contains another window of S.

Notice that, clearly, a derived SIG is proper. Moreover, the intersection graph H_d of the set of windows in S_d is an <u>induced</u> subgraph of H. Notice also that of course, $S_d = S$ iff S is proper, and that every graph $G \in \mathrm{F}_{S_d}$ is a <u>partial</u> subgraph of H.

3. Problem definition

In this Section we shall give a formal definition of the DOMINATING SET problem ON SIG's (DSS, for short).

Definition 9: Given an arbitrary graph $F = (V_F, E_F)$ a subset of nodes $D_F \subseteq V_F$ is a *dominating set* iff for any $u \in V_F \setminus D_F$ there exists a $v \in D_F$ such that edge $(u,v) \in E_F$,

DOMINATING SET problem ON SIG's:

Given: a SIG $S = \{t_i = <l_i, r_i, \lambda_i> \in Z_+^3 : r_i - l_i \geq \lambda_i > 0,$ and $i = 1,\ldots,n\}$

Find: placement $\overline{\varphi}$

Such That: a minimum dominating set $D_{G(\overline{\varphi})}$ on the interval graph $G(\overline{\varphi})$ satisfies $|D_{G(\overline{\varphi})}| = \min_{F \in \mathrm{F}_S} \{|D_F|\}$.

In what follows, $D(S)$ and $d(S) = |D(S)|$ will denote a minimum dominating set for the given SIG S, and its cardinality, respectively.

4. Computational Complexity

In this Section we shall prove the NP-completeness of DSS. For this reason, throughout the present Section we shall only refer to the decisional version of the problem, whose formal statement is obtained by replacing into the problem definition the requirement "$|D_{G(\overline{\varphi})}| = \min_{F \in \mathrm{F}_S} \{|D_F|\}$" by "$|D_{G(\overline{\varphi})}| \leq \delta$", where $\delta \geq 1$ is a given integer.

Theorem 10: Problem DSS is NP–complete in the strong sense.

Proof: DSS is immediately seen to be in NP. The reduction is from 3–PARTITION, which is NP–complete in the strong sense [6]. The formal statement of 3–PARTITION is the following [6]: given a nonnegative integer B, and a finite set $A = \{a_1, a_2, \ldots, a_{3m}\}$ of $3m$ integers such that $\frac{B}{4} < a_i < \frac{B}{2}$, for $i = 1, \ldots, 3m$, and such that $\sum_{a_i \in A} a_i = mB$, find a partition of A into (exactly) m disjoint sets $A_1, A_2, \ldots, A_m$ such that $\sum_{a_i \in A_j} a_i = B$ for all $j = 1, \ldots, m$ (in order to avoid some trivial cases assume $B \geq 6$). Notice that by the preceding hypothesis it follows that every set has exactly 3 elements. From a given instance of 3–PARTITION construct the following instance of DSS. δ is set to $3m$ and the triple set S is the union of two sets S_1 and S_2, where $S_1 = \{t_i = (0, m(B+7)–1, a_i)$: for $i = 1, \ldots, 3m\}$ and $S_2 = \{t_j = (j, j+1, 1)$: for $j = 0, \ldots, m(B+7)–2$, with $j \neq k(B+7)–1$ for $k = 1, \ldots, m–1\}$. In what follows, the windows and intervals of triples in S_1 will be called *large*, and the windows and intervals of triples in S_2 will be called *small*. In other words, S_2 consists of m sequences of $B+6$ unit windows with corresponding unit length intervals, each sequence being separated by the following one by a unit space (*jump*), while S_1 consists of $3m$ large windows within each of which an interval of length a_i is to be placed. Notice that each large window properly contains all the small windows, and the only feasible placement for all small intervals s is $\varphi_s = 0$. We claim that a feasible solution for DSS exists, that is a dominating set for the given SIG S of cardinality not larger than $\delta = 3m$, if and only if a partition with the required properties exists for 3–PARTITION. The IF part follows immediately by observing that a dominating set with cardinality $\delta = 3m$ is easily obtained from a (feasible) solution $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, with $j = 1, \ldots, m$, for 3–PARTITION, by setting $\varphi_s = 0$ for all small intervals, and by setting $\varphi_{j_1} = (j-1)(B+7)+1$, $\varphi_{j_2} = (j-1)(B+7)+a_{j_1}+3$, $\varphi_{j_3} = (j-1)(B+7)+a_{j_1}+a_{j_2}+5$ for all $j = 1, \ldots, m$. Let us prove the ONLY IF part. We shall say that the ith large interval is *nicely placed* if φ_i is such that no jump is contained in $[\varphi_i, \varphi_i+a_i]$ and there are two small windows w_s, w_t such that $r_s = \varphi_i$, and $l_t = \varphi_i+a_i$. It is easy to see that the ith large interval may dominate up to a_i+2 small windows, such maximum value being achieved if and only if it is nicely placed. Since the assumed hypothesis $B \geq 6$ implies $a_i > 1$ for all $i = 1, \ldots, 3m$, it is immediate to see that any dominating set of cardinality not larger than $3m$ is made of all large intervals, only, and all of them are nicely placed. More precisely, notice that for the same reason no dominating set for S exists with cardinality strictly less than $3m$. The facts that all large intervals are nicely placed and that $\frac{B}{4} < a_i < \frac{B}{2}$, for $i = 1, \ldots, 3m$, implies that the small intervals of a same sequence are dominated by exactly three large intervals. The corresponding 3–PARTITION is obtained by inserting into the same subset A_j the three intervals which dominate the small windows of the jth sequence. Since the reduction from 3–PARTITION to DSS is pseudopolynomial, the result is proved. o

5. Lower and Upper Bounds

Motivated by the preceding result we here state lower and upper bounds to the cardinality $d(S)$ of a minimum dominating set for the given SIG S.

