On the Problem of Finding All Maximum Weight Independent Sets in Interval and Circular-arc Graphs

Y. D. Liang, S. K. Dhall and S. Lakshmivarahan

Parallel Processing Institute
School of Electrical Engineering and Computer Science
University of Oklahoma
Norman, Oklahoma 73019

ABSTRACT

Leung (J. Algorithms, 5 (1984)) presented algorithms for generating all the maximal independent sets in interval graphs and circular-arc graphs. The algorithms take \(O(n^2+\beta)\) steps, where \(\beta\) is the sum of the number of nodes in all maximal independent sets. In this paper we use a new technique to give fast and efficient algorithms for finding all the maximum weight independent sets in interval graphs and circular-arc graphs. Our algorithms take \(O(\max(n^2, \beta))\) steps in \(O(n^2)\) space, where \(\beta\) is the sum of the number of nodes in all maximum weight independent sets. Our algorithms can be directly applied for finding a maximum weight independent set in these graphs in \(O(n^2)\) steps. Thus, our result is an improvement over the best known result of \(O(n^2 \log n)\) [4, 6] for finding the maximum weight independent set in circular-arc graphs.

1. Introduction

Consider a finite family \(S\) of nonempty sets. A graph \(G = (V, E)\) is called an intersection graph for \(S\) if there is a one-to-one correspondence between \(S\) and \(V\) such that two sets in \(S\) have nonempty intersection if and only if the corresponding nodes in \(V\) are adjacent to each other. If \(S\) is a family of intervals on real line, then \(G\) is called an interval graph. When \(S\) is a family of arcs on a circle, \(G\) is called a circular–arc graph. A graph is said to be a weighted graph if a real number \(w_i\) is associated with each node \(i\). The number \(w_i\) is called the weight of node \(i\). In this paper, we assume that all weights are positive. For \(R \subseteq V\), \(\sum_{i \in R} w_i\) is called the weight of the subset \(R\).

An independent set of a graph is a set of nodes such that no two nodes in the set are joined by an edge. In interval and circular-arc graphs, such a set corresponds to a set of mutually nonoverlapping intervals and arcs. An independent set is said to be maximal if it is not properly contained in another independent set. A maximum independent set is a maximal independent set with the largest cardinality. A maximum weight independent set is an independent set, such that its weight is maximum amongst all independent sets. A set of nodes is called a clique if the nodes are pairwise adjacent. A clique is said to be maximal if it is not properly contained in any other clique. A maximum clique is a maximal clique with the largest cardinality. A maximum weight clique is a maximal clique with the largest total weight. An algorithm for generating all maximum weight independent sets of a graph can be used to generate all maximum weight cliques, since a maximum clique of a graph has a one-to-one correspondence with a maximum weight independent set of the complementary graph.

Interval and circular-arc graphs have been used in many practical applications [1, 7, 8, 10, 11] and, as such a wide variety of algorithms have been developed [2–9]. Booth and Lueker [5] have given an \(O(n+m)\) time algorithm for recognizing interval graphs. Their algorithm also produces an interval model if the graph is indeed an interval graph. Tucker [12] has shown that recognizing circular-arc graphs can be done in \(O(n^2)\) time. His algorithm also constructs a circular-arc model if the graph is indeed a circular-arc graph. Gupta, Lee and Leung [8] have shown that finding a maximum independent set, a maximum clique, a minimum coloring, and a minimum coloring by maximal cliques can all be solved in \(O(n \log n)\) time, assuming the graph is given by an interval model. Also, Gupta, Lee and Leung [8] gave \(O(n^2)\) time algorithms for finding a maximum independent set, a minimum coloring by maximal cliques of circular-arc graphs, assuming the graph is given by a circular-arc model. Later Leung [9] developed efficient algorithms for generating all maximum independent sets of interval and circular-arc graphs. His algorithms take \(O(n^2+\beta)\) steps, where \(\beta\) is the sum of the number of nodes in all maximal independent sets.

In this paper, we present fast and efficient algorithms.
for generating all maximum weight independent sets in interval and circular-arc graphs. Our algorithms take $O(\max(n^2, \beta))$ steps using $O(n^2)$ space, where $\beta$ is the sum of the number of nodes in all maximum weight independent sets. Also, in this paper we use a technique different from the ones which have hitherto been used. This technique is likely to prove useful in other similar problems.

