SIAM J. DISC. MATH.
Vol. 1, No. 3, August 1988

TOTAL DOMINATION AND IRREDUNDANCE IN WEIGHTED INTERVAL GRAPHS*

ALAN A. BERTOSSI† AND ALESSANDRO GORI†

Abstract. In an undirected graph, a subset X of the nodes is a total dominating set if each node in the graph is a neighbor of some node in X. In contrast, X is an irredundant set if the closed neighborhood of each node in X is not contained in the union of closed neighborhoods of the other nodes in X. This paper gives $O(n \log n)$ and $O(n^4)$ time algorithms for finding, respectively, a minimum weighted total dominating set and a minimum weighted maximal irredundant set in a weighted interval graph, i.e., one that represents n intersecting intervals on the real line, each having a (possibly negative) real weight.

Key words. dominating set, total dominating set, irredundant set, interval graph, min-tree, shortest path

AMS(MOS) subject classifications. 05C70, 49B99, 68R10, 68Q25

1. Introduction. Let $G(N, E)$ be an undirected graph. The *open neighborhood* of a node x is $N(x) = \{y \in N: [x, y] \in E\}$, while its *closed neighborhood* is $N[x] = N(x) \cup \{x\}$. In general, let $N(X)$ and $N[X]$ denote, respectively, $\cup_{x \in X} N(x)$ and $N(X) \cup X$.

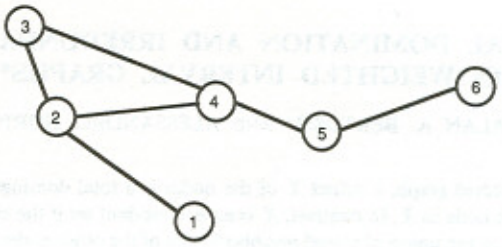
A subset X of the nodes *dominates* another subset Y if $Y \subseteq N(X)$. X is a *dominating set* if $N[X] = N$, while it is a *total dominating set* if in addition $X \subseteq N(X)$ [COC80]. In words, X is a dominating set if each node of $N - X$ is adjacent to at least one node in X, while it is also total if in addition each node in X is adjacent to some other node in X. As an example, Fig. 1(a) shows the "Capricorn" graph. It is easy to realize that $\{1, 4, 6\}$ is a dominating set whereas $\{2, 4\}$ is not; moreover, $\{1, 2, 5, 6\}$ is a total dominating set.

A node $x \in X$ is *redundant* in X if $N[x] \subseteq N[X - \{x\}]$. X is an *irredundant set* if it contains no redundant node. An irredundant set is *maximal* if it is not properly contained within another irredundant set. For instance, the set $\{2, 4\}$ is an irredundant set for the Capricorn graph, whereas $\{2, 4, 5\}$ is not, since node four is redundant; furthermore, $\{2, 4\}$ is also maximal, since adding any other node to it would induce redundancies.

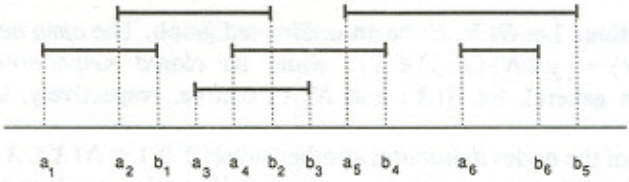
The problems of determining dominating, total dominating, or irredundant sets have obvious applications in the location of facilities in a network [ROB78]. In particular, the notion of redundancy is relevant in the context of communication networks, since any redundant node in a set can be removed from the set without affecting the totality of nodes that may receive communication from some node in the set [BOL79], [LAS85].

The problem of finding dominating sets along with several closely related variants has been remarkably investigated (e.g., see [HUJ84], [JOH84] for extensive references). In particular, finding a minimum cardinality total dominating set is NP-*complete*, even for special classes of graphs, such as *bipartite* or *chordal graphs* [BER84], [CHA84], [LAS84]. Moreover, finding a minimum cardinality maximal irredundant set is also NP-complete for chordal graphs [LAS83], but can be solved in linear time for *trees* [BER85] and *partial k-trees* [WIM87]. In this paper, we consider the weighted versions of these last two problems on a particular subclass of chordal graphs, known as *interval graphs*. Such graphs are one of the most useful discrete mathematical structures for modeling problems arising in the real world. In fact, they have found applications in

* Received by the editors December 1, 1986; accepted for publication (in revised form) February 9, 1988.
† Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56100 Pisa, Italy.



(a)



(b)

FIG. 1. (a) *The "Capricorn" graph*; (b) *an interval family for it*.

archaeology, biology, psychology, traffic control, job scheduling, and storage information retrieval (e.g., see [GOL80], [ROB78], [GOL85]).

