UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT

Optimal Algorithms to Compute the Closure of a Set of Iso-Rectangles

Eljas Soisalon-Soininen
Derick Wood

CS-82-26

August, 1982

OPTIMAL ALGORITHMS TO COMPUTE THE CLOSURE OF A SET OF ISO-RECTANGLES

Eljas Soisalon-Soininen & Derick Wood

Helsinki April 24, 1982

Report C-1982-31

Department of Computer Science
University of Helsinki
Tukholmankatu 2
SF-00250 Helsinki 25, Finland

ISSN 0357-3664
ISBN 951-45-2622-8

Optimal Algorithms to Compute the Closure of a Set of Iso-Rectangles*

by

Eljas Soisalon-Soininen† and Derick Wood††

* The work of the second author was partially supported by a Natural Sciences and Engineering Research Council of Canada Grant A-7700 and was carried out while visiting the University of Helsinki.

† Department of Computer Science
University of Helsinki
Tukholmankatu 2
SF-00250 Helsinki 25, FINLAND

††Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

A proposed running head: Computing the closure of iso-rectangles

Mailing address:

Prof. Derick Wood
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Abstract

We introduce three notions of the closure of a set of iso-(oriented) rectangles, that is rectilinearly-oriented rectangles, namely, uni-directional, diagonal and rectangular closure. We first prove a strong decomposition theorem for diagonal closure in terms of uni-directional closure. We then describe time and space optimal algorithms to compute uni-directional and diagonal closure, each running in O(nlogn) time and O(n) space. We also provide an O(nlogn) time and space algorithm for rectangular closure. Thus we obtain an optimal solution to the safeness problem for locked transaction systems, solving an open problem due to Lipski and Papadimitriou.

1. Introduction

Yannakakis, Papadimitriou and Kung [14], in an interesting paper, relate properties of locked transaction systems to geometrical properties of corresponding rectilinearly oriented rectangles, which we call iso-(oriented) rectangles. In particular they define the notion of the closure of a set of rectangles and prove that a locked transaction system is safe if and only if the closure of the corresponding set of rectangles consists of one connected region. In a follow-up paper, Lipski and Papadimitriou [7] derive an O(n log n log log n) time and O(n log n) space algorithm to determine the closure of a set of iso-rectangles. This is then used to solve the safeness problem with the same resource requirements, and also the deadlockfreeness problem, see [14]. They left open the problem of whether either of these problems can be solved in O(n log n) time, that is in optimal time, since the problems can easily be shown to require $\Omega(n \log n)$ time and $\Omega(n)$ space.

In this paper we focus attention on the notion of closure, and we introduce three varieties of closure. One of these, diagonal closure is the same as that considered by Lipski and Papadimitriou [7], while uni-directional and rectangular closure are new.

We decompose and characterize diagonal closure in terms of uni-directional closure and are, thereby, able to derive an O(n log n) time and O(n) space algorithm for both uni-directional and diagonal closure. Thus we obtain an optimal and simple algorithm to determine the safeness of locked transaction systems.

Finally we also provide an algorithm for rectangular closure requiring O(nlogn) time and space. Rectangular closure has applications in architectural, geographic and geometric data bases, when a set of planar geometrical objects must be partitioned such that the smallest bounding rectangles of the partitions are disjoint.

2. Preliminary Notions, Terminology and Theorems

In this section we will prove a decomposition theorem for diagonal closure in terms of the associated uni-directional closure operations, in particular, NESW-closure in terms of NE- and SW-closure. Although we are primarily interested in iso-rectangles, it turns out to be more convenient, in this section, to consider arbitrary regions in the plane. To this end a line will always denote a straight line, while a curve will always denote an arbitrary line, which has finite length, is not closed, is non-intersecting and its two endpoints have distinct x-values.

An iso-rectangle in the plane is defined by a quadruple (x_ℓ, x_r, y_b, y_t), which defines a rectangle whose corner points are (x_ℓ, y_b), (x_ℓ, y_t), (x_r, y_b) and (x_r, y_t).

We say two regions R_1 and R_2 in the plane intersect or overlap if they have at least one point in common. Let R be a region and p_1 and p_2 points in it. We say p_1 and p_2 are connected if there is a curve within R on which both p_1 and p_2 lie. We say that a region R is connected if all points p_1, p_2 in R are connected. If R is not connected then it can be expressed as the disjoint union of connected regions, that is $R = R_1 \cup \ldots \cup R_n$, R_i is connected, $1 \leq i \leq n$, $n \geq 1$ and no two R_i and R_j, $i \neq j$, overlap. The regions R_i are called the connected components of R.

In [6] an algorithm is derived to compute the connected components of a set of n iso-rectangles in O(nlogn) time and O(n) space. The present paper can be viewed as an investigation of variants of connectiveness.

Let R be a region and let $p_i = (x_i, y_i)$, $i=1,2$, be two connected points in R. We say p_1 and p_2 are comparable if

$x_2 < x_1$ and $y_2 < y_1$, and incomparable if $x_1 < x_2$ and $y_1 > y_2$. Hence comparable points determine a positive sloping line and incomparable ones a negative sloping line.

Let R be a region. Then the NE-conjugate of two incomparable points p_1 and p_2 in R is the point (x_2,y_1) and the SW-conjugate is (x_1,y_2). For p_1 and p_2 comparable we obtain the NW- and SE-conjugates (x_2,y_1) and (x_1,y_2), respectively. These are illustrated in Figure 2.1, they are the opposing corner points of the iso-rectangle determined by p_1 and p_2.

We say that R is NE-closed if for every two incomparable points p_1 and p_2 in R, the NE-conjugate of p_1 and p_2 is in R. In Figure 2.2 an NE-closed region and in Figure 2.3 an region which is not NE-closed are given. Similarly we obtain SW-closed, and also for comparable points NW- and SE-closed. If R is X-closed for some X in {NE,SE,SW,NW}, then R is said to be uni-directionally closed. If R is NE- and SW-closed (or NW- and SE-closed) it is said to be diagonally-closed. We abbreviate the particular direction as NESW-closed or NWSE-closed. Finally we say R is rectangular-closed if it is NE-, SE-, SW- and NW-closed, we also write R-closed in this case.

These notions lead quite naturally to the notion of closure. A region S is the X-closure of a region R, denoted by S = X(R), if S is the smallest X-closed region containing R, where X is in {NE,SE,SW,NW,NESW,NWSE,R}. In Figure 2.4 the NESW-closure of a region is displayed.

The importance of diagonal-closure comes from its correspondence to safeness and deadlock-freedom of locked transaction systems as can be seen in Section 5 and in [14]. Uni-directional closure plays an important role in the

computation of diagonal closure as Theorem 2.9 indicates. Finally, rectangular closure has applications in geometric data bases in which "well-formed" partitions of the objects are required. In particular, if R is connected then R(R) is the bounding box around R.