2. Maximum Weight Independent Set (MWIS) in interval graphs

We assume that the graph $G$ is given by a set of $n$ intervals $I_1, \cdots, I_n$, with each interval $I_i$ represented by its left endpoint, $l_i$, and its right endpoint, $r_i$, and $w_i$ as the weight of $I_i$. Without loss of generality, we assume that the left endpoints are distinct and sorted as $l_1 < \cdots < l_n$. (If the left endpoints are not sorted, it will take only $O(n \log n)$ time to sort them.)

From the given interval graph $G$, we construct the digraph $H = (V_H, E_H, W_H)$, where $V_H = \{1, 2, \cdots, n\}$, node $j$ corresponding to interval $I_j$, $j = 1, 2, \cdots, n$, and $w(j) = w_i$. An edge $(i, j)$ is in $E_H$ if

- (a) $r_i < l_j$, and
- (b) there is no $k$ such that $r_i < l_k$ and $r_k < l_j$.

In other words, $(i, j)$ is an edge in $H$ if the interval $I_j$ begins after the interval $I_i$ terminates, and no other interval lies completely in $[r_i, l_j]$. For example, let $G$ be an interval graph shown in Figure 2.1 with its interval model and the weight of each interval illustrated in Figure 2.2. The corresponding $H$ graph for $G$ is displayed in Figure 2.3.

![Figure 2.1: interval graph G](image)

![Figure 2.2: interval model for G](image)

![Figure 2.3: H graph for G](image)

A node in $V_H$ is called a source node if its indegree is 0, and is called a terminal node if its outdegree is 0. We say $P = (i_1, \cdots, i_k)$ is a full path in $H$ if $(i_l, i_l) \in E_H$ for $l = 2, \cdots, k$ and $i_1$ is a source node and $i_k$ is a terminal node.

**Lemma 2.1:** There is a one-to-one correspondence between a maximal independent set in $G$ and a full path in $H$.

**Proof.** From the definition of edges in $H$, it is obvious that $H$ is a digraph without cycles. Let $P = (i_1, \cdots, i_k)$ be any full path in $H$. By definition of edges in $H$, the interval corresponding to node $i_l$ must lie completely to the right of the interval corresponding to node $i_{l-1}$, for $l = 2$ to $k$. Hence no two intervals corresponding to the nodes on path $P$ intersect, implying that the corresponding nodes in $G$ form an independent set. Furthermore, we claim that $P$ corresponds to a maximal independent set of $G$. If not, let $i$ be a node which is not connected in $G$ to any of the nodes $(i_1, \cdots, i_k)$ of the path $P$. Assume $i_{l-1} < i < i_l$ for some $l \in \{2, \cdots, k\}$. Since $I_{i_{l-1}} \cap I_i = \emptyset$ and $I_i \cap I_{i_l} = \emptyset$, $I_i$ is a complete interval between $I_{i_{l-1}}$ and $I_{i_l}$. This implies that $(i_{l-1}, i)$ cannot be an edge in $H$ according to the rules for constructing the edges in $H$, which contradicts $(i_{l-1}, i)$ is an edge on the path $P$. Similar contradiction can be drawn if we assume $i < i_1$ or $i_k < i$.

Given any maximal independent set, let the nodes in the independent set be arranged in increasing order, say, $i_1 < i_2 < \cdots < i_k$. Hence the corresponding intervals $I_{i_1}, \cdots, I_{i_k}$ are pairwise disjoint. Since no other node can be added to this set for which the corresponding interval does not intersect with at least one of these nodes, the graph $H$ will have edges $(i_l, i_l)$, $l = 2, \cdots, k$, which shows that there is a path from $i_1$ to $i_k$. □

The following corollary is obvious.

**Corollary 2.2:** A maximum weight independent set in $G$ corresponds to a maximum weight path in $H$.

For example in Figure 2.1, $(2, 5, 7)$ is an MWIS in $G$, and is also a maximum weight full path in $H$ as shown.
For purposes of developing an efficient algorithm for computing all maximum weight paths, we use an alternative criterion for determining the edges in \( H \). In this connection, we first define two new terms. For each interval \( i \), \( \text{first}[i] \) is the left endpoint of the first interval to begin amongst all intervals which begin after the right endpoint of interval \( i \). If there is no such interval, then \( \text{first}[i] = \infty \). Symbolically,

\[
\text{first}[i] = \begin{cases} \
\min \{ l_k : r_i < l_k \}, & \text{if } \{ l_k : r_i < l_k \} \neq \emptyset, \\
\infty, & \text{otherwise.}
\end{cases}
\]