5.1. Lower Bound

Lemma 11: Given a SIG S one has

$$|D_H| \leq |D_G|, \text{ for any } G \in \mathrm{F}_S.$$

Proof: Let $\mathrm{Adj}_F(K)$ denote the set of nodes adjacent to at least one node of the subset $K \subseteq V_F$ in a graph $F = (V_F, E_F)$. The claimed thesis follows immediately: in fact

$\mathrm{Adj}_G(K) \subseteq \mathrm{Adj}_H(K)$, as $E_G \subseteq E_H$.

o

Clearly, if S is a degenerate SIG, the above relation holds with the equality sign.

From the above Lemma immediately follows that

Theorem 12: Given a SIG S one has

$$|D_H| \leq d(S).$$

5.2. Upper Bound

In this Section we propose an upper bound to $d(S)$.

Lemma 13: Given a SIG S one has

$$|D_G| \leq \alpha(H), \text{ for any } G \in \mathrm{F}_S.$$

Proof: Let $K = \{K_1, K_2, \ldots, K_{|K|}\}$ be a minimum cardinality covering by cliques of the node set V of interval graph H (i.e., $|K| = k(H)$, where $k(H)$ is the clique cover number of H). W.l.o.g. assume that a consecutive clique arrangement $<K_1, K_2, \ldots, K_{|K|}>$ is given, and let $\pi(K_i)$ be the rightmost coordinate such that $\{w_t : l_t \leq \pi(K_i) \leq r_t\} = K_i$. Take any two consecutive cliques K_i, K_{i+1}. By the maximality of each single clique it follows that there exists (at least) a node v_{j_i} belonging to K_i but not belonging to K_{i+1}. Consider the j_i^{th} interval and place it in such a way that it contains coordinate $\pi(K_i)$, that is choose φ_{j_i} such that $l_{j_i} + \varphi_{j_i} \leq \pi(K_i) \leq l_{j_i} + \varphi_{j_i} + \lambda_{j_i}$, for $i = 1, \ldots, |K|$. The set J of all intervals j_i's for $i = 1, \ldots, |K|$ is clearly a dominating set for some $G \in \mathrm{F}_S$. Since $|D_G| \leq |J| = k(H)$, and $k(H) = \alpha(H)$, as H is a perfect graph, the claimed thesis follows. o

From the above Lemma we immediately derive the following:

Theorem 14 : Given a SIG S one has

$$d(S) \leq \alpha(H).$$

A different upper bound can be obtained by relating the cardinality of a minimum dominating set of S to the cardinality of a minimum dominating set of the derived SIG S_d.

Theorem 15: Given a SIG S one has

$$d(S) \leq d(S_d).$$

Proof: Consider a triple $t_u \in S \setminus S_d$. By definition of derived SIG S_d, there exists (at least) a triple $t_v \in S_d$ whose window is properly contained into window u. If $(t_v, \varphi_v) \in D(S_d)$, for some φ_v, we are done. If $(t_v, \varphi_v) \notin D(S_d)$, consider a $(t_z, \varphi_z) \in D(S_d)$ which dominates t_v, that is either $l_v \leq l_z+\varphi_z \leq r_v$ or $l_v \leq l_z+\varphi_z+\lambda_z \leq r_v$ or both. Since $l_u \leq l_v$ and $r_v \leq r_u$, the triple t_z clearly dominates also t_u. o

The importance of this Theorem lays in the fact that problem DSS can be solved at optimality on S_d, in fact it is a *proper* SIG (see next Section).

It can also be proved that

Theorem 16: Given a SIG S one has

$$d(S_d) \leq \alpha(H).$$

In order to prove the theorem above we need the following

Lemma 17: Given a SIG S one has

$$\alpha(H_d) = \alpha(H).$$

Proof: Let A be an independent set of H with maximum cardinality, that is $|A| = \alpha(H)$. Consider $A \subseteq V(H_d)$. Then A is an independent set for H_d, and of course $\alpha(H_d) = \alpha(H)$, as H_d is an induced subgraph of H. Consider $A \nsubseteq V(H_d)$ and be $x \in A \setminus V(H_d)$. Let Adj(v) denote the set of vertices adjacent to a vertex v in H and v itself. We claim that among the vertices adjacent to x in H there exists one, call it y, such that $y \in V(H_d)$ and Adj(y) $\subseteq$ Adj(x). This follows from the fact that there exists a window w_y properly contained into window w_x. Thus $A \setminus \{x\} \cup \{y\}$ keeps being a maximum cardinality independent set for H. o

Proof of Theorem 16: By Theorem 14, $d(S_d) \leq \alpha(H_d)$. By Lemma 17, $\alpha(H_d) = \alpha(H)$, and the claimed thesis follows. o