More formally, an *interval family* is a set of intervals on the real line. A graph is an *interval graph* if there is a one-to-one correspondence between the nodes of the graph and the intervals of an interval family such that two nodes of the graph are joined by an edge if and only if their corresponding intervals overlap. An interval graph is *weighted* if a real (possibly negative) number w_i (the *weight*) is associated to each node i. For instance, Fig. 1(b) shows an interval family for the Capricorn graph, which thus results to be an interval graph.

Investigating the algorithmic complexity of total domination in interval graphs was mentioned in [LAS84] as a relevant open question. Successively, an $O(n + e)$ time algorithm for finding a minimum cardinality total dominating set in an n-node, e-edge interval graph has been devised (see [KEI86] and, for a more comprehensive exposition, [RAM88]). Independently, an $O(n^2)$ time algorithm has been proposed for the same problem that can be easily modified so as to work in the weighted case too [BER86]. In this paper, in § 3, we improve upon both these results, by giving an $O(n \log n)$ time algorithm for finding a minimum weighted total dominating set in an interval graph. We also present in § 4 an $O(n^4)$ time algorithm for determining a minimum weighted maximal irredundant set in an interval graph.

2. Background. Let $I = \{1, \cdots, n\}$ be an interval family such that the ith interval is equal to $[a_i, b_i]$, $i = 1, \cdots, n$. Without loss of generality, we assume that each interval contains both its endpoints and that no two intervals share a common endpoint. For the sake of simplicity, the interval graph G corresponding to the interval family I will be denoted as $G(I)$ and we shall deal directly only with the intervals instead of the nodes.

We assume that the interval family I is augmented with two extreme intervals, say zero and $n+1$, in such a way that $b_0 < a_1$, $b_n < a_{n+1}$, and $w_0 = w_{n+1} = 0$. We also assume that the intervals are indexed by increasing a_i's, namely, $a_0 < a_1 < \cdots < a_n < a_{n+1}$. We generically denote with $e_1, \cdots, e_{2n+4}$ the resulting $2n+4$ endpoints, sorted by increasing order.

To better understand the algorithms presented in this paper, we first need to review some known results. As shown in [MAN84], the problem of finding a dominating set having minimum overall weight in an interval graph $G(I)$ can be solved by reducing it to a shortest path problem on an appropriate directed graph H.

Let us temporarily assume that there is no negative weight and consider the augmented interval family $I' = I \cup \{0, n+1\}$. The nodes of H correspond to the intervals in I'. There is an arc (i, j) in H with length w_j if and only if $j \in P_i \cup Q_i$, where

$$P_i = \{k : a_i < a_k < b_i < b_k\}, \quad \text{and}$$

$$Q_i = \{k : a_k > b_i \text{ and there is no } h \text{ with } b_i < a_h < b_h < a_k\}.$$

We may readily verify that any shortest path from node zero to node $n+1$ in H corresponds to a minimum dominating set for $G(I')$. The minimum dominating set for $G(I)$ is obtained from such a path by deleting nodes zero and $n+1$.

Since H is acyclic, a shortest path can be found in $O(n^2)$ time [LAW76]. However, an $O(n \log n)$ time algorithm exists that uses a data structure known as *min-tree*. It consists of a complete binary tree with $n+2$ leaves (corresponding to intervals in I') that is almost exactly balanced in such a way that the leftmost set of leaves is at one level and the rightmost set of leaves at one level less. A value is assigned to each leaf of the tree, while each interior node maintains the minimum of the values of its sons. An example is shown in Fig. 2.

Let us define a *block* to be a set of intervals with consecutive indices, e.g., $\langle i, i+1, \cdots, i+d \rangle$. Given a block of leaves, it is possible to find in $O(\log n)$ time the minimum value in the block along with the index of the leaf in which this minimum is located [AHO74], [MAN84]. A new value may be inserted in a leaf, and updating the min-tree then also takes $O(\log n)$ time. Moreover, constructing such a min-tree takes $O(n)$ time.

Manacher and Smith's algorithm for finding a minimum dominating set sequentially scans the interval endpoints $e_1, \cdots, e_{2n+2}$ backward, from right to left (lines 5–14). As soon as a right endpoint b_i is encountered (line 6), the set Q_i is determined (line 7). Since Q_i is clearly a block, it is implemented by a pair $\langle Q_i[\text{lower}], Q_i[\text{upper}] \rangle$. The algorithm uses a min-tree T_{DIST} in which the value DIST $[i]$ associated to a leaf i is the distance in a shortest path of H between i and $n+1$. As soon as a left endpoint a_i is

