A Rudimentary Expansive-Insertion Algorithm for Embeddable Containment Trees

Higraph Editor Project*
Konstantinos Tourlas, Daniele Turi
Division of Informatics, University of Edinburgh
{kxt,dt}@inf.ed.ac.uk

January 14, 2003

1 Preliminaries

1.1 Rectangles

Here a *rectangle* is considered a geometric entity: a simple closed curve on the plane with the requisite properties.

Given any rectangle r write $loc(r)$ for its location, defined as the coordinate pair $\langle x, y\rangle$ of its top left-hand corner, also denoted as $\langle loc(r) \cdot x, loc(r) \cdot y\rangle$. Write $width(r)$ for the width of rectangle r, and $height(r)$ for its height. Further define $maxX(r)$ as $loc(r) \cdot x + width(r)$, and similarly for $maxY(r)$.

Given rectangles r, r' write $r < r'$ to mean that the interior $I(r)$ of r is strictly contained in that of $I(r')$: $I(r) \subset I(r')$.

$r \leq r'$ is defined as $r < r'$ or $r = r'$.

The *empty rectangle* is by convention taken to be the rectangle with location $\langle 0, 0\rangle$, zero width, and zero height.

*Research carried out under UK-EPSRC grant GR/N12480/01 (pricipal investigator: Stuart Anderson)

1.2 Nodes

Assume a set N of nodes, ranged over by n, n' and so on.

We shall need to consider functions which assign boundaries to some nodes in N. In general, it should suffice to regard these boundaries as simple closed curves on the plane, yet here we make the simplifying assumption that each assigned boundary is a rectangle.

We shall often write $loc(n)$ as a shorthand for $loc(B(n))$ when B is understood from the context; similar conventions apply to $width(n)$ and $height(n)$. Also, when both B and B' are understood, we shall write $loc(n)$ and $loc'(n)$ for $loc(B(n))$ and $loc(B'(n))$ respectively, and similarly for widths and heights.

1.3 Trees

Given a tree t we write $n \rightarrow_t n'$ to mean that n' is a child of n in t, omitting the subscript t when clear from the context. Given tree t and node $n_0 \in nodes(t)$ write $Sib_t(n_0)$ for the set of siblings of n_0 in t. In particular, $n_0 \notin Sib_t(n_0)$.

2 Embeddable Containment Trees

An embeddable containment tree τ is a pair $\langle t, B \rangle$ where

- t is a tree with set of nodes in N
- B is a function assigning a rectangle to each $n \in nodes(t)$.

These data are subject to the following conditions:

- whenever $n \rightarrow n'$, then $B(n') < B(n)$ and, moreover, no other node n'' exists in t such that $B(n') < B(n'') < B(n)$
- whenever n and n' are distinct nodes in t, then the intersection of $B(n)$ and $B(n')$ is empty.

We assume certain evident (and partial) functions, such as $root(t)$. Given embeddable tree $\tau = \langle t, B \rangle$ and a subset A of $nodes(t)$, we shall often write $B(A)$ to mean

- the empty rectangle if A is empty

- the smallest rectangle r such that $B(n) \leq r$ for all $n \in A$, otherwise.

$LSib_\tau(n_0)$ is the set of the "left siblings" of n_0 in the embeddable containment tree $\tau = \langle t, B \rangle$, namely

$$\{n \in nodes(t) \mid loc(B(n)) \cdot x < loc(B(n_0)) \cdot x\}$$

whereas $RSib_\tau(n_0)$ is

$$\{n \in nodes(t) \mid loc(B(n)) \cdot x \geq loc(B(n_0)) \cdot x\} \ .$$

Given $\tau = \langle t, B \rangle$ and $no \in nodes(t)$ define $Containers_\tau(n_0)$ to be the set

$$\{n \in nodes(t) \mid n \text{ is an ancestor of } n_0\} \ .$$

Also, we write $subtree_\tau(n)$ for the (embeddable containment) subtree of τ rooted at n.

3 The Expansive Insertion Algorithm

The algorithm creates and inserts a new node with given bounding rectangle *bounds* into an embeddable containment tree $\langle t, B \rangle$. If necessary, the new node and any node already in the tree that is located to the right of the new node are shifted further to the right, and any node which is to contain the new node is expanded. A margin integer value $M > 0$ is used to provide extra spacing between nodes which are shifted or expanded and the boundaries of their container nodes.

The algorithm invokes the following two operations, specified in the following sections:

- An operation for shift-inserting a new node into an embeddable containment tree by relocating to the right, if necessary, the new node as well as any nodes already in the tree which are to become siblings of the new node. Such relocation may be necessary so as to avoid intersections among the bounds of nodes.

