Enclosing many boxes by an optimal pair of boxes *

Bruno Becker[†] Paolo Giulio Franciosa[‡] Stephan Gschwind[†]
Thomas Ohler[†] Gerald Thiemt[†] Peter Widmayer[†]

Keywords: computational geometry, covering problems, axis-parallel rectangles

Abstract

We look at the problem: Given a set M of n d-dimensional intervals, find two d-dimensional intervals S, T, such that all intervals in M are enclosed by S or by T, the distribution is balanced and the intervals S and T fulfill a geometric criterion, e.g. like minimum area sum. Up to now no polynomial time algorithm was known for that problem. We present an $O(dn \log n + d^2 n^{2d-1})$ algorithm for finding an optimal solution.

1 Introduction

Throughout the years, several fast heuristics have been proposed for a combinatorial optimization problem that is important in the area of spatial data structures. Given a set of (axis-parallel) rectangles in the plane, the problem is to find two rectangles, say S and T, such that each given rectangle is enclosed by S or by T (or both), each of S and T enclose at least a certain number of given rectangles, and S and T together minimize some measure, e.g. the sum of their areas. This problem must be solved whenever an overflowing page of an R-tree or its variants must be split into two pages [BKSS90, Gre89, Gut84]. The bound on the number of rectangles enclosed by S and by T serves to balance the split; S and T are the bounding boxes for the two pages after the split. There is a number of suggestions on what to minimize in order to achieve a good data structure performance, apart from the sum of areas, e.g. the area of the overlap, the sum of perimeters, and combinations of these. Surprisingly, quite a sorry state of affairs seems to have been accepted: almost all heuristics compute approximate solutions that can be arbitrarily bad as compared with the optimal solution, even though no lower bounds on the problem complexity have been established — the only comment in this direction says that an exhaustive search for an optimal solution takes time proportional to 2^n for n rectangles [Gut84].

In this paper, we show that the optimal solution for all of the variants of the problem investigated in the literature can be obtained in polynomial time. We propose a conceptually simple algorithm that is easy to implement and runs in optimal time for the one-dimensional case and in time $O(n^3)$ for two-dimensional rectangles; for comparison, the better (in terms of data structure performance)

*Partially supported by grant Wi810/2–5 of the Deutsche Forschungsgemeinschaft DFG, and by the ESPRIT II BRA Project 3075 ALCOM of the European Community.

[†]Institut für Informatik, Universität Freiburg, Rheinstraße 10–12, D–7800 Freiburg

[‡]Università "La Sapienza", Dipartimento di Informatica e Sistemistica, via Salaria 113, I-00198 Roma, Italy. Work done while the author was partially supported by Istituto di Analisi dei Sistemi ed Informatica, CNR, Viale Manzoni 30, I–00185 Roma, Italy

of the two originally proposed heuristics takes time $O(n^2)$. Our algorithm is general enough to work for any dimension and any objective function in a fairly large class. For $d \geq 2$ dimensions, it takes time $O(d^2 n^{2d-1})$ in the worst case, using linear space. Furthermore, we propose a variation of our algorithm that is faster in low-dimensional cases, if the value of the objective function is sufficiently low.

The problem attacked in this paper can be considered as a member of a family of *covering and approximation* problems. For instance, if we drop the requirement that each given rectangle be enclosed by S or by T, and instead merely request that each rectangle be enclosed by the union of S and T, the problem changes its nature considerably [BFG+91]. As a fringe benefit, our algorithm slightly improves on the best solution known for a special case of another problem in the family: for n input points instead of rectangles, a minimum perimeter rectangle that contains k points can be found with higher-order Voronoi diagrams in time $O(k^2 n \log n)$ [AIKS89], whereas our algorithm solves this problem in time $O(n^3)$, an improvement at least for $k = \Theta(n)$.

We describe the enclosing problem formally in chapter 2 where we also explain the most important ideas of our solution. In chapter 3 we propose an efficient algorithm for the problem and inspect its performance in detail. We show in chapter 4 that $O(n \log n)$ is a lower time bound and that our algorithm is optimal in the one–dimensional case. Chapter 5 describes a further development of the algorithm, which can in special cases result in an enhanced time performance for low dimensions.

