Dynamic Programming on Intervals

Takao Asano *

Abstract

We consider problems on intervals which can be solved by dynamic programming. Specifically, we give an efficient implementation of dynamic programming on intervals. As an application, an optimal sequential partition of a graph $G = (V, E)$ can be obtained in $O(m \log n)$ time, where $n = |V|$ and $m = |E|$. We also present an $O(n \log n)$ time algorithm for finding a minimum weight dominating set of an interval graph $G = (V, E)$, and an $O(m \log n)$ time algorithm for finding a maximum weight clique of a circular-arc graph $G = (V, E)$, provided their intersection models of n intervals (arcs) are given.

1 Introduction

Dynamic programming is one of the most popular techniques for designing efficient algorithms and has been applied to many problems. For example, an optimal sequential partition of a graph and a maximum weight clique of a circular-arc graph can be obtained by dynamic programming on intervals [4, 6]. In this paper we propose an efficient implementation of dynamic programming on intervals.

For an interval z, we denote by $x(z)$ and $y(z)$ the left and right endpoints of z, respectively. We assume $x(z)$ and $y(z)$ are integers from 1 to p. Then the problem we consider is: for a set Z of intervals, a set $SZ = \{z_1, z_2, \ldots, z_p\}$ with $y(z_i) = i$ and a real-valued cost function $c(z)$ for $z \in Z$, compute

$$E[i] = \min\{D[j] + C(j, i)\}, \tag{1}$$

for each $1 \leq i \leq p$. Here we assume that the minimum is taken over all the integers $j < i$ contained in $z_i \in SZ$. We also assume that $D[i]$ can be computed from $E[i]$ in

*Department of Mechanical Engineering, Sophia University, Chiyoda-ku, Tokyo 102, Japan. This work was supported in part by Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan under Grant-in-Aid for Co-operative Research (A) 02302047 (1990,1991).

Although this observation can be easily shown by using the segment tree in computational geometry [2],[7], we will prove briefly later in Section 2.2 for completeness. Now we assume Observation 1 and show that each $E[i]$ and $D[i]$ can be computed in $O(\log p)$ time.

2.1 Implemetation of Algorithm DP

In this subsection we give an implementation of Algorithm DP. It is based on a kind of (line) sweep. In the procedure, M is supposed to be a sufficiently large positive constant.

Procedure IMPLDP;
begin {comment computing $E[i]$ and $D[i]$;}
1 $X := \{1, 2, \ldots, p\}$; $DZ := \emptyset$;
2 $z_0 := [1, p]$; $cost(z_0) := M$; $insert(z_0, DZ)$;
3 **for** $i := 1$ **to** p **do begin**
4 **if** $i = 1$ **then** $E[i] := 0$ **else** $E[i] := findmin(z_i, DZ)$;
5 compute $D[i]$ from $E[i]$;
6 $y_i := [i, i]$; $cost(y_i) := D[i] - M$; $insert(y_i, DZ)$;
7 **if** i is the right endpoint of an interval in Z **then**
8 **for each** $z \in Z$ with $y(z) = i$ **do** $delete(z, DZ)$;
9 **if** i is the left endpoint of an interval in Z **then**
10 **for each** $z \in Z$ with $x(z) = i$ **do**
11 **begin** $cost(z) := \alpha c(z)$; $insert(z, DZ)$ **end**
end
end;

The following observations are useful to show the correctness of Procedure IMPLDP.

(A) If we choose M such that $M > 2\sum_{z \in Z} |\alpha c(z)|$, then just before the i-th iteration of Line 3, $csum(x, DZ) > M/2$ for all points $x \in [1, p]$ except integer points $x < i$.
(B) Just before the i-th iteration, $csum(x, DZ) = D[x] + C(x, i)$ for an integer point $x < i$.

We will prove Observations (A) and (B) by induction on i.