Further, let \( \text{next}[i] \) be the right endpoint of the first nonoverlapping (with \( I_i \)) interval to end amongst all intervals which begin after the right endpoint of interval \( i \). If there is no such interval, then \( \text{next}[i] = \infty \). As before we can write

\[
\text{next}[i] = \begin{cases} \
\min \{ r_k : r_i < l_k \}, & \text{if } \{ l_k : r_i < l_k \} \neq \emptyset, \\
\infty, & \text{otherwise.}
\end{cases}
\]

For example in Figure 2.2, \( \text{first}[1] = l_5 \) and \( \text{next}[1] = r_5 \). Computation of \( \text{first}[i] \) and \( \text{next}[i] \) takes \( O(\log n) \) steps by using binary search on the ordered list of the interval endpoints. Thus computing \( \text{first}[i] \) and \( \text{next}[i] \) for all nodes in \( H \) takes \( O(n \log n) \) steps. If \( i \) is a left endpoint, then let \( p[i] \) be the index of the interval whose left endpoint is \( l_i \). If \( r_i \) is a right endpoint, then let \( q[r] \) be the index of the interval whose right endpoint is \( r_i \), that is \( q[r_i] = k \) if \( r_i = r_k \). The following lemma gives a relation between the set of intervals and the edges in graph \( H \).

**Lemma 2.3.** Let \( i,j \in V_H \). Then:

1. If \( l_i < l_j < \text{first}[i] \), then \( r_i > l_j \) and \((i,j) \in E_H \);  
2. If \( \text{first}[i] \leq l_j < \text{next}[i] \), then \((i,j) \in E_H \);  
3. If \( \text{first}[i] \leq l_j < \text{next}[i] \), then \( \text{next}[i] \leq r_j \);  
4. If \( r_i > \text{next}[i] \), then \((i,j) \in E_H \).

**Proof.** (1) We prove it by contradiction. Let \( k=p[\text{first}[i]] \). Assume \( r_i < l_j \). Since \( r_i < l_j \) and \( l_i < l_k \), it implies that \( \text{first}[i] \leq \min \{ l_j, l_k \} \leq l_j \), contradicting \( \text{first}[i] > l_j \). Therefore \( r_i > l_j \). Furthermore, since \( l_i < l_j \) and \( r_i > l_j \), \( I_i \) intersects \( I_j \) and therefore \((i,j) \notin E_H \).

(2) Let us verify the conditions (a) and (b) of the rules for constructing edges of \( H \). If \( \text{first}[i] \leq l_j < \text{next}[i] \), then \( l_j \geq \text{first}[i] > r_i \), thus \( I_i \) and \( I_j \) do not overlap. This implies that condition (a) is satisfied. If there exists a complete interval \( I_k \) between \( r_i \) and \( l_j \), then \( \text{next}[i] \leq r_j < l_j \). This contradicts \( l_j < \text{next}[i] \). Thus, condition (b) is also satisfied. Therefore, \((i,j) \in E_H \).

(3) Again, we prove it by contradiction. Let \( k=q[\text{next}[i]] \). Assume \( r_k > r_j \). Since \( \text{first}[i] < l_j \) and \( \text{first}[i] \leq l_k \), then \( r_k = \text{next}[i] \leq \min \{ r_j, r_k \} \leq r_j \). This contradicts \( r_k > r_j \). Therefore, \( \text{next}[i] \leq r_j \).

(4) When \( l_j > \text{next}[i] \), \( I_{q[\text{next}[i]]} \) is a complete nonoverlapping interval between \( I_i \) and \( I_j \), violating condition (b) for \((i,j) \) to be in \( E_H \).

To compute all MWIS efficiently, we introduce the **expansive weight** (ew) and **prefix expansive weight** (pew).

**Definition 2.4.** Let expansive weight of a path from node \( i \) to node \( j \) in \( H \), denoted as \( \text{ew}[i,j] \), be the maximum weight amongst all paths from node \( i \) to \( j \). Let \( \text{ew}[i,i] = w[i] \) for \( i = 1, \ldots, n \) and \( \text{ew}[i,j] = 0 \) if there is no path from \( i \) to \( j \) and \( i \neq j \).

