Pseudo-code for the Ordering Step

June 27, 2013

We sort the entity lines vertically at each time frame to minimize the number of line crossings while satisfying the hierarchy constraints. Our sorting algorithm has two parts: (1) sorting the location nodes; (2) ordering the session and entity nodes within each location.

We assume that the locations are hierarchically organized as a location tree. We use a greedy algorithm to order the location nodes from bottom to top, recursively. Algorithm 1 describes the pseudo-code, namely, $\text{OrderLocationTree}(r)$, for sorting the locations with the location tree.

**Algorithm 1:** $\text{OrderLocationTree}(r)$.

- **Data:** the root of a location tree: \( r \)
- **Result:** the location tree with all the locations ordered

1. Sort \( r\.\text{children} \) by their numbers of sessions;
   
2. Create an empty list: \( \text{bestPosition} \);
   
3. foreach child in root\.children do
   
4. \( \text{OrderLocationTree}(\text{child}); \)
   
5. Insert \( \text{child} \) to the best place in \( \text{bestPosition} \) to minimize the number of crossings between locations; // See Fig.4(c) in the paper.

6. end
7. \( r\.\text{children} = \text{bestPosition}; \)

Next, we order the sessions and entities under each location node. The problem is similar to minimizing edge crossings in a multi-level directed acyclic graph (DAG) with a fixed ordering of some of the non-leaf levels that correspond to the locations. We use a method that starts by generating an initial ordering of the first time frame, which satisfies the hierarchy constraints. It then treats the ordering of the first frame as a reference to calculate the ordering of the second one. This step is repeated until the last frame is reached. Once the order of the last step is fixed, we then treat it as the reference and sweep back. Algorithm 2 describes the pseudo-code, namely, $\text{OrderRelationshipTreeSet}(r_1, r_2, \ldots, r_n)$, for ordering the sessions and entities, which are organized as a sequence of relationship trees.
Algorithm 2: OrderRelationshipTreeSet($r_1, r_2, \ldots, r_{nt}$).

**Data:** the roots of relationship trees for all time frames: $r_1, r_2, \ldots, r_{nt}$

**Result:** the ordered relationship trees

1. Initialize the order of the first frame;
2. for $i = 1$ to $MaxSweepTimes$ do
   /* $MaxSweepTimes$ is the maximum iteration number */
   3. for $j = 2$ to $nt$ do
      4. OrderRelationshipTree($r_{j-1}, r_j$);
      /* Sweep forward and invoke Algorithm 3 */
   5. end
   6. for $j = nt - 1$ to 1 do
      7. OrderRelationshipTree($r_{j}, r_{j-1}$);
      /* Sweep back and invoke Algorithm 3 */
   8. end
9. end

Algorithm 3: OrderRelationshipTree($r_p, r_c$).

**Data:** $r_p, r_c$: the roots of the relationship trees at the previous time frame and the current time frame, respectively

**Result:** the ordered relationship tree of $r_c$ at the current time frame

1. if $r_c$ has children and not all children are location nodes then
   2. foreach child in $r_c$.children do
      3. OrderRelationshipTree(child);
   4. Compute the barycenter of child based on $r_p$;
5. end
   6. Sort the children of $r_c$ by their barycenter values;