Initially (just after Line 2), $csum(x, DZ) = M > M/2$ for all $x \in [1, p]$ and thus, just before the first iteration ($i = 1$), (A) and (B) hold.

We assume that (A) and (B) hold just before the i-th iteration and consider the i-th iteration. Note that $csum(x, DZ) < M/2$ holds only if virtual interval y_x has been inserted into DZ in Line 6. Thus, during the i-th iteration, only $csum(i, DZ)$ can newly become less than or equal to $M/2$ (this will happen when $D[i]$ is less than or equal to $M/2$). Thus, (A) holds immediately after i-th iteration (i.e., just before the $i+1$-th iteration). Next we consider (B). Define $Z_R(x)$ to be the set of intervals in Z having x as the right endpoint, i.e.,

$$Z_R(x) = \{z \in Z \mid y(z) = x\}.$$

Similarly, define $Z_L(x)$ and $Z(x,y)$ for $x < y$ as follows.

$$Z_L(x) = \{z \in Z \mid x(z) = x\},$$

$$Z(x,y) = \{z \in Z \mid x, y \in z\}.$$

Note that $C(x,y) = \alpha \sum_{z \in Z(x,y)} c(z)$. During the i-th iteration, all intervals in $Z_R(i)$ are deleted from DZ and all intervals in $Z_L(i)$ are inserted to DZ. Furthermore, $D[x]$ is not changed for an integer $x < i$. Thus, $Z(x, i+1) = Z(x,i) - Z_R(i)$ and just before the $i+1$-th iteration, we have

$$csum(x, DZ) = D[x] + C(x, i+1),$$

for an integer $x < i$. Similarly, for a sufficiently small $\epsilon > 0$,

$$Z(i, i+1) = (Z(i-\epsilon, i) - Z_R(i)) \cup Z_L(i).$$

Thus,

$$csum(i, DZ) = D[i] + C(i, i+1)$$

after i-th iteration, since y_i is inserted. This implies that (B) holds just before the $i+1$-th iteration. Thus, we complete our proof of (A) and (B).

These observations imply that each $E[i]$ (and $D[i]$) is correctly computed by $E[i] := findmin(z_i, DZ)$ (or $E[i] := 0$) in Line 4 during the i-th iteration when $i = y(z_i)$ and $findmin(z_i, DZ) < M/2$, since $csum(x, DZ) = D[x] + C(x,i)$ for an integer point $x < i$ and

$$findmin(z_i, DZ) = \min_{x \in z_i} \{csum(x, DZ)\}$$

$$= \min_{x<i, x \in z_i, x:integer} \{D[x] + C(x,i)\},$$

by Observations (A) and (B) (note that $csum(i, DZ) > M/2$ just before the i-th iteration). If $findmin(z_i, DZ) > M/2$, then the correct value of $E[i]$ is ∞ and in the procedure $E[i]$ (and $D[i]$) is set a value $> M/2$. Thus we can obtain the following theorem by Observation 1 since the number of operations $findmin$, *insert* and *delete* is $O(q)$.

Theorem 1. All $E[i]$ can be correctly computed in $O(q \log p)$ time.

2.2 Data Structure Supporting $findmin$, *insert* and *delete*

In this subsection, we shall describe a data structure supporting the operations $findmin$, *insert* and *delete* defined before. Since $delete(z, DZ)$ can be replaced by $insert(z', DZ)$ using $z = z'$ and $cost(z') = -cost(z)$, we consider only $findmin$ and *insert*.

We consider a balanced binary search tree $T(X)$ for a set $X = \{1, 2, \ldots, p\}$. That is, $T(X)$ satisfies the following (i)-(iii).

(i) Each node of $T(X)$ has either no children or two children, its left child $\ell(v)$ and its right child $r(v)$ (a node with no children is a *leaf* and a node with children is an *inner node*).

constant time and that $C(j,i)$ is a constant multiple of the total cost of intervals $z \in Z$ that contain both i and j, i.e.,

$$C(j,i) = \alpha \sum_{j,i \in z \in Z} c(z).$$

Furthermore, we assume $E[1] = 0$. This problem can easily be solved by the following Algorithm DP based on dynamic programming:

For $i := 1$ to p, compute $E[i]$ by (1) and $D[i]$.

