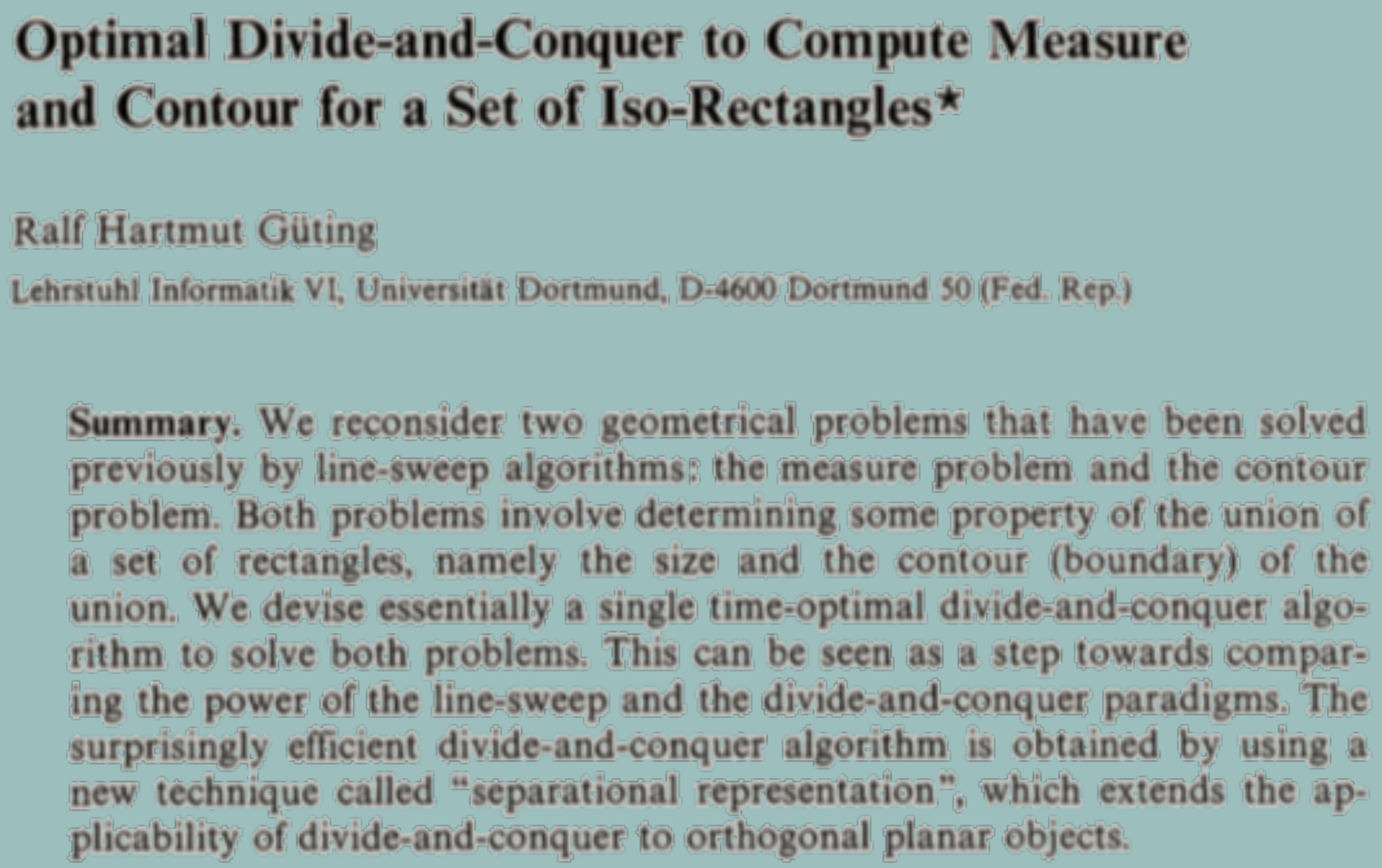


Optimal Divide-and-Conquer to Compute Measure and Contour for a Set of Iso-Rectangles*

Ralf Hartmut Güting

Lehrstuhl Informatik VI, Universität Dortmund, D-4600 Dortmund 50 (Fed. Rep.)