There may be many paths from node \( i \) to \( j \). However, we are interested only in paths for which the weight is maximum. The maximum expansive weight amongst \( \text{ew}[i,j] \) \( i=1,n; j=1,n \) is thus the weight of maximum weight independent set. Let \( \text{maxw} = \max \{ \text{ew}[i,j] : i=1,n, j=1,n \} \). In other words, all the paths in \( H \) with a weight \( \text{maxw} \) correspond to the maximum weight independent sets in \( G \). Thus our aim is to identify all full paths with weight \( \text{maxw} \).

**Definition 2.5.** Let **prefix expansive weight** from \( i \) to \( j \), denoted as \( \text{pew}[i,j] \), be the maximum expansive weight amongst \( \text{ew}[i,j], \text{ew}[i+1,j], \ldots \), and \( \text{ew}[j,j] \).

It is easy to show that

\[
\text{pew}[i,j] = \max \{ \text{ew}[i,j], \text{pew}[i+1,j] \}
\]

and

\[
\text{pew}[i,k] \geq \text{pew}[j,k] \text{ for } i < j < k.
\]

Computing \( \text{pew}[*,j] \) is, in fact, a prefix computation. The following properties are crucial for computing \( \text{ew}[i,j] \) efficiently.

**Lemma 2.6.**

1. If \( \text{ew}[i,j] = w_i \) if \( i = j \) and \( \text{ew}[i,j] = 0 \) if \( i > j \);
2. \( \text{ew}[i,j] = 0 \) and \( \text{pew}[i,j] = \text{pew}[i+1,j] \text{ if } i < j < p[\text{first}[i]] \);
3. \( \text{ew}[i,j] = \text{pew}[p[\text{first}[i]], j] + \text{ew}[i,l] \text{ if } p[\text{first}[i]] \leq j \).

**Proof.**

(1) Directly from definition 2.4.
(2) If \(i < j < p[\text{first}[i]]\), then \(l_i < l_j < \text{first}[i]\). By Lemma 2.3 (1), there exists no path from \(i\) to \(j\). Therefore, \(\text{ew}(i, j) = 0\) and

\[
\text{pew}(i, j) = \max\{\text{ew}(i, j), \text{pew}(i+1, j)\}
\]

\[
= \text{pew}(i+1, j).
\]

(3) Let \(S = \{x : \text{first}[i] \leq l_x < \text{next}[i]\}\). By Lemma 2.3 (2), \((i, x) \in E_H\) for any node \(x \in S\). For \(p[\text{first}[i]]\), we consider two cases.

Case 1. When \(\text{first}[i] \leq l_i < \text{next}[i]\), by Lemma 2.3 (3), we know \(r_j > \text{next}[i] > l_j\). However, \(l_i < \text{next}[i] < r_x\) for \(x \in S\), therefore \(l_i\) and \(r_x\) intersect. It implies that \(\text{ew}(x, j) = 0\). Therefore, \(\text{pew}(p[\text{first}[i]], j) = \max\{\text{ew}(\text{first}[i], j), \text{ew}(\text{first}[i]+1, j), \ldots, \text{ew}(x, j)\}\)

\[
= \text{ew}(j, j). \quad \Box
\]

These properties result in a natural algorithm for computing \(\text{ew}(i, j)\) by using \(\text{pew}(i, j)\). The algorithm, named \text{compute_ew}, initializes \(\text{ew}\) and \(\text{pew}\) and computes \(\text{ew}(i, j)\) and \(\text{pew}(i, j)\) for \(i\) from \(n\) down to 1. At the \(i^{th}\) iteration, \(\text{ew}(i, j)\) and \(\text{pew}(i, j)\) can be determined by \(\text{ew}(i, k)\) and \(\text{pew}(i, k)\) \((k > i)\) obtained from the previous steps. We give the detailed implementation of this algorithm in Figure 2.4.

Once \(\text{maxw} = \max\{\text{ew}(i, j) : i=1, n, j=1, n\}\) is available as the weight of an \(\text{MWIS}\), to obtain all maximum weight independent sets we create a subgraph of \(H\) called \(M = (V_M, E_M)\), \(V_M \subseteq V_H\), which contains all the maximum weight paths of \(H\). Let \(l\text{w}(x)\) be the maximum weight amongst all the paths from a source node to a node \(x\). Since weights are all positive, \(\text{ew}(i, x)\) is maximal only if \(i\) is a source node. Then \(l\text{w}(x) = \max \{ \text{ew}(i, x) : i=1, x-1 \}\). Let \(r\text{w}(x)\) be the maximum weight amongst all the paths from a node \(x\) to a terminal node. Then \(r\text{w}(x) = \max \{ \text{ew}(x, j) : j=x, n \}\). It follows

