COMPUTING THE CONNECTED COMPONENTS OF D-RANGES

by

Bernard Chazelle and Janet Incerpi

Department of Computer Science

Brown University, USA

This short note reports on a new algorithm for computing the connected components of d-ranges (details can be found in [2]). Our method requires $O(n \log^{d-1} n)$ time and $O(n)$ space, and thus improves on the best algorithm known by a factor of $O(\log n)$ time and $O(\log^{d-1} n)$ space for $d > 2$ [3]. We assume the reader's familiarity with the work in [3,4]. We use the concept of ***streaming***, first introduced in [4], (and discovered independently in [2]), as the basic tool for reducing the dimensionality of the problem and at the same time requiring only linear storage. Unlike in previous algorithms, however, we reduce the problem to a one-dimensional subproblem, thus granting the overall method greater simplicity.

For the sake of illustration, assume that $d = 2$. Let S be a database of n rectangles parallel to the axes. We organize the X-projection of S into a segment tree T, thus inducing a vertical ***slicing*** of each rectangle into $O(\log n)$ canonical sub-rectangles. Let $L(v)$ denote the node-list at node v. Recall that $L(v)$ contains each rectangle whose X-projection *covers* the canonical interval associated with v, yet does not cover the interval associated with v's father. Applying the idea of ***streaming***, we set up the connected components of S *level-by-level*. In a top-down fashion (bottom-up would work, too) we consider each level in the segment tree in turn, applying the computation to each node v on the level. What this computation involves is now described.

Assuming that the components are maintained with a standard ***union-find*** scheme allowing us to perform $O(n \log n)$ finds and $n - 1$ unions in $O(n \log n)$ time [1], we must investigate the interaction of rectangles at node v to see if any components should be merged. We distinguish between two cases: 1) taking into account intersections between rectangles in $L(v)$, and 2) recognizing the interaction between rectangles in $L(v)$ (blue) and other rectangles (red) overlapping with the vertical slab associated with v. Note that these red rectangles have canonical parts in $L(w)$, where w is a descendant of v, therefore retrieving them is an easy matter.

We handle both cases uniformly by solving the following problem: Let $B = \{[a_1, b_1], \ldots, [a_p, b_p]\}$ and $R = \{[c_1, d_1], \ldots, [c_q, d_q]\}$ be two sets of p blue intervals and q red intervals, respectively; determine the connected components of the bipartite graph, $G(B, R)$, whose nodes are the intervals

in $B \bigcup R$ and whose edges connect intersecting pairs blue-red. Solving this problem for $G(B, B)$ and $G(B, R)$ successively will produce the desired effect at node v.

It is easy to find an $O(p+q)$ algorithm for solving this problem, and we only outline the method. With $O(n \log n)$ preprocessing, we can assume that the two lists $a_1, \ldots, a_p$ and $c_1, \ldots, c_q$ appear in non-decreasing order. The key observation is the following: the intervals $[a_i, b_i]$ and $[c_j, d_j]$ intersect if and only if either 1) $a_i \leq c_j \leq b_i$ or 2) $c_j \leq a_i \leq d_j$. We may then proceed in two passes, one involving B and the points c_j, and the other R and the points a_i. Because of the symmetry of these cases, we can restrict our attention to the first one. We could report all occurrences of condition 1 in one pass through R and B, performing a *union* at each occurrence. To do so, we would scan the list $a_1, \ldots, a_p$, keeping track of the last c_{j_i} smaller than the a_i currently visited. Every time i was incremented, we would update the value of j_i and report all values $c_{j_i+1}, c_{j_i+2}, \ldots$ that satisfy condition 1. This scheme would be wasteful, however, since it would essentially involve computing all intersections. Instead, we will keep track of the largest value c_{j_i+k} achieved when processing $[a_i, b_i]$. Let *skip* be this value at time t and let $[a_m, b_m]$ be the interval under consideration when the value *skip* was achieved. Assume that when dealing with $[a_i, b_i]$ at time t, the procedure detects an intersection between $[a_i, b_i]$ and $[c_j, d_j]$ for $j < skip$. All the points $c_j, c_{j+1}, \ldots, c_{skip-1}$ were previously visited, therefore all the corresponding intervals are already in the same component as $[a_m, b_m]$. This shows that if at time t any *union* is to be taken between $[a_i, b_i]$ and $[c_u, d_u]$, for $j \leq u < skip$, it consists of putting $[a_i, b_i]$ in the same component as $[a_m, b_m]$. It is then clear that examining the first value of u should be sufficient to do so. We can then avoid all the others and jump directly to the current value of *skip*. This leads directly to an $O(p+q)$ scheme for each pass.

Lemma: It is possible to compute the connected components of the bipartite graph $G(B, R)$ in $O(p+q)$ time and space, using $O((p+q)\log(p+q))$ preprocessing.

Proof: Omitted. ∎

Streaming the computation will thus allow us to apply the previous procedure to each node of T, so as to deal with the two types of intersection mentioned earlier; furthermore this scheme will necessitate only $O(n)$ storage. Generalizing the algorithm to higher dimensions is straightforward, and we omit the details. Note that the algorithm is optimal in two dimensions, since we can easily reduce *element uniqueness* to it. We conclude.

Theorem: It is possible to compute the connected components of n d-ranges in $O(n \log^{d-1} n)$ time and $O(n)$ space.

REFERENCES

[1] Aho, A.V., Hopcroft, J.E. and Ullman, J.D. *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[2] Chazelle, B., Incerpi, J. *Unraveling the segment tree*, Tech. Rep. No. CS-83-15, Brown Univ., June 1983.

[3] Edelsbrunner, H., van Leeuwen, J., Ottmann, T., Wood, D. *Computing connected components of orthogonal geometric objects*, RAIRO (1983), to appear.

[4] Edelsbrunner, H.,Overmars, M.H. *Batched dynamic solutions to decomposable problems*, Tech. Rep., RUU-CS-83-8, Univ. Utrecht, March 1983.

A Note on DTOL Systems

by

Jürgen Dassow
Technological University Magdeburg
Department of Mathematics and Physics
DDR - 3014 Magdeburg
German Democratic Republic

The ambiguity of OL systems is considered in /ReS/, and it is proved that any DOL system is unambiguous and that the ambiguity of a OL system is undecidable. In this note we consider the ambiguity of DTOL systems.

We assume that the reader is familiar with the basic concepts of formal language theory, especially of the theory of L systems and the Post correspondence problem (see /RoS/).

Definition: Let $G = (V, \{h_1, h_2, \ldots, h_n\}, w)$ be a DTOL system. We say that G is unambiguous iff

$$h_{i_1} h_{i_2} \ldots h_{i_r}(w) = h_{j_1} h_{j_2} \ldots h_{j_s}(w)$$

implies

$$i_1 i_2 \ldots i_r = j_1 j_2 \ldots j_s .$$