Electronic Journal of Graph Theory and Applications **5** (1) (2017), 7–20

Electronic Journal of
Graph Theory and Applications

Efficient maximum matching algorithms for trapezoid graphs

Phan-Thuan Do[a], Ngoc-Khang Le[b], Van-Thieu Vu[a]

[a]*Department of Computer Science, Hanoi University of Science and Technology, Vietnam*
[b]*LIP, ENS de Lyon, Lyon, France*

thuan.dophan@hust.edu.vn, ngoc-khang.le@ens-lyon.fr, thieu.vuvan@hust.edu.vn

Abstract

Trapezoid graphs are intersection graphs of trapezoids between two horizontal lines. Many NP-hard problems can be solved in polynomial time if they are restricted on trapezoid graphs. A matching in a graph is a set of pairwise disjoint edges, and a maximum matching is a matching of maximum size. In this paper, we first propose an $O(n(\log n)^3)$ algorithm for finding a maximum matching in trapezoid graphs, then improve the complexity to $O(n(\log n)^2)$. Finally, we generalize this algorithm to a larger graph class, namely k-trapezoid graphs. To the best of our knowledge, these are the first efficient maximum matching algorithms for trapezoid graphs.

Keywords: trapezoid graphs, maximum matching
Mathematics Subject Classification : 05C70, 05D15
DOI:10.5614/ejgta.2017.5.1.2

1. Introduction

A graph G is a *trapezoid graph* if there exists a set of trapezoids between a pair of horizontal lines such that each vertex v_i of G corresponds to a trapezoid T_i and there is an edge (v_i, v_j) iff $T_i \cap T_j \neq \emptyset$. We call this family of trapezoids a *trapezoid representation* (or *trapezoid model*) for G, see Figure 1 in Section 3 for an example. Trapezoid graphs were first introduced by Dagan, Golumbic and Pinter in 1988 [6]. Corneil and Kamula independently introduced the same class [5], but they refer to them as interval-interval (II for short) graphs. Felsner, Müller, and Wernisch

Received: 10 August 2016, Revised 14 January 2017, Accepted: 28 January 2017.

[9] introduced an equivalent box representation for trapezoid graphs. It is noticeable that we can easily get the box representation from trapezoid model by mapping the lower and upper lines of the model to the x-axis and y-axis in the box representation, respectively.

Trapezoid graphs are a class of cocomparability graphs that contains interval graphs and permutation graphs as subclasses. Interval graphs are widely applied for modeling real world problems. They appear in many different scientific domains such as biology, chemistry and archaeology [13]. On the other hand, trapezoid graphs are mainly applied in modeling channel routing problems in VLSI circuit design [6]. Besides, they are very simple in the sense that many graph problems that are NP-hard in general can be solved in polynomial time. For instance, in [9], some of the most classical problems in graph theory such as finding chromatic number, maximum weighted independent set, minimum clique cover and maximum weighted clique are solved in $O(n \log n)$ time by using their box representation and sweeping line techniques. Recently, by exploring the simplicity of this graph class, many well-known problems have a more efficient solution on trapezoid models, such as some $O(n^2)$ algorithms for several counting problems on vertex covers [20], efficient algorithms on K-terminal residual reliability of d-trapezoid graphs [21, 26], an $O(n \log n)$ algorithm for calculating the vertex connectivity [16].

Given a graph $G = (V, E)$, a *matching* M in G is a set of pairwise disjoint edges; that is, no two edges share a common vertex. A *maximum matching* is a matching that contains the largest possible number of edges. For finding a maximum matching in general graphs, Even and Kariv [8] presented an $O(n^{2.5})$ algorithm; Micali and Vazirani [23] improved it to $O(\sqrt{n}m)$, which is proportional to the best-known algorithm for this problem in bipartite graphs [15]. Although a maximum matching can be found in polynomial time in general, it is still interesting to improve the time complexity of this problem for more restricted classes. As pointed out by Moitra and Johnson in [24], there is a strong relationship between the maximum matching problem in a cocomparability graph and the scheduling problem on its complement. Coffman and Graham [4] propose an $O(n + m)$ algorithm for the two-processors scheduling problem when the dependency graph among tasks is transitively closed, therefore we propose an algorithm for the maximum matching problem on cocomparability graphs. Frank et al. [1] present an $O(n(\log n)^2)$ algorithm for finding a maximum matching in a permutation graph. Rhee and Liang [25] improved it to $O(n \log \log n)$. They also present an efficient $O(n \log n)$ maximum matching algorithm for interval graphs and circular-arc graphs [19], which could be refined to $O(n \log \log n)$ as claimed in [25]. Ghosh and Pal [12] proposed an $O(n^2)$ maximum matching algorithm for trapezoid graphs that uses $O(n+m)$ space. Unfortunately, [17] shows that their algorithm turns out not to be correct by giving a simple counterexample.