2 The Minimum Measure Enclosing Problem

For any positive integer d, a *d-dimensional (closed) interval* I is written as a $2d$-tuple $(I_1, I_2, \ldots, I_{2d-1}, I_{2d})$, where $I_j \in \mathcal{R}$ for each $j \in \{1, \ldots, 2d\}$, and for each $k \in \{1, \ldots, d\}$, we have $I_{2k-1} \leq I_{2k}$. Let $\mathcal{I}^{2d} \subset \mathcal{R}^{2d}$ denote the set of all d-dimensional intervals. I_{2k-1} is the *lower*, I_{2k} the *upper boundary* of interval I in dimension k. In the sequel, we restrict ourselves to intervals of dimension d, unless the dimension is stated explicitly. An interval J *encloses* an interval I, denoted by $I \subseteq J$, if $I_{2k-1} \geq J_{2k-1}$ and $I_{2k} \leq J_{2k}$ for each $k \in \{1, \ldots, d\}$; if at least one of the inequalities is strict, J *strictly encloses* I, for short $I \subset J$. The *bounding box bbox(M)* of a set M of intervals is the unique interval for which $I \subseteq$ *bbox(M)* for all $I \in M$, and there is no other interval J with $J \subset$ *bbox(M)* and $I \subseteq J$ for all $I \in M$; in this sense, *bbox(M)* is the smallest interval enclosing all intervals in M. For an interval S, let $M_S := \{I | I \in M, I \subseteq S\}$ be the set of intervals in M that are enclosed by S, and let $|M_S|$ be their number. For the set $\mathcal{I}^{2d}$ of all possible d-dimensional intervals, let a *combined measure* function of *two* d-dimensional intervals be any function in the set $F := \{f | f : \mathcal{I}^{2d} \times \mathcal{I}^{2d} \rightarrow \mathcal{R}_+; \forall I, I', J \in \mathcal{I}^{2d} : f(I, J) = f(J, I)$ and $(I \subseteq I' \Rightarrow f(I, J) \leq f(I', J))\}$. Note that all measure functions that are of interest for splitting pages in data structures [BKSS90, Gre89, Gut84] belong to set F.

The Minimum Measure Enclosing Problem now can be defined as follows:

Minimum Measure Enclosing Problem ME(M, b): Given a set M of n d-dimensional intervals and a positive integer b, with $b \leq \lfloor n/2 \rfloor$, compute two intervals S and T with minimum measure such that

1. every interval in M is enclosed by S or by T (*enclosing condition*).

2. $|M_S| \geq b, |M_T| \geq b$ (*balance condition*).

To solve this problem, first observe that the boundaries of the intervals S, T of an optimal solution must be chosen from the set of boundary values of the intervals in M, i.e., for each boundary S_k of S there is an interval $J \in M$ whose boundary J_k equals S_k. This can be shown as follows. If *bbox*(M_S) is strictly enclosed by S, a solution of smaller measure can be found by shrinking S to *bbox*(M_S). The same statement and argument holds for T. A pair of intervals is called an *admissible solution*, if it fulfills the enclosing condition, the balance condition, and its boundaries are chosen from the respective set of boundaries of the intervals in M. An interval pair is called *optimal solution*, if it is an admissible solution and its measure is minimum.

For simplicity of the presentation, we assume that the intervals are in general position, i.e., they have pairwise different boundaries in each dimension. This is no loss of generality, since identical boundary values can be made distinct by shrinking the i-th given interval by $i \cdot \varepsilon$ at each boundary, for $\varepsilon > 0$ and $i = 1, \ldots, n$. It can be shown that for sufficiently small ε, this transformation preserves all admissible solutions in such a way that the relative order of the measure is unchanged among different admissible solutions, and hence any optimal solution is preserved.

A polynomial time algorithm

A brute force algorithm creates at least all instances of S with boundaries matching boundaries of some $J \in M$ by a combination of all possible values for each of the $2d$ boundaries. This can be done in time $O(n^{2d})$. T is created analogously. For all pairs (S, T) the enclosing condition and balance condition can be checked in time $O(d \cdot n)$. Not every measure function in our class F can be computed fast, but many of the interesting ones, like sum of the areas of S and T, sum of the perimeters of S and T, area of the union of S and T, perimeter of the union of S and T, area of the intersection of S and T, perimeter of the intersection of S and T, and combinations of these, can be computed in time $O(d)$. Since the complexity of evaluating other functions in F is not our primary concern, we assume throughout the paper that $O(d)$ time is enough for computing the value of the objective function for given S and T. Hence, the brute force algorithm needs total time of $O(d \cdot n^{4d+1})$.

