View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Kyutacar : Kyushu Institute of Technology Academic Repository

九州工業大学学術機関リポジトリ Kyutacar Kyushu Institute of Technology Academic Repository

Simple Efficient Algorithm for MPQ-tree of an Interval Graph

著者	Saitoh Toshiki, Kiyomi Masashi, Uehara Ryuhei
journal or publication title	Proceedings of the KOREA-JAPAN Joint Workshop on Algorithms and Computation
volume	2007
page range	121-126
year	2007-08-09
URL	http://hdl.handle.net/10228/00006366

Simple Efficient Algorithm for $\mathcal{MPQ}$-Tree of an Interval Graph

Toshiki SAITOH[1] **Masashi KIYOMI**[1] **Ryuhei UEHARA**[1]

[1]Japan Advanced Institute of Science and Technology
Asahidai 1-1, Nomi, Ishikawa, Japan

E-mail: {toshikis,mkiyomi,uehara}@jaist.ac.jp

Abstract

$\mathcal{MPQ}$-tree is an informative data structure for an interval graph. We propose a simple algorithm that constructs an $\mathcal{MPQ}$-tree for an interval graph $G = (V, E)$ given in the interval representation. If endpoints of the interval representation are already sorted, the algorithm runs in $O(|V|)$ time and space. The complexities are theoretically optimal. Further, our algorithm is much simpler than the previously known algorithms.

1 Introduction

Interval graphs were introduced independently in the 1950's by Hajós, a mathematician, and Benzer, a molecular biologist[4]. A number of important applications for interval graphs have been studied since then, including applications on modeling the topological structures of the DNA molecules [2, 8]. In such applications, we have to deal with tons of data given in interval representations.

Recognition problem is one of the basic problems in the algorithmic graph theory. For the interval graph recognition problem of an interval graph, there are two approaches from the historical point of view. Given a graph, algorithms in early time check if it has a corresponding $\mathcal{PQ}$-tree (e.g., [1, 5]). On the other hand, recent algorithms simply check if a given graph can be embedded into a specific interval representation, by sorting the vertices several times with lexicographically breadth first search (LexBFS) (e.g., [3]).

It is well known that it is quite complicated to construct the $\mathcal{PQ}$-tree although some algorithms run in linear time. These algorithms require to apply dozens of templates [1, 5]. However, once a $\mathcal{PQ}$-tree is constructed, we can use it as an informative and compact data structure of an interval graph. In this paper, we focus on $\mathcal{MPQ}$-trees (labeled $\mathcal{PQ}$-trees) introduced by Korte and Möhring [5]. For an interval graph $G = (V, E)$, the $\mathcal{MPQ}$-tree is uniquely determined up to the isomorphism. It contains information of all vertices and edges of G. Using the $\mathcal{MPQ}$-tree, we can solve the graph isomorphism problem for interval graphs in linear time (see also [6]). The $\mathcal{MPQ}$-tree takes only $O(|V|)$ space, and essentially represents all possible interval representations, though there exist several interval representations for an interval graph. Hence, we can use it as a good basic data structure of an interval graph that can be used for some applications including random generation, enumeration, and other graph problems.

In [5], Korte and Möhring reduce the number of templates for construction of the $\mathcal{MPQ}$-tree from the algorithm by Booth and Lueker [1]. However, the algorithm still has to handle several templates. Hence, though the algorithm runs in $O(|V| + |E|)$ time and space, it is still complicated.

Recently, McConnell and Montgolfier propose a framework of tree structures that generalizes $\mathcal{PQ}$-trees, the modular decompositions, and certain decompositions of boolean functions [7]. In the paper, they state that the $\mathcal{PQ}$-tree (and hence $\mathcal{MPQ}$-tree) can be constructed in $O(|V|)$ time if the endpoints of the intervals are integers from 1 to $O(1)$, and in $O(|V| \log |V|)$ time if they are given as real numbers.

In this paper, we focus on the construction of the $\mathcal{MPQ}$-tree for an interval graph $G = (V, E)$. Especially, we concentrate on the case that inputs are given in interval representations. This assumption is reasonable from the practical point of view. By this assumption, the input size is $O(|V|)$ even if $|E| = \Theta(|V|^2)$. We propose an algorithm that constructs the $\mathcal{MPQ}$-tree in $O(|V|)$ time and space if the endpoints of the intervals are given in an ordered list sorted by the coordinates. The algorithm is quite simple compared to the previously known ones. It only uses basic data structures like arrays and stacks, and no templates are required any longer. As a result, the implementation of the algorithm is quite easy and it runs quickly even in a practical sense.