Summary. We reconsider two geometrical problems that have been solved previously by line-sweep algorithms: the measure problem and the contour problem. Both problems involve determining some property of the union of a set of rectangles, namely the size and the contour (boundary) of the union. We devise essentially a single time-optimal divide-and-conquer algorithm to solve both problems. This can be seen as a step towards comparing the power of the line-sweep and the divide-and-conquer paradigms. The surprisingly efficient divide-and-conquer algorithm is obtained by using a new technique called "separational representation", which extends the applicability of divide-and-conquer to orthogonal planar objects.

OPTIMAL DIVIDE-AND-CONQUER TO COMPUTE MEASURE AND CONTOUR FOR A SET OF ISO-RECTANGLES

Ralf Hartmut Güting, 1984

4. Implementation

謝德威 B05401009

REVIEW - STRIPES ALGORITHM

Given *set V: vertical edges, interval x_ext*

Return

set L, R: y-proj of left/right unpaired edges

set P: y-proj of end pts of V with $\pm\infty$

set S: stripes(rect(V), (x_ext, [-∞, +∞]))

Function STRIPES (V, x_ext):

If V.size() == 1: // only side v

If v.side == left: L = {v.y_interval}

Else: R = {v.y_interval}

P = {-∞, v.y_interval.bottom, v.y_interval.top, +∞}

S = {(i_x, i_y, ∅)| i_x = x_ext and i_y ∈ partition(P)} **[A]**

For s∈S s.t. s.y_interval = v.y_interval:

If v.side == left:

s.x_union = {v.coord, x_ext.top} **[B]**

Else:

s.x_union = {x_ext.bottom, v.coord} **[C]**

WHAT WAS THAT?

Stripe s (s.t. s.y_interval = v.y_interval)

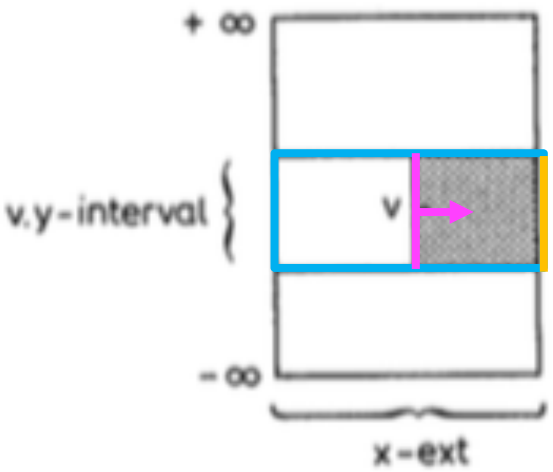


Fig. 5

If v.side == left:
    s.x_union = {v.coord, x_ext.top}

REVIEW - STRIPES ALGORITHM

If V.size() > 1:

Divide

Choose x_m and divide V into V_1 and V_2 s.t. V_1.size() ≈ V_2.size() .

Conquer

L_1, R_1, P_1, S_1 = STRIPES(V_1 ,[x_ext.bottom, x_m])

L_2, R_2, P_2, S_2 = STRIPES(V_2 ,[x_m, x_ext.top])

Merge

$L = (L_1 \setminus LR) \cup L_2$

$R = R_1 \cup (R_1 \setminus LR)$

$P = P_1 \cup P_2$

S_L = **copy**(S_1, P, ,[x_ext.bottom, x_m])

S_R = **copy** (S_2, P,[x_m, x_ext.top])

blacken(S_L, $R_1 \setminus LR$)

blacken(S_R, $L_1 \setminus LR$)

Return L, R, P, S = **concat**(S_L, S_R, P, x_ext)