We will summarize some of the basic properties of closed regions in the following lemma.

Lemma 2.1

Let R be an arbitrary region and the connected components of R be $R_1,\ldots,R_n$, $n\geq 1$, where $R = R_1 \cup \ldots \cup R_n$. Let X be taken from {NE,SE,SW,NW,NESW,NWSE,R} unless specified otherwise. Then

(i) $X(R_i) \subseteq C_j$, for some j, $1\leq j\leq m$, where $X(R) = C_1 \cup \ldots \cup C_m$, $m\geq 1$.

(ii) $X(R) \subseteq X(S)$, for all regions $S \supseteq R$.

(iii) $Y(R) \subseteq NESW(R) \subseteq R(R)$, where Y equals NE or SW.

(iv) $Y(R) \subseteq NWSE(R) \subseteq R(R)$, where Y equals NW or SE.

(v) $NE(SW(R)) \subseteq NESW(R)$, $SW(NE(R)) \subseteq NESW(R)$, and similarly for NWSE.

(vi) For all sequences Y of operations X, $Y(R) \subseteq R(R)$.

Proof: They all follow almost immediately from the definitions. (v) and (vi) make use of (ii). □

In order to prove our main theorem we first need some further notions and lemmas.

Let C be a curve and p_1 and p_2 two points on it. Then we say p_1 precedes p_2 if p_1 is reached before p_2 when traversing C from its left end.

A curve is said to be non-decreasing if for all distinct points p_1 and p_2 on it, p_1 precedes p_2 implies $y_1 \leq y_2$. We say it is increasing if it is non-decreasing and for all distinct points p_1 and p_2 on it, p_1 precedes p_2 implies $x_1 \leq x_2$.

Let R be a region and $q = (x,y)$ a point, not necessarily in R. The point q determines four quadrants of the plane, for example the NW-quadrant is composed of all points $q' = (x',y')$ for which $x' \leq x$ and $y' \geq y$. We say R blocks a quadrant if all semi-infinite lines from q in the quadrant pass through R.

In the following development for simplicity of presentation we only consider NE-, SW- and NESW-closure and R-closure.

Lemma 2.2

Let R be a connected region. Then

(i) NE(R) has as NE corner the NE corner of R(R),

(ii) SW(R) has as SW corner the SW corner of R(R), and

(iii) for NESW(R) both (i) and (ii) hold.

(See Figure 2.4.)

Proof: R(R) is the bounding box of R, when R is connected, hence there is a point p_1 in R and on the N edge of R(R), similarly there is a point p_2 in R on the E edge of R(R). But this means that the corner point q of R(R) is in NE(R) since it is the NE-conjugate of p_1 and p_2. Moreover every point q' in the SW-quadrant of q such that q' lies above R and to the right of p_1 is in NE(R) since there are two points in R of which q' is the NE-conjugate. The other two cases follow similarly. □

The (outer) contour of a connected region R is the closed curve C within R such that all points in R - C are inside C, that all four quadrants are blocked by C. For an

NE- or NESW-closed connected region R, the upper contour is the portion of the contour, lying above R, from the leftmost point in R (and if there is more than one such point, the bottommost one) to the NE corner point in R. If R is a SW- or NESW-closed connected region, then the lower contour is the portion of the contour, lying below R, from the SW corner point in R to the rightmost point in R (and if there is more than one then the topmost one).

Lemma 2.3

Let R be a connected NE-closed region. Then the upper contour of R is a non-decreasing curve.

Proof: Assume the contrary. Then there exists two points p_1, p_2 on the contour such that p_1 precedes p_2 but $y_1 > y_2$. Pictorially, see Figure 2.5. But this contradicts the assumption that R is NE-closed since there exists a point q not in R which is the NE-conjugate of two points r and p_2 in R. □

Similarly we obtain:

Lemma 2.4

Let R be a connected SW-closed region. Then the lower contour of R is a non-decreasing curve.

And also:

Lemma 2.5

Let R be a connected NESW-closed region. Then its contour consists of two increasing curves, the upper and lower contour.

Lemma 2.6

Let R be a connected NE-closed region and let S = SW(R). Then the contour of S consists of two increasing curves. The same result holds if R is SW-closed and S = NE(R). In both cases S is NESW-closed.

Proof: The upper contour of R is non-decreasing by Lemma 2.3. If it is increasing then SW-closure has no effect upon the upper contour, since for all points q above R, the NW-quadrant of q never cuts R; pictorially see Figure 2.6. But this ensures that there are no two points p_1 and p_2 connected in R of which q is the SW-conjugate.

On the other hand if the upper contour of R is properly non-decreasing, then there exist points $p_1 = (x,y_1)$ and $p_2 = (x,y_2)$ on the contour such that $y_1 < y_2$ and that $x' > x$ for all points (x',y') on the contour with $y_1 < y' < y_2$. Consider any point $q = (x,y)$ such that $y_1 < y < y_2$. Clearly there exists a point $p = (x',y)$ in R (on the upper contour) such that q is the SW-conjugate of p_2 and p. But this means SW-closure fills these "dents" smoothing the upper contour, until all such points q are absorbed. Thus we are left with an increasing upper contour.

Second consider the lower contour. Since S is SW-closed, by Lemma 2.4 the lower contour is nondecreasing. If it is increasing, the required result has been obtained. Hence assume it is not increasing. But this means that there exist points $p_1 = (x,y_1)$ and $p_2 = (x,y_2)$ on the lower contour such that $y_1 < y_2$ and that $x' < x$ for any point (x',y') on the contour with $y_1 < y' < y_2$. Furthermore p_1 and p_2 can be chosen

such that $x' < x$ also for any point (x',y') on the contour such that $y' < y_1$. Now p_1 and p_2 are in R also, because the SW-quadrant neither at p_1 nor at p_2 ever cuts R; pictorially see Figure 2.7. Furthermore, all points q on the straight line joining p_1 and p_2 are NE-conjugates in R of p_1 and some point p_3 in R, because R is connected. Hence q is in R, which is a contradiction, and therefore the lower contour is increasing.

The other statements follow similarly. □

Thus we have obtained:

Theorem 2.7

Let R be a connected region. Then NE(SW(R)) = SW(NE(R)) = NESW(R).

Proof: By Lemmas 2.1 and 2.6. □

We now extend Theorem 2.7 to disconnected regions by means of the following lemma.

Lemma 2.8

Let R be an arbitrary region and the connected components of R be $R_1,\ldots,R_n$, $n\geq 1$, where $R = R_1 \cup \ldots \cup R_n$, and let R be NE-closed. Then SW(R) is NESW-closed. Similarly if R is SW-closed then NE(R) is NESW-closed.