It is clear that Algorithm DP correctly computes all $E[i]$ (and $D[i]$) since all $D[j]$ $(j < i)$ are already computed when computing $E[i]$, and if we let $q = |Z|$, it requires $O(q+p^2)$ time. In this paper we shall present an efficient implementation of Algorithm DP and thus obtain an $O(q \log p)$ time algorithm for computing all $E[i]$.

As an application, an optimal sequential partition of a graph $G = (V,E)$ can be obtained in $O(m \log n)$ time, where $n = |V|$ and $m = |E|$. Similarly, we can obtain efficient algorithms on interval graphs and circular-arc graphs. For example, a minimum-weight dominating set of a weighted interval graph (circular-arc graph) can be obtained in $O(n \log n)$ $(O(n^2 \log n))$ time and a maximum-weight clique of a weighted circular-arc graph can be obtained in $O(m \log n)$ time, provided that their intersection models are given. These except for the clique algorithm improve the complexity of existing algorithms [3, 5, 6]. For the maximum-weight clique problem on circular-arc graphs, Shih and Hsu [8] has already proposed a faster algorithm with $O(n \log n + m \text{ loglog } n)$ time. However, we believe our algorithm proposed here is easier to implement and more practical.

2 Efficient Implementation

In this section, we present an efficient implementation of Algorithm DP. By this implementation, all $E[i]$ in (1) can be computed in $O(q \log p)$ time and $O(p)$ space.

For the efficient implementation, we consider the following problem. We are given a set of numbers $X = \{1, 2, \ldots, p\}$ $(p \geq 2)$ and a set DZ of intervals each of which has its both endpoinds in X. Also each interval $z \in DZ$ has a cost $cost(z)$. For a number x, $csum(x, DZ)$ is defined as $csum(x, DZ) = \sum_{x \in z \in DZ} cost(z)$ if $DZ \neq \emptyset$ and $csum(x, DZ) = \infty$ if $DZ = \emptyset$. We consider the following three operations.

$$findmin(z, DZ): \text{ return } \min_{x \in z}\{csum(x, DZ)\},$$

$$insert(z, DZ): \text{ replace } DZ \text{ with } DZ \cup \{z\},$$

$$delete(z, DZ): \text{ replace } DZ \text{ with } DZ - \{z\}.$$

A crucial observation is:

Observation 1. Each of the operations *findmin*, *insert* and *delete* can be done in $O(\log p)$ time with a data structure of $O(p)$ space constructed by an $O(p)$ time preprocessing.

(ii) Each node v of $T(X)$ contains a key $key(v) \in X$ and keys are arranged in symmetric order: each $i \in X$ is stored in exactly one leaf of $T(X)$ as a key, and each inner node v has $key(v)$ satisfying

$$\max_{u \in L(v)} \{key(u)\} = key(v) < \min_{u \in R(v)} \{key(u)\},$$

where $L(v)$ denotes the set of descendants of the left child $\ell(v)$ of v and $R(v)$ denotes the set of descendants of the right child $r(v)$ of v.

(iii) $depth(v) = O(\log p)$ for each node v of $T(X)$.