COPY

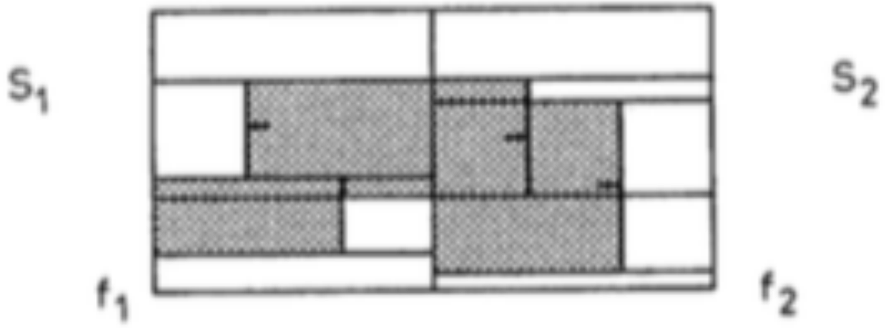


Fig. 8

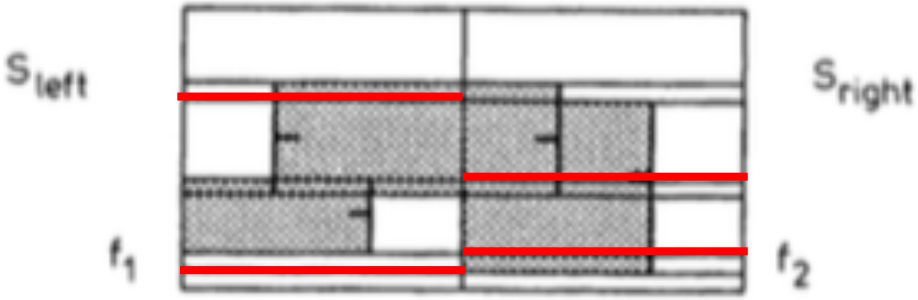


Fig. 9

Function copy (set S: stripes, set P, interval x_int):

$S' = \{(i_x, i_y, \varnothing) \mid i_x = x\_int \text{ and } i_y \in \text{partition}(P)\}$

Forall $s' \in S$:

For $s \in S$ s.t. s.y_interval $\supseteq$ s'.y_interval:

s'.x_union = s.x_union **[D]**

Return S'

BLACKEN

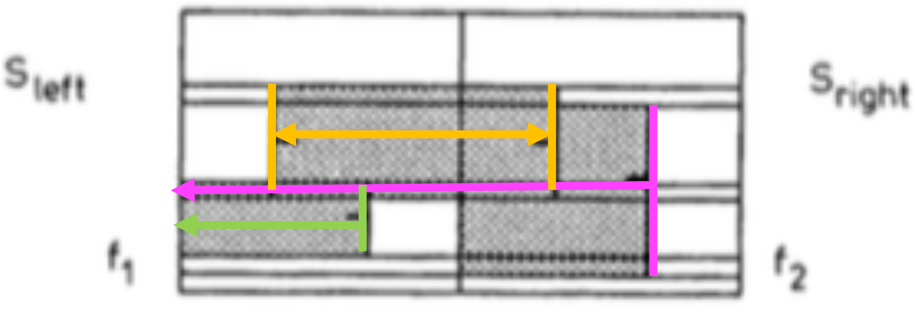


Fig. 9

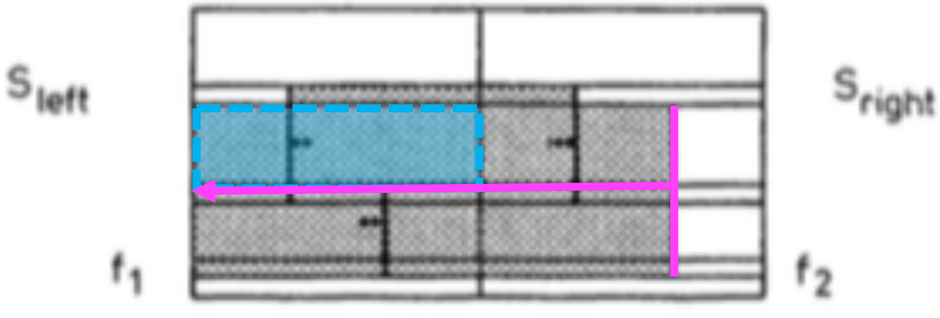


Fig. 10

Function blacken (set S: stripes, set P, set J: y intervals):

Forall s∈S:

if ∃i ∈j s.t. s.y_interal⊆i:

