Discrete Applied Mathematics 178 (2014) 58–70

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

In-place algorithms for computing a largest clique in geometric intersection graphs☆

Minati De[a,*], Subhas C. Nandy[b], Sasanka Roy[c]

[a] The Technion – Israel Institute of Technology, Haifa - 32000, Israel
[b] Indian Statistical Institute, Kolkata - 700108, India
[c] Chennai Mathematical Institute, Chennai - 603103, India

ARTICLE INFO

Article history:
Received 28 June 2013
Received in revised form 18 June 2014
Accepted 25 June 2014
Available online 10 July 2014

Keywords:
Geometric intersection graphs
Clique
In-place algorithms
Bipartite matching

ABSTRACT

In this paper, we study the problem of designing in-place algorithms for finding the maximum clique in the *intersection graphs* of axis-parallel rectangles and disks in $\mathbb{R}^2$. First, we propose an $O(n^2 \log n)$ time in-place algorithm for finding the maximum clique of the intersection graph of a set of n axis-parallel rectangles of arbitrary sizes. For the intersection graph of fixed height rectangles, the time complexity can be slightly improved to $O(n \log n + nK)$, where K is the size of the maximum clique. For disk graphs, we consider two variations of the maximum clique problem, namely geometric clique and graphical clique. The time complexity of our algorithm for finding the largest geometric clique is $O(m \log n + n^2)$ where m is the number of edges in the disk graph, and it works for disks of arbitrary radii. For graphical clique, our proposed algorithm works for unit disks (i.e., of same radii) and the worst case time complexity is $O(n^2 + m(n + K^3))$; m is the number of edges in the unit disk intersection graph and K is the size of the largest clique in that graph. It uses $O(n^3)$ time in-place computation of maximum matching in a bipartite graph, where the vertices are given in an array, and the existence of an edge between a pair of vertices can be checked by an oracle on demand (from problem specification) in $O(1)$ time. This problem is of independent interest. All these algorithms need $O(1)$ work space in addition to the input array.

1. Introduction

Geometric intersection graphs are very interesting to study both from geometric and graph theoretic point of view. The geometric intersection graph $G = (V, E)$ of a set of geometric objects S is a graph whose nodes V correspond to the set of objects in S. Between a pair of nodes v_i and v_j, there is an edge (v_i, v_j) if the corresponding objects in S intersect. The intersection of a pair of objects is defined depending on the problem specification. For example, sometimes proper containment is considered to be an intersection and sometimes it is not.

In sophisticated database query and VLSI physical design, several optimization problems are formulated using the intersection graph of axis-parallel rectangles [2]. Similarly, the disk graphs play important role in formulating different problems in mobile ad hoc network [15]. From now onwards, by rectangle intersection graph we will mean the intersection graph of axis-parallel rectangles.

☆ A preliminary version of this paper appeared in the proceedings of Frontiers in Algorithmics and Algorithmic Aspects in Information and Management - Joint International Conference, FAW-AAIM 2012, Beijing, China, May 14–16, 2012.

* Correspondence to: Department of Computer Science, The Technion – Israel Institute of Technology, Haifa - 32000, Israel.
E-mail addresses: minati.isi@gmail.com, minati@cs.technion.ac.il (M. De).

http://dx.doi.org/10.1016/j.dam.2014.06.025
0166-218X/

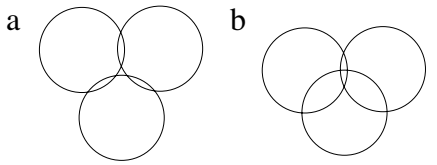

Fig. 1. (a) Graphical clique and (b) geometric clique of disk graph.

Lee and Preparata [14] proposed the first polynomial time algorithm for finding the maximum clique in a rectangle intersection graph in $\mathbb{R}^2$. The best known algorithm for this problem runs in $O(n \log n)$ time and $O(n)$ space [12,21]. The best known algorithm for finding the maximum clique among a set of axis-parallel rectangles in $\mathbb{R}^d$ runs in $O(n^{d-1})$ time [13]. Since axis-parallel rectangles satisfy Helly property,[1] the members of a clique in the rectangle intersection graph have a non-empty intersection [12].