Proof: We only prove the first part of the lemma. The proof of the second part is completely analogous. The proof is by induction on the number of connected components of R. If n=1, Theorem 2.7 implies the lemma. Then let k be a positive integer, and assume that SW(R) is NESW-closed, whenever NE-closed R has n connected components, $n \leq k$. Consider the case $n = k+1$, i.e.,

R has k+1 connected components. First observe that each connected component R_i of R is NE-closed. Let $S = R_1 \cup \ldots \cup R_k$, $S' = SW(S)$ and $R' = SW(R_{k+1})$. We claim that $SW(R) = SW(R' \cup S')$ is NESW-closed. It is enough to show that $SW(R' \cup S')$ is NE-closed, since then it is NESW-closed. Clearly, if $R' \cap S' = \emptyset$, $R' \subseteq S'$ or $S' \subseteq R'$, then the result holds trivially. Hence assume that R' and S' are incomparable.

Assume $SW(R' \cup S')$ is not NE-closed. Then there exist two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in $SW(R' \cup S')$, where $x_1 < x_2$ and $y_1 > y_2$, such that the NE-conjugate $q = (x_2, y_1)$ is not in $SW(R' \cup S')$. Now p_1 and p_2 cannot both be in R' or in S' because both R' and S' are NE-closed by the induction hypothesis. Further because both R' and the connected components of S' have increasing upper and lower contours, it should be clear by the geometrical properties of $SW(R' \cup S')$ that both p_1 and p_2 must be in $R' \cup S'$. Assume, without loss of generality, that p_1 is in R' and p_2 is in S'. Now S' contains a connected region S_1' such that p_2 is in S_1', and the upper contour of S_1' passes between q and p_2 and the lower contour of R' between p_1 and q. Moreover these contours must intersect because p_1 and p_2 are connected in $R' \cup S_1'$ and thus R' and S_1' overlap. Consider the topmost intersection point of these contours, which we call r. We derive a contradiction by examining the point r.

Now r is on the upper contour of S_1'. Therefore if S_1' is neither vertical nor horizontal at r, then r is in S. This follows by observing that in this case the NW-quadrant of r is not cut by S_1', and hence r is not the SW-conjugate of any two points in S_1'. If S_1' is horizontal at r, then r is in S,

since otherwise S is not NE-closed, as assumed.

Similarly r is on the lower contour of R'. Again if R' is neither vertical nor horizontal at r, r is in R_{k+1}. If R' is vertical at r then r is in R_{k+1}, because otherwise R_{k+1} is not NE-closed.

In all combinations of the above cases for R' and S_1', we see that r is in $R_{k+1} \cap S$. This contradicts the assumption that $R_{k+1} \cap S$ is empty.

Consider the remaining cases:

(a) S_1' vertical and R' not vertical at r. Then r is not the highest intersection point.

(b) S_1' not horizontal and R' horizontal at r. As (a).

(c) S_1' and R' both vertical at r. Then r must also be in S, since it is a corner point, which cannot have been added by SW-closure. Thus, r is in $R_{k+1} \cap S$.

(d) S_1' and R' both horizontal at r. Then r must also be in R_{k+1}, if r is the rightmost intersection point. Also in this case $R_{k+1} \cap S$ is nonempty.

Hence in all cases a contradiction has been obtained, therefore SW(R'US') = SW(R) is NE-closed, as desired. □

Theorem 2.9 The Decomposition Theorem

Let R be an arbitrary region. Then NE(SW(R)) = SW(NE(R)) = NESW(R).

Proof: By Lemma 2.8 SW(NE(R)) is NESW-closed and by Lemma 2.1 SW(NE(R)) is included in NESW(R). Hence we obtain equality. □

In concluding this section, we return from the considerations of general regions to those of primary interest to us, namely sets of rectangles. In Figures 2.8, 2.9, and 2.10 we illustrate the NE-, SW- and NESW-closure of a set of rectangles. In this regard it should be observed that the upper contour of NE-closed regions and the lower contour of SW-closed regions are nondecreasing step functions, while the lower and upper contours of the NESW-closed regions are both increasing step functions.

3. Uni-Directional and Diagonal Closure

In reading the algorithm of Lipski and Papadimitriou [7] one certainly gets the feeling that there must be an easier way to compute the diagonal closure. They use the scan line paradigm, that is a left-to-right sweep over the rectangles, to compute the NESW-closure of a set of rectangles. Unfortunately their stratagem ensures that what happens at the scan line can affect objects lying completely to the left of it, that is they are dead but they are not allowed to lie down. Typically the power of the scan line approach stems from the fact that what is at the left of the scan line is now dead and cannot be further affected. See [3] and [11] for example. In our approach based on the Decomposition Theorem of Section 2, we carry out two scans, one after the other. We first carry out a left-to-right scan to compute the NE-closure and second, a right-to-left scan to compute the SW-closure of the NE-closure. The Decomposition Theorem asserts that this results in the NESW-closure, as required.

However the output of our algorithm is the partition of the set of rectangles defined by their NESW-closure, rather than the contours of the NESW-closed regions. But in a third phase we can also compute the contour as well, and this can be absorbed into the first two-pass algorithm. We, therefore, have a two-pass algorithm operating in O(nlogn) time and O(n) space as we shall demonstrate. The correctness of each pass is straightforward to verify, because of their simplicity. However as we shall see in the next section the rectangular

algorithm is descriptively more complex, because it, too, fails to meet the "dead is dead" criterion outlined above.

Recall that given a set of rectangles, the generic position of the scan line divides them into three classes. These are composed of those to the left of the scan line, those cutting the scan line and those to the right of the scan line, known as the dead, active and inactive rectangles respectively. Since these classes only change whenever a left or right end of a rectangle is met, the scan line moves in discrete steps from left to right through the corresponding x-values. We shall also be using a right to left scan line, when the classification of dead and active rectangles will be reversed.

The basic idea on the left to right scan is to compute the NE-closure of the set of rectangles. Thus there will also be dead and active (connected) components, which correspond to the completed and partially completed NE-closure, respectively, at each scan line position. The currently active components correspond to disjoint intervals on the scan line, we call these waves. On meeting a new rectangle it forms a wave W, whose extremities are given by the bottommost and topmost y-values in the rectangle. Since W is a newly created wave, it possibly overlaps a number of those already present. If so these are all replaced by one new wave and if not W corresponds to the beginning of a new component. Observe that the properties of NE-closure prevent any rewriting of history here. When the right end of a rectangle is met, this can mean the completion of a component, the shrinking of a wave or perhaps have no affect at all. It is worth remarking that in

computing the NE-closure, the top of a wave can never become lower. The bottom of a wave can, however, become higher, when a rectangle is deleted, or expand, when a merger takes place.