<u>s.x_union = {s.x_interval} **[E]**</u>

CONCAT

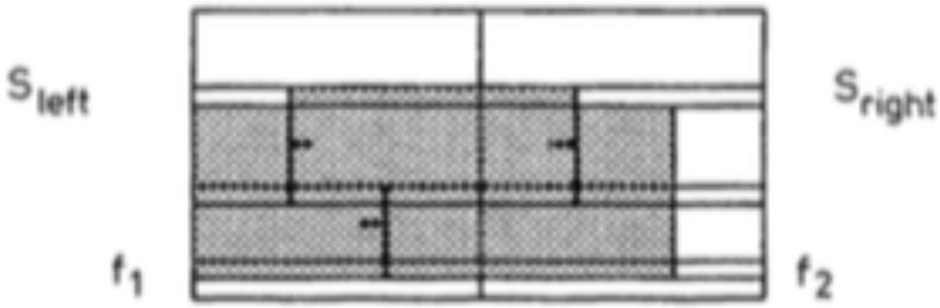


Fig. 10

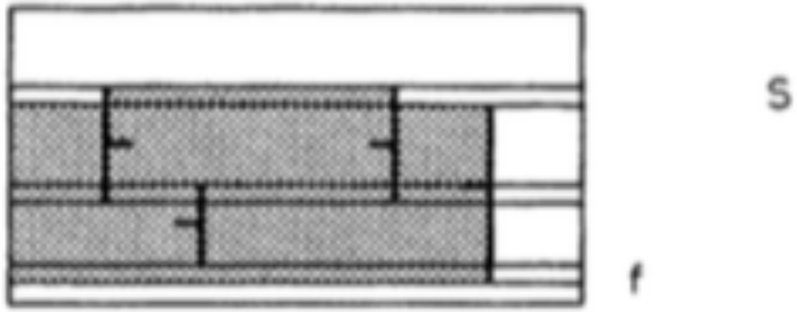


Fig. 11

Function concat (set S: stripes, set P, set J: y intervals):

$S = \{(i_x, i_y, \varnothing) \mid i_x = x\_int \text{ and } i_y \in partition(P)\}$

Forall $s \in S$:

For $s_1 \in S_1$, $s_2 \in S_2$ s.t. s.y_interval == s_1.y_interval and s.y_interval == s_2.y_interval:

s.x_union = s_1.x_union merge s_2.x_union **[F]**

Return S

THE MEASURE PROBLEM

Just replace s.x_union as s.x_measure (at the underlined parts of the algorithm, and “merge” = “+” in this case), which denotes the total length of the intervals contained.

THE CONTOUR PROBLEM

Need to compute: given disjoint x-intervals J, query q=[x1, x2], return $q \setminus (q \cap \text{union}(J))$ (i.e. free subintervals of q with respect to J)

Use Binary Search Tree to store endpoints of s.x_union in its leaves.

1. Each node contains a value x, an side-type (either left, right or undef) and pointers to left child / right child.
2. Replace s.x_union with s.tree (at the underlined part of the algorithm).
3. Handle tree operations with pointers.
4. Return interval [a, b] within $[x_1, x_2]$ for every pair of (a, right, empty, empty) and (b, left, empty, empty)

