ON THE INTERSECTION OF ORTHOGONAL OBJECTS

H. EDELSBRUNNER and H.A. MAURER
Institut für Informationsverarbeitung, Technical University of Graz, Steyrergasse 17, A-8010 Graz, Austria

Received 5 August 1981

Searching, data structures, tree structures, multidimensional algorithms, range searching, rectangle intersection, line segment intersection, interval tree, segment tree, range tree

1. Introduction

Over the past half a dozen years, many aspects of multidimensional algorithms and data structures have been investigated. This is best illustrated by the recent bibliographies of Edelsbrunner and van Leeuwen [8, 9] which contain over 300 entries.

In many cases, collections of geometrical objects in two or higher dimensions are subjected to queries, where each query is itself a geometrical object. The answer of such a query provides information concerning the query object in relation to the collection of objects available.

In this paper we demonstrate that a number of such problems which are usually dealt with separately are special cases of a single general problem. By presenting an efficient (static and dynamic) solution for this general case – which involves the intersection of certain simple geometric objects – we provide a unified solution to many problems considered before and to problems not explicitly examined in earlier papers.

More specifically, we consider the intersection of what we call orthogonal objects. An *orthogonal object* of dimension d is the Cartesian product of d intervals (each of which may reduce to a single value), one on each coordinate-axis. Thus, typical orthogonal objects are points, and objects such as line segments, rectangles, cubes, etc., provided all edges are axis-parallel.

We claim that a large variety of problems dealt with in the literature can be reduced to what we call orthogonal intersection searching. *Orthogonal intersection searching* involves a collection of orthogonal objects and a sequence of queries, where each query is also an orthogonal object; the desired answer is the set of objects in the given collection intersecting the query object. (We say that two orthogonal objects *intersect* if they have at least one point in common.)

Historically, the first problem of this kind is the range searching problem. It involves a set of points in the plane. Each query asks for all points that lie within a specified orthogonal rectangle. Bentley and Friedman [2] present a survey of practical algorithms solving this problem. More sophisticated algorithms are offered by Bentley and Maurer [3], Bentley [1], Lueker [11,12], and Willard [20].

The *inverse range searching problem* involves a set of planar orthogonal rectangles. Each query asks for all rectangles that contain a specified query point. Vaishnavi [17] designed structures based on the segment tree, consult Leeuwen and Wood [10], for this task.

Given vertical and horizontal line segments, the *line segment intersection searching problem* asks for all segments that intersect a vertical or horizontal query segment. This problem was recently investigated by Vaishnavi and Wood [19]. The related problem that asks for all intersecting pairs of a set of vertical and horizontal line segments was considered before by Bentley and Ottmann [4].

The *rectangle intersection searching problem*

0020-0190/81/0000–0000/$02.75

involves a set of orthogonal rectangles and asks for all rectangles that intersect an additional one. This problem was considered by Edelsbrunner [6], who applied his results to the enumeration of all intersecting pairs of a given set. Earlier work on the latter issue can be found in Bentley and Wood [5], Vaishnavi [16], Vaishnavi, Kriegel and Wood [18], and Six and Wood [14,15].

The above problems occur, directly or indirectly, in a variety of applications such as VLSI design techniques, computer graphics, algorithms for data-base queries, and others. The interested reader is referred to the papers quoted for further background information.

It should be clear that all of the above problems can be generalized to more than two dimensions. We have mentioned their planar instances in order to aid the geometric intuition of the reader.

The first step towards a unified approach to all these related problems was undertaken by Edelsbrunner [7]. He introduced the notion of orthogonal objects and developed structures that are based on results due to Six and Wood [14,15], Willard [21], and Edelsbrunner [6].

Like most multidimensional searching problems, orthogonal intersection searching comes in a static and a dynamic variety. In the *static* case the collection of orthogonal objects remains unchanged throughout the sequence of queries, while in the *dynamic* case the collection of orthogonal objects can be updated inbetween queries by inserting and deleting individual objects.