A first improvement

Obviously, many useless interval pairs are built by this algorithm, and many pairs of boundaries do not even define intervals at all. Clearly, after S has been created, T should enclose at least all those intervals in M which are not enclosed by S. For this purpose we define the *complement* $I^c(M)$ of an interval I with respect to a set of intervals M (I^c for short) as the bounding box of those intervals of M that are not enclosed by I. Then we can state the following:

Lemma 2.1 *For any optimal solution (S, T) of ME(M, b) there is $T = S^c$ or $S = T^c$.*

Proof. Assume (S, T) is an optimal solution with $T \neq S^c$ and $S \neq T^c$. Because both S^c and T^c are bounding boxes, $T \subset S^c$ or $S \subset T^c$ implies that not each interval of M is enclosed by S or by T. Therefore, we only need to consider the case $T \supset S^c$ and $S \supset T^c$. Then, S^c and T^c each enclose less than b intervals, because otherwise either (S, S^c) or (T^c, T) would be an admissible solution with smaller measure. Hence, S and T each enclose more than $n - b \geq b$ intervals. T strictly encloses S^c, i.e., at least one of its boundaries differs from the boundaries of S^c. Therefore, an interval i in M exists that is enclosed by T and not by S^c, and hence is enclosed by S. Because T encloses more than b intervals, it is possible to shrink T by modifying this boundary without enlarging S or violating the balance condition. Therefore, (S, T) cannot be an optimal solution. □

Now we can modify the brute force algorithm slightly. As before, all admissible instances of S are created by the combination of boundary values. But then, for each instance S we compute T as the complement of S. For each instance of S, each interval J in M is inspected; if J is not enclosed by S, T is enlarged so as to enclose J. In this way, time $O(d \cdot n)$ is sufficient to create the complement $T = S^c$. The balance condition can be checked by keeping track of the numbers of intervals in S and T. Because of symmetry it is not necessary to create all admissible intervals T and S as the complement of T. The overall time complexity of this algorithm is therefore $O(d \cdot n^{2d+1})$.

A second improvement

Another observation can be made: Let I denote the bounding box of M. If (S, T) is an optimal solution, then each boundary of I must be matched by a boundary of either S or T (but not both, because of the general position assumption and Lemma 2.1). If one interval of S and T, say T, equals I, then it must be the complement of an interval S that has no boundary in common with I. In this case, S must enclose exactly b intervals, because otherwise S could be shrunk without violating the enclosing or the balance condition.

This fact can be used for a second improvement of the algorithm. We distinguish whether S has at least one boundary in common with I, or no boundary of S matches a boundary of I. In the first case, the instances of S can be created by equating one boundary of S with the corresponding boundary of I and combining all other $(2d - 1)$ boundaries of S as before. This results in no more than $2d \cdot n^{2d-1}$ different instances of S, a change in the number of instances by a factor of $O(d/n)$. In the second case, S is restricted to enclose exactly b intervals. In this case, the number of instances of S decreases, because $2d - 1$ given boundaries for S uniquely determine the last boundary. We leave the details to the next section, where we also describe how to compute the complementary T efficiently.

3 An Efficient Algorithm

Let us now describe an algorithm for the d-dimensional $ME(M, b)$. Let n be the number of intervals in M and I the bounding box of M. We say an admissible solution (S, T) of the $ME(M, b)$ is of *type 1*, if one or more boundaries of S equal the corresponding boundaries of I and T is the complement of S. An admissible solution (S, T), where S has no common boundary with I, therefore the complementary T is equal to I and S encloses exactly b intervals, is said to be of *type 2*. The first algorithm we propose computes admissible solutions of *type 1* in a first and admissible solutions of *type 2* in a second part.