COPY

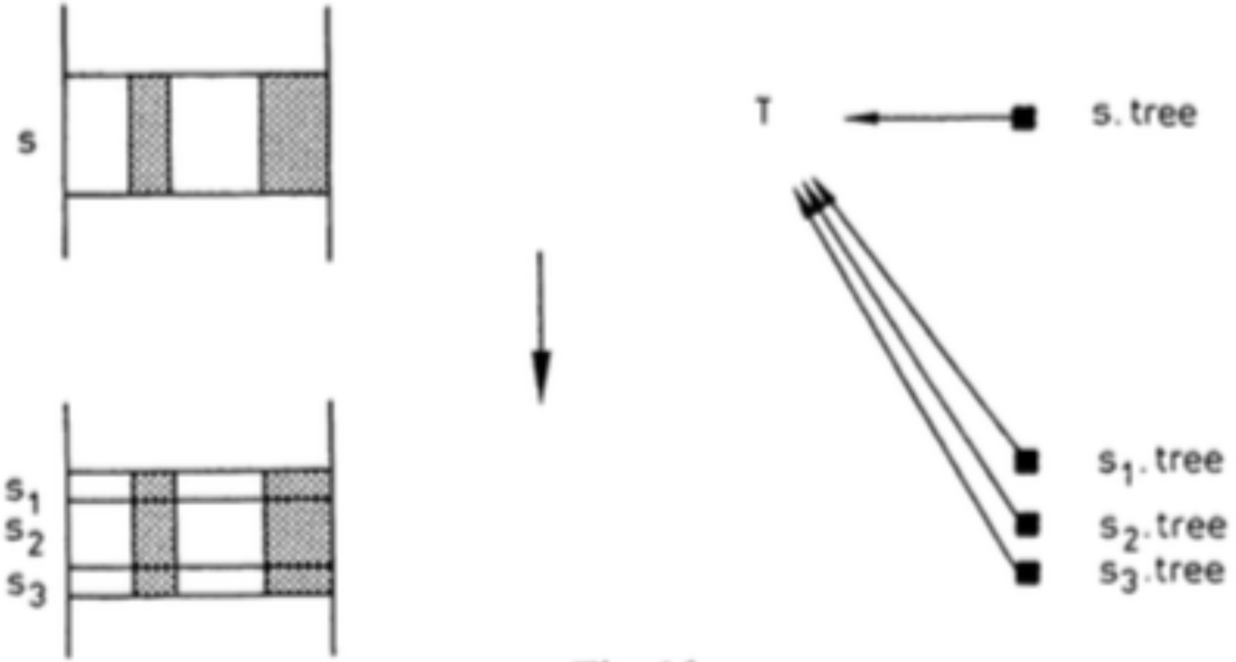
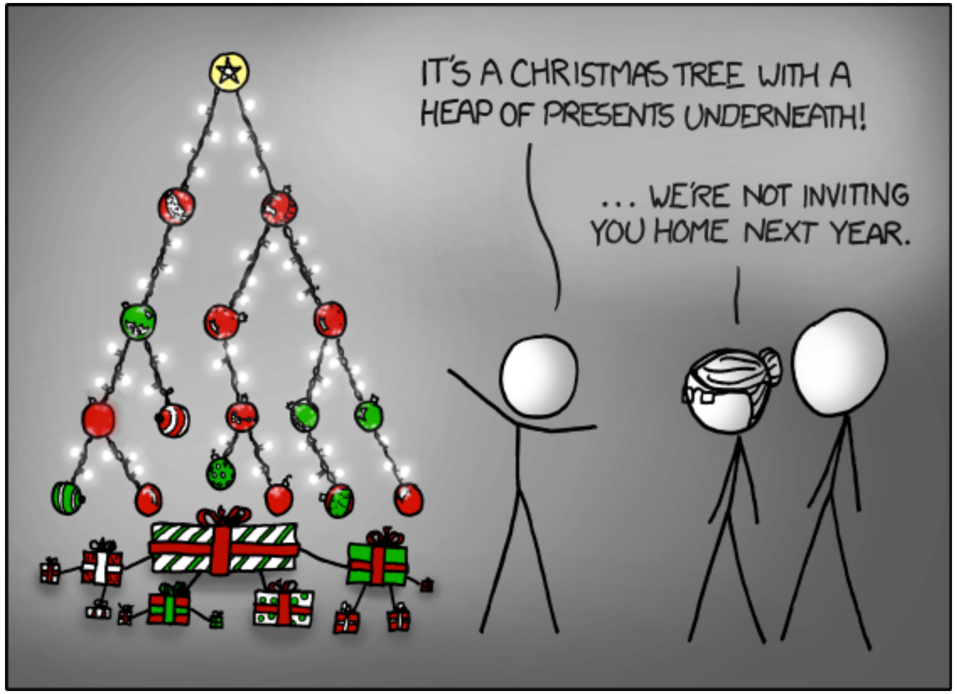