At the conclusion of the left-to-right scan the rectangles have been partitioned into classes corresponding to NE-closed regions.

The secret of the second scan is simply that it begins with the classes of rectangles given in the first pass. Otherwise the second pass is very similar to the first one. Before giving the complete algorithm in more detail, consider the primitive operations in a little more detail.

First, since we are essentially computing equivalence classes, that is the NE- and NESW-closed classes are equivalence classes, we assume the existence of a basic UNION-FIND structure to deal with these. It is sufficient for our purposes that both operations can be carried out in O(logn) time, and that the structures used require O(n) space. However such structures are abundant, see [1], for example. Minor modifications to these structures are needed, as we shall see, but they clearly do not have a deleterious affect on time and space bounds.

Second, we need a structure in which we can insert and delete waves and also carry out a wave query to determine all waves the given one overlaps. However since waves are by their nature disjoint y-intervals, we can base our structure on a leaf search tree. For example in Figure 3.1 we have three waves, whose bottom and top y-values are (1,2), (3,4) and (5,6) respectively. The internal nodes have been filled in using the maximum-value-in-the-left-subtree routing scheme

and the leaves have been doubly liked together. Note that it is a height-balanced or AVL-tree. The insertion or deletion of a wave can be carried out in O(logn) time and the resulting tree can be restructured within O(logn) time to once more be an AVL-tree. The addition of the double linking at the leaves and the leaf search nature of the tree only cause minor changes in the standard algorithms. Checking for the overlaps that occur with a query wave $W = (w_b, w_t)$ corresponds to carrying out two searches, from root-to-leaves, one for w_b and the other for w_t. For example letting $W = (1.5, 4.5)$, then the searches end up at the leaves containing 2 and 5, respectively. Since $4.5 < 5$ it falls between the second and third waves, and since $1.5 < 2$ W overlaps both (1,2) and (3,4). Note the necessity to also keep with the leaf values whether they are bottom or top values. This ensures it can be determined that the query (1.5,1.8) overlaps (1,2) but the query (2.5,2.8) overlaps no wave. Again this additional information causes little change to the AVL-maintenance algorithms.

Let us now give a more detailed description of the algorithm.

NESW-closed components algorithm

Input: A set of n iso-rectangles ($n \geq 1$) each given by a quadruple (x_ℓ, x_r, y_b, y_t).

Output: The NESW-closed components defined by the rectangles.

Phase I: The left-to-right scan

Step1: Sort the x_ℓ- and x_r-values into ascending order (with ties broken with the left-before-right rule). The sorted $x_1,\ldots,x_{2n}$ form the scan list. Let AW denote the active wave structure, which is initially empty. The UNION-FIND structure is also initialized to empty. Let i=1.

Step2: If i = 2n+1 then goto Phase II otherwise consider x_i. It is either the left or right end of a rectangle R, say.

2.1. x_i is the left end of R

Create a new active component C consisting of R, and the corresponding new wave W defined by $y_{i,b}$ and $y_{i,t}$.
Set the active count of C to 1 and let the status of R in C be "active".
Determine the waves in AW which W overlaps.
If none overlaps then insert W into AW and C into the UNION-FIND structure, increase i by one and goto Step2.
Otherwise merge W and the overlapping waves in AW, updating the UNION-FIND structure (the union of C and the components corresponding the overlapping waves is formed and the active count of the new component is the sum of the counts in the unioned components), deleting the merged waves from AW and inserting the newly created wave in their place.
Increase i by one and goto Step2.

2.2. x_i is the right end of R

Determine the component C to which R belongs using the UNION-FIND structure, and set the status of R to "dead", decrementing the active count of C.

If the active count is zero then delete the wave of C from AW, mark C as "dead" in the UNION-FIND structure, increase i by one and goto Step2.
Otherwise if R is the bottommost rectangle among the active rectangles in C, then shrink the wave upwards to y_b of the next lowest active rectangle in C.
Finally, increase i by one and goto Step2.

End of Phase I.

There are only two new ideas introduced in this more detailed version of the algorithm and both are in Step 2.2. The first is that each active component C needs to keep track of its wave in the structure AW. This is another simple addition to the UNION-FIND structure. The second is that we need to keep the bottommost y-values of the active rectangles in each component C. This can be done by adding a separate dictionary to C, using an AVL-tree once again, which is updated when meeting the left and right ends of rectangles. This structure could also be used instead of the active count, since it is empty if and only if the active count is zero.

After this interlude let us continue with the next phase.

Phase II: The right-to-left scan

Step3: The sorted x-values $x_{2n},\ldots,x_1$ form the scan list. Let AW be empty once more and leave the UNION-FIND structure as it is. Let i=1.

Step4: If i = 2n+1 then finish otherwise consider x_i. It is either the right or left end of a rectangle R, say.

4.1. x_i is the right end of R

Determine the component C of which R is a member. If C is "dead" (from Phase I) then change its status to "active", and let its wave be W = (bottom(R),top(C)).
Otherwise determine the wave W associated with C.
In all cases change R to "active".
If bottom(R) < bottom(W) then delete W from AW and let W be expanded to bottom(R).
If W has been expanded or activated then determine the waves in AW, which W overlaps.
If none overlaps then insert W into AW, increase i by one and goto Step4.
Otherwise merge W and the overlapping waves, updating the UNION-FIND structure, deleting the merged waves from AW and inserting the newly created wave in their place.
Increase i by one and goto Step4.

4.2. x_i is the left end of R

Determine the component C to which R belongs using the UNION-FIND structure and set the status of R to "dead", decrementing the active count of C.
If the active count is zero then delete the wave W of C from AW, mark C as "dead" in the UNION-FIND structure, increase i by one and goto Step4.
Otherwise if R is the topmost rectangle among the active rectangles in C, then shrink the wave downwards to the y_t of the next highest active rectangle in C.
Finally increase i by one and goto Step4.

End of Phase II.

Readers should compare Phases I and II to find that they are almost identical. The major distinctiveness of Phase II being that it begins with the knowledge of the NE-closed components. This means that an incoming rectangle can only overlap other waves, if it either activates its NE-component or expands its wave. In Phase II a wave can never shrink upwards, while in Phase I it can never shrink downwards. The notations top(C), bottom(R), bottom(W) should be self-explanatory. However in the case of C this means that we should keep with it the coordinate of the NE-corner point, which is of course easily available after Phase I.

End of NESW-closed components algorithm

That this algorithm does indeed correctly compute the NESW-closed components is the result of the following theorem.

Theorem 3.1

Given a set of n iso-rectangles the NESW-closed components algorithm correctly computes the NESW-closed components of them and Phase I correctly computes the NE-closed components.