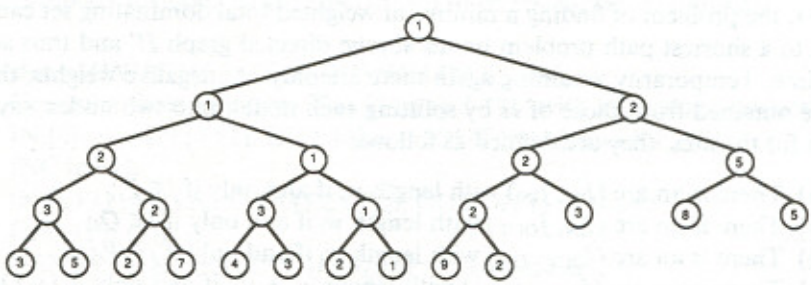


FIG. 2. *A min-tree with* 13 *leaves.*

encountered (line 8), the leaf k having smallest value in the block $\langle i+1, Q_i[\text{upper}]\rangle$ is determined and DIST $[k]$ is added to w_i in order to get DIST $[i]$ (lines 9–11).

```
1.  procedure MINIMUM-DOMINATING-SET;
2.      Construct a min-tree T_DIST with leaf values DIST [0], ··· ,DIST [n+1];
3.      DIST [n+1] := 0 and UPDATE (T_DIST);
4.      NEXT [n+1] := "end-of-list"; AVAIL [lower] := AVAIL [upper] := n+1;
5.      for s from 2n+2 to 1 by −1 do
6.          if e_s is a right endpoint b_i then
7.              Q_i[lower] := AVAIL [lower]; Q_i[upper] := AVAIL [upper];
8.          else // e_s is a left endpoint a_i //
9.              k := MINEL (T_DIST, i+1, Q_i[upper]);
10.             NEXT [i] := k;
11.             DIST [i] := w_i + DIST [k] and UPDATE (T_DIST);
12.             AVAIL [lower] := i; AVAIL [upper] := min {AVAIL [upper],
                Q_i[lower] − 1};
13.         endif
14.     endfor.
```

The output of the algorithm is a minimum dominating set the weight of which is DIST [0] and the elements of which are those in the NEXT list beginning with NEXT [0]. The subroutine *MINEL* (T, i, j) selects the leaf of the min-tree T having minimum value in the block $\langle i, j\rangle$. In case of ties, it chooses the rightmost leaf in the block. Let

$$C_i = \{h : a_i < a_h < b_h < b_i\}.$$

The correctness of the algorithm is based on the fact that for the block

$$\langle i+1, Q_i[\text{upper}]\rangle = C_i \cup P_i \cup Q_i,$$

MINEL does not select any interval k which either is in C_i or is "sandwiched" (i.e., with NEXT $[k] \in C_i \cup P_i$). Indeed, in both cases, either DIST $[k]$ is not the minimum value in the block (since $w_k > 0$) or it is (because $w_k = 0$), but k is not the rightmost leaf [MAN84, Lemma 5b]. Since *MINEL* and *UPDATE* both take $O(\log n)$ time, the overall $O(n \log n)$ time bound follows.

In the case there are negative weights, each $w_i < 0$ is temporarily set to zero. Then the above procedure is applied and a minimum weighted dominating set for $G(I)$ is obtained by deleting intervals zero and $n+1$ and by adding all the intervals having negative weights [MAN84].

3. Total domination. As we may easily prove by extending the results given in [BER86], the problem of finding a minimum weighted total dominating set can also be reduced to a shortest path problem on an acyclic directed graph H' and thus solved in $O(n^2)$ time. Temporarily assuming again there are only nonnegative weights, the nodes of H' are obtained from those of H by splitting each node i into two nodes, say i_{IN} and i_{OUT}. As for the arcs, they are defined as follows:

(i) There is an arc $(i_{\text{IN}}, j_{\text{IN}})$ with length w_i if and only if $j \in P_i$;
(ii) There is an arc $(i_{\text{IN}}, j_{\text{OUT}})$ with length w_i if and only if $j \in Q_i$;
(iii) There is an arc $(i_{\text{OUT}}, j_{\text{IN}})$ with length w_i if and only if $j \in P_i$;
(iv) There is an arc $(i_{\text{OUT}}, j_{\text{OUT}})$ with length $w_i + w_k$ if and only if $j \in Q_i$, $k \in C_i$ and $w_k = \min\{w_h : h \in C_i\}$.