We can practically use our algorithm with the recent LexBFS algorithms to construct an $\mathcal{MPQ}$-tree for a given interval graph in the graph representation. Given graph $G = (V, E)$, we first determine whether or not G is an interval graph in $O(|V| + |E|)$ time and $O(|V|)$ space by the recognition algorithm based on LexBFS described in [3]. We obtain a specific interval representation as a by-product of the algorithm, when G is an interval graph. We use our algorithm to construct the $\mathcal{MPQ}$-tree from the interval representation in $O(|V|)$ time and $O(|V|)$ space. This approach admits us to recognize an interval graph and to obtain the $\mathcal{MPQ}$-tree as in [5] in a quite efficient way.

2 Preliminaries

The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N_G(v) = \{u \in V | \{u, v\} \in E\}$, and the *degree* of a vertex v is $|N_G(v)|$ denoted by $\deg_G(v)$. A subset $V' \subseteq V$ is a *clique* in G if every pair of vertices in V' are joined by an edge in G. Given graph $G = (V, E)$, a vertex $v \in V$ is *simplicial* in G if $G[N_G(v)]$ (the graph induced by $N_G(v)$ from G) is a clique in G.

A graph (V, E) with $V = \{v_1, v_2, \cdots, v_n\}$ is an *interval graph*, if there is a set of intervals $\mathcal{I} = \{I_{v_1}, I_{v_2}, \cdots, I_{v_n}\}$ such that $\{v_i, v_j\} \in E$ if and only if $I_{v_i} \cap I_{v_j} \neq \emptyset$ for each i and j with $1 \leq i, j \leq n$. We call the set $\mathcal{I}$ of intervals *interval representation* of the graph.

We define basic notations of an interval representation.

Definition 1 *For an interval i, we denote the coordinate of the left endpoint by $L(i)$, and that of the right endpoint by $R(i)$.*

Definition 2 *Interval i and interval j overlap if and only if*
$L(i) < L(j),\ R(i) \geq L(j),$ *and* $R(i) < R(j),$ *or* $L(j) < L(i),\ R(j) \geq L(i),$ *and* $R(j) < R(i).$

A $\mathcal{PQ}$*-tree* is a rooted tree T whose internal nodes are of two types, $\mathcal{P}$- and $\mathcal{Q}$-nodes. Each leaf of T is labeled with distinct maximal clique of G. The *frontier* of a $\mathcal{PQ}$-tree T is the permutation of the maximal cliques obtained by ordering the leaves of T from left to right. $\mathcal{PQ}$-tree T corresponds to an interval graph G as follows.

Definition 3 [1] *$\mathcal{PQ}$-tree T corresponds to an interval graph G, if and only if, for every $\mathcal{PQ}$-tree T' obtained from T by applying the following operations (1) and (2) a finite number of times, there is a consecutive arrangement of the maximal cliques on G that represents for the frontier of T'.*

(1) arbitrarily permute the successor nodes of a $\mathcal{P}$-node, or

(2) reverse the order of the successor nodes of a $\mathcal{Q}$-node.

An $\mathcal{MPQ}$*-tree* , which stands for *modified $\mathcal{PQ}$-tree*, was developed by Korte and Möhring [?] to simplify the algorithm. An $\mathcal{MPQ}$-tree T^* assigns sets of vertices (or intervals from the view of interval representation) to the nodes of a $\mathcal{PQ}$-tree T representing an interval graph $G = (V, E)$. Empty set may be assigned to some nodes. A $\mathcal{P}$-node is assigned only one set, while a $\mathcal{Q}$-node may have multiple sets of its sons (ordered from left to right according to the ordering of the sons). For a $\mathcal{P}$-node P, this set consists of those vertices of G contained in all maximal cliques represented by the subtree rooted at P in T, but in no other cliques.

For a $\mathcal{Q}$-node Q, the definition is more involved. Let $Q_1, \cdots, Q_m$ be the set of the sons (in consecutive order) of Q, and let T_i be the subtree of T with root Q_i (note that $m \geq 3$). We then assign a set S_i, called *section*, to Q for each Q_i. Section S_i contains all vertices that are contained in all maximal cliques of T_i and some other T_j, but not in any clique belonging to some other subtree of T that is not below Q.

