Pseudo-Code and Computational Complexity of Water-Filling Algorithm for Independent Innovations Trees


structure node{
    var; /* variance of node */
    M; /* N_γ, where γ is this node */
    N; /* |L_γ| where L is the current leaf node set */
    curr_u; /* μ_γ(N) */
    next_u; /* μ_γ(N + 1) */
    curr_v; /* μ_{γ+1,N} where γ is this node */
    next_v; /* μ_{γ+1,N} */
    num_child; /* P_γ */
    child; /* pointer to location of first child, γ1 */
    cov_fac; /* ξ_{γ+1,k} where γ ↑ k = γ */
    parent; /* pointer to parent, γ ↑ */
    const; /* ϱ_γ */
    var_W; /* var(W_γ) */
    max_v_loc; /* child number with highest (next_v-curr_v) */
}

*node create_nodes()
/*create an array of ‘node’ structures which sets the parent, child, num_child, var_W, and const fields for all nodes. Initialize the root node variance. Return a pointer to the root location.*/
}

init(node *node){
    node *parent, *child; /*pointers to parent and child of node*/
    parent=node→parent; /*set parent to parent of current node*/
    child=node→child; /*set child to first child of current node*/
    node→N=0;
    if (node ≠ root){ /* if the node is not the root */
        node→var=((node→const)² * parent→var) +node→var_W ;
        node→cov_fac=node→var/parent→var)*[(node→const)² + node→var_W/parent→var]
        / (node→const)² ; /* set node variance */
        node→curr_v=1/patient→var;
    } /* end if (node≠ root) */
node->curr_u=1/node->var;

if (node->child != 0) { /* if not a leaf node */
  for k=1 to node->num_child {
    init(child+k-1); /* initialize all children */
    node->M+=(child+k-1)->M;
  } /* end for loop */

  max_v_loc=find_max_v(node); /* find child with maximum (next_v-curr_v) */
  node->max_v_loc=max_v_loc;
  node->next_u=node->curr_u+(child+max_v_loc-1)->next_v-(child+max_v_loc-1)->curr_v;
  if (node!=root){
    node->next_v=1/(parent->var - ((1/node->cov_fac)*(node->var-(1/node->next_u))));
    /* end if (node != root) */
  }
}
/* end if (node->child != 0)*/
else { /* else if a leaf node */
  node->M=1;
  node->next_v=1/(parent->var - ((node->const*parent->var)^2/node->var));
}
return ;
}

int find_max_v(node *node){ /* find child with maximum (next_v - curr_v) */
  int i;
  double max=0.0;
  int max_v_loc; /* child number */
  node *child; /* pointer to child */
  child=node->child;

  for i=1 to node->num_child {
    if (child->next_v - child->curr_v->max){
      max=child->next_v - child->curr_v;
      max_v_loc=i;
      child++; /* move to next child node */
    } /* end if */
  } /* end for loop */
}

/* returns pointer to the next optimal leaf node */
find_next_leaf(node *node){
  node *parent, *child; /* pointers to parent and child */
  parent=node->parent;
  child=node->child;
  node->N++;

  if (node->num_child==0){ /* if a leaf node */
    best_child=node; /* return this leaf node */
  }
  else {
    best_child=find_next_leaf(child+max_v_loc-1);
    node->curr_u=node->next_u;
    node->curr_v=node->next_v;
  }
}
if (node→N < node→M) { /* if nodes remaining in sub-tree*/
    max_v_loc=find_max_v(node); /*find child with maximum (next_v-curr_v) */
    node→max_v_loc=max_v_loc;
    node→next_u=node→curr_u+(child+max_v_loc-1)→next_v-(child+max_v_loc-1)→curr_v;
    if (node≠root){
        node→next_v=1/(parent→var - ((1/node→cov_fac)*(node→var-(1/node→next_u))));
    } /* end if (node ≠ root) */
} /*end if (node→num_child=0) */
return(best_child);
}

leaves best_leaves(int n){ /* Determine the best n leaf nodes to estimate the root*/
    node *root;
    root=create_nodes(); /* create array of node structures */
    init(root); /* initialize the structure array starting with root */
    leaves=empty set; /* leaves contains the set of leaf indexes selected so far*/
    for i=1 to n
        leaves=leaves∪find_next_leaf(root); /* each leaf entry consists of a set of child indexes of nodes connecting the root to the leaf*/
    return(leaves);
}

**Computational Complexity:** Denote the number of nodes in the tree by $N$ and the depth of the tree by $D$. The complexity of `create_nodes()` and `init(root)` is $O(N)$ since they create and initialize $N$ structures of type `node`. The complexity of `find_next_leaf(root)` is $O(D)$ since it updates a fixed amount of information at each node along a path from the root to a leaf along of length $D$. The overall complexity of the water-filling algorithm is thus $O(N + nD)$. If every node in the tree has the same number of children then $D$ is approximately $\log(N)$ thus giving an overall complexity of $O(N + n \log(N))$. 