The rest of the paper is organized as follows. In Section 2, we present some data structures and a known algorithm which solves the problem for cocomparability graphs. In Section 3, we define S-Range tree and propose an $O(n(\log n)^3)$ maximum matching algorithm for trapezoid graphs. This approach is inspired by the range tree method used in [1]. In section 4, we refine the previous algorithm to $O(n(\log n)^2)$ by introducing C-Range tree and generalize it for k-trapezoid graphs. We give some remarks and open questions in the last section. To the best of our knowledge, these are the first efficient maximum matching algorithms for trapezoid graphs.

2. Preliminaries

We first present some data structures and a prior result about the maximum matching algorithm for cocomparability graphs, which will be used later in our algorithms.

2.1. Segment tree

The segment tree was discovered by Bentley [2] in 1977; this is a data structure to store intervals, or segments. Let A be an array of n elements. By considering each element in A as an elementary interval in a normal segment tree, and each internal node contains a maximum element corresponding to its interval, a segment tree can be used to answer the problem of Range Maximum Query (RMQ for short). This data structure uses $O(n)$ storage and can be built in $O(n)$ time. It supports the following basic operations:

- Given i and j, where $1 \leq i \leq j \leq n$, find the maximum element in the interval $[i, j]$ of array A in $O(\log n)$ time.
- Update the value of an element in array A in $O(\log n)$ time.

2.2. CRMQ

This data structure was introduced in [11] to answer RMQ in constant time. Its time-efficiency is achieved thanks to Cartesian tree [27] and the lowest common ancestor query [14]. For convenience, we call this data structure ***CRMQ*** (constant-time RMQ) throughout this paper. It can be constructed in $O(n)$ time, uses $O(n)$ storage and answers RMQ optimally in $O(1)$. However, the only disadvantage of CRMQ compared to the segment tree is that it does not support the update operation.

2.3. Range tree

The range tree is a data structure to hold a list of points. It allows to report all points in a given range efficiently, and can be used in two or higher dimensions. Range tree was separately introduced by different people [3, 18, 22, 29]. In this paper, we first use 2D range trees while working on trapezoid graphs and use k-dimensional range trees later to apply for k-trapezoid graphs. The range tree of dimension k (with $k \geq 2$) can be built in $O(n(\log n)^{k-1})$ time and uses $O(n(\log n)^{k-1})$ storage. It allows to report all points in a k-dimensional region given by a range (for both closed and open regions) in $O((\log n)^{k-1} + m)$ time, where n is the number of points stored in the tree and m is the number of points reported in a given query. In the 2D case, the query region is a rectangular region of the form $[x_1, x_2] \times [y_1, y_2]$ and the corresponding query time is $O(\log n + m)$. We refer to [7] for more details about both segment trees and range trees.

2.4. Maximum matching algorithm for cocomparability graphs

The maximum matching algorithm for cocomparability graph is derived from the following result.

Theorem 2.1. *([10]) Given a directed acyclic and transitively closed graph G with n vertices, then there is a two processors scheduling for G of length ℓ iff there is a matching of size $n - \ell$ in the undirected complement graph G'.*

It is noticeable that G' is a cocomparability graph and a matching in G' can be obtained from the schedule by simply matching all pairs of vertices that are assigned to the same time unit. The two processors scheduling problem is solved efficiently in [4]. Now, we describe briefly their idea to find a maximum matching in a cocomparability graph. The algorithm uses a vertex-labeling that assigns numbers from $1, 2, \ldots, n$ to n vertices of graph G. Given two vertices u, v of G, we say that v is a *successor* of u if there is a directed edge from u to v in G. Let $\mathfrak{L}(u)$ be the label of vertex u and $N(u)$ be the label-list $(\mathfrak{L}(v_1), \mathfrak{L}(v_2), \ldots, \mathfrak{L}(v_k))$ of the successors of u in G such that $N(u)$ is sorted in decreasing order. First, we label the outdegree-zero vertices starting by 1 in an arbitrary order. Suppose that the labels from 1 to $k-1$ have already been assigned, then a vertex u is labeled by k if:

1. all successors of u are labeled, and
2. $N(u)$ is the smallest list in lexicographic order among all vertices that satisfy the condition 1.

Once the vertex-labeling process is complete, the matching can be found in a greedy manner: start from the highest-label vertex, match it with the highest-possible-label vertex that remains, then delete these two vertices from the graph and repeat this process. This yields an $O(n^2)$ time algorithm for finding a maximum matching in the cocomparability graph G'. It is noticeable that a trapezoid graph is also a cocomparability graph, therefore we wish to apply this procedure on trapezoid graphs to get a more efficient maximum matching algorithm based on their special structure.