Key properties of $\mathcal{MPQ}$-trees are summarized as follows:

Theorem 4 [5, Theorem 2.1] *Let T be a $\mathcal{PQ}$-tree for an interval graph $G = (V, E)$ and let T^* be the associated $\mathcal{MPQ}$-tree. Then we have the following:*

(a) T^* can be obtained from T in $O(|V| + |E|)$ time and represents G in $O(|V|)$ space.

(b) Each maximal clique of G corresponds to a path in T^* from the root to a leaf, where each vertex $v \in V$ is as close as possible to the root.

(c) In T, each vertex v appears in either one leaf, one $\mathcal{P}$-node, or consecutive sections $S_i, S_{i+1}, \cdots, S_{i+j}$ in a $\mathcal{Q}$-node with $j > 0$.

Property (b) is the essential property of an $\mathcal{MPQ}$-tree. For example, the root of T^* contains all vertices belonging to all maximal cliques, and the leaves contain the simplicial vertices of G. In [5], Korte and Möhring did not state Theorem 4(c) explicitly. Theorem 4(c) is immediately obtained from the fact that the maximal cliques containing a fixed vertex occur consecutively in T.

Other key properties of $\mathcal{MPQ}$-tree is summarised as follows:

Lemma 5 [5] *Let N be a $\mathcal{Q}$-node. Let $S_1, \ldots, S_m$ (in this order) be the sections of N, and let V_i denote the set of vertices occurring below S_i in the $\mathcal{MPQ}$-tree T with $1 \le i \le m$. Then we have the following:*

(d) $S_{i-1} \cap S_i \neq \emptyset$ *for* $i = 2, \ldots, m$.

(e) $S_1 \subset S_2$ *and* $S_{m-1} \supset S_m$.

(f) $V_1 \neq \emptyset$ *and* $V_m \neq \emptyset$.

(g) $S_i \cap S_{i+1} \setminus S_1 \neq \emptyset$ *and* $S_{i-1} \cap S_i \setminus S_m \neq \emptyset$ *for* $i = 2, \ldots, m-1$.

(h) $S_{i-1} \neq S_i$ *for* $i = 2, \ldots, m-1$.

Korte and Möhring state the properties from (d) to (g) in [5, Lemma 2.2]. The property (h) is necessary to determine the $\mathcal{MPQ}$-tree of an interval graph uniquely. Although Korte and Möhring did not mention the fact [5, 9], we note that two algorithms in [5] certainly construct the $\mathcal{MPQ}$-tree with property (h).

3 Algorithm

The outline of our algorithm is as follows.

[0] (sort given interval representation I if necessary;)

[1] convert I to a "compact" and "ordered" interval representation I';

[2] partition intervals into sets such that each set corresponds to a ($\mathcal{P}$- or $\mathcal{Q}$-) node in the $\mathcal{MPQ}$-tree.

[3] determine the parent-child relations of the nodes, and create sections of $\mathcal{Q}$-nodes.

Figure 1(a) is an example of input interval representation. We assume that, after step 0, we have an interval representation in the form of a linked list of endpoints; e.g., $(A, B, C, D, a, c, E, F, d, G, f, b, e, g)$, where the upper and lower case letters stand for the left and right endpoints, respectively. We also assume that each endpoint has its own coordinate. An interval representation is said to be *compact* if it has no redundancy (Figure 1(b)). More precisely, the compact interval representation is defined as follows:

Definition 6 *An interval representation I of an interval graph G is* compact *if and only if it satisfies the conditions below.*

1. *Coordinates of endpoints of intervals in I are consecutive integers (assume that the values vary in $\{0, 1, \ldots, K\}$) for some positive integer K.*
2. *There exists at least one endpoint whose coordinate is k for every $k \in \{0, 1, \ldots, K\}$.*
3. $\{i \in I \mid k \in i\}$ *and* $\{i \in I \mid k+1 \in i\}$ *do not include each other, for every* $k \in \{0, 1, \ldots, K-1\}$.

The algorithm uses an array of lists storing endpoints of intervals to represent a compact interval representation; Figure 1(c) shows of the data structure of the compact interval representation drawn in Figure 1(b). A compact interval representation graph has some good properties. For example, the coordinates of endpoints are bounded by n. See [10] for the details.