Algorithm 1: *Compute an optimal solution (S, T) for ME(M, b).*

Part 1:

(1) **for** $j := 1$ **to** $2d$ **do**
    let $S_j := I_j$;
    { one boundary of S is fixed, $2d - 1$ boundaries left to be fixed }
(2) choose an arbitrary $k \in \{1, .., 2d\}, k \neq j$; $\{S_k$ will be fixed last$\}$
(3) **forall** combinations of admissible boundary values for
        all boundaries of S excluding S_j and S_k **do**
      { $2d - 1$ boundaries of S fixed, boundary S_k left to be fixed }
(4)   **foreach** admissible boundary value for S_k **do**
        { all boundaries of S are fixed }
(5)   let T be S^c;
(6)   **if** both S and T enclose at least b intervals each **then**
        { (S, T) is an admissible solution }
(7)   keep track of the minimum measure of the (S, T)
            computed so far

Part 2:

(8) **foreach** candidate S that has no boundary in common
      with I and that encloses exactly b intervals **do**
    { T as complement of S is always equal to I }
    { $|M_T| = n$, $|M_S| = b$, balance condition is fulfilled }
(9) keep track of minimum measure interval S

Before filling in the details in this algorithm, let us describe the data structure on which we base our operations.

Data Structure, Space and Preprocessing Complexity

Altogether the n intervals in M have $2dn$ different lower and upper boundaries. The boundary values are stored in $2d$ linear, doubly linked boundary lists $B_1, B_2, \ldots, B_{2d}$. The kth boundary of an interval $J \in M$ is stored in the list B_k, for all $k \in \{1, \ldots, 2d\}$. We need the values of each list to be sorted, lists with lower boundaries (lists with odd index) in ascending order, lists with upper boundaries (lists with even index) in descending order. The bounding interval $I = bbox(M)$ is then represented by the first elements of the boundary lists. For the purpose of simplifying the description of loops, we append to each boundary list one more element: At the end of each lower boundary list B_{2k-1} there is an element with the value of I_{2k}, and at the end of each upper boundary list B_{2k} there is an element with the value of I_{2k-1}. Elements of the same interval in different lists are said to *correspond* to each other. The set M is stored as a set L_M of n intervals, where each interval is an array containing $2d$ pointers to the corresponding elements in the boundary lists. In our algorithm, we want to access the boundary value of a given interval known from some interval list element and a given boundary index in constant time. To this end, we store with each element of each boundary list a pointer to the corresponding interval (array) in the interval set L_M. Because we need to mark an interval of M for different purposes, e.g.

to assign it to S or T, each element in L_M contains space for two binary markers. S and T are realized as arrays of $2d$ pointers to boundaries in the boundary lists, so called S- and T-pointers. Hence, there exist one set L_M with space $O(d)$ per element and $2d$ boundary lists with space $O(1)$ per element. In each of these lists, there are $n+1$ elements. The representation of S and T consumes space $O(d)$. Altogether, we need space $O(dn)$; that is, the required working space is linear in the length of the input.

Since the boundary lists must be sorted, we need the preprocessing time $O(dn \log n)$ for the $2d$ boundary lists.

Details of Algorithm 1

On the basis of precomputed interval boundary lists, let us now describe how the high-level operations of Algorithm 1 can be performed efficiently.

In Part 1, fixing one boundary S_j to be I_j amounts to setting the pointer of S_j to the beginning of list B_j, a constant time operation. In line (3), all requested combinations of boundary values can be obtained by repeatedly traversing all necessary $2d-2$ boundary lists; this amounts to a constant time operation for each combination of boundary values. In the innermost loop at line (4), the last boundary of S gets fixed. At the beginning of the loop, the pointer S_k is initialized to the last element of B_k; it is moved through list B_k to the first element during the loop. If w.l.o.g. k is an odd index, the last element of B_k equals the first element of B_{k+1}. This can be interpreted as S being flat in the dimension of k and therefore not enclosing any interval of M. As the pointer S_k moves to the beginning of B_k, S gets enlarged in each step of the loop.

The computation of $T := S^c$ in line (5) and the checking of the balance condition in line (6) can be done incrementally as follows. Because S does not enclose any interval at the beginning of the innermost loop, the complement T of S must enclose all intervals of M. Therefore, initially T equals I. At each step of this loop, S grows; by growing, S may or may not enclose a new interval J. Whenever S encloses a new interval in T that touches $T's$ boundary, T shrinks; otherwise, T is unchanged. To control the shrinking of T use for each interval J in L_M a binary marker, which tells wether or not S covers J. If T shrinks, it may only shrink to boundaries of intervals, which are not enclosed by S, i.e. that are not marked.