3. An $O(n(\log n)^3)$ maximum matching algorithm on trapezoid graphs

In this section, we present our $O(n(\log n)^3)$ maximum matching algorithm for trapezoid graphs. Let G be a trapezoid graph with n vertices, and its trapezoid model is given, as in Figure 1. Each vertex u in G is associated with its four endpoints in the trapezoid model, which are denoted by a_u, b_u, c_u, d_u for coordinates of the top-left, top-right, bottom-left, bottom-right of its corresponding trapezoid, respectively. In the box representation (Figure 2), each vertex corresponds to a box (a rectangular region), and we consider only the coordinate of the bottom-left (c_u, a_u) and the top-right (d_u, b_u) endpoints of that box. As described in Section 2.4 above, our maximum matching algorithm has two main steps: (1) labeling the vertices of the complement graph G' (comparability graph), and (2) finding a maximum matching of G by the above greedy method. We use the notions of $\mathfrak{L}(u)$ and $N(u)$ as defined in Section 2.4.

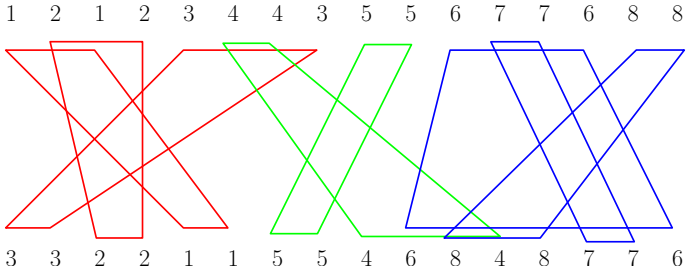

Figure 1. A trapezoid model.

Since G' is the complement of G, two vertices of G' are adjacent iff their two corresponding trapezoids do not intersect in the trapezoid model, *i.e.* one lies entirely to the right of the other (*e.g.* vertices 1 and 5 in Figure 1). We orient the edges of G' from the vertex corresponding to the left trapezoid to the vertex corresponding to the right one.

Definition 1. The *level* of a vertex v of G' is the length of the longest path from v to an outdegree-zero vertex.

For the labeling purpose, we first put the vertices into levels. We denote the level of vertex u by ℓ_u. For example, in Figure 1, $\ell_1 = 2$ because one longest path from 1 to an outdegree-zero vertex in G' is 1-4-7 with length 2. Let k be the maximum level among all the vertices and L_i be the set of vertices of level i $(0 \leq i \leq k)$. This putting-into-levels step can be done in $O(n \log n)$ time by the technique of finding maximum independent set in [17]. The algorithm uses a dynamic process. We assign a number for each vertex while scanning the upper line of the trapezoid model. This number is the length of the longest chain ending at that vertex in the context of maximum independent set. It is the level of that vertex if we scan in the reverse direction. It is noticeable that for every i $(0 \leq i \leq k)$, L_i is a clique in G and is an independent set in G'. By the definition, the level of every successor of a vertex u in G' is always lower than ℓ_u. We prove by induction the following lemma:

Lemma 3.1. *If level of vertex u is higher than level of vertex v, then $\mathfrak{L}(u) > \mathfrak{L}(v)$.*

Proof. The lemma is true for every vertex whose level is 0 or 1. This is because the level-0 vertices are exactly the outdegree-zero vertices (*i.e.* do not have any successor). They are always labeled from the beginning of the algorithm. Hence their labels are smaller than any level-1 vertex. Assume that the lemma is true for every vertex whose level is below i $(i \geq 2)$, we show that it is also true for level i. Let u be any level-i vertex and v be any vertex having level ℓ_v less than i. By the definition of level above, u must have at least one successor of level $(i-1)$. Since $\ell_v < i$, the level of any successor of v is less than $(i-1)$. Therefore, by the induction hypothesis, $N(v) < N(u)$ in lexicographic order, hence v is always labeled before u, thus $\mathfrak{L}(v) < \mathfrak{L}(u)$. □

According to Lemma 3.1, after putting into levels, the rest of the labeling process is to sort and label the vertices in each level, from the lowest to the highest level. We do this by using the box representation of the trapezoid graph.