\[\text{algorithm compute_ew}
\]

\[
\begin{align*}
\text{begin} \\
\{\text{initialize \text{ew} and \text{pew}}\} \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\text{for } j = 1 \text{ to } n \text{ do} \\
\text{\text{ew}(i, j) } \leftarrow 0; \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\text{begin} \\
\text{\text{ew}(i, j) } \leftarrow \text{w}(i); \\
\text{\text{pew}(i, j) } \leftarrow \text{w}(i) \text{ end; }
\end{align*}
\]

\[\text{for } i = n \text{ down to } 1 \text{ do}
\]

\[
\begin{align*}
\text{\text{begin}} \\
\text{if } \text{first}[i] \neq \infty \text{ then} \\
\text{begin} \\
\text{for } j = i+1 \text{ to } p[\text{first}[i]]-1 \text{ do} \\
\text{\text{pew}(i, j) } \leftarrow \text{pew}(i+1, j); \\
\text{for } j = p[\text{first}[i]] \text{ to } n \text{ do} \\
\text{begin} \\
\text{\text{ew}(i, j) } \leftarrow \text{ew}(p[\text{first}[i]], j) + \text{ew}(i, j); \\
\text{\text{pew}(i, j) } \leftarrow \max \{ \text{pew}(i+1, j), \text{ew}(i, j) \}; \\
\text{end; }
\end{align*}
\]

\[\text{end; }
\]

\[\text{end; if} \\
\text{end;}
\]

\[\text{end;}
\]

\[\text{Figure 2.4}
\]

\[\text{Figure 2.5: M graph obtained from H}
\]

\[\text{algorithm create_max_graph}
\]

\[
\begin{align*}
\text{begin} \\
\text{for } i := 1 \text{ to } n \text{ do} \\
\text{for } j := p[\text{first}[i]] \text{ to } n \text{ do} \\
\text{if } (l\text{w}(i) + r\text{w}(j) = \text{maxw}) \text{ then add}(i, j); \\
\text{end;}
\end{align*}
\]

\[\text{Figure 2.6}
\]

that an edge \((i, j) \in E_M\) iff \(l\text{w}(i) + r\text{w}(j) = \text{maxw}\). For example in Figure 2.2, \(\text{maxw} = 8\), and \(M\) created from \(H\) is displayed in Figure 2.5.

Note that \(M\) extracts only the maximum weight full paths of \(H\). Thus, there is a one-to-one correspondence
between the maximal paths of \( M \) and the maximum weight independent sets in \( G \). For example in Figure 2.5, \([2, 5, 7], [2, 5, 8], [3, 5, 7] \) and \([3, 5, 8] \) are all full paths in \( M \), and they are all MWIS for \( G \). In order to output all full paths in \( M \), we first create a reversed adjacency list and then apply a standard technique for reducing the time for depth-first-search. Specifically, algorithm \text{create_max_graph} \) described in Figure 2.6, creates a linked list for each node \( j \) in \( M \) such that \( i \) is in the linked list iff \((i, j) \in E_M \).

Procedure \text{add}(i, j) \) is to add node \( i \) to the linked list of \( j \). Each node \( j \) keeps a linked list for all the nodes pointing to it. This is for identifying the path \text{via} \) backtracking at a later stage for printing out all MWIS. To generate all full paths in \( M \) from a terminal node to a source node, our algorithm starts from each terminal node and traverses across the path through the link by using a stack. When a source node is reached, a full path is found, and we continue searching for paths \text{via} \) backtracking until all the paths are found. Thus, for each node in a maximum weight independent set, only one edge in \( M \) is visited. The entire algorithm is outlined in Figure 2.7.

\begin{algorithm}
begin
compute first, next, \( p \) and \( q \);
compute \( ew \);
compute \( lw \) and \( \text{rw} \);
create \text{max_graph} \;
output all mwis;
end;
\end{algorithm}