6. Special cases solvable at optimality

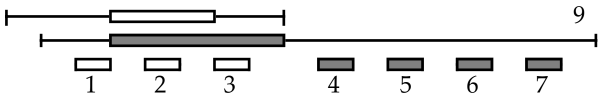
This Section is devoted to study special cases solvable at optimality by a simple greedy approach. We shall first describe Algorithm G and then characterize the special classes of SIG's on which it finds the optimal solution. For the sake of simplicity we shall describe the algorithm on the interval model of the windows, each time suitably positioning the corresponding interval (in the algorithm $t_i \in S \setminus D$ is an improper writing for $t_i \in S \setminus \{t_j: (t_j, \varphi_j) \in D\}$).

```
ALGORITHM G:
Input: a SIG S= $\{t_i = <l_i,r_i,\lambda_i> \in Z_+^3 : r_i - l_i \geq \lambda_i > 0$, and $i = 1,\ldots,n\}$;
Output: a feasible placement $\varphi$ and a subset $D$;
Initially all the windows are unmarked and the set $D$ is empty.
Repeat
    Consider the leftmost right endpoint $\pi$ of an unmarked window of $S \setminus D$ ;
    Let $A = \{i: l_i \leq \pi \leq r_i, i \notin D\}$;
    Set $\varphi_i = \min\{r_i - l_i - \lambda_i, \pi - l_i\}$ for all $i \in A$;

    SELECTION PHASE:
        Let $j \in A$ be such that $l_j + \varphi_j + \lambda_j = \max\{l_i + \varphi_i + \lambda_i$, for $i \in A\}$;

    Insert $j$ into $D$;
    Mark all windows $w_i$ verifying $l_i \leq l_j + \varphi_j + \lambda_j$ ;
Until all windows are marked.
```

The algorithm produces a feasible placement φ and a subset D of indices. The subset D is a dominating set for the interval graph $G(\varphi)$. As for the algorithm behaviour we notice what follows: (i) by setting $\varphi_i = \min\{r_i - l_i - \lambda_i, \pi - l_i\}$ for all $i \in A$ we are actually placing each interval in the rightmost position within its window, so as to make it cross π; (ii) the φ_j are usually set more than once until either j is inserted into D or it does not belong to the current A anymore; (iii) at each iteration we insert into the current set D the index of an interval with rightmost right endpoint.

The computational complexity of Algorithm G amounts to $O(n^2)$. In fact, in the worst case the algorithm considers $O(n)$ different subsets A, whose cardinality is bounded by n, and the choice of j takes globally $O(n^2)$ throughout the whole execution of the algorithm.

Algorithm G cannot guarantee to determine the optimal solution on arbitrary SIG's since the problem is NP-Hard (Theorem 10), as it is shown in particular by the example of Figure 2 where the solution output by Algorithm G (above) on the SIG $S = \{t_1$=<3,4,1>, t_2=<5,6,1>, t_3=<7,8,1>, t_4=<10,11,1>, t_5=<12,13,1>, t_6=<14,15,1>, t_7=<16,17,1>, t_8=<1,9,3>, t_9=<2,18,5>$\}$ and an optimum solution (below) on the same SIG are drawn. The thin rectangles represent the intervals, and in particular the grey ones are those in the dominating set.



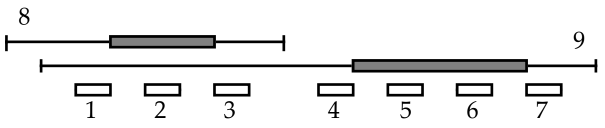


Figure 2 – Solution output by Algorithm G (above) and an optimum solution (below).

However there are special cases solved at optimality by Algorithm G, as illustrated in the following of the present section.

Definition 18: A SIG is *good* if throughout the execution of Algorithm G the current π is greater than the right endpoint r_j of the window corresponding to the last index inserted into D.

Theorem 19: Let S be a good SIG. Then, the placement vector φ output by Algorithm G is an optimal solution of DSS.

Proof: By contradiction. Assume that there exists a placement φ^* such that a minimum dominating set $D(\varphi^*)$ on $G(\varphi^*)$ has cardinality smaller than $|D|$. Let K, K^* denote the set of intervals corresponding to indices in D, $D(\varphi^*)$ placed according to φ, φ^*, respectively. The proof consists in showing that taken any coordinate τ the number of intervals of K whose right endpoint lays on the left of τ is never smaller than the number of intervals of K^* whose right endpoint lays on the left of the same point τ, the contradiction being found in the fact that $D(\varphi^*)$ is not a dominating set for $G(\varphi^*)$. For the sake of simplicity we shall denote with $l(s)$, $r(s)$ the left and right endpoint of an interval $s \in K \cup K^*$, namely $l(s) = l_s + \varphi_s$ and $r(s) = l_s + \varphi_s + \lambda_s$.
Sort the intervals in $K(K^*$, respectively) by non decreasing right endpoint, resulting in the sequence $x_1, \ldots, x_{|K|}$ ($y_1, \ldots, y_{|K^*|}$). Let also w_a be the window with leftmost right endpoint r_a in the given SIG. Consider x_1 and y_1, it must be the case that $l(x_1)$, $l(y_1) \leq r_a$, otherwise node a would not be dominated contradicting the hypothesis. Because of the algorithm behaviour, it is also the case that $r(y_1) \leq r(x_1)$.
Now consider the next pair x_2, y_2. We shall prove that $r(y_2) \leq r(x_2)$. Infact: after having fixed the position of x_1, the algorithm moves to the leftmost right endpoint r_b of a not-yet dominated window w_b (thus $r(x_1) < l_b$), and sets the placement φ_2 of x_2, resulting in $l(x_2) \leq r_b \leq r(x_2)$. Notice that the algorithm chooses x_2 in the set $C(r_b) = \{i$: such that $l_i \leq r_b \leq r_i\}$, which x_1 does not belong to, as by hypothesis $r_b > r_{x_1}$. As $r(y_1) \leq r(x_1)$, clearly, window w_b is not dominated by y_1. However there must exist another interval in K^* which dominates window w_b. Indeed such interval must be y_2

and one has $l(y_2) \le r_b$. If $r(y_2) \le r_b$, it is also $r(y_2) \le r(x_2)$. If $r(y_2) > r_b$ then $y_2 \in C(r_b)$, and the result follows from the algorithm behaviour.
The reasoning can be repeated, always comparing the pair of intervals x_i, y_i, for $i = 3, \ldots, |K^*|$, concluding that $r(y_{|K^*|}) \le r(x_{|K^*|})$. Consider $x_{|K^*|+1}$. The placement of this interval is set in order to dominate a not-yet dominated window w_C, verifying $r(x_{|K^*|}) < l_c$. This contradicts the hypothesis that K^* is a dominating set, and the claimed thesis follows. o