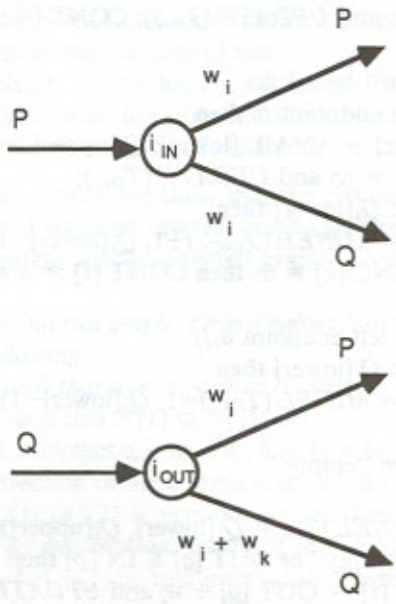


FIG. 3. *Nodes and arcs in H'.*

As illustrated in Fig. 3, only "P-arcs" enter into an "IN-node," while both P- and "Q-arcs" can leave from it. In contrast, only Q-arcs enter into an "OUT-node." This is because a total dominating set cannot contain any *isolated* interval, that is, one that does not overlap with any other interval in the set. Since P-arcs are induced by interval overlapping, but Q-arcs are not, no two consecutive Q-arcs may appear in H', unless an interval properly contained within another (indeed, that having minimum weight) is taken into account as in (iv).

As before, our algorithm scans the interval endpoints $e_1, \cdots, e_{2n+2}$ backward. However, we use three min-trees T_{IN}, T_{OUT}, and T_{INC}. The leaves of T_{IN} (T_{OUT}) correspond to IN-nodes (OUT-nodes) of H', and the value IN [i] (OUT [i]) associated to a generic leaf i is the shortest path distance between i_{IN} (i_{OUT}) and $n + 1_{\text{OUT}}$ (lines 24–30 and 31–42, respectively). T_{INC}, instead, is used to determine, for each i, a $k \in C_i$, if any, such that $w_k = \min \{w_h: h \in C_i\}$. The value INC [$i$] initially associated to each leaf i is w_i (lines 6–8). Since C_i is not a block, the block $\langle i + 1, Q_i[\text{lower}] - 1 \rangle = C_i \cup P_i$ is considered as soon as the right endpoint b_i is encountered (lines 13–16). The algorithm acts in such a way that leaves in P_i have value equal to infinity at this time. This is accomplished by setting INC [i] to ∞ as soon as b_i is scanned, since hereafter interval i cannot be properly included in any interval yet to be encountered (line 12).

1. **procedure** *MINIMUM-TOTAL-DOMINATING-SET*;
2. Construct three min-trees T_{IN}, T_{OUT} and T_{INC} with leaf values, respectively, IN [0], $\cdots$,IN [n+1], OUT [0], $\cdots$,OUT [n+1], and INC [0], $\cdots$, INC [n+1];
3. IN [n+1] := 0 and *UPDATE* (T_{IN}); OUT [n+1] := 0 and *UPDATE* (T_{OUT});
4. NEXT [$n+1_{\text{IN}}$] := NEXT [$n+1_{\text{OUT}}$] := "end-of-list";
5. AVAIL [lower] := AVAIL [upper] := n+1;
6. **for** i **from** 0 **to** n+1 **do**

```
7.      INC [i] := w_i and UPDATE (T_INC); CONT [i] := "empty";
8.    endfor
9.    for s from 2n+2 to 1 by −1 do
10.     if e_s is a right endpoint b_i then
11.       Q_i[lower] := AVAIL [lower]; Q_i[upper] := AVAIL [upper];
12.       INC [i] := ∞ and UPDATE (T_INC);
13.       if i+1 < Q_i[lower] then
14.         k := MINEL (T_INC, i+1, Q_i[lower]−1);
15.         if INC [k] ≠ ∞ then CONT [i] := k endif
16.       endif
17.     else // e_s is a left endpoint a_i//
18.       if i+1 < Q_i[lower] then
19.         p := MINEL (T_IN, i+1, Q_i[lower]−1);
20.       else
21.         p := "empty"
22.       endif
23.       q := MINEL (T_OUT, Q_i[lower], Q_i[upper]);
24.       if p = "empty" or OUT [q] ≦ IN [p] then
25.         IN [i] := OUT [q] + w_i and UPDATE (T_IN);
26.         NEXT [i_IN] := q_OUT;
27.       else
28.         IN [i] := IN [p] + w_i and UPDATE (T_IN);
29.         NEXT [i_IN] := p_IN;
30.       endif
31.       if p = "empty" then
32.         OUT [i] := ∞ and UPDATE (T_OUT);
33.       else
34.         k := CONT [i];
35.         if k = "empty" or IN [p] ≦ OUT [q] + w_k then
36.           OUT [i] := IN [p] + w_i and UPDATE (T_OUT);
37.           NEXT [i_OUT] := p_IN;
38.         else
39.           OUT [i] := OUT [q] + w_i + w_k and UPDATE (T_OUT);
40.           NEXT [i_OUT] := q_OUT;
41.         endif
42.       endif
43.       AVAIL [lower] := i; AVAIL [upper] := min {AVAIL [upper],
          Q_i[lower]−1};
44.     endif
45.   endfor.
```

The output of the algorithm is a minimum total dominating set the weight of which is IN [0] and the elements of which are those in the NEXT list beginning with NEXT [0_{IN}]; of course, elements zero and $n + 1$ are not to be included in such a set; moreover, if two consecutive OUT-nodes, say i_{OUT} and j_{OUT}, appear in the NEXT list, then also CONT [i] belongs to the optimum solution.