Figure 2.7

Theorem 2.7: Algorithm \text{MWIS} \) correctly generates all MWIS in \( O(\text{max}(n^2, \beta)) \) steps using \( O(n^2) \) space.
Proof. Finding \text{first}[i], next[i], \( p[\text{first}[i]], q[\text{next}[i]], \) \( \text{lw}[i] \) and \( \text{rw}[i] \) takes \( O(n \log n) \) steps for each node \( i \). Thus for all nodes it takes \( O(n \log n) \) steps. Algorithm \text{compute}\_\text{ew} \) takes \( O(n^2) \) steps. Adding an element to the linked list of \( i \) takes constant time. Thus procedure \text{create}\_\text{max}\_\text{graph} \) takes \( O(n^2) \) steps. Note that for each node in a maximum weight independent set, only one edge in \( M \) is visited. It, therefore, takes \( O(\beta) \) steps, where \( \beta \) is the total number of nodes in all MWIS \( G \) to generate all full paths in \( M \). The data structures are two \( nxn \) arrays \( \text{ew} \) and \( \text{pew} \), and six linear size arrays \text{first}, \text{next}, \( p, q, \) \( lw \) and \( \text{rw} \). Since the length of any full path in \( M \) is no longer than \( n \), the size of the stack used for generating all full paths in \( M \) \text{via} \) backtracking is no larger than \( n \). Thus the algorithm takes \( O(\text{max}(n^2, \beta)) \) steps using \( O(n^2) \) space. \( \quad \square \)

3. MWIS in circular-arc graphs

The input to the algorithm is a set of \( n \) arcs. Let \( A = \{A_1, \cdots, A_n\} \). Arc \( A_i \) is denoted by \( [h_i, t_i], \) \( i = 1, \cdots, n \), where \( h_i \) is called the head point of \( A_i \) and \( t_i \) is called the tail point of \( A_i \). Without loss of generality, assume that all endpoints of \( n \) arcs are distinct and that there is no arc extending across the whole circle. We sort the 2n endpoints of the arcs in \( A \) such that we encounter the endpoints in increasing order if we start at \( h_1 \) and travel along the circle in the clockwise direction. This sorting can be done in \( O(n \log n) \) steps. Thus we assume that the 2n points in \( A \) are available in this order. We also assume that the arcs in \( A \) have been relabeled such that \( i < j \) implies \( h_i < h_j \). Let the circular-arc graph based on \( A \) be \( C \). We construct an interval graph \( G \) based on a set of intervals, \( I = \{I_1, \cdots, I_n\} \) such that for \( i = 1, \cdots, n \), \( I_i = [h_i, t_i] \) and

\[ r_i = \begin{cases} t_i, & \text{if } h_i < t_i, \\ h_i + \epsilon_i, & \text{if } t_i < h_i. \end{cases} \]

Where \( \epsilon_i \) is chosen such that \( h_i + \epsilon_i \) does not cross arc \( A_i \) when going clockwise around the circle, and \( \epsilon_i \)'s are chosen so that no two intervals have the same terminal point. We first construct \( H \) from \( G \) and use algorithm \text{compute}\_\text{ew} \) to find \( \text{ew}[i, j] \) for all \( i \) and \( j \) in \( H \). We must readjust \( \text{ew}[i, j] \) such that \( \text{ew}[i, j] = 0 \) if \( A_i \) intersects \( A_j \). For \( i < j \), consider the following two cases.

Case 1. If \( l_j < r_i \) then \( A_i \) intersects \( A_j \), and \( \text{ew}[i, j] \) is set to 0 by algorithm \text{compute}\_\text{ew} \).

Case 2. If \( l_j > r_i \), \( l_j < h_i \) and \( h_i < t_j \) then \( A_i \) intersects \( A_j \). Therefore, we set \( \text{ew}[i, j] \) to be 0. This can be simply done in linear time.

Now \( \text{maxw} = \max \{ \text{ew}[i, j] \} \) is the weight of an MWIS in the circular-graph based on \( A \). With this modification, we can find all maximum weight paths using the same method as in Section 2.

It is easy to prove that our algorithm correctly generates all maximum weight independent sets in a weighted circular-arc graph in \( O(\text{max}(n^2, \beta)) \) steps using \( O(n^2) \) space.
4. Conclusion

We presented fast and efficient algorithms for generating all maximum weight independent sets in interval and circular-arc graphs. The algorithms take $O(\max(n^2, \beta))$ steps using $O(n^2)$ space, where $\beta$ is the sum of the number of nodes in all maximum weight independent sets. These algorithms can be used to find maximum weight cliques in the same time and space complexity. Our algorithms can also be applied for finding one maximum weight independent set on interval graphs and circular-arc graphs in $O(n^2)$ steps. Since the best known algorithm for finding the maximum weight independent set in circular-arc graphs is $O(n^{2.10})$, our result is an improvement over the best known result.

References