This Theorem, clearly, does not allow to know if Algorithm G will output an optimal solution before running it. But, of course, it is a sufficient condition to prove the optimality of the output solution. There are special SIG's which can be proved *a–priori* to be good: on these SIG's we know that Algorithm G will output an optimal solution. This happens, for example, for the proper SIG's.

Theorem 20: A proper SIG is good.

Proof: Let i be the last index inserted by the algorithm into the current D, and let j be the index of window with leftmost right endpoint among the not yet marked (i.e. dominated) ones. Then $l_j > l_i + \varphi_i + \lambda_i$. Since, clearly, $l_i + \varphi_i + \lambda_i \ge l_i$, and the SIG is a proper SIG, we may conclude that $r_j > r_i$, and the claimed thesis follows. o

Theorem 21: A (0,1)–SIG S is good.

Proof: Let i denote the last index inserted into the current D, and j be the index of an unmarked window with leftmost right endpoint r_j . One has: $r_j > l_j$, by definition; $l_j \ge l_i + \varphi_i + \lambda_i + 1$, as j is unmarked; $l_i + \varphi_i + \lambda_i + 1 \ge r_i$, as S is a (0,1)–SIG. Since $\pi = r_j$, the thesis follows. o

Theorem 22: A degenerate SIG S is good.

Proof: Let i denote the last index inserted into the current D, and j be the index of an unmarked window with leftmost right endpoint r_j (unmarked w.r.t. the current φ and D). One has: $r_j > l_j$, by definition; $l_j \ge l_i + \varphi_i + \lambda_i + 1$, as j is unmarked; $l_j > r_i$, as S is degenerate (in fact all edges are strong). Since $\pi = r_j$, the thesis follows o
In the last case problem DSS can be solved very easily. In fact it reduces to finding a dominating set on H as any feasible placement φ gives rise to the unique interval graph belonging to $F_S = \{H\}$.

Algorithm G establishes the position of an interval of D within its window in a greedy way, that is, as soon as the chosen interval is the best candidate to solve the subproblem “in the neighbor of π”. And, clearly, this may not be the right strategy to optimize over the global problem.

7. General case

In this Section we shall discuss some algorithms for problem DSS on arbitrary SIG's, which are more sophisticated than the greedy algorithm of Section 6.

7.1. Algorithm MEC

The first algorithm we discuss is the MEC algorithm (MEC=Minimal element of an Equivalence Class). Its algorithmic scheme is quite similar to Algorithm G. It differs from it in two facts only: the choice of coordinate π w.r.t. the current set D, and the choice of which index j inserting into D.

π is chosen as the leftmost right endpoint of an unmarked window among those belonging to the derived SIG S_d, only. In fact a dominated position for any interval u such that $t_u \in S \setminus S_d$ does always exist, whatever is the position of an interval whose window is properly contained into window w_u (see proof of theorem 15). In this sense, u has a “passive” role in the domination. However, an interval u such that $t_u \in S \setminus S_d$ might play an “active” role in the domination, and this is why subset A is constructed upon S, and not upon S_d only.

As for the choice of the element j to insert into D, the following SELECTION PHASE of Algorithm G,

```
SELECTION PHASE:
    Let $j \in A$ be such that $l_j + \varphi_j + \lambda_j = \max\{l_i + \varphi_i + \lambda_j,$ for $i \in A\}$;
```

has to be replaced by the following instructions, where U_i denotes the set of unmarked windows which are dominated by interval i, placed according to φ_i;

```
SELECTION PHASE:
    Let $h \in A$ be such that $l_h + \varphi_h + \lambda_h = \max\{l_i + \varphi_i + \lambda_j,$ for $i \in A\}$;
    Compute $U_h$;
    Let $L = \{i \in A$: $U_i = U_h\}$;
    Let $\Theta = \min\{l_j + \varphi_j + \lambda_i,$ for $i \in L\}$;
    Let $j \in L$ be such that $r_j - l_j = \min\{r_i - l_i,$ for $i \in L$: $l_i + \varphi_j + \lambda_j = \Theta\}$;
```

The MEC algorithm is better than Algorithm G in the fact that it individuates a sort of “equivalence class” for local optimality among whose elements it chooses a “minimal” one to be inserted into D. In fact, the interval selected by Algorithm MEC is one which dominates exactly the same set of windows dominated by the interval selected by Algorithm G and whose window has minimum length. However, the general strategy is still a greedy one, in the sense that an interval is chosen in a subset all of whose elements give rise to a locally optimal solution.

The computational complexity of Algorithm MEC amounts to $O(n^2 \log n)$, which corresponds to the complexity of selecting j for all $O(n)$ different subsets A. In fact, for a given A, the computation of h, Θ, and j takes $O(n)$, as well as the computation of U_h assuming that the left and right endpoints are arranged in a segment tree. {pre/s]. On the other hand the computation of L can be carried out in $O(n \log n)$ time if one observe that $U_i \subseteq U_l$ if $l_i + \varphi_i + \lambda_j \leq l_l + \varphi_l + \lambda_l$. In fact, we first sort the elements $i \in A$ by non increasing value of $l_i + \varphi_i + \lambda_j$, then we determine the last i which verifies $U_i = U_h$ by performing a binary search.