Lemma 7 *Let T be an $\mathcal{MPQ}$-tree corresponding to an interval graph G. Each $\mathcal{Q}$-node q on T consists of intervals I' on any interval representation I of G, where I' satisfies the following properties.*

1. *$i \in I'$ overlaps with some other interval in I', or*

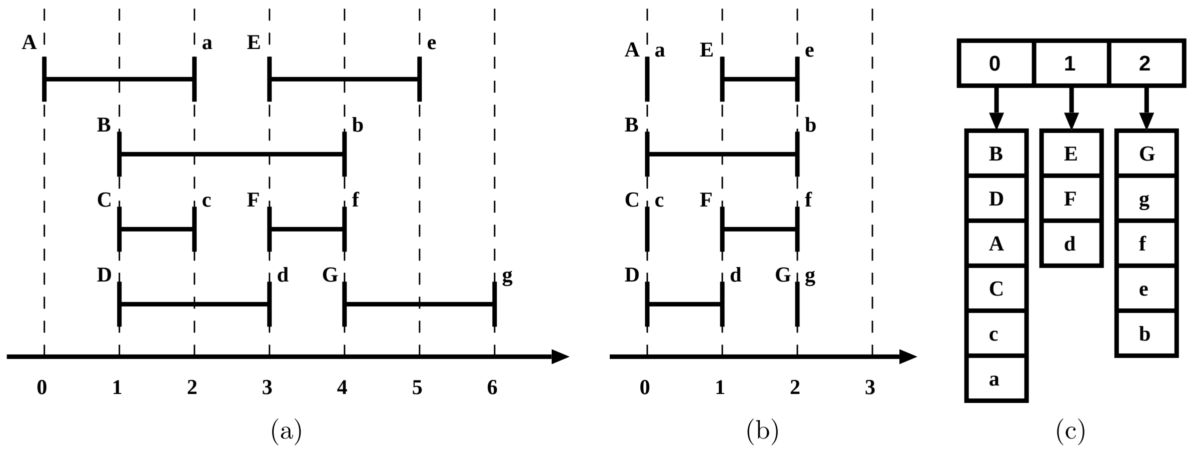

Figure 1: (a): an input interval representation. (b): the compact interval representation corresponding to (a). (c): data structure of ordered compact interval representation.

2. $i \in I'$ is union of the other intervals in I'.

We here introduce an ordering of intervals that makes our algorithm simple.

Definition 8 *We say that a compact interval representation is* ordered *if the representation satisfy the following conditions:*

(1) Left endpoints $L(i)$ precede right endpoints $R(j)$ with $L(i) = R(j)$.

(2) A left endpoint of an interval i precedes any left endpoints of intervals j with $L(i) = L(j)$ and $R(i) > R(j)$.

(3) A right endpoint of an interval i precedes any right endpoints of intervals j with $R(i) = R(j)$ and $L(i) < L(j)$.

(4) For a pair i and j with $R(i) = R(j)$ and $L(i) = L(j)$, the right endpoint of i precedes the right endpoint of j if and only if the left endpoint of j precedes the left endpoint of i.

Lemma 9 *For any given compact interval representation (in a form of an array of lists storing endpoints of intervals), we can compute the ordered compact interval representation (in the same form) in $O(n)$ time and $O(n)$ space.*

Proof. (Outline.) We sweep the endpoints in a given compact interval representation one by one. When we read a left endpoint, we only reserve a place for the endpoint. When we read a right endpoint, we determine the index of the interval, and write it to the places for both the left and right endpoints. Using an array that maintains the correspondence of left and right endpoints, we can write the index in $O(1)$ time. Thus, the ordered compact interval representation can be computed in $O(n)$ time and space. ∎

For the ordered compact interval representation in an array of lists storing endpoints of intervals, the algorithm partitions it into sets of nodes in step (2), and constructs the $\mathcal{MPQ}$-tree in step (3). Hereafter, for lack of space, we only show how to partition intervals into sets of nodes.

3.1 Partition of intervals

The partition of intervals consists of three substeps;

[2.1] find all $\mathcal{Q}$-nodes.

[2.2] create sections in every $\mathcal{Q}$-node obtained in step [2.1].

[2.3] create $\mathcal{P}$-nodes.

```
function find-Q-nodes :
  input: list of endpoints on ordered compact interval representation of interval graph $G$;
  stack: stack;
  array of integer: state;
  /* For interval $i$, state[$i$]=                                                  */
  /*   0: right endpoint of $i$ is not processed, and $i$ belongs to no $Q$-node.      */
  /*   1: right endpoint of $i$ is not processed, and $i$ belongs to a certain $Q$-node. */
  /*   2: right endpoint of $i$ is processed.                                          */
 1 :  initialize stack and state;
 2 :  foreach endpoint $i$ do
 3 :     if $i$ is a left endpoint then
 4 :        push $i$ to stack;
 5 :     else if the top of stack is the left endpoint of $i$ then
 6 :        pop the left endpoint from stack;
 7 :     else
 8 :        if state[$i$] $= 0$ then
 9 :           make a new $Q$-node $q$ of intervals on stack from the left endpoint of $i$ to the top;
10 :       else
11 :           merge intervals in the $Q$-node to which $i$ belongs and
              intervals on stack from the left endpoint of $i$ to the top into a new $Q$-node $q$;
12 :       end if;
13 :       set states of intervals in $q$ whose state is 0 to 1;
14 :       set state[$i$] to 2;
15 :       if every state of interval in $q$ is 2 then
16 :           remove endpoints of intervals in $q$ from stack;
17 :       end if
18 :    end if
19 :  end for
end function
```