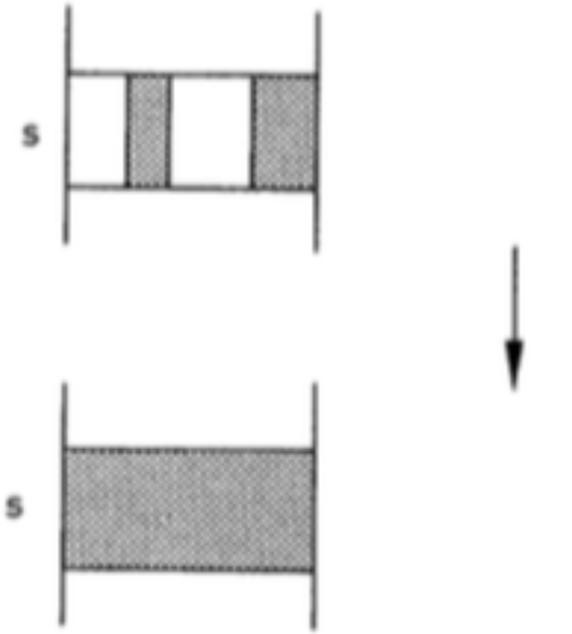
Fig. 16

[D] s'.tree = s.tree



https://xkcd.com/835/

BLACKEN



[E] s.tree = empty
(because no free subintervals left)

Fig. 17

CONCAT

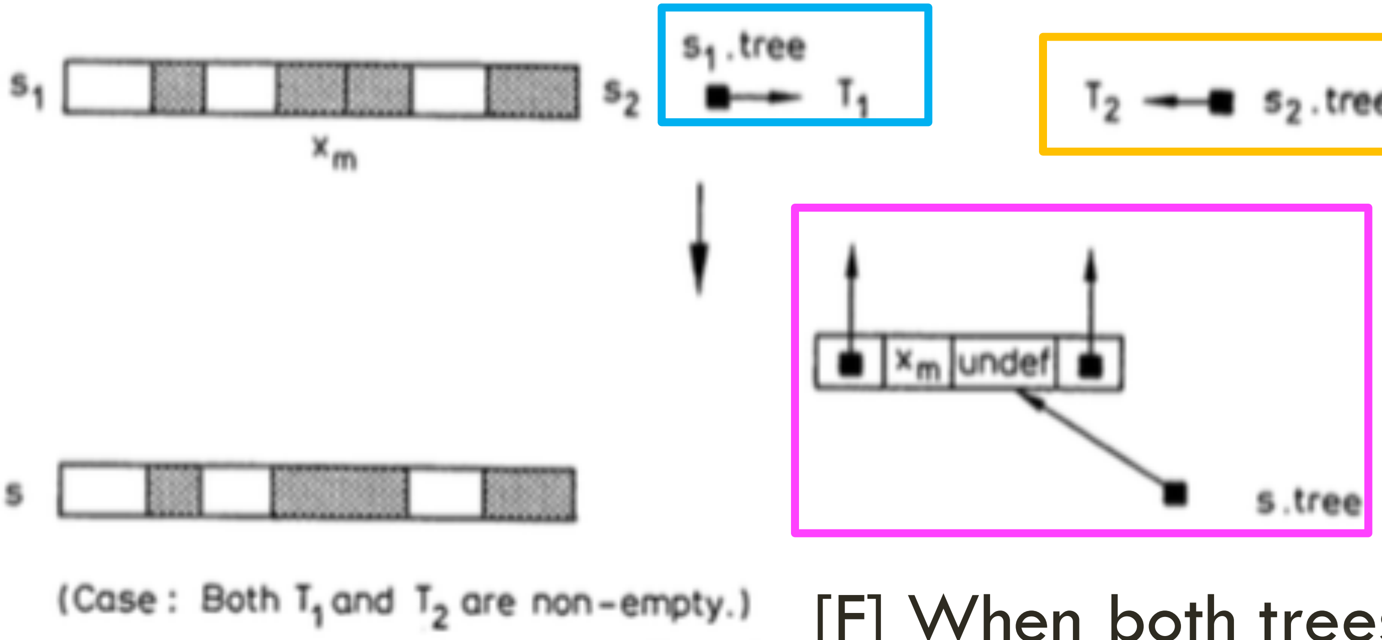


Fig. 18

[F] When both trees aren't empty,

s.tree = (s_1.x_interval.top, undef, s_1.tree, s_2.tree)

TIME COMPLEXITY

- ✓ STRIPES algorithm

  $T(n) = O(1) + 2T(n/2) + O(n)$

  $T(n) = O(n\log n)$

- ✓ For the contour problem:

  Each query takes O(tree height + # of free intervals)

  Total Time $O(n\log n + p)$, p = # of contour pieces.

  (Same as time-optimal line-sweep algorithms)