The numerical experiments have been conducted running two different version of the algorithm, whose only difference consists in the direction chosen to scan the problem. In particular, the one we presented scans the problem from left to right, while the other one scans the problem from right to left.

Examples can be constructed where both versions of the algorithm fail. In Figure 3, the solutions output by the two versions of Algorithm MEC when applied to SIG $S = \{t_1=<2,3,1>,\ t_2=<4,5,1>,\ t_3=<6,7,1>,\ t_4=<9,11,2>,\ t_5=<12,13,1>,\ t_6=<14,15,1>,\ t_7=<16,17,1>,\ t_8=<19,20,1>,\ t_9=<23,24,1>,\ t_{10}=<1,25,5>\}$, while an optimum solution is one where the 10th interval is placed so as to dominate t_4, t_5, t_6, t_7, all other triples being dominated by themselves.

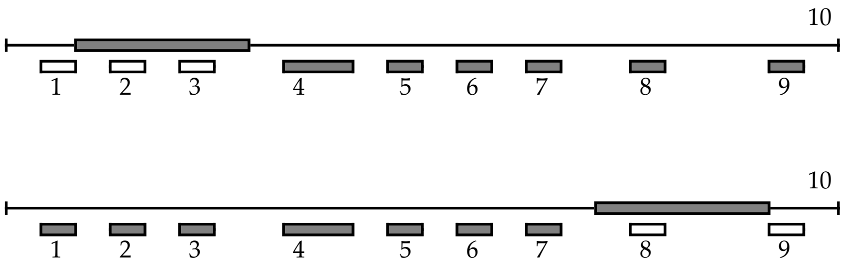


Figure 3 – Solution output by the left to right version of Algorithm MEC (above) and solution output by the right to left version of Algorithm MEC(below).

In order to avoid that an element is inserted into D as soon as it is an optimal local solution, we decided to design algorithms S1-MEC and S2-MEC.

7.2. Algorithm S1_MEC

Algorithm MEC places and inserts an element into the current dominating set as soon as it turns out to be an element yielding a local optimum. However, it may be the case that a better solution is found if the interval is placed on the right of the just computed position. The algorithm discussed in the present section has been designed in order to evaluate the effect of placing an interval in many different positions within its windows. In order to do this, in Algorithm S1_MEC we consider the (linear) problem as a cyclic problem in the sense that we first process the triples whose windows lay on the right or cross a prespecified coordinate τ and then we process the remaining ones, and we go on step by step moving τ more and more rightwards until we reach the right end of the problem. At each step we run Algorithm MEC, keeping track of the best current solution. By doing so, in some sense, we search the solution space for different configurations of a feasible solution.

```
ALGORITHM S1_MEC:

$\tau := 0$;
$D := \{1, \ldots, n\}$;
$\varphi_i := 0$, for $i = 1, \ldots, n$;
Repeat
    $\tau := \min \{r_j : r_j > \tau$ and $t_j \in S_d\}$;
    Let $S' = \{t_i \in S : r_i \geq \tau\}$;
    Run algorithm MEC on $S'$, and let $D'$, $\varphi'$ be the resulting solution;
    If $\{t_i \in S_d : r_i < \tau\}$ is non empty
    Then    Begin
                Mark all the windows dominated by some element in $D'$;
                Let $S'' = \{t_i \in S$ and not marked$\} \cup \{t_i \in S \setminus S_d$ and $i \notin D' : l_i < \tau \leq r_i\}$;
                Run algorithm MEC on $S''$ keeping track of the already marked windows;
                Let $D''$, $\varphi''$ be the resulting solution;
                End
    Else $D'' = \emptyset$;
    If $|D' \cup D''| < |D|$
    Then $D := D' \cup D''$;
Until $\tau = \max \{r_j : t_j \in S_d\}$.
```

SIG's S' and S'' are clearly subSIG's of S, and there are triples which belong to both of them (most of the triples whose window crosses coordinate τ), while S', S'' do induce a partition of the derived SIG S_d. Notice that in the first cycle $S' = S$ and S'' is empty.

The computational complexity of S1_MEC is $O(n^3 \log n)$, as the cycle is repeated at most n times and in each cycle algorithm MEC is applied twice to SIG's whose size is bounded by n.

In this case too, the numerical experiments were conducted running algorithm S1–MEC in the present version, which scans the problem from left to right, and in a mirror–version, which scan the problem in the opposite direction, from right to left.

7.3. Algorithm S2_MEC

During the execution of algorithm S1_MEC, S'' keeps growing at each cycle, but the "starting point" of algorithm MEC is always the left endpoint of the whole problem. In order to further reduce this the effects of the greedy behaviour, we designed algorithm S2–MEC. It simply consists in applying algorithm S1_MEC to the subproblem S'', instead of applying algorithm MEC to it. Clearly, three subsets are derived from S, and still the leftmost one of them RISENTE DEL GREEDY APPROACH. However, the numerical results and the fact that the computational complexity was already large enough, suggested not to further subdivide the problem.

The computational complexity of S2_MEC is $O(n^4 \log n)$, as the cycle is repeated at most n times and in each cycle algorithm S1_MEC is applied to a SIG whose size is bounded by n.

The numerical results reported in the next Section are the results of having applied algorithm S2_MEC both in its left–to–right and in its right–to–left versions.