The time needed to perform an update or a query is usually measured in worst-case or in average complexity. If an algorithm performs a command (i.e. a query or update) in time f(n) even in the worst of all possible cases then the command has *worst-case complexity* f(n) (with respect to that algorithm), where n denotes the actual size of the problem (in most cases, n denotes the current number of objects stored). A command has *average complexity* f(N) (with respect to an algorithm) if a sequence C of |C| commands is executed in at most |C|f(N) time, where N denotes the maximal number of objects in the initially empty set.

The next section of this paper presents efficient static and dynamic solutions for orthogonal intersection searching. It has to be emphasized that this paper, for the first time, considers the searching problem with potentially inhomogeneous sets of orthogonal objects. The reader is assumed to have a working knowledge of the techniques contained in the above mentioned references, since the main contribution of this paper is a clever combination of a variety of techniques and data structures. The final section of this paper shows how the results can also be applied to the determination of all intersecting pairs of a given set of objects.

2. The theorem

Theorem. Let S be an arbitrary collection of n orthogonal objects in d-dimensional space, for d no less than 2. Let x denote an arbitrary orthogonal query object and let A(S, x) denote the set of t objects of S that intersect x.

There exists a static data structure which allows one to locate and report the objects of A(S, x) in $O(\log^{d-1} n + t)$ time and requires $O(n \log^d n)$ space and preprocessing.

Further, the same problem can be solved with a dynamic data structure that allows for a query to be answered in $O(\log^d n + t)$ time and requires $O(n \log^d n)$ space and $O(\log^d n)$ time for an update.

All bounds are given in worst-case complexity.

Before we present the proof of the above theorem by exhibiting efficient data structures supporting intersection queries involving arbitrary (in general inhomogeneous) sets of orthogonal objects, one remark is in order: It is possible to improve the space requirements of the above structures by a factor of log n. However, this increases the query time of the static structure by the same factor, and affects the dynamic structure either by increasing the bound for performing an update by a factor of log n, or by weakening the original $O(\log^d n)$ worst-case complexity of the update time to $O(\log^d N)$ average complexity.

Proof. For simplicity, we confine our attention to the 2-dimensional space. Extension to three and higher dimensions can then be obtained by straightforward generalizations.

Let x and y denote two 2-dimensional orthogonal

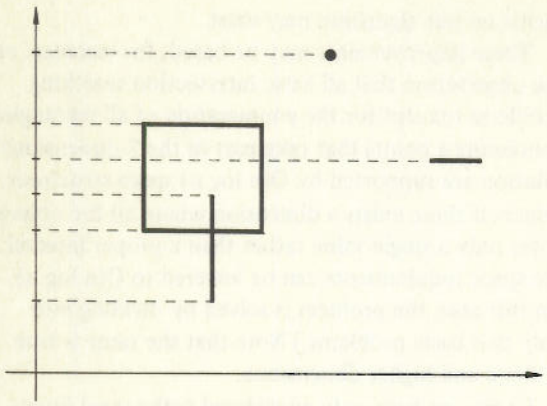

Fig. 1. Four orthogonal objects.

objects. Either x contains the left bottom point of y, or the left border line of x intersects the bottom border line of y, or the bottom border line of x intersects the left border line of y, or the left bottom point of x is contained in y. Note that exactly one of these cases occurs, unless x does not intersect y. Observe further that this is also true if degenerated rectangles like vertical or horizontal line segments or even points are involved.

In what now follows, we will establish a data structure for each of the four types of intersection.

These structures are composed of three basic types of trees, two of which are the outcome of recent research on 1-dimensional intersection searching. The common binary leaf search tree (called *range tree* in this paper) storing n values in linear space is employed

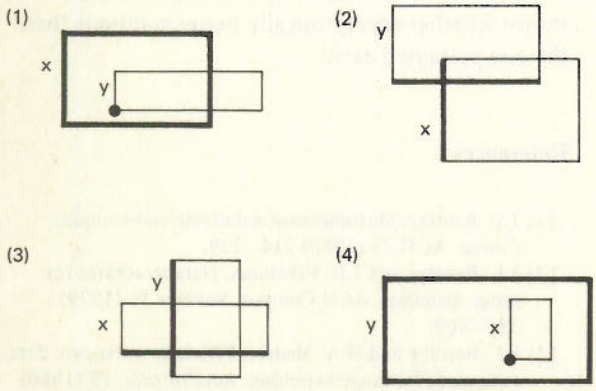

Fig. 2. The four types of intersection.

to answer 1-dimensional range queries (which in fact are intervals) in O(log n + t) time. The inverse problem, i.e. given a set of intervals – report those which contain a query value, is supported by the *segment tree.* O(n log n) space is required for n intervals and a query is answered in O(log n + t) time. The third basic type is called the *interval tree* which stores n intervals in linear space and supports the enumeration of those which intersect a query interval in O(log n + t) time.