Proof: We split the proof into two steps corresponding to the two phases. If $n = 1$ (or 0) the theorem is immediately true.

1) Assume that the scan line is at the ith position, $i \geq 1$, and that up to and including this position the partial NE-closed components have been correctly computed. Now consider the (i+1)st position.

Either we meet the left end of a

rectangle R, in which case a new component C and a new wave W is created for it. The waves at this scan line position correspond to NE-closed components and in particular each wave can be viewed as in Figure 3.2. The bottom of the wave corresponds to the bottom of an active rectangle and the top of the wave to the top of a, possibly dead, rectangle. Whenever W overlaps such a wave W', it is clear that it either overlaps a rectangle in the corresponding component C' or it overlaps a region defined by the topmost rectangle in C', and this region is in the NE-closure of it. In both cases R and C' must be in the same component of the NE-closure of the rectangles.

Or we meet the right end of a rectangle. The key issue here is that a bottommost rectangle causes the wave to shrink upwards, and this is indeed correct, while any other rectangle including a topmost one produces no change in the wave. The latter action is also correct since "holes" and NE-openings are filled in by NE-closure. Hence we conclude that Phase I operates correctly.

2) A similar argument to that used in (1) will demonstrate that after Phase II has been completed the NESW-closed components will have been found correctly. □

The time and space bounds are the content of the next result.

Theorem 3.2

Given a set of n iso-rectangles, the uni-directional and diagonal closed components can be computed in O(nlogn) time and O(n) space, that is in optimal time and space.

Proof: The space bound is immediate from the structures used in our algorithm, while at least $\Omega(n\log n)$ time is needed because sorting occurs in Step1. The querying of the active wave structure AW takes $O(\log n + k)$ time, where k is the number of intervals responding to a query. Thus Step2 and Step4 may at one time need to carry out $O(n)$ merges at worst. However at most n-1 merges of n objects can take place. Hence we obtain $O(n\log n)$ time overall. Similarly the time needed for other primitive operations is globally bounded by $O(n\log n)$.

Optimality for space follows because n objects need $O(n)$ space, while optimality for time follows in the usual way for these geometric problems, that is via the element uniqueness problem. □

To complete this section we briefly discuss how to modify the NESW-algorithm to also report the contour of the NESW-closed region. We can basically do this in another two passes of the final NESW-connected components that have been obtained. Basically a left-to-right scan enables us to compute the upper contour of each region, while a right-to-left scan enables us to compute the lower contour. The key is to consider the way the currently active wave changes, by expanding upwards on the first pass and expanding downwards on the second. No other changes are monitored. Clearly the first pass changes correspond to left ends of rectangles, while the second pass changes correspond to the right ends of rectangles. Each such change defines a horizontal line segment, from the last changing x-position and a vertical line segment

corresponding to the extent of the change.

These observations bound the number of edges in the contour of each NESW-closed region to be O(n), if there are n rectangles in the region. Finally, it should be clear that the contour computation can be carried out during Phases I and II, except that some of the upper contours found in Phase I will either disappear in Phase II or be merged with others. We leave to the reader the laborious but straightforward details of this change, which leads to our final theorem.

Theorem 3.3

Given a set of n rectangles, the NESW-closed contours can be computed in O(n) space and O(nlogn) time.

4. Rectangular Closure

Since the decomposition of diagonal closure into the composition of two unidirectional closure operations has been so succesful, the reader might, initially, be seduced into taking a similar approach for rectangular closure. However there does not seem to be any decomposition theorem corresponding to Theorem 2.9. Basically, this is simply because closure in any one direction can interfere with the closure in other directions in this setting. In Figure 4.1 this is illustrated using 4 NENWSW-closed regions. The reader should observe that when applying SE to "complete" the R-closure, then regions 3 and 4 are not R-closed. This is because the shaded region cannot be added under SE. Because of this counterexample it is necessary to rethink R-closure from the beginning, since one would, intuitively, expect it to be simpler that diagonal closure.

To this end we once more consider the scan line paradigm together with the ideas of Lipski and Papadimitriou [7] for diagonal closure. Thus at a generic position of the scan line we have some active components (note that these are always rectangles), some "dead" components, which have already been passed over and some inactive rectangles, still to be processed. See Figure 4.2 for an example of this situation. The dotted rectangles are "dead" components, the shaded ones are the "active" components and the remaining ones are the inactive rectangles.

As in the diagonal closure algorithm there are only two critical positions of the scan line, namely at the left or right end of an inactive rectangle. Consider what might happen

in each case. On meeting the left end of a rectangle as illustrated, for example in Figure 4.2, it is necessary first to determine which of active components it overlaps and second to update the set of active components accordingly. In Figure 4.2 the incoming rectangle intersects two active components with which it should be merged to form a replacement active component. However note that this can also cause a merger with a "dead" component. In Figure 4.3 a possible interpretation is given. Observe that a "dead" component should be merged with the new active component and this in turn causes another merger with an active and dead component, and so on. Finally all of the components bar the bottommost dead one end up as a single new active component.

Let us now consider what happens on meeting the right end of a rectangle. In this case the component to which it belongs dies if it is its last active rectangle. Otherwise it has no effect whatsoever.

Examining the above comments we are led to search for a data structure which can support the following operations:

1) INSERT/DELETE active and dead components.
2) QUERY active components to determine if any overlaps a given interval (the left end of a new rectangle, for example).
3) QUERY dead components to determine if any overlaps a new active component.

We also, of course, need to keep for each component its coordinates and the rectangles it contains, but first is trivial and the second can be carried out using a UNION-FIND

data structure. So let us turn to each of the above operations.

First, observe that the components do not introduce any new x- or y-values, when considering their x- and y-projections. This means that a component's y-projection, for example, is made up of "fragments" determined by the original rectangles. Thus a semi-dynamic data structure may be used. Second, observe that active components can be represented by disjoint y-intervals. Third, observe that it is necessary to keep more than the y-interval for dead components, otherwise the querying of dead components cannot be accomplished correctly.

These observations lead us to represent components by their y-intervals, while also keeping the rightmost x-value for dead components. We will store the "dead intervals" in a segment tree and a range tree. The segment tree is a structure invented by Bentley [2], while first appeared in [4] and subsequently in many places, including [7]. For completeness we will briefly describe it. Assume we have points $y_1 < y_2 < \ldots < y_{10}$ on the y-axis, which we renumber for convenience as $1,\ldots,10$. We represent this point set as a leaf search tree, see Figure 4.4. However we consider each leaf i to represent a closed-open interval $[i,i+1)$, $1 \leq i < 10$, and leaf 10 to represent $[10,\infty)$. Similarly each internal node u represent the interval defined by the leaves of its subtree, denoted by int(u). In particular the root represents $[1,\infty)$.