We denote by $DomReg(u)$ the upper right quadrant of an axis aligned coordinate system whose origin is at point (d_u, b_u), or $DomReg(u) = (d_u, +\infty) \times (b_u, +\infty)$. Therefore, a vertex v is not adjacent to u in trapezoid graph G and has lower level than u iff $(c_v, a_v) \in DomReg(u)$. Let $Reg(u)$ be the subregion of $DomReg(u)$ which shares no point with $DomReg(v)$, or $Reg(u) = DomReg(u) \setminus DomReg(v)$. We define $Reg(v)$ similarly (see Figure 2). Let $Max(R)$ be the maximum label of all points in R, where R is any region in the plane. $Max(R)$ is defined to be zero if R contains no point. In the plane, each vertex is represented by only the bottom-left point of its box in the box representation of the trapezoid graph (*i.e.* we discard all points of the form (d_u, b_u)). Moreover, we label the bottom-left point of each vertex by its label. The next lemma is very important in our algorithm:

Lemma 3.2. *If $Max(Reg(u)) > Max(Reg(v))$, then $\mathfrak{L}(u) > \mathfrak{L}(v)$.*

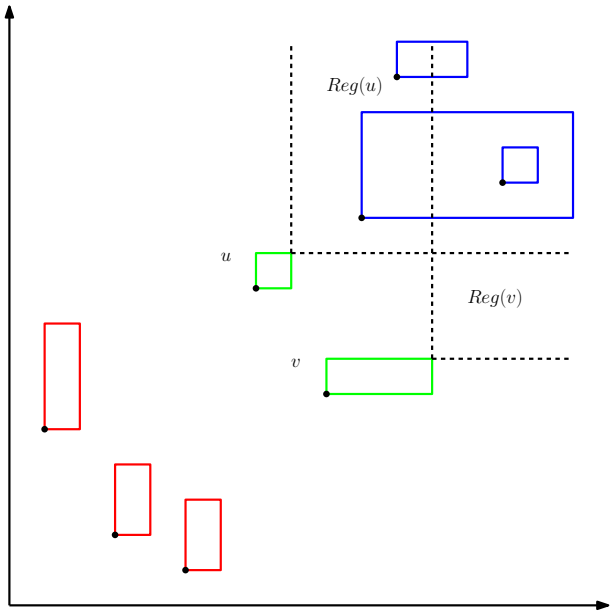

Figure 2. Box representation and the regions $Reg(u)$, $Reg(v)$.

Proof. Note that we consider the complement graph G' of the trapezoid graph in the labeling process. A vertex u' has its corresponding point in $DomReg(u)$ iff u' is a successor of u. Hence, each point in $DomReg(u) \cap DomReg(v)$ is a common successor of u and v. Therefore, to compare $N(u)$ and $N(v)$, we only need to consider the remaining points in $Reg(u)$ and $Reg(v)$. The lemma is then followed. □

The comparison in Lemma 3.2 is a critical operation that will be used in our algorithm for sorting vertices in each level from L_0 to L_k. So, we need a data structure which allows to query the maximum label in a given rectangular region, *i.e.* 2D range maximum query, and allows to update the label of a point efficiently.

3.1. S-Range tree

We use a range tree data structure to store only the bottom-left points of vertices in their box representation (the point at coordinate (c_u, a_u) for each vertex u, see black points in Figure 2). Recall that a range tree can be constructed by using the fractional cascading technique [22, 28] as follows. First, sort the points with respect to the increasing x-coordinate, and build a binary search tree over them. Second, at each internal node, sort the points in its subtree with respect to the y-coordinate by using the merge sort method. Each element in the point list of any internal node always has two pointers that point to the appropriate element in the list of its left child and right child. Specifically, an element α has both a pointer to the element corresponding to the same point in a child and a pointer to the element in the other one having smallest y-coordinate that is bigger than the y-coordinate of α. The goal of these pointers is to indicate the exact segment to query while searching for points in a given rectangular region. For querying the maximum label and updating labels, we add a segment tree over the labels of points at each internal node (see

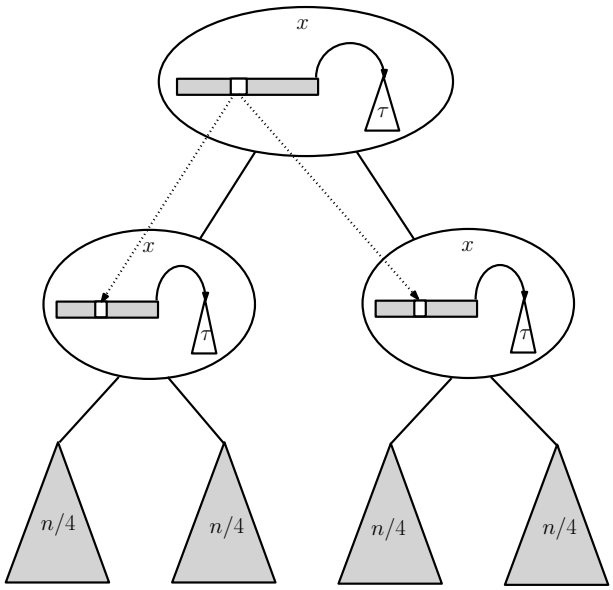

Figure 3. S-Range tree of n points.