A certain kind of combination of these three basic types of trees is employed to yield structures that handle the four instances of 2-dimensional orthogonal intersection searching listed above. We are going to demonstrate the specific combination for the segment tree and the range tree.

The specific combination is used to create what we call the segment–range tree which stores horizontal line segments as follows: The projections of the line segments onto the x-axis are stored in a segment tree. Each node of this tree corresponds to a set of line segments and gets attached to the range tree storing the projections of these line segments onto the y-axis. In a similar fashion the range–range tree, the range–segment tree and the segment–segment tree are obtained. (Note that the range–segment tree may be replaced by the segment–range tree by interchanging the coordinates.)

These four structures solve our four basic intersection problems with query time $O(\log^2 n + t)$ each. The segment–segment tree reports all rectangles containing a query point, the segment–range tree enumerates all horizontal (respectively vertical) line segments that intersect a vertical (respectively horizontal) query segment, and the range–range tree reports all points inside a query rectangle. These structures can be constructed in $O(n \log^2 n)$ time and require $O(n \log^2 n)$ space.

Let us summarize what we have obtained so far: a data structure is exhibited that solves the 2-dimensional orthogonal intersection searching problem with $O(n \log^2 n)$ preprocessing and space, and with $O(\log^2 n + t)$ query time. Next, a few modifications assure that the bounds stated in the theorem are achieved.

For the static case, the segment–segment tree, the two instances of the segment–range tree, and the range–range tree are improved by adding additional pointers that save a factor O(log n) query time and

leave the other bounds unchanged. (This approach is called using a layered tree in the literature.)

This proves the first part of the theorem concerning the static solution of the orthogonal intersection searching problem.

The mentioned improvement of the space requirements accompanied by an increase in query time is obtained by replacing the pair of the segment–segment and segment–range tree by the segment–interval tree and by replacing the pair of the range–segment and range–range tree by the range–interval tree. It should be noted that the task supported by a pair of segment and range trees (namely interval intersection searching) is accommodated by a single interval tree.

Concerning dynamic environments, the segment–segment, the two instances of the segment–range, and the range–range tree are replaced by their dynamic relatives that are based on the trees of bounded balance introduced by Nievergelt and Reingold [13].

The dynamic relatives of the four trees achieve the same bounds for space and query time as the static trees (but not the layered static trees) by simultaneously allowing updates to be performed in $O(\log^2 n)$ time in the worst case.

The saving of a factor O(log n) space is again attained by replacing the four trees by the dynamic relatives of the segment–interval tree and the range–interval tree. The update time deteriorates either to $O(\log^3 n)$ in the worst case or to $O(\log^2 N)$ on the average. Nevertheless, we do not know whether there exist strategies that realize the above replacement without affecting the time needed to perform an update.

This proves the second part of the theorem and the second part of the remark.

3. Applications and conclusions

In the main section of this paper a method that treats arbitrary and potentially inhomogeneous sets of orthogonal objects in multidimensional space with only one algorithm is given. In the past, all special cases were handled separately, leading to a large number of seemingly unrelated papers. Our algorithm not only clarifies and unifies the situation, it also can be used for problems not dealt with previously. On the other hand, for special sets of objects, minor improvements on our algorithm may exist.

These improvements may be based, for instance, on the observation that all basic intersection searching problems (except for the enumeration of all rectangles containing a point) that take part in the 2-dimensional solution are supported by O(n log n) space structures. Hence, if there exists a dimension where all the objects cover only a single value rather than a proper interval the space requirements can be lowered to O(n log n). (In this case, the problem is solved by dividing into only two basic problems.) Note that the same is true in three and higher dimensions.