7.4. Algorithm OLGA

We also designed OLGA (Optimum Local Greedy Algorithm), which is based on a completely different approach to the problem. It starts with a feasible solution and tries to decrease its cardinality until a (local) minimum is reached, maintaining feasibility at each step.

D is initially chosen as the set of all indices whose triple belongs to S_d. This defines a trivial solution: in fact the pair D, φ defines a dominating set of S for any feasible placement φ. For each triple $t_x \in S \setminus S_d$, and $x \notin D$, let $Q_x(\varphi_x) = \{i \in D$: $t_i \in S_d$ and $[l_i, r_i] \cap [l_x+\varphi_x, l_x+\varphi_x+\lambda_x] \neq \varnothing$, and be $q_x(\varphi_x)$ its cardinality. At each iteration a tentative placement φ'_x, is determined so as to maximize $q_x(\varphi_x)$. An interval is selected which attains the maximum over all $q_x(\varphi_x)$, say y, and it is inserted into D in place of $Q_y(\varphi_y)$. The algorithm is formally described below:

```
ALGORITHM OLGA;

Let $D = \{i: t_i \in S_d\}$;
Repeat
    For each triple $t_x \in S \setminus S_d$ and $x \notin D$ do
        Let $q_x = \max\{q_x(\varphi_x)$ for all feasible $(\varphi_x)\}$;
    Let $y$ be such that $q_y = \max\{q_x: t_x \in S \setminus S_d$ and $x \notin D\}$;
    If $q_y > 1$
    Then $D := D \setminus Q_y(\varphi_y) \cup \{y\}$
Until $|\{i: i \in D$ and $t_i \in S_d\}| \geq 2$.
```

Notice that once the index y is inserted into D, it will not be considered any more, which guarantees that the algorithm terminates in $O(n)$ steps. The complexity of each step amounts to $O(n^3 \log n)$ performing a binary search on a segment tree [11].

We also designed another algorithm based on local improvements, which can be applied both to the trivial solution and to the solutions output by the S1_MEC, S2_MEC, and OLGA algorithms, but the computational results were not encouraging [2]. However we noticed that it greatly improved the solution when applied to the trivial solution , while it improved the solution output by OLGA on 4 cases over 200 only, and it could never improve the solution output by S1_MEC or S2_MEC. We could design one example, only, where it outperformed all other algorithms (the second one in Table 7). The execution time of this algorithm is often quite large compared to the other algorithms.

8. Experimental results

We run the algorithms on 220 randomly generated test problems. Each test problem is defined by a 4-uple TEST = (n,d,λ_{max},p), where n is the number of triples of S, $d = \max\{r_i: t_i \in S\} - \min\{l_i: t_i \in S\}$, λ_{max} is an upper bound on the length of an interval, and p is an upper bound on the ratio $(r_i - l_i)/\lambda_{max}$. In particular, in the first step we randomly decide if we shall next generate a left or a right endpoint of a window. If a left (right, respectively) endpoint of a window is to be generated, we randomly choose a value $0 \leq l_i < d$ ($0 < r_i \leq 100$, resp.). Then, the interval length $\lambda_i \leq \min\{\lambda_{max}, d-l_i\}$ ($\lambda_i \leq \min\{\lambda_{max}, r_i\}$, resp.) is randomly generated, and finally the right endpoint r_i (left endpoint l_i,resp.) is randomly generated so as to verify $l_i + \lambda_i \leq r_i \leq \min\{d, l_i + p\lambda_{max}\}$ ($\max\{0, r_i - p\lambda_{max}\} \leq l_i \leq r_i - \lambda_i$,resp.). The 4 parameters of a TEST assumed values in the following sets: $n \in \{20,40\}$, $d \in \{50,100\}$, $\lambda_{max} \in \{5,10\}$, and $p \in \{1.2,2,2.8,5,10\}$. For each fixed 4-uple TEST we generated 10 different problems. The algorithms behaved on the very exact way on all TEST problems. For this reason we decided to show 4 tables, only, among the 20 ones we had at hand.

Tables 1 through 4 show the results obtained on the following problems TEST = (40,100,5,2), TEST = (40,100,5,5), TEST = (20,100,5,2), TEST = (20,100,5,10).

In order to (possibly) emphasize the differences among the algorithms, by observing the structure of 5 problems which were constructed *ad hoc* for this purpose, we decided to test the algorithms on a second group of problems. Each problem of this group is given by the union of two 4-uple TEST, that is, TEST' = $(n,d,\lambda_{max},p) \cup (n',d',\lambda'_{max},p')$. These group of problems is characterized by having triples with small windows and interval lengths and triples with large windows and interval lengths, in a relative sense. We tested the algorithm on 10 problems having TEST' = (35,100,2,1.5) ∪ (5,100,10,5) as characteristic tuple and on 10 problems having TEST' = (20,100,2,1.5) ∪ (20,100,10,5) as characteristic tuple; the result we obtained are listed in Tables 5 and 6.

All the algorithms have been implemented with MATLAB on a PC 486DX 33Mhz.

In the tables below each column (but for the last element) reports the cardinality of the dominating set output by the generic algorithm on each of the 10 instances of the corresponding TEST, or TEST'. These are the listed results:

- column $|S_d|$: cardinality of the derived S_d;
- column LB: lower bound to $d(S)$ (see Theorem 12);
- column UB: upper bound to $d(S)$ (obtained by applying Algorithm G to the derived S_d, see Theorem 15);
- columns G, S1_MEC, S2_MEC, OLGA: cardinality of the solution output by Algorithms G, S1_MEC, S2_MEC, OLGA, respectively;
- column MEC: cardinality of the solution output by algorithm MEC from left to right (left), and by algorithm MEC from right to left (right);

The last element of each column represents the execution time (averaged over the 10 TEST or TEST' instances) of the corresponding algorithm, and reported in seconds. Notice that the average execution time of algorithm MEC when applied from left to right is always smaller than the average execution time of algorithm MEC when applied from right to left. This depends on the implementation of the *min* and *max* functions in MATLAB. Notice that the time required to compute the upper bound UB is exactly the time required to solve DSS at optimality on the derived SIG.