Note that this balanced binary search tree $T(X)$ can be constructed in $O(p)$ time and $O(p)$ space. For simplicity, we write x_i to denote the leaf containing i as a key. We consider two values $cmin(v)$ and $cin(v)$ for each node v of $T(X)$. We maintain $cmin(v)$ and $cin(v)$ so that they satisfy

$$cmin(v) = \min\{cmin(\ell(v)) + cin(\ell(v)),\ cmin(r(v)) + cin(r(v))\} \qquad (2)$$

for each inner node v of $T(X)$. Let $P(u, v)$ be the simple path of $T(X)$ from node u to node v. We also use $P(u, v)$ to denote the set of nodes on the path $P(u, v)$. Initially (when $DZ = \emptyset$), $cmin(v) = 0$ and $cin(v) = 0$ for each node v of $T(X)$. Equation (2) together with $cmin(x_i) = 0$ for each leaf x_i will imply $cmin(v) = \min\{\sum_{u \in P(v,x_i)-\{v\}} cin(u)\}$ for each inner node v of $T(X)$, where the minimum is taken over all the leaves x_i that are descendants of v.

We first consider $insert(z, DZ)$. Let $z = [i, j]$ $(i \leq j)$. Then $insert(z, DZ)$ can be done as follows. Let r_{ij} be the nearest common ancestor of x_i and x_j. Consider the paths $P(r_{ij}, x_i)$ and $P(r_{ij}, x_j)$ from r_{ij} to x_i and x_j. Let v_i be the deepest node on $P(r_{ij}, x_i)$ such that v_i is the right child of a node on $P(r_{ij}, x_i)$ if such v_i exists, otherwise we set $v_i := r_{ij}$. Similarly, let v_j be the deepest node on $P(r_{ij}, x_j)$ such that v_j is the left child of a node on $P(r_{ij}, x_j)$ if such v_j exists, otherwise we set $v_j := r_{ij}$. Define $R(r_{ij}, v_i)$ and $L(r_{ij}, v_j)$ as follows. If $v_i = v_j = r_{ij}$ then $R(r_{ij}, v_i) = L(r_{ij}, v_j) = \{r_{ij}\}$. If $v_i = r_{ij} \neq v_j$ then $R(r_{ij}, v_i) = \{\ell(r_{ij})\}$. If $v_i \neq r_{ij} = v_j$ then $L(r_{ij}, v_j) = \{r(r_{ij})\}$. If $v_i \neq r_{ij}$ then

$$R(r_{ij}, v_i) = \{v = r(p(v)) \mid p(v) \in P(\ell(r_{ij}), p(v_i)),\ v \text{ is not in } P(\ell(r_{ij}), p(v_i))\}$$

that is, $v \in R(r_{ij}, v_i)$ if and only if v is the right child of a node on the path $P(\ell(r_{ij}), p(v_i))$ but v itself is not on the path $P(\ell(r_{ij}), p(v_i))$. Thus, $v_i \in R(r_{ij}, v_i)$ if $v_i \neq r_{ij}$. If $v_j \neq r_{ij}$ then

$$L(r_{ij}, v_j) = \{v = \ell(p(v)) \mid p(v) \in P(r(r_{ij}), p(v_j)),\ v \text{ is not in } P(r(r_{ij}), p(v_j))\}$$

that is, $v \in L(r_{ij}, v_j)$ if and only if v is the left child of a node on the path $P(r(r_{ij}), p(v_j))$ but v itself is not on the path $P(r(r_{ij}), p(v_j))$. Thus, $v_j \in L(r_{ij}, v_j)$ if $v_j \neq r_{ij}$. Note that, for each $k \in X$, $k \in z = [i, j]$ if and only if there is exactly one node $v \in R(r_{ij}, v_i) \cup L(r_{ij}, v_j)$ such that x_k is a descendant of v. We first set $cin(v) := cin(v) + cost(z)$ for each $v \in R(r_{ij}, v_i) \cup L(r_{ij}, v_j)$. Next, we have to modify $cmin(u)$ for

all $u \in P(r_{ij}, p(v_i)) \cup P(r_{ij}, p(v_j)) \cup P(r, r_{ij})$, since $cin(v)$ for all $v \in R(r_{ij}, v_i) \cup L(r_{ij}, v_j)$ are now changed ($P(r, r_{ij})$ is the path from the root r to r_{ij}). This can be done by traversing nodes along the paths $P(r_{ij}, p(v_i))$, $P(r_{ij}, p(v_j))$ and $P(r, r_{ij})$ in decreasing order of their *depth* and looking their children. Thus, $insert(z, DZ)$ can be done in $O(\log p)$ time, since the $depth(v)$ of each node v of $T(X)$ is $O(\log p)$.