Let us consider the intersection graph of a set of disks. Let C be a subset of disks such that each pair of members in C intersect. In the aforesaid graph, the nodes corresponding to C define a clique. However, the members of C may not have a common intersection region. Thus, a clique in a disk graph is usually referred to as a *graphical clique*. For demonstration, see Fig. 1. In particular, if the members in a clique have common intersection region, then that clique is referred to as a *geometric clique*. Clark et al. [8] proposed an $O(n^{4.5})$ time algorithm for computing the largest (graphical) clique in a unit disk graph. Later the time complexity of the problem was improved to $O(n^{3.5} \log n)$ [3]. Given a set of points in $\mathbb{R}^2$, the problem of finding the position of a unit disk to contain maximum number of points can be mapped to finding the maximum geometric clique of a unit disk graph. Using the plane sweep method, this problem can also be solved in $O(n^2)$ time using $O(n)$ extra-space [7].

In this paper, we study the problem of computing the largest clique for some classes of geometric intersection graphs in the in-place model, where the geometric specification of the objects are given. In this model, the following constraints are maintained: (i) the input objects are given as array elements, (ii) during the execution of the algorithm, swapping of the elements in the input array is permissible, (iii) after the execution, all the input elements should be present in the array (may be in some different permutation), and (iv) only $O(1)$ extra-space (each of $O(\log n)$ bits) usage during the execution.

In-place algorithms have many profound advantages compared to traditional algorithms where the main stress is on running time. As apart from the array containing the input data, only $O(1)$ extra-space is used during the execution, a larger part of the data can be kept in the faster memory. As a result of which the number of slower memory access is very less. Thus, they practically perform faster than the traditional algorithms. In mission critical applications, the traditional algorithms may fail because of out of memory in runtime; whereas the in-place algorithms are less prone to failure [4]. As in-place algorithms use only constant amount of extra-space apart from the input, it is a big advantage for handling big data where one cannot afford too much extra memory space apart from the input. A detailed survey on the in-place algorithms for the geometric optimization and search problems is available in [5].

Our results

First, we propose an in-place $O(n \log n)$ time algorithm for computing the maximum clique of an intersection graph of a set of n intervals on a real line. We use this algorithm to design an in-place algorithm for finding the maximum clique of the intersection graph of a set of n axis-parallel rectangles of arbitrary size in $O(n^2 \log n)$ time. For fixed height rectangles, the time complexity can be improved to $O(n \log n + nK)$, where K is the size of the largest clique.

Next, we consider the maximum clique problem for the disk graph. Our proposed in-place algorithm for computing the largest geometric clique of the intersection graph of a set of disks of arbitrary radii needs $O(m \log n + n^2)$ time, where m is the number of edges in this disk graph, which may be $O(n^2)$ in the worst case. However, if this disk graph is a sphere of influence graph [24–26] for a set of points in $\mathbb{R}^2$, then $m = O(n)$ [23]. For graphical clique, our in-place algorithm works for unit disks only, and it runs in $O(n^2 + m(n + K^3))$ time, where n and m are the number of vertices and edges in the unit disk graph, and K is the size of the maximum clique in that graph.

To compute the largest graphical clique of unit disk graph, we need the maximum matching of a bipartite graph (see [8]). In this context, we propose an $O(n^3)$ time in-place algorithm for computing maximum matching in a bipartite graph $G = (V_1, V_2, E)$ where the two sets of nodes V_1 and V_2 are stored in two arrays, and the existence of an edge between a pair of nodes can be checked on demand by an oracle in $O(1)$ time.

Table 1 summarizes the results presented in the paper along with the comparative study with the best known algorithm available in the literature. Notice that, the (*time* × *extra-space*) is less than or equal to the best-known results for all the problems we have considered excepting the last one.

We also show that the 2-approximation algorithm by Agarwal et al. [1] for computing the maximum independent set of a set of axis-parallel rectangles of fixed height can be made in-place easily, and it runs in $O(n \log n)$ time using $O(1)$ extra-space. This follows from the fact that if we split the plane into strips of height δ using horizontal lines, where δ is the height of the rectangles, then each rectangle intersect exactly one horizontal line. Similarly, the 5-approximation algorithm

[1] A set of convex objects in $\mathbb{R}^2$ is said to satisfy *Helly property*, if the members of a subset of this set are mutually intersecting then they have a common intersection region, and vice versa.