$\|S_d\|$	LB	UB	G	MEC		S1_MEC	S2_MEC	OLGA
23	8	12	9	9	9	9	9	10
28	7	11	9	9	9	9	9	10
25	7	11	8	8	8	8	8	8
24	8	12	9	9	9	9	9	11
24	7	13	10	10	10	10	10	10
21	8	11	8	8	8	8	8	9
19	6	12	7	7	7	7	7	8
22	6	12	9	9	9	9	9	9
22	8	11	9	9	9	9	9	10
26	6	11	8	8	8	8	8	9
time	1.6	1.33	2.99	7.77	8.58	158	1151	261

Table 1 – TEST =(40,100,5,2)

$\|S_d\|$	LB	UB	G	MEC		S1_MEC	S2_MEC	OLGA
18	4	8	7	7	7	7	7	8
17	3	7	6	6	6	6	6	7
21	3	9	8	7	7	7	7	9
14	4	8	6	6	6	6	6	6
14	4	8	7	6	6	6	6	6
17	5	7	7	7	7	7	7	7
15	3	8	6	6	6	6	6	7
18	4	7	6	6	6	6	6	7
13	4	8	6	6	6	6	6	7
14	4	8	7	7	7	7	7	8
time	1.02	0.908	2.395	6.1	6.66	84	457	682

Table 2 – TEST =(40,100,5,5)

$\|S_d\|$	LB	UB	G	MEC		S1_MEC	S2_MEC	OLGA
15	6	9	8	8	8	8	8	8
14	6	7	7	7	7	7	7	7
16	5	5	5	5	5	5	5	5
14	7	8	7	7	7	7	7	8
13	5	7	6	6	6	6	6	7
14	7	9	8	8	8	8	8	8
14	7	8	8	8	8	8	8	8
16	7	8	8	8	8	8	8	8
16	7	7	7	7	7	7	7	7
14	6	8	8	8	8	8	8	9
time	0.92	0.702	1.144	2.77	2.89	42.5	209.5	11.4

Table 3 – TEST =(20,100,5,2)

$\mid S_d \mid$	LB	UB	G	MEC		S1_MEC	S2_MEC	OLGA
11	3	6	4	4	4	4	4	5
8	3	4	4	4	4	4	4	4
9	3	6	5	5	5	5	5	5
8	2	5	5	4	4	4	4	4
7	2	4	4	4	4	4	4	4
10	4	7	6	6	6	6	6	6
9	3	7	6	6	6	6	6	6
11	3	5	5	5	5	5	5	5
11	3	6	6	6	6	6	6	6
9	2	5	3	3	3	3	3	4
time	0.43	0.458	1.049	2.072	2.222	20.4	67.4	76.8

Table 4 – TEST =(20,100,5,10)

From Tables 1 through 4, and from the remaining 16 ones, we derive the following conclusions.

Objective function: Algorithms G, MEC, S1_MEC, and S2_MEC give the very same result, but for three examples where Algorithm G outputs a solution which is one unit larger than the solutions output by the other three algorithms. On all the remaining 197 problems, algorithms G, MEC, S1_MEC, and S2_MEC gave the same solution. The behaviour of OLGA was definitely worse than the behaviour of G, MEC, S1_MEC, and S2_MEC: in fact OLGA output a solution of larger cardinality on 76 problems over 200.

Execution times: The execution times reported in the tables show the general trend on all the 200 problems, and confirm the computational complexities stated in the previous Section. Algorithm G is always the fastest algorithm, followed by MEC which still has reasonable execution times. The running times of both algorithms depend on n and decrease for increasing p. S1_MEC takes 10 to 20 times the time required by MEC, and S2_MEC requires 5 to 10 times the time required by S1_MEC, approximatively. The running times of these two algorithms also depend on n and decrease for increasing p. The execution time of OLGA is quite homogeneous within the set of problems characterized by the same 4-uple TEST.

$\lvert S_d \rvert$	LB	UB	G	MEC		S1_MEC	S2_MEC	OLGA
26	5	14	10	10	11	10	10	11
29	5	12	8	7	7	7	7	8
31	13	17	15	15	15	15	15	15
26	7	17	13	13	13	13	13	13
28	5	16	10	10	11	10	10	10
30	9	18	14	14	14	14	14	15
32	4	16	11	11	12	10	10	10
30	7	20	13	13	15	13	13	14
29	4	18	12	11	13	11	11	12
30	13	21	17	17	17	17	17	18
time	1.76	2.79	4.2	9.23	10.4	149	2516	168

Table 5 – TEST' =(35,100,2,1.5) ∪ (5,100,10,5)

$\lvert S_d \rvert$	LB	UB	G	MEC		S1_MEC	S2_MEC	OLGA
20	5	15	9	9	9	9	9	10
20	4	14	7	7	8	7	7	9
19	2	15	8	7	8	7	7	9
19	4	15	8	8	8	8	8	8
18	4	13	7	7	7	7	7	8
19	3	12	7	7	7	7	7	7
20	3	12	6	6	6	6	6	8
18	3	12	5	5	5	5	5	5
19	3	12	6	6	6	6	6	6
22	3	15	8	8	8	8	8	8
time	0.94	1.645	2.632	6.28	7.14	116.2	778	599

Table 6 – TEST' =(20,100,2,1.5) ∪ (20,100,10,5)

Tables 5 and 6 summarize the results on the TEST' problems, which have been constructed in order to emphasize the differences among the algorithms.