Figure 2: The algorithm to find every Q-node.

Since substeps [2.2] and [2.3] are straightforward, we only describe substep [2.1].

Our algorithm maintains Q-node candidates in a stack. The algorithm is shown in Figure 2.

On Step 2, endpoints are searched according to the ordering under the conditions (1) to (4) in Definition 8. This can be done by simply sweeping the data structure of the compact interval representation.

Since intervals overlapping with no other intervals will be thrown off at Step 6, this algorithm finds intervals overlapping to some other intervals. Since Q-nodes whose intervals have been processed will be removed on Steps 15 and 16, a Q-node obtained by this algorithm exactly contains intervals that must be in the node.

We here show the complexity. A naive implementation of this algorithm does not achieve $O(n)$ time. We introduce a good data structure to run the algorithm in $O(n)$ time. We maintain each Q-node candidate as a pair of two indices in the stack. The lemma below guarantees the validness of the method.

Lemma 10 *Vertices in a Q-node candidate are consecutive on stack.*

Therefore, the algorithm can maintain a set of Q-node candidates by top and bottom of the set on stack. Thus, two sets of Q-node candidates are merged in $O(1)$ time.

Lemma 11 *The total computational cost of merging Q-nodes in the algorithm is $O(n)$.*

Proof. At first, the number of sets of Q-node candidates is at most n. One merger, which takes $O(1)$ time, decreases the number of sets by one. The total number of mergers is thus at most n. Therefore, it takes $O(n)$ time in total to merge the Q-node candidate sets. ∎

We obtain the following theorem from the Lemmas 10 and 11.

Theorem 12 *If the input is given in the interval representation with the endpoints sorted by the coordinates, we can obtain an $\mathcal{MPQ}$-tree corresponding to G in $O(n)$ time.*

4 Concluding Remarks

By combining with the algorithm in [3], we can recognize an interval graph and we can obtain the $\mathcal{MPQ}$-tree of the interval graph efficiently. We can use the $\mathcal{MPQ}$-tree for various problems on interval graphs. For example, we may be able to use the $\mathcal{MPQ}$-trees for random generation and/or enumeration of all possible affirmative interval representations. One of the interesting problems is listing of all unlabeled interval graphs (up to) n vertices, where "unlabeled" means that we do not admit to generate two isomorphic graphs. Random generation of all unlabeled interval graphs (up to) n vertices is another interesting problem.

References

[1] K.S. Booth and G.S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using *PQ*-Tree Algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

[2] A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey.* SIAM, 1999.

[3] D.G. Corneil, S. Olariu, and L. Stewart. The Ultimate Interval Graph Recognition Algorithm? In *Proc. 9th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 175–180. ACM, 1998.

[4] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Annals of Discrete Mathematics 57. Elsevier, 2nd edition, 2004.

[5] N. Korte and R.H. Möhring. An Incremental Linear-Time Algorithm for Recognizing Interval Graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989.

[6] G.S. Lueker and K.S. Booth. A Linear Time Algorithm for Deciding Interval Graph Isomorphism. *Journal of the ACM*, 26(2):183–195, 1979.

[7] R. M. McConnell and F. de Montgolfier. Algebraic Operations on PQ Trees and Modular Decomposition Trees. In *Graph-Theoretic Concepts in Computer Science (WG 2005)*, pages 421–432. Lecture Notes in Computer Science Vol. 3787, Springer-Verlag, 2005.

[8] T.A. McKee and F.R. McMorris. *Topics in Intersection Graph Theory.* SIAM, 1999.

[9] R.H. Möhring. Personal communication. 2003.

[10] R. Uehara and Y. Uno. On Computing Longest Paths in Small Graph Classes. In *ISAAC '04*, pages 871–883. Lecture Notes in Computer Science Vol. 3341, Springer-Verlag, 2004.