Furthermore each node of the segment tree has an associated node list, which is initially empty. The node list is organized as a two-way list. Let I be an interval, whose

endpoints are contained in $\{y_1,\ldots,y_{10}\}$. We insert I into the segment tree as follows:

Let u be the root of the tree initially.

If int(u) $\subseteq$ I then add I to the nodelist of u and return.

Otherwise: if int(left(u)) $\cap$ I $\neq \emptyset$ then repeat the insertion for the left subtree of u,

if int(right(u)) $\cap$ I $\neq \emptyset$ then repeat the insertion for the right subtree of u.

Inserting A = [3,6), B = [1,5), C = [3,9) into the tree of Figure 4.4 we obtain the node lists seen there.

Readers should convince themselves that insertion is indeed an O(height(tree)) time algorithm, and hence O(logn) if it is a balanced tree. Moreover since each interval can appear in O(logn) node lists, n intervals require O(nlogn) space. Deletion can also be carried out in O(logn) time, if an additional dictionary giving for each interval its position in the node-lists is provided (this is necessary because a node list can be of O(n) size). A query q is simply an x- and y-value. The x-value corresponds to the leftmost x-value of the newly created active component, while y is its bottommost y-value. The query returns all intervals, that is dead components, "stabbed" by y, whose rightmost end is between x and the scan line.

To achieve this, a standard search is made in the segment tree with y. At each node u, visited by y we report only those components satisfying the x-condition. In order for this to be carried out efficiently, the node list is organized in descending x-value order, so that only the initial portion of each node list need be examined. Finally, this ordering is

achieved naturally when a component dies because it is inserted into the segment tree at that time.

The segment tree only deals with half of the problem, namely it is used to determine the dead components which overlap the bottom of the query component. Thus as in [12], we use a range tree to complete the picture, that is to determine the bottommost values of dead components which stab the query component. Basically the tree structures are very similar. The range tree is also organized as a leaf search tree, but only on bottommost y-values of dead components. An insertion is a y-value (to which there must be a corresponding leaf). At each node u on the search path the corresponding dead component is added to u's node list. Again observe that the latest dead component, that is the one with the largest rightmost x-value appears first in all its nodelists. Just as insertion in range tree is analogous to a query in the segment tree, a query in the range tree is analogous to an insertion in the segment tree. In other words a query consists of a y-interval together with an x-value, and this defines a forked search path in the range tree. Again only the initial components in each accepted node are taken. See [12] for further details.

Remark: The segment tree constructed as described above solves the following problem:

Given a set of non-overlapping line segments in the plane, describe a data structure, which answers y-queries efficiently. A y-query is a y-value, y, and an x-value, x; it requires all line segments stabbed by the y whose x-values are greater than x.

This problem is similar to that solved in [7], when performing "dripping".

But if the x-axis is interpreted as time, then it is also an example of searching in the past, see [5,9,10].

Having disposed of the structure for dead components we turn our attention to the structure needed for active components. We could use the segment tree and range tree structures for these components as well, however since they correspond to disjoint y-intervals we use the simpler structure of Section 3, called AW. This structure can be maintained with O(logn) insertion and deletion time and O(logn + k) query time, where k is the number of intervals responding to a query, see Section 3 for details.

To complete the picture we will give a more detailed outline of the R-closure algorithm.

R-closure algorithm

Input: A set of n iso-rectangles ($n \geq 1$) each given by a quadruple (x_ℓ, x_r, y_b, y_t).

Output: The R-closed components defined by the rectangles.

Step1: Sort the x_ℓ- and x_r-values into ascending order (with ties broken by the left-before-right rule). The sorted x-values $x_1, \ldots, x_{2n}$ form the scan list. Similarly, sort the y-values. Form the "skeletal" balanced segment and range tree structures for the dead components (these have empty node lists and dictionaries). Let this total structure be denoted by DC. Let NC denote the search tree structure for

active components or waves. This is initially empty. Let i=1.

Step2: If i = 2n+1 then finish, otherwise consider x_i. It is either the left of right end of a rectangle R, say. In the former case it determines a new component, and in the latter an active component.

2.1. R determines a new component C = R and a new wave W.

(a) Determine all waves in NC which overlap W.
If none overlaps, then insert W into NC, increase i by one and goto Step2.
Otherwise determine the new merged component C' and the new wave W'.
Delete the merged waves from NC.

(b) Determine all waves in DC which overlap C'.
If none overlaps, then insert W' into NC, increase i by one and goto Step2.
Otherwise determine the new merged component C and the new wave W.
Delete the merged components from DC.
Goto 2.1(a).

2.2. R determines an active component C

If R is the only active rectangle in C then delete C from NC and insert C into DC.
In either case change the status of R to dead, increase i by one and goto Step2.

End of R-closure algorithm.

Note that Step 2.2 implies that we are keeping a UNION-FIND structure to enable the rapid determination of the active component to which R belongs. This also means that in Step 2.1,

the creation of a new component always requires the UNION operation. The same structure can, with minor modifications, also hold the status information for the rectangles together with a count of those which are currently active.

Finally let us examine the time and space requirements of this algorithm. By earlier remarks we need O(nlogn) space for the DC structure in the worst case. The NC structure on the other hand only requires O(n) space. As far as timing is concerned Step1 needs O(nlogn) time, since sorting takes place, while the building of the initial DC and NC structures is an O(n) time operation. The querying of NC and DC we have already seen take O(logn + k) time, where k is the number of answers. Thus Step 2.1 (a) or 2.1 (b) can have O(n) merges to carry out at worst. However as for diagonal closure there can never be more than n-1 merges of n objects. Hence we obtain O(nlogn) time overall. The various deletions, each of which take O(logn) time are also charged to global rather than local expences. Thus Step 2.1 takes O(nlogn) time, overall. Finally Step 2.2 clearly takes O(logn) time, thus yielding final worst-case time and space bounds of O(nlogn).

To summarize:

Theorem 4.1

Given a set of n iso-rectangles, their rectangular closure can be computed in O(nlogn) time and space.

Remark: It is to be expected that the space bound can be reduced to O(n), while retaining the O(nlogn) time bound. This time and space optimal algorithm we leave for some reader to discover.

5. Applications of Diagonal Closure

In this section we briefly discuss how the results of Section 3 solve problems posed by Lipski and Papadimitriou [7].

In [14] the following results are proved, where T is a locked transaction system and B(T) is the union of its corresponding rectangles.

Proposition 5.1

T is safe if and only if NESW(B(T)) is a connected region.

Proposition 5.2

T is deadlock-free if and only if all lower horizontal and all left vertical segments of the contour of NESW(B(T)) also belong to the contour of B(T).

We refer to the two papers mentioned above for further details of locked transaction systems and the geometric interpretation of them.