the construction of segment trees in Section 2.1). We call this modified range tree *S-Range tree*. As we can see in Figure 3, S-Range tree has a recursive structure: an S-Range tree of n points is formed by its root node and two S-Range trees of $\frac{n}{2}$ points. Each node contains a key x serving for searching by x-coordinate like a binary search tree, a list of its corresponding points sorted by y-coordinate and a segment tree τ built over the labels of these points. Note that S-Range tree is very similar to the data structure introduced in Section 3 of [11], which they called a kind of *scaling tree*. It is also used to answer efficiently RMQ in any dimension. For convenience, we call their data structure *C-Range tree* for dimension 2 and *multidimensional C-Range tree* in general. The only difference between S-Range tree and C-Range tree is that instead of a segment tree, C-Range tree uses CRMQ (introduced in Section 2.2) at each node. Therefore, they differ a bit in the time complexity and supported operations, that we will discuss more in Section 4.

The construction of S-Range tree takes $O(n \log n)$ time as a normal range tree since the building time for a segment tree at each internal node is just $O(n')$, where n' is the number of points in that node. So, given a rectangular region, we need to traverse through $O(\log n)$ nodes in the S-Range tree. At each node, querying the maximum label by a segment tree takes $O(\log n)$. Hence the total time for querying the maximum label in any given rectangular region is $O((\log n)^2)$. For the update operation, we only need to follow the sequence of pointers corresponding to the point whose label needs to be updated. Therefore, this takes $O((\log n)^2)$ time for updating, similar to the time complexity for searching for the maximum label.

3.2. Algorithm

We can summarize the whole vertex-labeling process as follows:

1. Put the vertices into levels $L_0, L_1, \ldots, L_k$.
2. Initialize the label of every vertex to zero.

3. Label the vertices in L_0 starting from 1 in an arbitrary order.
4. Construct the S-Range tree as described above.
5. For each i from 1 to k:
   (a) Sort the vertices in L_i in the increasing order of labels based on the comparison in Lemma 3.2.
   (b) Label the vertices in L_i in increasing order from $(Maximum\ label\ in\ L_{i-1}) + 1$.
   (c) Update the segment tree in each internal node of the S-range tree with respect to the new labels.

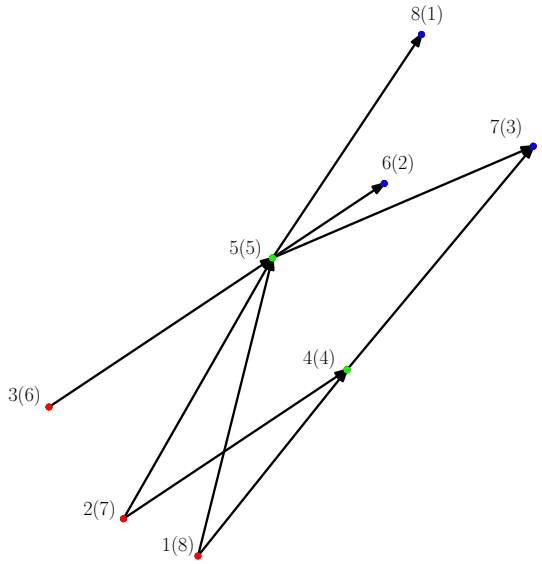

Figure 4. The comparability graph corresponding to the trapezoid graph in Figure 1 with the reduced edge set (consider only the edges between two consecutive levels) and an appropriate vertex labeling.

Recall that operations 1) and 4) take $O(n \log n)$ time, operations 2) and 3) trivially take $O(n)$ time. Suppose that the set L_i has n_i vertices. Since the comparison takes $O((\log n)^2)$ time, the sorting operation in 5a) takes $O((\log n)^2 n_i \log n_i)$ time (by using any kind of $O(n \log n)$ sorting algorithm). Operation 5b) simply takes $O(n_i)$ time and operation 5c) takes $O((\log n)^2 n_i)$ time since there are n_i vertices to be updated. Since $\sum_{i=1}^{k} n_i \log n_i < n \log n$, operation 5) for all k sets takes $O(n(\log n)^3)$ time. Therefore, the overall time for the labeling process is $O(n(\log n)^3)$.

After labeling every vertex, all that remains is to perform a greedy matching step in trapezoid graph G. The S-Range tree constructed in the above step is still useful for the matching purpose. Recall that $DomReg(u)$ is a region that contains only the points corresponding to vertices that are not adjacent and have a lower level than u. We denote by $MaxLabel(u)$ the maximum label of the vertices in the whole plane that is outside $DomReg(u)$, or inside the region $(0, +\infty) \times (0, +\infty) \setminus DomReg(u)$. We can compute $MaxLabel(u)$ efficiently by dividing the query region into two rectangular regions (*e.g.* region $(0, d_u) \times (0, +\infty)$ and $(d_u, +\infty) \times (0, b_u)$), and use the range tree structure above to find the maximum label in these two regions. Obviously, this $MaxLabel(u)$ query takes $O((\log n)^2)$ time. Our matching step can be described as follows:

We go through every set from L_k to L_0. At each set, visit from the highest-label vertex to the lowest one, and for each vertex u:

- If $\mathfrak{L}(u) = 0$, do nothing and continue with the next vertex, since u was already matched with a higher-label vertex.
- If $\mathfrak{L}(u) > 0$, update the label of u to 0 and do the $MaxLabel(u)$ query:
  - If $MaxLabel(u) = 0$, then u has no free adjacent vertex, so u is not matched.
  - If $MaxLabel(u) > 0$, then match u with the vertex v having that $MaxLabel$, and update the label of v to 0.