The balance condition is checked on the fly by counting the intervals of M that are enclosed by S and those enclosed by T. The number of intervals enclosed by S is maintained in a counter. This counter needs to be incremented whenever S encloses an interval as a result of a growth step; initially, it is zero. Similarly, a counter for the number of intervals enclosed by T is decremented whenever, as a result of a shrink step, an interval is no longer enclosed by T. However, the test whether a given interval leaves a shrinking T costs $\theta(d)$ computation steps, if it is performed by comparing all $\theta(d)$ boundary values. To get the complexity of this test down to $\theta(1)$ – an essential step for the overall efficiency of our algorithm –, we make use of the fact that each interval leaves T only once: a second binary marker for each interval indicates whether or not the interval is enclosed by T.

In more detail, the computation of the complement of S and the verification of the balance condition can be initialized before line (4) as follows:

remove all marks assigning intervals to S;
$Counter_S$:=0;
initialize the T-pointers to the first elements of the boundary lists;
mark all intervals to be assigned to T;
$Counter_T$:=n;

The body of the innermost loop now can be refined as:

```
{ let J denote the interval corresponding to the new
      boundary value of S_k chosen for S in the innermost loop }
if   J is enclosed by S   then
      mark interval J to be assigned to S;
      increment Counter_S;
      foreach   boundary T_i that equals the corresponding boundary J_i  do
          { change the boundary T_i }
          repeat
              if   the T_i-pointer refers to an interval J' assigned to T   then
                  decrement Counter_T;
                  remove the mark that assigns J' to T;
              move the T_i-pointer to the next element in its boundary list
          until   the T_i-pointer refers to an interval not assigned to S
else
      { the boundaries of T do not change }
```

The assignment of an interval J to S may cause a T-pointer not only to move to the next element of its list, but to skip over several boundaries of intervals that were assigned to S before J. But every T-pointer moves through its list in only one direction and therefore visits each of the n elements at most once during the execution of the innermost loop. Therefore, each T-pointer is moved in $O(n)$ steps through its list. For $2d$ lists, altogether $O(dn)$ steps are sufficient.

To analyze the time that suffices for Part 1 of Algorithm 1, note that the body of the outer loop runs exactly $2d$ times. Since there are n^{2d-2} combinations of the $2d-2$ boundary values for each looping, $O(dn^{2d-2})$ steps altogether suffice for all of Part 1 except the innermost loop times. Under the assumption that the evaluation of the objective function takes time $O(d)$, the innermost loop needs time $O(nd)$. Hence, Part 1 needs time $O(d^2n^{2d-1})$.

In Part 2 of Algorithm 1, we generate all candidates of S that have no boundary in common with I and enclose exactly b intervals. Hence, all $2d$ boundaries of S need to be fixed. But if $2d-1$ boundaries are already fixed, the last variable boundary must be placed such that exactly b intervals of M are enclosed by S. Therefore, we need only $2d-1$ nested loops to fix all boundaries. Figure 1 shows different instances of S in a two-dimensional case with already fixed lower and left boundary. While the right boundary moves to the right, the upper boundary moves down.

Similar to Part 1, all combinations of boundaries except S_{2d-2} and S_{2d} can be fixed by repeatedly traversing the necessary $2d-2$ lists. For each of these combinations, the boundary values of S_{2d-2} and S_{2d} are fixed as described below.

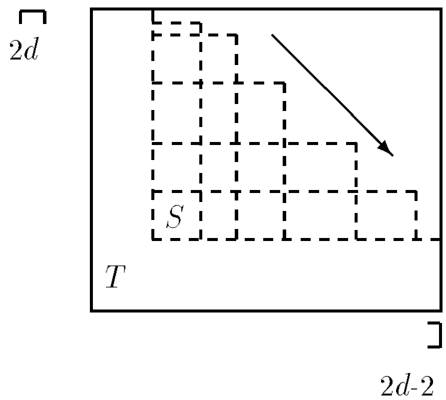

Figure 1

We define an interval or an element of a boundary list to be *active* with respect to some partially fixed S, if the already fixed boundaries of S enclose the corresponding boundaries of the interval. That is, active intervals are the only candidates that may be enclosed by the completed interval S. After having fixed all but two boundaries of S, all intervals can be marked as active or not active in time $O(dn)$. To do the marking, one of the two binary markers used in Part 1 for assigning intervals to S or T can be used here. If less than b intervals are active, S need not be completed. Otherwise, S is completed as follows:

let S_{2d} point to the first element of B_{2d}
    { i.e. highest upper boundary in dimension d };
let S_{2d-2} point to the next to last element of B_{2d-2}
    { i.e. lowest upper boundary in dimension $d-1$ };
move S_{2d-2} to the *b-th* active element of B_{2d-2},
    assign the corresponding b active intervals to S, and
    move S_{2d} to the boundary of the first interval in B_{2d} assigned to S;
keep track of the minimum measure interval S
while the first element of B_{2d-2} is not reached **do**
    move S_{2d-2} to the next active element towards the front of the list
        and assign the corresponding active interval to S;
    move S_{2d} to the next element towards the rear of the list
        whose corresponding active interval is assigned to S;
    keep track of the minimum measure interval S

To analyze the time complexity of Part 2 of Algorithm 1, observe that n^{2d-2} combinations of boundary values are fixed first. The remaining two boundary values are fixed by traversing two lists in time $O(n)$. The complement T of S equals I, and the balance condition is fulfilled. To compute the measure for a given S we spend time $O(d)$, as explained for Part 1 (for quite a few functions, measure computation can even be performed incrementally, so time $O(1)$ suffices). Hence, Part 2 takes time $O(dn^{2d-1})$. Together with the time complexity of preprocessing and that of Part 1, we have:

Theorem 3.1 *The d-dimensional Minimum Measure Enclosing Problem ME(M, b) can be solved in time $O(dn \log n + d^2 n^{2d-1})$ and space $O(dn)$.*

Corollary 3.2 *The one-dimensional Minimum Measure Enclosing Problem $ME(M,b)$ can be solved in time $O(n \log n)$ and space $O(n)$.*

Corollary 3.3 *The two-dimensional Minimum Measure Enclosing Problem $ME(M,b)$ can be solved in time $O(n^3)$ and space $O(n)$.*

4 A lower bound

To obtain a lower bound for the problem $ME(M,b)$ we look at the one-dimensional problem. Let I denote the bounding box of a set M of one-dimensional intervals. Because each of the boundaries of I has to be matched by either S or T, there are two different types of solutions (see Figure 2): Either each interval has one boundary in common with I, or one interval equals I and the other one encloses exactly b intervals.

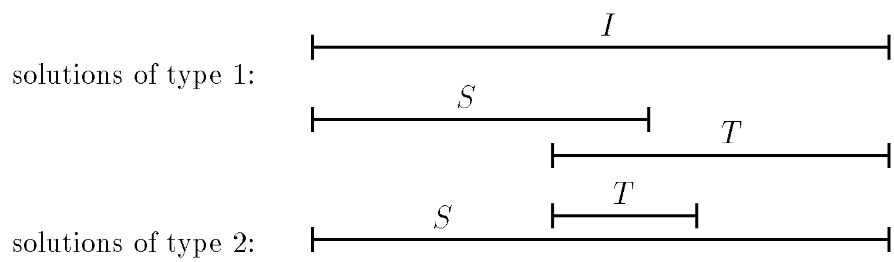

Figure 2

Lemma 4.1 *$ME(M,b)$ needs $\Omega(n \log n)$ operations in the algebraic decision tree model.*

Proof. We show that the *ε-Closeness Problem* [PS85] can be transformed in linear time to the Minimum Measure Enclosing Problem with balance constant $b = 2$. The ε-Closeness Problem is defined as follows: Given a set M of n real numbers $x_1, ..., x_n$ and a real number $\varepsilon > 0$, determine whether in M there exist x_i, x_j, $i \neq j$, such that $\mid x_i - x_j \mid < \varepsilon$. Let x_{max} denote the maximum and x_{min} denote the minimum in M. Transform each real number x_k into an interval $[x_k, x_k + \varepsilon]$, $k = 1, \ldots, n$, and build one interval $J = [x_{min}, x_{max} + \varepsilon]$. Finding the maximum and minimum value and the transformation of the given reals into intervals can be done in linear time.