The correctness of the algorithm follows from Lemma 5b of [MAN84]. In particular, the subroutine *MINEL* selects the proper leaf having minimum value also in the block $\langle i + 1, Q_i[\text{lower}] - 1 \rangle = C_i \cup P_i$ (line 22), excluding, as before, selection of an interval

p that is in C_i, as we may readily check. Notice that the "$\leqq$" condition in line 24 allows us to select leaf q (the rightmost one) in case of ties.

The overall time complexity is $O(n \log n)$. Of course, in the presence of negative weights, the same maneuver seen in the previous section can be performed, and the actual minimum total dominating set is thus found with the same computational effort.

4. Irredundance. In this section, we show that the problem of finding a minimum weighted maximal irredundant set of an interval graph can also be reduced to a shortest path problem on an appropriate acyclic directed graph. To do this, we first need the following result.

LEMMA. *Let $G(I)$ be an interval graph. Then a subset X of intervals has redundancies if and only if we have the following*:

(1) *There are $i, j \in X$ such that $a_i < a_j < b_j < b_i$, or*

(2) *There are $i, j \in X$ such that $N[i] \subseteq N[j]$, or*

(3) *There are $i, j, h \in X$ such that $a_i < a_h < a_j$, $b_i < b_h < b_j$, and $N[h] \subseteq N[i] \cup N[j]$.*

Proof. Verifying one direction of the lemma is trivial. So assume X to have redundancies. If either condition (1) or (2) is verified, we are done. If none of them holds, assume h to be redundant. By assumption, intervals in I, and hence in X, are indexed by increasing left endpoints. Let i (j) be that interval in X which immediately precedes (follows) h in the ordering. Note that both i and j must exist, because otherwise either condition (1) or (2) should be verified. Since h is redundant, the case $b_i < a_h < b_h < a_j$ cannot occur. Thus h overlaps with at least one interval between i and j. Without loss of generality, assume $a_i < a_h < b_i$ (the case $a_j < b_h < b_j$ can be dealt with similarly). Each interval in $N[h]$ including one point p, $a_h < p < b_i$, is dominated by i. Thus consider the remaining intervals in $N[h]$. Each such interval k must contain a point q, $b_i < q < b_h$. Since there is no interval in X properly contained within another interval in X and h is redundant, $b_i < a_k < b_k < a_j$ cannot occur. Thus, k overlaps with j and $N[h] \subseteq N[i] \cup N[j]$. □

Let $J = \{(i, j) : i, j \in I,\ a_i < a_j < b_i < b_j,\ N[i] \not\subseteq N[j]$ and $N[j] \not\subseteq N[i]\}$. By the lemma, we can define a directed graph D having node set equal to $I' \cup J$. There is an arc (x, y) in D if and only if

$$y \in Q_x \cup R_x \cup S_x \cup T_x \cup U_x,$$

where:

(a) for $i \in I'$, $Q_i = \{j : a_j > b_i$ and there is no v with $b_i < a_v < b_v < a_j\}$,

(b) for $i \in I'$, $R_i = \{(j, k) : a_j > b_i$, $N[j] \not\subseteq N[i] \cup N[k]$ and there is no v with $b_i < a_v < b_v < a_j$ and $N[j] \not\subseteq N[v] \cup N[k]\}$,

(c) for $(i, j) \in J$, $S_{i,j} = \{k : a_k > b_j$, $N[j] \not\subseteq N[i] \cup N[k]$ and there is no v with $b_j < a_v < b_v < a_k$ and $N[j] \not\subseteq N[i] \cup N[v]\}$,

(d) for $(i, j) \in J$, $T_{i,j} = \{(h, k) : a_h > b_j$, $N[j] \not\subseteq N[i] \cup N[h]$, $N[h] \not\subseteq N[j] \cup N[k]$ and there is no v with $b_j < a_v < b_v < a_h$ for which $N[j] \not\subseteq N[i] \cup N[v]$ and $N[h] \not\subseteq N[v] \cup N[k]\}$,

(e) for $(i, j) \in J$, $U_{i,j} = \{(j, h) : a_h > b_i$, and $N[j] \not\subseteq N[i] \cup N[h]\}$.