Objective function: S1_MEC and S2_MEC do always find the best solution on all the TEST' problems (such solution can not be proved to be an optimum one, of course), while MEC found the best solution on all examples but one. Algorithm G fails to find the best solution on 5 over 20 examples. Nevertheless, all these algorithms outperform OLGA, which fails to find the best solution on 50% of the examples, approximatively.

Execution times: The same conclusions drawn for the TEST problems hold.

It is worth noticing that an all TEST and TEST' problems, as espected, the execution times are slightly related to $\lvert S_d \rvert$.

#triple $\mid S_d \mid$	LB	UB	G	MEC		S1_MEC	S2_MEC	OLGA
16	1	12	8	8	7	7	7	7
12	0.2	0.88	1.16	2.14	2.2	30	132	45
18	1	15	11	11	10	10	10	10
15	0.2	1.32	1.76	2.9	2.9	60	310	130
20	1	17	13	13	13	12	12	11
17	0.2	1.6	2.19	3.7	4	80	500	120
28	1	23	11	11	10	7	7	7
23	0.3	3	2.7	5.3	5.5	120	1020	280
16	1	13	8	8	9	8	7	7
13	0.2	1.1	1.21	2.14	2.47	380	180	80

Table 7 – *Ad hoc* designed examples

The last table shows the results obtained on *ad hoc* designed examples. Each element of the first column reports the number of triples of the given S (above) and of the derived (below). Each element of the remaining columns (see also explanations of Tables 1 through 6) reports the objective function value and the time required to compute it.

The situation is somewhat different from the randomly generated ones. The first and second examples were designed to show that Algorithm G can be easily made to fail. The third one shows that OLGA also has some peculiar property. The fourth one clearly separates the two simplest algorithms (G and MEC) from the more sophisticated ones. Finally, the fifth example distinguish S1_MEC from S2_MEC, showing that the latter may outperform the former one. The execution times result in being very different from one another. All the examples are characterized by having a very small lower bound, due to the presence of a window containing all the other ones, and by having an upper bound which always coincides with $\mid S_d \mid$, because the windows of S_d are pairwise disjoint by construction. Notice that the instance of DSS constructed in the NP-completeness proof has exactly this structure. Notice also that the difference among UB and LB is larger when the problem instance has both triples with small windows and interval lengths and triples with large windows and interval lengths, in a relative sense.

In conclusion, we think that the most effective algorithms were MEC and S1–MEC, which gave the better results both from the view point of the trade off among objective function and execution time. Indeed, S1–MEC achieves slightly better results than MEC, but it definitely requires longer computation time.

9. Conclusions

We studied the problem of computing the dominating set of a Shiftable Interval Graph (SIG). The problem is proved to be NP-complete in the strong sense on arbitrary SIG's, and it is proved to be polynomial time solvable on special classes. Lower and upper bounds are proposed for the general case. Exact algorithms for the special cases considered are also proposed, as well as several heuristic algorithms for the general case. The tables reported show that some of the heuristic algorithms find good solutions in a reasonable amount of time, which compares well if we recall that the problem is NP-complete in the strong sense. Many examples were studied which helped in understanding the most difficult problem structures.

References

[1] Bertossi, A.A., "Total Domination in Interval Graphs", *Inf. Proc. Lett.* 23, (1986), 131-134.

[2] Bonfiglio, P.: *Lo Scheduling e il Problema dell'Insieme Dominante sui SIG*, Università di Roma "La Sapienza", Tesi di Laurea in Ingegneria Elettronica, 1994.

[3] Booth, K.S., and J.H. Johnson, "Dominating Sets in Chordal Graphs", *SIAM J. Comput.* 11, (1982), 191-199.

[4] Farber, M., "Independent Domination in Chordal Graphs", *Op. Res. Lett.* 1, (1982), 134-138.

[5] Farber, M., "Domination, Independent Domination, and Duality in Strongly Chordal Graphs", *Discr. Appl. Math.* 7, (1984), 115-130.

[6] Garey, M.R., and D.S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W.H. Freeman & Co., New York,1979.

[7] Gilmore, P.C., and A.J. Hoffman, "A Characterization of Comparability and of Interval Graphs", *Canad. J. Math.* 16, (1964), 539–548.

[8] Golumbic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, London 1980.

[9] Keil J.M., "Total Domination in Interval Graphs", *Inf. Proc. Lett.* 22, (1986), 171-174.

[10] Malucelli, F., and S. Nicoloso, "Shiftable Interval Graphs", IASI-CNR Rep. 435 (1996), revised 1998 and submitted to Discreta Applied Mathematics.

[11] Preparata, F.P., and M.I. Shamos: *Computational Geometry*, Springer Verlag, New York, 1985.

[12] Ramalingan, G., and C. P. Rangan, "A Unified Approach to Domination Problems on Interval Graphs", *Inf. Proc. Lett.* 27, (1988), 271-274.

[13] White, K., M. Farber, and W.R. Pulleyblank, "Steiner Trees, Connected Domination, and Strongly Chordal Graphs", *Networks* 15, (1985), 109-124.

***mi accorgo solo ora che non abbiamo mai verificato se un era buono tra quelli utilizzati nelle prove sperimentali.