- An operation to expand a given node, and all of its containers, by given amounts along its width and height. Such expansions may become necessary as a result of performing the shifting operation above.

The algorithm proceeds as follows:

1. locate a suitable container for the new node;
2. the bounding rectangle of the new node is relocated (if necessary) and the new node is inserted using shift-insertion;
3. the bounds of the containers of the new node are expanded, if necessary.

Or in pseudo-code:

```
expansiveInsert(n: Node, bounds: Rectangle, margin: int) {
  container = container of the location of bounds
  shift-insert(container, node, margin);
  W = amount by which container must be expanded along its width,
      including margin
  H = amount by which container must be expanded along its height,
      including margin
  expandNode(container, W, H);
}
```

4 The Node Expansion Operation

This section is concerned with the specification of a simple operation which expands a given node in an embeddable containment tree by W units along its width and by H units along its height.

4.1 Input

- An embeddable containment tree $\tau = \langle t, B \rangle$;
- a node $n_0 \in \mathit{nodes}(t)$;
- two natural numbers W and H.

4.2 Output

An embeddable tree $\tau' = \langle t, B' \rangle$ with the same underlying tree as the input, but with a new "bounds" function B' determined as follows:

1. for all $n \in nodes(t)$ such that $B(n) < B(n_0)$,

$$B'(n) = B(n)$$

2. for all $n \in Containers_\tau(n_0)$

   - $loc'(n) = loc(n)$
   - $width'(n) = width(n) + W$
   - $height'(n) = height(n) + H$

3. for all nodes n such that $n \in Sib_t(n')$ for some $n' \in Containers_{\tau'}(n_0)$,

$$subtree_{\tau'}(n) = transl(subtree_\tau(n), x_n, y_n)$$

   where

$$x_n = \begin{cases} W, & \text{if } loc(n) \cdot x > loc(n') \cdot x + width(n') \\ 0, & \text{otherwise} \end{cases}$$

   and

$$y_n = \begin{cases} H, & \text{if } loc(n) \cdot x > loc(n') \cdot x + height(n') \\ 0, & \text{otherwise} \end{cases}$$

   (Notice that, for each n in this clause, the required n' is unique when it exists.)

5 The Shift-insertion Operations

This operation inserts a new node under a given parent node in an embeddable containment tree. This is done by shifting the insertion point (ie the supplied location of the new node) to the right, if necessary, to avoid intersecting any of the parent's children lying to the left of the insertion point. Correspondingly, those children of the parent lying to the right of the new insertion point are also shifted further to the right.

5.1 Input

- An embeddable containment tree $\tau = \langle t, B \rangle$
- a node $n_0 \in nodes(t)$

- a node $n \notin nodes(t)$ with associated bounding rectangle b such that $loc(b) \cdot x > loc(B(n_0)) \cdot x$ and $loc(b) \cdot y > loc(B(n_0)) \cdot y$.
- a margin value $M > 0$ as a natural number.

Write L for the set of children of n_0 in t lying to the left of b

$$L = \{n \in nodes(t) \mid n_0 \rightarrow n \text{ and } loc(n) \cdot x < loc(b) \cdot x\} \ ,$$

and $B(L)$ for the collective bounds of all nodes in L.

5.2 Output

If either $loc(b) \cdot x > loc(B(n_0)) \cdot x$ or $loc(b) \cdot y > loc(B(n_0)) \cdot y$ do not hold of the input, the operation is undefined. Otherwise, the operation returns

- a tree t'
- a function B' from $nodes(t')$ to rectangles; and
- a pair $\langle W, H \rangle$ of natural numbers representing the amounts by which $B'(n_0)$ must be extended along its width and height to yield a rectangle b_0 such that b_0 contains all the children n' of n_0 in t' (i.e. $B'(n') \leq b_0$ for all children n') plus the specified margin M.

More specifically,

- t' is simply t augmented with $n_0 \rightarrow n$
- B' is determined as follows:
  1. $B'(n) = transl(B(n), loc(B(L)) \cdot x + width(B(L)) + M, 0)$
  2. for all children n' of n_0 which are not in L,

     $B'(n') = transl(B(n'), loc(B'(n)) \cdot x - loc(b) \cdot x + width(b) + M, 0)$
  3. for all other nodes n' in t', $B'(n') = B(n')$.
- $W = \max(0, maxX(B'(R)) + M - maxX(B(n_0)))$, where

  $$R = \{n' \in nodes(t') \mid n_0 \rightarrow n' \text{ and } loc(n') \cdot x \geq loc(B'(n)) \cdot x\}$$

  (i.e. R is the set of children of n_0, including the new node n, which lie to the right of the new node n)
- $H = \max(0, maxY(B'(n)) + M - maxY(B(n_0)))$.