Next we consider $findmin(z, DZ)$. Let $z = [i, j]$ $(i \leq j)$. Let r_{ij}, $P(r_{ij}, x_i)$, $P(r_{ij}, x_j)$, v_i, v_j, $R(r_{ij}, v_i)$, $L(r_{ij}, v_j)$ and $P(r, r_{ij})$ be the same as above. If we let

$$cs(v) := cmin(v) + \sum_{u \in P(r,v)} cin(u)$$

for each $v \in R(r_{ij}, v_i) \cup L(r_{ij}, v_j)$, then $\min_{v \in R(r_{ij}, v_i) \cup L(r_{ij}, v_j)}\{cs(v)\}$ is the desired value $findmin(z, DZ)$. This value can be computed in $O(\log p)$ time by traversing nodes along the paths $P(r_{ij}, p(v_i))$, $P(r_{ij}, p(v_j))$ and $P(r, r_{ij})$ in decreasing order of their *depth* and looking their children. Thus, $findmin(z, DZ)$ can also be done in $O(\log p)$ time. Note that $T(X)$ with $cmin(v)$ and $cin(v)$ for each node v of $T(X)$ can be represented in $O(p)$ space and is almost the same as the segment tree in computational geometry [2], [7].

3 Applications and Related Problems

In this section, we present some applications of an efficient implementation of Algorithm DP. The first application is to optimal sequential partitions of graphs proposed by Kernighan [6].

3.1 Optimal Sequential Partitions of Graphs

We are given a graph $G = (V, E)$, a cost function $c : E \rightarrow R^+$, a weight function $w : V \rightarrow R^+$ and a positive number K. Here we assume the vertices of the graph are labeled from 1 to n. For a subset $B = \{b_1, b_2, \ldots, b_s\}$ of V such that $b_1 = 1$ and $b_i < b_j$ for $i < j$, we obtain a family of graphs $G(B) = \{G_1, G_2, \ldots, G_s\}$, where each G_i is the subgraph of G induced by the set $\{b_i, b_i + 1, \ldots, b_{i+1} - 1\}$ (for convenience, we assume $b_{s+1} - 1 = n$). We will call $P = G(B)$ a sequential partition (or partition, for short) of the graph $G = (V, E)$ defined by B. The cost of partition $P = G(B)$, denoted by $cost(P)$, is defined to be the total cost of the edges of G joining different graphs in $G(B)$. A partition $G(B)$ is called *admissible* if all the graphs in $G(B)$ have the weights $< K$ (the weight of $G_i \in G(B)$ is equal to $w(b_i) + w(b_i + 1) + \cdots + w(b_{i+1} - 1)$). An admissible partition P of G is called *optimal* if $cost(P) \leq cost(P')$ for all admissible partitions P' of G.

We would like to find an optimal partition of G. Kernighan [6] first proposed an algorithm for finding an optimal partition and later Kaji and Ohuchi [5] corrected the complexity analysis of his algorithm. They showed that the complexity of the Kernighan's algorithm is $O(n^2)$.

Here we will show that the Kernighan's algorithm can be implemented to run in $O(m \log n)$ time, where $m = |E|$. We first review the Kernighan's algorithm. For a

subset B_i of $\{1, 2, \ldots, i\}$ containing 1 and i, the corresponding partition $G(B_i)$ of G will be called a *partial admissible partition* if all graphs G_j in $G(B_i)$ except the graph containing the vertex i have the weights $\leq K$. Let $T(i)$ be the minimum cost of a partial admissible partition $G(B_i)$ of G for a subset $B_i \subset \{1, 2, \ldots, i\}$ containing 1 and i. Then Kernighan observed the following.

$$T(i) = \min\{T(j) - A(j-1, i) + A(i-1, i)\}, \tag{3}$$

where minimum is taken over all j's such that $w(j) + w(j+1) + \cdots + w(i-1) \leq K$ and $A(x, y)$ $(x < y)$ is the total cost of the edges which contain both x and y (we consider an edge $e = (i, j)$ to be an interval $[i, j]$ on the real line).