Examples of arcs in D are given in Fig. 4. The length of the arcs is clearly equal to w_i in cases (a), (b), and (e), while it is equal to $w_i + w_j$ in cases (c) and (d). Note that in case (e) there is an arc between nodes (i, j) and (j, k), which share the *same* j. This

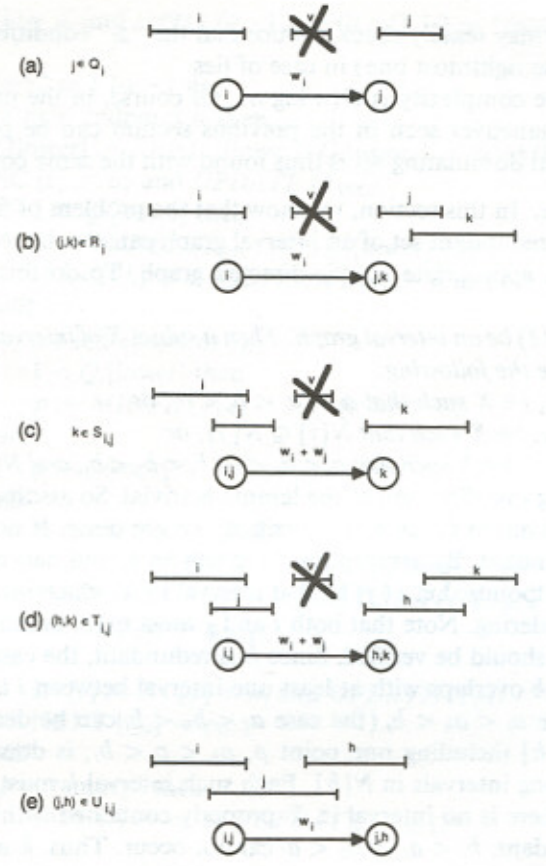


FIG. 4. *Examples of arcs in D.*

allows us to consider sequences of (two or more) pairwise overlapping intervals. A complete example is exhibited in Fig. 5.

THEOREM. *All the paths in D between zero and $n + 1$ correspond to maximal irredundant sets of $G(I)$, and vice versa.*

Proof. By construction, any path p in D between zero and $n + 1$ corresponds to an irredundant set $X \subseteq I$. Indeed, for any three (consecutive) intervals in p, conditions (1)–(3) of the lemma have been tested. To prove maximality, let us add to X any interval $v \in I - X$ and show that the resulting set $X \cup \{v\}$ has redundancies.

Clearly, if there is a j in X such that $a_j < a_v < b_v < b_j$ (or $a_v < a_j < b_j < a_v$) then v (j) is redundant. Moreover, this is true also if $N[v] \subseteq N[j]$ (or $N[j] \subseteq N[v]$). Therefore, let i, j be two consecutive intervals appearing in p such that $a_i < a_v < a_j$, $N[v] \not\subseteq N[i]$, $N[v] \not\subseteq N[j]$, $N[i] \not\subseteq N[v]$, and $N[j] \not\subseteq N[v]$ (of course, $i = 0$ and/or $j = n + 1$ can occur). Five cases may come up.

(a) If $(i, j) \in J$, then v is "sandwiched" by i and j, since $a_i < a_v < a_j < b_i < b_v < b_j$, and thus it is redundant (Fig. 6(a)).

(b) If $j \in Q_i$, then clearly the case $b_i < a_v < b_v < a_j$ cannot occur; moreover, neither can there exist an interval $w \in N(v)$ with $b_i < a_w < b_w < a_j$. Thus v is redundant (Fig. 6(b)).

(c) Assume there is a k in p for which $(j, k) \in R_i$. If $b_i < a_w < b_w < a_j$ for a $w \in N[v]$, then, by definition of R_i, $N[j] \subseteq N[i] \cup N[w]$ and thus j is redundant (Fig. 6(c)). Otherwise, v is redundant.