An optimal solution (S,T) for the Minimum Measure Enclosing Problem with these $n+1$ intervals and balance constant $b = 2$ must be of type 2 (see Figure 2), because J has to be enclosed. Let S be the interval equal to I. Hence, T must enclose exactly two intervals. The measure of T is less than $2 \cdot \varepsilon$ if and only if there are two numbers in M with a distance less than ε. It follows that we can solve the ε-Closeness Problem, which needs $\Omega(n \log n)$ operations in the algebraic decision tree model, with the Minimum Measure Enclosing Problem. $\square$

Together with Corollary 3.2 we get

Theorem 4.2 *The one-dimensional Minimum Measure Enclosing Problem $ME(M,b)$ can be solved in optimal time $\Theta(n \log n)$.*

5 A faster algorithm for low dimension

In the previous section, the bound on the runtime of Algorithm 1 has been shown to match the lower bound for the one-dimensional problem. In any dimension, Algorithm 1 always actually needs the runtime stated as an upper bound in Theorem 3.1. In this section, we propose an alternative way of solving the Minimum Measure Enclosing Problem, and we show that this second algorithm may sometimes be faster, especially for low dimension, a practically interesting case.

To see how a finer grained case analysis leads to a potentially more efficient algorithm, let us recall the working principle of Algorithm 1. There, we create various instances of the first enclosing interval S, and we compute the second enclosing interval T as the complement of S. By fixing one boundary of S to the corresponding boundary of $I = bbox(M)$, on the one hand we reduce the number of created instances of S by a factor of $1/n$; on the other hand, since there are $2d$ choices for the boundary to be fixed to I, the net effect on the number of created instances of S is only a factor of $2d/n$. In addition, if S must enclose exactly b intervals, the number of created instances of S is reduced by a factor of $1/n$, since there is just one correct position for the last free upper boundary, and this position can be determined incrementally at constant cost.

Based on these observations, Algorithm 1 distinguishes between instances of S with one fixed boundary and $2d-1$ free boundaries and instances with $2d$ free boundaries. This distinction is useful only because in the latter case it is sufficient to create instances of S with exactly b enclosed intervals, whereas in the former case, for each instance of S the complement must be computed. This immediately raises the crucial question of whether the number of created instances of S can be cut down further, similar to the case of exactly b enclosed intervals. To this end, let us distinguish the instances of S according to the number of fixed boundaries. Algorithmically, let us create all instances of S with exactly k free boundaries, for $k = 0, 1, \ldots, 2d$; as before, all combinations of values for free boundaries, together with the fixed ones, yield the instances of S.

To see which of these instances can be created efficiently or need not even be created, recall that the complement of S with all boundaries fixed is the empty interval, and that the complement of S with no boundary fixed is I. In the general case, let us look at an instance of S with exactly k free boundaries; that is, exactly $2d-k$ boundaries of $T = S^c$ are not boundaries of I. We can see that not all instances of S with exactly k free boundaries need to be created by distinguishing between $k \leq d$ and $k > d$: the complements of S for $k \leq d$ coincide with instances of S for $k > d$. Let us now restrict k to $k \leq d$, and consider an instance of S with $2d-k$ free boundaries, together with its complement $T = S^c$. Due to the restriction on k, T has already been created (as a former instance of S, so to speak), and if (T, T^c) is an admissible solution, its measure certainly cannot be worse than the measure of (T, S), since $T^c \subseteq S$. If (T, T^c) is not an admissible solution, T^c does not enclose b intervals, and therefore we can shrink S so as to enclose just b intervals. We conclude that for more than d free boundaries it is sufficient to create instances of S with exactly b enclosed intervals. More formally, as a refinement of Lemma 2.1, we have:

Lemma 5.1 *For any optimal solution (S,T) of $ME(M,b)$, either ($S \neq T^c$ and $T = S^c$ and $|M_S| = b$) or ($S = T^c$ and $T \neq S^c$ and $|M_T| = b$) or ($S = T^c$ and $T = S^c$)*