Thus, if we let $E[i] := T(i) - A(i-1, i)$, $D[j] := T(j)$ and $C(j-1, i) := -A(j-1, i)$ by setting $\alpha := -1$, then we can rewrite (3) as follows.

$$E[i] = \min\{D[j] + C(j-1, i)\}.$$

This is almost the same as (1) and can be solved by a similar method described before. Here we reduce (3) to the same form as in (1). We consider the following intervals and their costs.

$$Z(E) = \{z(e) = [2x + 1, 2y] \mid e = [x, y] \in E\},$$

$$Z(V) = \{z(i) = [2x(i), 2i] \mid i \in V, x(i) = \min\{j \mid w(j) + w(j+1) + \cdots + w(i-1) \leq K\}\},$$

and

$$c(z(e)) = c(e) \text{ for } z(e) \in Z(E).$$

Let $Z = Z(E)$ and $SZ = Z(V) \cup \{z = [2i - 1, 2i - 1] \mid i = 1, 2, \ldots, n\}$. We use the same notation as in Section 3 ($q = m$ and $p = 2n$). Note that if we set $\alpha := -1$ then $C(2j, 2i) = -A(j-1, i)$. Thus if we let $E[2i] := T(i) - A(i-1, i)$ and $D[2j] := T[j]$, then we have the problem of computing $E[2i]$ and $D[2i]$ defined by (1) ($E[2i-1]$ and $D[2i-1]$ will become ∞). Note that all $A(i-1, i) = -C(2i, 2i)$ can be computed in $O(m)$ time in advance and we store them in a table. Thus we can compute $D[2i]$ from $E[2i]$ in costant time. Since there are $m + n$ intervals and $2n$ endpoints, all $E[2i]$ and $D[2i]$ can be computed in $O(m \log n)$ time by the method in Section 2.

3.2 Maximum Weight Clique of a Circular-Arc Graph

There are several variations of the problem described in Section 1. Here we consider one variation which appears in Hsu's algorithm for finding a maximum weight clique of a circular-arc graph [4] and dominates the complexity of the algorithm. We are given a sequence $Z = (z_1, z_2, \ldots, z_q)$ of intervals and a subsequence $SZ = (z_{\sigma(1)}, z_{\sigma(2)}, \ldots, z_{\sigma(r)})$ of Z and a real-valued cost function $c(z)$ for $z \in Z - SZ$, and compute

$$E[i] = \max\{D[j] + C(j, i)\}, \tag{4}$$

for each $1 \leq i \leq r$. Here we assume that the maximum is taken over all the integers $j < i$ such that $\ell(z_{\sigma(j)})$ is contained in $z_{\sigma(i)} \in SZ$. We also assume that $D[i]$ can be computed from $E[i]$ in constant time and that $C(j, i)$ is a constant multiple of the total

cost of intervals $z_k \in Z - SZ$ such that z_k appears after $z_{\sigma(j)}$ and before $z_{\sigma(i)}$ in the sequence Z (i.e., $\sigma(j) < k < \sigma(i)$) and contains both $\ell(z_{\sigma(i)})$ and $\ell(z_{\sigma(j)})$. Furthermore, we assume $E[1] = 0$.