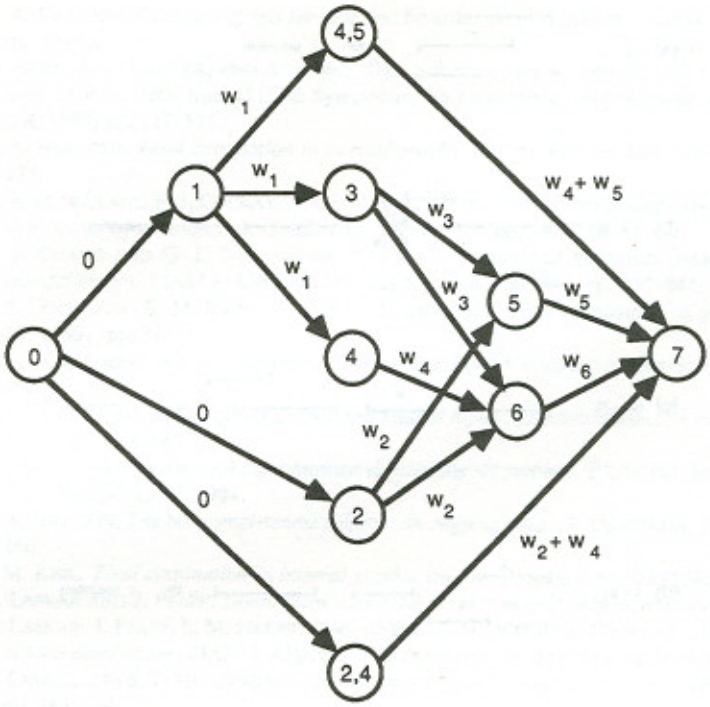


FIG. 5. *The directed graph D for the Capricorn graph.*

(d) Let $j \in S_{h,i}$ for an h in p. Again, if there is a $w \in N[v]$ for which $b_i < a_w < b_w < a_j$, then i is redundant, since $N[i] \subseteq N[h] \cup N[w]$, by definition of $S_{h,i}$ (Fig. 6(d)). If no such w exists, v itself is redundant.

(e) Finally, when the case $(j, k) \in T_{h,i}$ holds for h and k in p, then either i or j, or both, are redundant (if there is $w \in N[v]$ with $b_i < a_w < b_w < a_j$: see Fig. 6(e)), or v itself is redundant (otherwise).

Conversely, let X be a maximal irredundant set and assume its elements to be ordered by increasing a_i's. Suppose X does not correspond to any path in D. Thus there exist two consecutive intervals in X, say i and j, for which neither $(i, j) \in J$ nor is there an arc joining them.

Since $(i, j) \notin J$, then i and j do not overlap, i.e., $b_i < a_j$. Moreover, since $j \notin Q_i$, there exists a $v \in I - X$ with $b_i < a_v < b_v < a_j$. Thus $N[i] \not\subseteq N[v]$, $N[j] \not\subseteq N[v]$ and $N[v] \not\subseteq N[i] \cup N[j]$. Four cases may appear: in each of them we shall reach a contradiction.

(i) If both i and j do not overlap with any other interval in X, then $X \cup \{v\}$ is irredundant.

(ii) If only interval i overlaps with another interval in X, say h, then $N[i] \not\subseteq N[h] \cup N[v]$, since $j \notin S_{h,i}$. Thus $X \cup \{v\}$ is again irredundant.

(iii) If only interval j overlaps with another interval in X, say k, then $N[j] \not\subseteq N[v] \cup N[k]$, since $(j, k) \notin R_i$. As before, $X \cup \{v\}$ results to be irredundant.

(iv) If intervals i and j overlap, respectively, with h and k, then $N[i] \not\subseteq N[h] \cup N[v]$ and $N[j] \not\subseteq N[v] \cup N[k]$, since $(j, k) \notin T_{h,i}$. Even in this case, $X \cup \{v\}$ comes out to be irredundant. □

As a consequence of the above theorem, any minimum weighted maximal irredundant set can be found by means of the following algorithm.

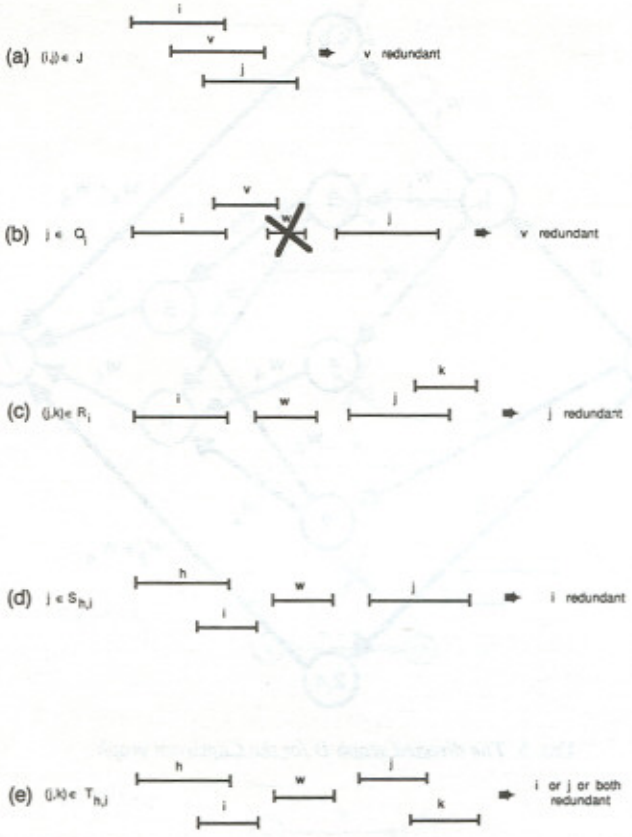


FIG. 6. *Examples of redundancies*.