By the results of Section 3 we immediately obtain the optimal algorithm for safeness, since it is trivial to inspect the UNION-FIND structure to determine whether or not it contains only one component.

Theorem 5.3

Let $d \geq 2$ be a fixed integer and let T be a transaction system consisting of d transactions. Then the safeness of T can be computed in $O(n \log n)$ time and $O(n)$ space, where n is the maximum size of the d transactions. Furthermore this is optimal in both time and space.

Thus this solves the first open problem of Lipski and Papadimitriou [7], their algorithm runs in O(nlogn loglogn) time and O(nlogn) space.

Using the technique of Section 3 to compute the diagonally-closed components, we can in further two passes compute the left vertical and lower horizontal segments of the NESW-contour. Essentially we carry out a left-to-right and a bottom-to-top scan. As they are computed it is a simple matter to check whether or not they belong to the contour of the set of rectangles. Hence we obtain:

Theorem 5.4

Let T be a transaction system consisting of two transactions. Then the deadlock-freedom of T can be computed in O(nlogn) time and O(n) space, which is time and space optimal, where n is the maximum size of the transactions.

References

1. A.V.Aho, J.E.Hopcroft, and J.D.Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, Mass., 1974.
2. J.L.Bentley, Algorithms for Klee's rectangle problems, Unpublished notes, Carnegie-Mellon University, 1977.
3. J.L.Bentley and Th.Ottman, Algorithms for reporting and counting geometric intersections, IEEE Transactions on Computers, C-28 (1979), 643-647.
4. J.L.Bentley and D.Wood, An optimal worst-case algorithm for reporting intersections of rectangles, IEEE Transactions on Computers, C-29 (1980), 571-577.
5. D.P.Dobkin and J.I.Munro, Efficient uses of the past, in "Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, 1980," pp. 200-206.
6. H.Edelsbrunner, J. van Leeuwen, Th.Ottman, and D.Wood, "Computing the Connected Components of Orthogonal Geometric Objects," Technical Report 81-CS-04 , Unit for Computer Science, McMaster University, Hamilton, Canada, 1981.
7. W.Lipski and C.H.Papadimitriou, "A Fast Algorithm for Testing for Safety and Detecting Deadlocks in Locked Transaction Systems," Report MIT/LCS/TM-181, Laboratory for Computer Science, MIT, Mass., 1980.
8. W.Lipski and F.P.Preparata, Finding the contour of a union of iso-oriented rectangles, Journal of Algorithms 1 (1980), 235-246.
9. M.Overmars, "Searching in the Past I," Computer Science Technical Report RUU-CS-81-0, University of Utrecht, The Netherlands, 1981.
10. M.Overmars, "Searching in the Past II: General Transforms," Computer Science Technical Report RUU-CS-81-1, University of Utrecht, The Netherlands, 1981.
11. M.I.Shamos and D.J.Hoey, Geometric intersection problems, in "Proceedings of the 17th Annual IEEE Symposium on Foundations of Computer Science, 1976," pp. 241-247.
12. H.-W.Six and D.Wood, The rectangle intersection problem revisited, BIT 20 (1980), 426-433.

14. M.Yannakakis, C.H.Papadimitriou and H.T.Kung, Locking policies: safety and freedom from deadlock, in "Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science, 1979," pp. 286-297.

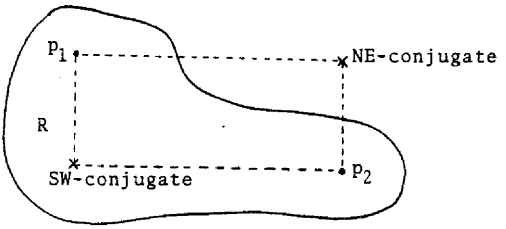

Figure 2.1 The conjugates of incomparable points.

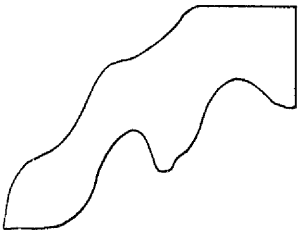

Figure 2.2 A NE-closed region.

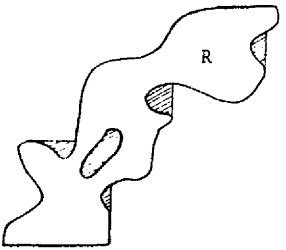

Figure 2.3 A non-NE-closed region.

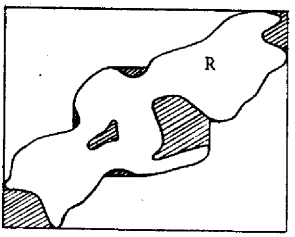

Figure 2.4 NESW-closure of a region R and its bounding rectangle.