Lemma 3.3. *Our greedy matching step for trapezoid graph works correctly with respect to the greedy matching step for cocomparability graph stated in Section 2.4.*

Proof. Note that the region in query $MaxLabel(u)$ contains only two kinds of points: one is adjacent to u, and the other is not adjacent to u that must have a higher level and must be updated to the label of 0 before u. So, $MaxLabel(u)$ always gives the vertex having the highest label that is adjacent to u, or returns 0 if no such vertex exists. Hence, our greedy algorithm works correctly. □

The algorithm goes through every vertex. For each vertex, we use at most two update operations and one $MaxLabel$ query that both take $O((\log n)^2)$ time. The total time for greedy matching step is then $O(n(\log n)^2)$. For example, from the vertex-labeling showed in Figure 4, the greedy matching step produces a maximum matching of the trapezoid graph in Figure 1 consisting of three edges: $(1, 2), (3, 4)$ and $(6, 7)$.

Theorem 3.1. *A maximum matching in a trapezoid graph can be found in $O(n(\log n)^3)$ time.*

Proof. Our maximum matching algorithm on a trapezoid graph is correct because its two main steps are right with respect to the maximum matching algorithm for cocomparability graph described in Section 2.4. Since the time complexity of the labeling step is $O(n(\log n)^3)$, and the greedy matching step takes $O(n(\log n)^2)$, the total time for our algorithm is $O(n(\log n)^3)$. □

4. Improved algorithm and generalization

4.1. Improved algorithm by using C-Range trees

In this section, we describe an $O(n(\log n)^2)$ maximum matching algorithm for trapezoid graph by improving the complexity of the labeling step to $O(n(\log n)^2)$.

In the labeling process, the most important operation which affects the complexity of the algorithm is sorting the vertices in each level based on the label-comparison in Lemma 3.2. The previous algorithm used an S-Range tree to perform this comparison in $O((\log n)^2)$. In this improved algorithm, we introduce *C-Range tree* to execute this comparison only in $O(\log n)$ time, based on the idea of CRMQ. The construction of a C-Range tree is analogous to an S-Range tree. Due to the similarity of the segment tree and CRMQ, we provide some comparisons for the operations performed by these two types of range tree in Table 1.

Table 1. Operations supported by S-Range tree and C-Range tree

	S-Range tree	**C-Range tree**
Construction time	$O(n \log n)$	$O(n \log n)$
Query maximum label in a given rectangular region	$O((\log n)^2)$	$O(\log n)$
Update the label of a point	$O((\log n)^2)$	Not supported

To gain the efficiency in the maximum label query, we need to change the algorithm by means of ignoring the update operations. Unlike the previous algorithm that uses only one S-Range tree, in this algorithm we need to construct $(k + 2)$ C-Range trees including:

- A *lvl-C-Range tree* to query the maximum level of the points in a given rectangular region. Since the levels of points are fixed after putting into levels, this data structure can be built from the beginning of the algorithm.
- $(k+1)$ C-Range trees to query the maximum label in each level (denoted by *C-Range tree*-0, *C-Range tree*-1,. . . , *C-Range tree*-k). C-Range tree-i stores only the points corresponding to the level-i vertices, and will be constructed after knowing every label of the vertices in that level. We use C-Range tree-i to compare the labels of the higher-level vertices.

We need to use many C-Range trees since the label of every vertex is not known from the beginning, it is obtained based on the labels of other lower-label vertices. To be able to apply in our algorithm, querying the maximum label in a given rectangular region must be divided into two sub-operations:

1. Find the maximum level of the points in this region; denote this level by t (it is obvious that the maximum-label point also has the maximum level).
2. Query the maximum label in this region using C-Range tree-t.

Since these two sub-operations take $O(\log n)$ time, querying the maximum label in a given rectangular region takes only $O(\log n)$ time. We can summarize the entire improved labeling process as follows:

1. Put the vertices into levels $L_0, L_1, \ldots, L_k$.
2. Construct a lvl-C-Range tree to query the maximum level in a rectangular region.
3. Label the vertices in L_0 starting from 1 in an arbitrary order.
4. Construct the C-Range tree-0 to query the maximum label of level 0.
5. For each i from 1 to k:
   (a) Sort the vertices in L_i in an increasing label order based on the comparison as described above.
   (b) Label the vertices in L_i in increasing order from $(Maximum\ label\ in\ L_{i-1}) + 1$.