So far, we have only considered orthogonal intersection searching. In certain applications one is interested in all intersecting pairs of a given set of orthogonal objects.

This problem can be solved by sweeping the d-dimensional space with a $(d-1)$-dimensional hyperplane normal to the x-axis from left to right. Associated with this hyperplane is the structure that supports dynamic $(d-1)$-dimensional orthogonal intersection searching with $O(n \log^{d-2} n)$ space, $O(\log^{d-1} n + t)$ query time, and $O(\log^{d-1} N)$ average update time. The task is then accomplished by executing a sequence of O(n) insertions, intersection queries, and deletions leading to a $O(n \log^{d-1} n)$ time and $O(n \log^{d-2} n)$ space solution.

Concluding, we want to emphasize once more that the approach described in this paper offers a clear and unified way to deal with a variety of tasks that were considered as separate problems in the past.

A great challenge for future research is the uniform description of non-orthogonal intersection problems. However, many open questions concerning orthogonal intersection also remain. In particular it remains to be shown whether asymptotically better solutions than the one presented exist.

References

[1] J.L. Bentley, Multidimensional divide-and-conquer, Comm. ACM 23 (1980) 214–229.

[2] J.L. Bentley and J.H. Friedman, Data structures for range searching, ACM Comput. Surveys 11 (1979) 397–409.

[3] J.L. Bentley and H.A. Maurer, Efficient worst-case data structures for range searching, Acta Inform. 13 (1980) 155–168.

[4] J.L. Bentley and Th. Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput. 28 (1979) 643–647.
[5] J.L. Bentley and D. Wood, An optimal worst-case algorithm for reporting intersections on rectangles, IEEE Trans. Comput. 29 (1980) 571–577.
[6] H. Edelsbrunner, A new approach to rectangle intersections, Technical University of Graz, Institut für Informationsverarbeitung (1980). Reports 47 and 50.
[7] H. Edelsbrunner, Dynamic data structures for orthogonal intersectionqueries, Technical University of Graz, Institut für Informationsverarbeitung (1980). Report 59.
[8] H. Edelsbrunner and J. van Leeuwen, Multidimensional algorithms and data structures (Bibliography), Bull. EATCS 11 (1980) 46–74.
[9] H. Edelsbrunner and J. van Leeuwen, Supplement to 'Multidimensional algorithms and data structures (Bibliography)', Bull. EATCS 13 (1981) 79–85.
[10] J. van Leeuwen and D. Wood, The measure problem for rectangular ranges in d-space, University of Utrecht, Department of Computer Science (1979). Report RUU-CS-79-6.
[11] G. Lueker, A data structure for orthogonal range queries, Proc. of the 15th FOCS Symposium (1978) 28–34.
[12] G. Lueker, A transformation for adding range restriction capability to dynamic data structures for decomposable searching problems, University of California, Irvine, Department of Information and Computer Science (1979). Report 129.
[13] J. Nievergelt and E.M. Reingold, Binary search trees of bounded balance, SIAM J. Comput. 2 (1973) 33–43.
[14] H.-W. Six and D. Wood, The rectangle intersection problem revisited, BIT 20 (1980) 426–433.
[15] H.-W. Six and D. Wood, Counting and reporting intersections of d-ranges, McMaster University, Unit for Computer Science (1980). Report 80-CS-6.
[16] V.K. Vaishnavi, Optimal worst-case algorithms for rectangle intersection and batched range searching problems, McMaster University, Unit for Computer Science (1979). Report 79-CS-12.
[17] V.K. Vaishnavi, Computing point enclosures, Concordia University, Computer Science Department (1980).
[18] V.K. Vaishnavi, H.P. Kriegel and D. Wood, Space and time optimal algorithms for a class of rectangle intersection problems, Inform. Sci. 21 (1980) 59–67.
[19] V.K. Vaishnavi and D. Wood, Rectilinear line segment intersection, layered segment trees and dynamization, McMaster University, Unit for Computer Science (1980). Report 80-CS-8.
[20] D.E. Willard, New data structures for orthogonal queries, Comm. ACM, to appear.
[21] D.E. Willard, An introduction to super-B-trees, University of Iowa, Department of Computer Science (1979). Report 79-01.