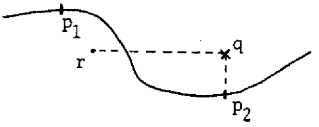

Figure 2.5

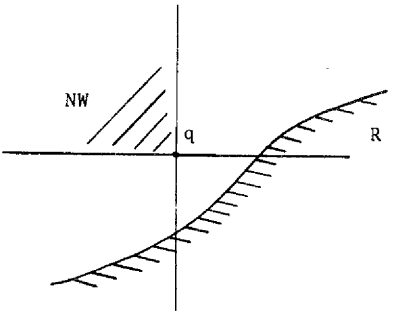

Figure 2.6

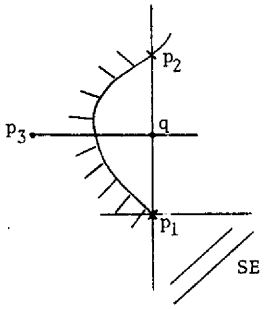

Figure 2.7

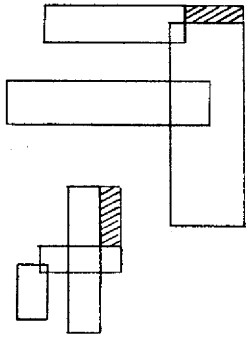

Figure 2.8

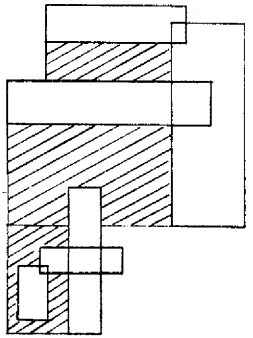

Figure 2.9

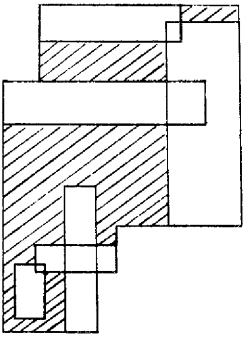

Figure 2.10

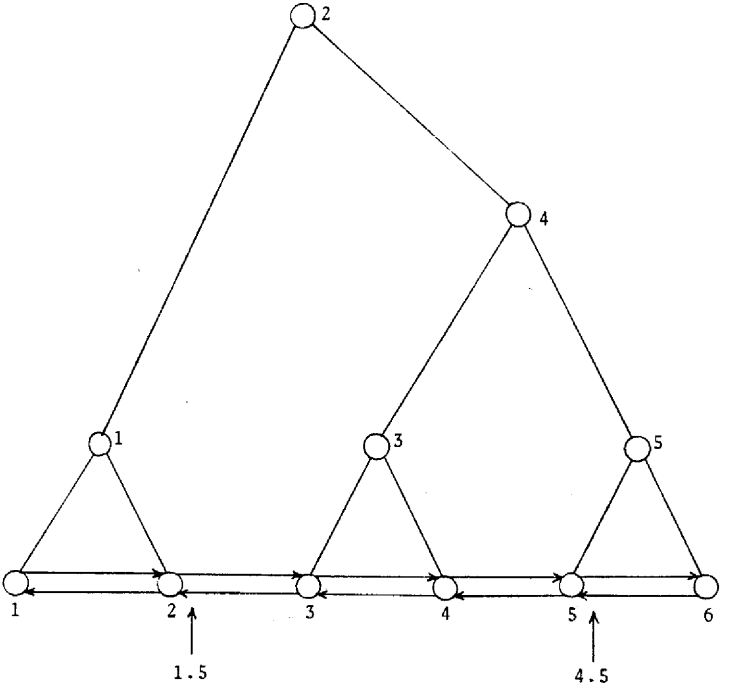

Figure 3.1

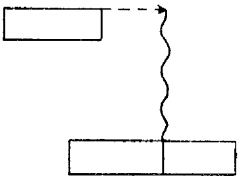

Figure 3.2 A NE-wave.

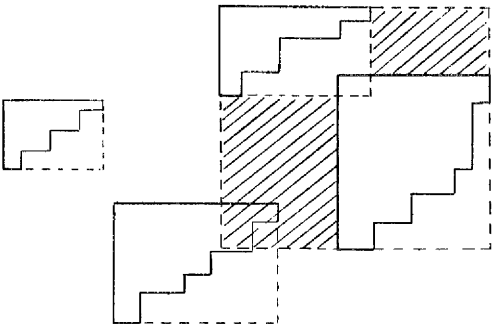

Figure 4.1 Computing SE-closure of NENWSW-closed region.

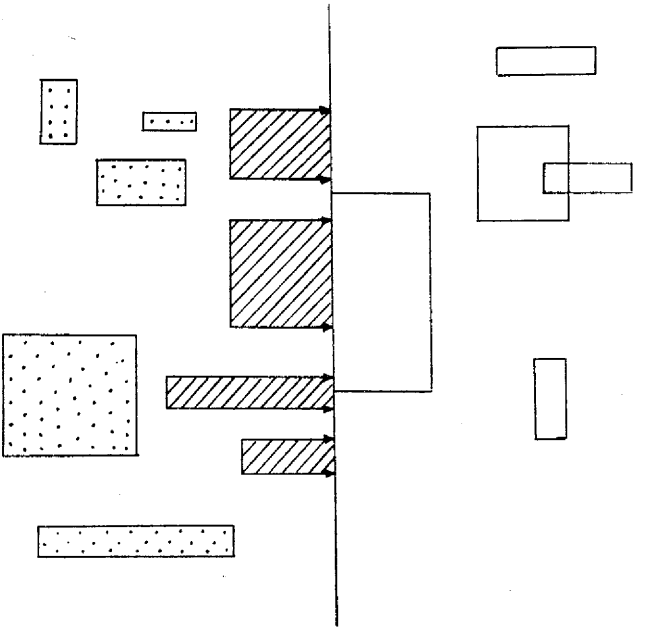

Figure 4.2 Generic position of the scan line for R-closure.

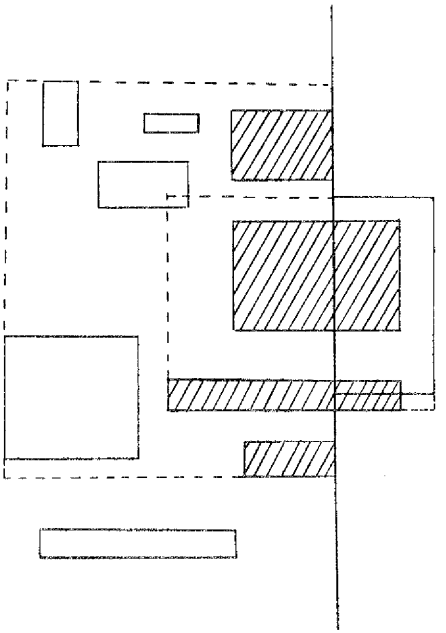

Figure 4.3

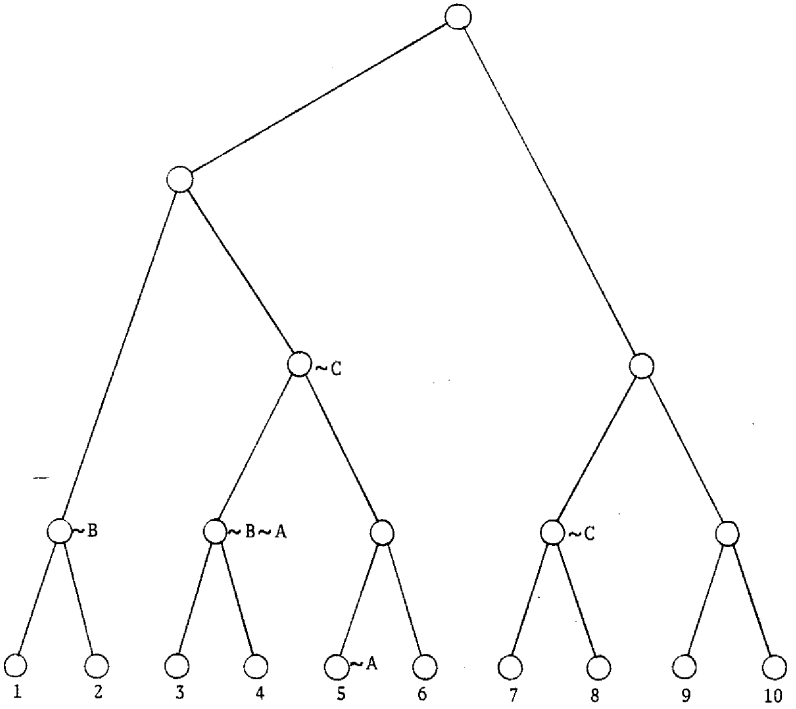

Figure 4.4