1. **procedure** *MINIMUM-MAXIMAL-IRREDUNDANT-SET*;
2. Construct the directed graph D;
3. Find a shortest path $p = 0, x_1, \cdots, x_k, n+1$ in D;
4. The minimum maximal irredundant set X for $G(I)$ is given by all intervals i and $j \in I$ such that either $i = x_h$ or $(i, j) = x_h$, $1 \leq h \leq k$. The weight of X is equal to the length of p.

The time required to build up D is $O(n^4)$, since it is dominated by that needed to compute all the $T_{h,i}$'s and $U_{h,i}$'s. Indeed, let us assume each $N[i]$ to be represented by an ordered list. Then verifying $N[h] \subseteq N[i] \cup N[j]$ for given h, i, j, takes $O(n)$ time. Since there are $O(n^3)$ such triples, we can set up in $O(n^4)$ time a Boolean tri-dimensional array B whose generic entry $B[h, i, j]$ is equal to one if and only if $N[h] \subseteq N[i] \cup N[j]$. Then, all the R_i's and $S_{i,j}$'s can be computed in $O(n^4)$ time. Moreover, since $(h, k) \in T_{i,j}$ if and only if both $(h, k) \in R_j$ and $h \in S_{i,j}$, computing all the $T_{h,i}$'s also requires $O(n^4)$ time.

Since D contains $O(n^2)$ nodes and is acyclic, as we may easily verify, a shortest path can be found in $O(n^4)$ time [LAW76]. Hence, the overall running time required to find a minimum weighted maximal irredundant set of an interval graph is $O(n^4)$.

REFERENCES

[AHO74] A. AHO, J. HOPCROFT, AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[BER84] A. A. BERTOSSI, *Dominating sets for split and bipartite graphs*, Inform. Process. Lett., 19 (1984), pp. 37–40.

[BER85] M. BERN, E. L. LAWLER, AND A. WONG, *Why certain subgraph computations require only linear time*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, Portland, OR, 1985, pp. 117–125.

[BER86] A. A. BERTOSSI, *Total domination in interval graphs*, Inform. Process. Lett., 23 (1986), pp. 131–134.

[BOL79] B. BOLLOBÁS AND E. J. COCKAYNE, *Graph-theoretic parameters concerning domination, independence, and irredundance*, J. Graph Theory, 3 (1979), pp. 241–249.

[CHA84] G. J. CHANG AND G. L. NEMHAUSER, *The k-domination and k-stability problems on sun-free chordal graphs*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 332–345.

[COC80] E. J. COCKAYNE, R. M. DAWES, AND S. T. HEDETNIEMI, *Total domination in graphs*, Networks, 10 (1980), pp. 211–219.

[GOL80] M. C. GOLUMBIC, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[GOL85] M. C. GOLUMBIC, ED., *Interval graphs and related topics*, Discrete Math., 55 (special issue: July 1985), pp. 113–243.

[HUJ84] M. HUJTER, *Bibliography on the minimum dominating set problem*, RUTCOR, Rutgers University, New Brunswick, NJ, 1984.

[JOH84] D. S. JOHNSON, *The* NP*-completeness column: an ongoing guide*, J. Algorithms, 5 (1984), pp. 147–160.

[KEI86] J. M. KEIL, *Total domination in interval graphs*, Inform. Process. Lett., 22 (1986), pp. 171–174.

[LAS83] R. LASKAR AND J. PFAFF, *Domination and irredundance in split graphs*, manuscript, 1983.

[LAS84] R. LASKAR, J. PFAFF, S. M. HEDETNIEMI, AND S. T. HEDETNIEMI, *On the algorithmic complexity of total domination*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 420–425.

[LAS85] R. LASKAR AND S. T. HEDETNIEMI, *Irredundance in graphs: a survey*, Congress. Numer., 48 (1985), pp. 183–194.

[LAW76] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart & Winston, New York, 1976.

[MAN84] G. K. MANACHER AND C. J. SMITH, *Efficient algorithms for new problems on interval graphs and interval models*, manuscript, 1984.

[RAM88] G. RAMALINGAM AND C. PANDU RANGAN, *A unified approach to domination problems in interval graphs*, Inform. Process. Lett., to appear.

[ROB78] F. S. ROBERTS, *Graph Theory and Its Applications to Problems of Society*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1978.

[WIM87] T. V. WIMER, *Linear algorithms on k-terminal graphs*, Ph.D. thesis, Dept. of Computer Science, Clemson University, Clemson, SC, August 1987.