(c) Construct C-Range tree-i to serve for the label-comparisons of higher-level vertices.

Suppose that L_i has n_i vertices, operations 1) and 2) take $O(n \log n)$ time, operation 3) takes $O(n_0)$ time, operation 4) takes $O(n_0 \log n_0)$ time. Since the comparison described above takes $O(\log n)$ time, operation 5a) takes $O((\log n) n_i \log n_i)$ time, operation 5b) takes $O(n_i)$ and operation 5c) takes $O(n_i \log n_i)$ time. Hence, entire operation 5) for all k sets takes $O(n(\log n)^2)$ time, and it is also the time complexity of the whole improved labeling process. The remaining step of our algorithm - the greedy matching step, is still implemented as in the previous algorithm since the updating operations here cannot be ignored. Therefore, the overall time complexity of our algorithm is $O(n(\log n)^2)$. So we have the following theorem:

Theorem 4.1. *A maximum matching in a trapezoid graph can be found in $O(n(\log n)^2)$ time.*

4.2. Generalization

A *k-trapezoid graph* ($k \geq 1$) is an intersection graph of *k-trapezoids* between k parallel lines. A *k-trapezoids* is a polygon formed by k intervals on each line by both joining the starting points and joining the ending points of every consecutive interval. This generalization of trapezoid graph was first proposed in [9]. Note that an interval graph is a 1-trapezoid graph and a trapezoid graph is a 2-trapezoid graph. Since k-trapezoid graphs are cocomparability graphs (the complement of k-trapezoid graphs are comparability graphs), we can apply the algorithm described in section 2.4 to find a maximum matching in this graph class. Similar to trapezoid graph, a k-trapezoid graph also has a box representation. Hence we only need to consider the coordinates of the *bottom* and *top* points corresponding to each box. The *bottom* (*top*) point of a box is the point whose coordinate is formed by k starting (ending) points of each interval in k-trapezoid representation. Since almost all crucial operations of maximum matching algorithm on trapezoid graphs use a range tree of dimension 2, we can easily extend our method to k-trapezoid graphs by using a multidimensional range tree. Here are some specific details:

- Putting the vertices into levels: Instead of using [17], we apply the technique for solving the maximum independent set or the minimum clique cover problem in a k-trapezoid graph from [9]. Therefore, this process takes $O(n(\log n)^{k-1})$ time.
- Construction time for both k-dimensional S-Range tree and k-dimensional C-Range tree is $O(n(\log n)^{k-1})$.
- Querying the maximum label in a rectangular region is extended to querying the maximum label in a k-dimensional region. This operation takes $O((\log n)^k)$ and $O((\log n)^{k-1})$ time on the extended S-Range tree and C-Range tree, respectively.
- The update operation of an extended S-Range tree takes $O((\log n)^k)$ time.

Therefore, by extending the dimension of the range trees, we can get an $O(n(\log n)^k)$ algorithm for finding a maximum matching in a k-trapezoid graph.

Theorem 4.2. *A maximum matching in a k-trapezoid graph ($k \geq 2$) can be found in $O(n(\log n)^k)$ time.*

Remark that not only could our algorithm adapt to the larger graph class by extending the dimension of trapezoid graphs, but it is also possible if we lower the dimension. If we consider 1-dimensional S-Range tree as a segment tree and 1-dimensional C-Range tree as a CRMQ, then we could similarly obtain an $O(n \log n)$ algorithm for finding a maximum matching in 1-trapezoid graphs, *i.e* interval graphs. Therefore, Theorem 4.2 is also true for $k = 1$. This does not mean too much since we already had an $O(n \log \log n)$ maximum matching algorithm for interval graph from [19, 25]. However, it confirms the flexibility of our algorithm.

5. Conclusion

In this paper, we present an $O(n(\log n)^2)$ algorithm for finding a maximum matching in a trapezoid graph, and extend the result to obtain an $O(n(\log n)^k)$ algorithm for k-trapezoid graphs. To the best of our knowledge, these are the first efficient algorithms to solve the problem. Since we do not know any lower bound of this problem except the trivial bound $\Omega(n)$, we believe that the complexity we obtained is not optimal. One hypothesis is that if we can answer Range Maximum Query in constant time on permutation or trapezoid models, there could be a linear maximum matching algorithm for both of them. We leave the following conjecture as an open question.

Conjecture 1. *There exists an $O(n)$ algorithm to find a maximum matching in a trapezoid graph given its trapezoid representation, where n is the number of vertices.*

Acknowledgement

This research is funded by Vietnam Ministry of Education and Training (MOET) under grant number B2015 - 01 - 90.