Proof. For any optimal solution (S, T) by Lemma 2.1 there is $S = T^c$ or $T = S^c$. This leads to three cases, namely ($S \neq T^c$ and $T = S^c$) or ($S = T^c$ and $T \neq S^c$) or ($S = T^c$ and $T = S^c$). For the first case we will show that $|M_S| = b$. Assume $|M_S| > b$. Because $S \neq T^c$, S encloses T^c and has at least one boundary different from T^c. Therefore, some interval J in M is enclosed by S and not by T^c, hence it is enclosed by T. Because $|M_S| > b$, it is possible to shrink S by modifying one boundary without increasing T, and the balance condition is still satisfied, a contradiction. A symmetric argument holds for the second case and the third case is the same as in Lemma 2.1. □

An algorithm making use of Lemma 5.1 can then be formulated as follows:

Algorithm 2: *Minimum Measure Enclosure with more fixed boundaries*

Part 1:

(1) **for** $k := 0$ **to** d **do**
(2)   **foreach** combination of $2d - k$ boundaries of S fixed to I **do**
(3)   **forall** combinations of boundary values for the remaining
            k boundaries of S **do**
(4)   let T be S^c;
(5)   **if** both S and T enclose at least b intervals **then**
(6)   keep track of minimum measure;

Part 2:

(7) **for** $k := d + 1$ **to** $2d - 1$ **do**
(8)   **foreach** combination of $2d - k$ boundaries of S fixed to I **do**
(9)   **forall** combinations of boundary values for the remaining
            k boundaries of S such that S encloses exactly
            b intervals **do**
(10)   let T be S^c;
            { balance condition is always fulfilled }
(11)   keep track of minimum measure;

Part 3:

(12) **if** minimum measure computed so far > measure(I,empty) **then**
(13)   **forall** instances S with $2d$ free boundaries
        that enclose exactly b intervals **do**
(14)   let $T = I$;
        { balance condition is always fulfilled }
(15)   keep track of minimum measure

For the efficiency of Algorithm 2, the fast computation of the complement in line (10) is crucial. Similar to Algorithm 1, all instances of S are created by combining free boundary values (line (9)). Within the innermost loop (see also Figure 1), the two last free boundary values are fixed simultaneously. To compute the complement incrementally, we make use of the fact that each interval may enter S once and may leave S once within each innermost looping. An interval that enters S may shrink the complement $T = S^c$, and an interval that leaves S may grow T. Since each interval can shrink and grow T at most once, an implementation similar to the one described for Algorithm 1 achieves $O(d \cdot n)$ runtime to compute the complements of n instances of S. By adding the costs for preprocessing and for Parts 1, 2 and 3 of Algorithm 2, we get:

Theorem 5.2 *The d-dimensional Minimum Measure Enclosing Problem ME(M,b) can be solved in time*

$$O\left(d \cdot n \log n + d \cdot \left(\sum_{k=0}^{d}\left(\binom{2d}{k} \cdot n^k\right) + \sum_{k=d+1}^{2d-1}\left(\binom{2d}{k} \cdot n^{k-1}\right) + n^{2d-1}\right)\right).$$

Because Part 3 of Algorithm 2 is the most costly part, we get for the special case $d = 2$:

Corollary 5.3 *The two-dimensional Minimum Measure Enclosing Problem can be solved in time* $O(n^2)$ *if the minimum measure is smaller than the measure of (bbox(M),empty).*

Even though in some situations, such as those characterized in Corollary 5.3, Algorithm 2 is faster than Algorithm 1, in others the reverse may be true. Especially in cases where d is large compared to n, Algorithm 1 is clearly superior. That is, the proper choice of the algorithm must be made on the basis of the problem input at hand.

References

[AIKS89] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *Proc. of the 5th Annual ACM Symposium on Computational Geometry*, 5:283–291, 1989.

[BFG+91] B. Becker, P. G. Franciosa, S. Gschwind, T. Ohler, G. Thiemt, and P. Widmayer. An optimal algorithm for approximating a set of rectangles by two minimum area rectangles. In *7th Workshop on Computational Geometry, Bern, Lecture Notes in Computer Science*, page to appear, 1991.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *ACM SIGMOD International Conf. on Management of Data*, 19:322–331, 1990.

[Gre89] D. Greene. An implementation and performance analysis of spatial access methods. *Fifth IEEE International Conference on Data Engineering, Los Angeles*, 5:606–615, 1989.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. *ACM SIGMOD International Conf. on Management of Data*, 12:47–57, 1984.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry, an Introduction.* Springer-Verlag, New York, 1985.