This problem can also be solved by a similar method as one described before. Thus, a maximum weight clique of a circular-arc graph $G = (V, E)$ can be obtained in $O(m \log n)$ time and $O(n)$ space provided that G is given by its intersection model of arcs of a circle, where $m = |E|$ and $n = |V|$. See [1] for details. It should be noted that Shin and Hsu [8] has already proposed a faster algorithm with $O(n \log n + m \log\log n)$ time based on another elegant observation. However, we believe that our algorithm is easier to implement and practically runs faster.

3.3 Minimum Weight Dominating Set

A dominating set of a graph $G = (V, E)$ is a subset U of V such that, for every vertex $v \in V$, $v \in U$ or there is an edge $e = (u, v) \in E$ connecting a vertex $u \in U$ and v. A minimum weight dominating set is a dominating set of minimum weight, where each vertex has a positive weight. To find a minimum dominating set of an interval graph, we first consider a shortest path problem on the interval graph. Here the length of a path is the sum of the weights of vertices on the path. We assume an interval graph $G = (V, E)$ is given by its intersection model of intervals $Z = \{z_1, z_2, \ldots, z_n\}$ on the line. We also assume $x(z_i) \leq x(z_j)$ for $i < j$. We now want to compute the distances of shortest paths from z_1 to other vertices z_i. Let $D[i]$ be the distance of a shortest path from z_1 to z_i. Then $D[1] = w(z_1)$ and, for $i = 2, 3, \ldots, n$,

$$D[i] = \min_{z_j \cap z_i \neq \emptyset} \{D[j] + w(z_i)\}.$$

Thus, if we let $E[i] := D[i] - w(z_i)$ then we have

$$E[i] = \min_{z_j \cap z_i \neq \emptyset} \{D[j]\}.$$

This can be solved by a similar method described before. In fact, a simpler method can be applied and all $E[i]$ can be computed in $O(n \log n)$ time.

To find a minimum weight dominating set of an interval graph $G = (V, E)$, we consider new intervals obtained from Z. Specifically, we consider an interval z_i' with $w(z_i') = w(z_i)$ corresponding to z_i as follows: $x(z_i') = x(z_i)$ and $y(z_i')$ is the leftmost right endpoint $y(z)$ of the intervals $z \in Z$ such that $y(z_i) < x(z)$. Let z_0 be a virtual interval with weight 0 such that $x(z_0) = x(z_1)$ and $y_0 = \min\{y(z) \mid z \in Z\}$. Then the distance of a shortest path from z_0 to z_n' is the weight of a minimum dominating set of the graph $G = (V, E)$. Thus, a minimum weight dominating set of an interval graph can be obtained in $O(n \log n)$ time. Since a minimum weight dominating set of a circular-arc graph $G = (V, E)$ can be obtained by solving n times the minimum weight dominating set problems of interval graphs, it can be obtained in $O(n^2 \log n)$ time. This improves the previous complexity [3].

References

[1] T. Asano, *An faster algorithm for finding a maximum weight clique of a circular-arc graph,* Technical Report of Institut für Operations Research, Universität Bonn, 90624-OR, 1990.

[2] J.L. Bentley, *Decomposable searching problems*, Information Processing Letters, 8 (1979), pp.244-251.

[3] A.A. Bertossi and S. Moretti, *Parallel algorithms on circular-arc graphs* Information Processing Letters, 33 (1989/1990), pp.275-281.

[4] W.-L. Hsu, *Maximum weight clique algorithms for circular-arc graphs and circle graphs*, SIAM Journal on Computing, 14 (1985), pp. 224-231.

[5] T. Kaji and A. Ohuchi, *Optimal sequential partitions of graphs by branch and bound*, Technical Report 90-AL-10, Information Processing Society of Japan, 1990.

[6] B.W. Kernighan, *Optimal sequential partitions of graphs*, Journal of ACM, 18 (1971), pp.34-40.

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

[8] W.-K. Shih and W.-L. Hsu, *An $O(n \log n + m \log\log n)$ maximum weight clique algorithm for circular-arc graphs*, Information Processing Letters, 31 (1989), pp.129-134.