References

[1] F. Bauernöppel, E. Kranakis, D. Krizanc, A. Maheshwari, J.R. Sack, and J. Urrutia, An improved maximum matching algorithm in a permutation graph, 1995.

[2] J.L. Bentley, *Solutions to klees rectangle problems*, Technical Report, Carnegie-Mellon Univ., Pittsburgh, PA, 1977.

[3] J.L. Bentley, Decomposable searching problems, *Information Processing Letters* **8** (5) (1979), 244–251.

[4] E.G. Coffman and R.L. Graham, Optimal scheduling for two-processor systems, *Acta Informatica* **1** (3) (1972), 200–213.

[5] D.G. Corneil and P.A. Kamula, Extensions of permutation and interval graphs, *Congressus Numerantium* **58** (1987), 267–275.

[6] I. Dagan, M.C. Golumbic, and R.Y. Pinter, Trapezoid graphs and their coloring, *Discrete Appl. Math.* **21** (1) (1988), 35–46.

[7] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars, *Computational geometry: algorithms and applications*, Springer, 2008.

[8] S. Even and O. Kariv, An $O(n^{2.5})$ algorithm for maximum matching in general graphs, in *16th Annual Symposium on Foundations of Computer Science* (16th FOCS, berkeley, California, 1975), IEEE, New York (1975), 100–112.

[9] S. Felsner, R. Müller, and L. Wernisch, Trapezoid graphs and generalizations, geometry and algorithms, *Discrete Appl. Math.* **74** (1) (1997), 13–32.

[10] M. Fujii, T. Kasami, and K. Ninomiya, Optimal sequencing of two equivalent processors, *SIAM Journal on Applied Mathematics* **17** (4) (1969), 784–789.

[11] H.N. Gabow, J.L. Bentley, and R.E. Tarjan, Scaling and related techniques for geometry problems, In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, ACM (1984), 135–143.

[12] P.K. Ghosh and M. Pal, An algorithm to find a maximum matching of a trapezoid graph, *The journal of the Korean Society for Industrial and Applied Mathematics* **9** (2) (2005), 13.

[13] M.C. Golumbic, *Algorithmic graph theory and perfect graphs*, Volume 57, North Holland, 2004.

[14] D. Harel and R.E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM Journal on Computing* **13** (2) (1984), 338–355.

[15] J.E. Hopcroft and R.M. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, *SIAM Journal on Computing* **2** (4) (1973), 225–231.

[16] A. Ilic, Efficient algorithm for the vertex connectivity of trapezoid graphs, *Information Processing Letters* **113** (2013), 398–404.

[17] A. Ilic and A. Ilic, On vertex covers and matching number of trapezoid graphs, *arXiv preprint arXiv:1106.2351*, 2011.

[18] D.T. Lee and C.K. Wong, Quintary trees: a file structure for multidimensional database systems, *ACM Transactions on Database Systems (TODS)* **5** (3) (1980), 339–353.

[19] Y.D. Liang and C. Rhee, Finding a maximum matching in a circular-arc graph, *Information Processing Letters* **45** (4) (1993), 185–190.

[20] M.S. Lin and Y.J. Chen, Counting the number of vertex covers in a trapezoid graph, *Information Processing Letters* **109** (2009), 1187–1192.

[21] M.S. Lin and C.C. Ting, A polynomial-time algorithm for computing K-terminal residual reliability of d-trapezoid graphs, *Information Processing Letters* **115** (2015), 371–376.

[22] G.S. Lueker, A data structure for orthogonal range queries, In *Foundations of Computer Science 1978, 19th Annual Symposium*, IEEE (1978), 28–34.

[23] S. Micali and V.V. Vazirani, An $O(\sqrt{V}E)$ algoithm for finding maximum matching in general graphs, In *Foundations of Computer Science, 1980, 21st Annual Symposium*, IEEE (1980), 17–27.

[24] A. Moitra and R.C. Johnson, Parallel algorithms for maximum matching and other problems on interval graphs, Technical Report, Cornell University, 1988.

[25] C. Rhee and Y.D. Liang, Finding a maximum matching in a permutation graph, *Acta Informatica* **32** (8) (1995), 779–792.

[26] S. Roy, K. Daripa, and A.K. Datta, K-terminal reliability of d-trapezoid graphs, *IEEE Transactions on Reliability* **65** (3) (2016), 1240–1247.

[27] J. Vuillemin, A unifying look at data structures, *Communications of the ACM* **23** (4) (1980), 229–239.

[28] D.E. Willard, *Predicate-oriented database search algorithms*, Technical Report, DTIC Document, 1978.

[29] D.E. Willard, *The super-B-tree algorithm*, Technical Report, DTIC Document, 1979.