Contents

1  Main program  

2  Formatting rules for HPF/F90 files  

FWEB version 1.60  

January 1, 1997
1 Main program

This program uses the lower level functions of Graph Algorithms to plot some interesting pictures of the depth and cardinality labels for nodes and arcs in a graph. It is a bit outdated from the present version of Graph Algorithms!
"WEAVE.f90" 1.1

PROGRAM Time_MST
    USE Precision     // Kind parameters
    USE Error_Handling
    USE System_Monitors
    USE Random_Numbers
    USE Initialization_Termination
    USE Network_Data_Structures
    USE Lattice_Geometry
    USE Network_Geometry
    USE Network_Graphics
    USE Lattice_Network_Optimization
    USE Graph_Algorithms
    USE Network_Spanning_Trees
    IMPLICIT NONE

    REAL (KIND = r_wp), DIMENSION (:), ALLOCATABLE :: weights_changes, arcs_weights
    INTEGER (KIND = i_wp) :: arc, node, index, i
    REAL (KIND = r_wp) :: disorder, total_weight
    LOGICAL :: color_arcs, resize_arcs, color_nodes, resize_nodes

    CALL StartProgram
    CALL InitializeLatticeNetworkProblem
    CALL CreateLatticeNetworkProblem
    ALLOCATE (arc, node, index, i)

    WRITE(*, *) "Enter the disorder:"
    READ(*, *) disorder

    nodes_mask = T
    arcs_weights = arcs_cost_parameters

    CALL ResetTimer(1)
    CALL StartTimer(1)
    CALL CreateSpanningTree(tree_type = min_cost_tree, arcs_weights = arcs_weights, 
                          weights_distribution = "Random", total_weight = total_weight)
    CALL StopTimer(1)

    WRITE(*, *) "Making the MST tree from scratch took:", ReadTimer(1)

    WRITE(*, *) "Total tree weight (min)=", total_weight

    arcs_mask = T
    DO node = n_special_nodes, n_nodes         // Plot only tree or only non-tree arcs
        arcs_mask (tree_nodes_parents_node) = T
    END DO

    arcs_mask0 = T
    color_arcs = T;
    resize_arcs = T;
    color_nodes = T;
    resize_nodes = T;

    CALL RandomUniform (weights_changes, range = (/ 1.0_wp - disorder, 1.0_wp + disorder /))
arcs_weights = weights_changes * arcs_weights  // Change the weights of the arcs
CALL ResetTimer(5)
CALL StartTimer(5)
CALL ReBuildSpanningTree(arc_offset = n_special_arcs, node_offset = n_special_nodes,
    arcs_list = (% (arc, arc = -n_special_arcs, n_arcs) %),
    heads_tails = heads_tails,
    arcs_weights = arcs_weights, orientations = tree_arcs_orientations, parents = tree_nodes_parents,
    cardinalities = tree_nodes_cardinalities, total_weight = total_weight, tree_type = min_cost_tree)
CALL StopTimer(5)
WRITE(*, *) "Total tree weight (min rebuilt)=", total_weight
CALL ResetTimer(6)
CALL StartTimer(6)
CALL BuildTreeStructures(arc_offset = n_special_arcs, node_offset = n_special_nodes,
    heads_tails = heads_tails, orientations = tree_arcs_orientations, parents = tree_nodes_parents,
    cardinalities = tree_nodes_cardinalities, level_ordering = tree_nodes_ordering,
    rebuild_cardinalities = F)
CALL StopTimer(6)
arcs_mask = F
color_arcs = F;
resize_arcs = F;
color_nodes = F;
resize_nodes = F;
(PlotSpanningTree 1.2)
WRITE(*, *) "Rebuilding the MST tree took ":, ReadTimer(5) + ReadTimer(6)
WRITE(*, *) "------------------ Arc pivoting:" , ReadTimer(5)
WRITE(*, *) "------------------ Thread building:" , ReadTimer(6)
CALL DestroyLatticeNetworkProblem
CALL EndProgram

END PROGRAM Time_MST
\begin{verbatim}
(PlotSpanningTree 1.2) \equiv

DO node = -n_special_nodes, n_nodes
   IF (tree_arcs_orientations[node] \equiv tail_is_parent) THEN
      arcs_voltages(tree_nodes_parents[node]) = 1.0 \_wp
   ELSE
      arcs_voltages(tree_nodes_parents[node]) = -1.0 \_wp
   END IF
END DO

nodes_potentials_0,1 = 0.0 \_wp

CALL PropagateNodesPotentials(arcs_offset = n_special_arcs, node_offset = n_special_nodes,
                               heads_tails = heads_tails, orientations = tree_arcs_orientations, parents = tree_nodes_parents,
                               level_ordering = tree_nodes_ordering, arcs_potential_drops = arcs_voltages,
                               nodes_potentials = nodes_potentials)

supplies_demands = 1.0 \_wp

CALL PropagateArcsFlows(arcs_offset = n_special_arcs, node_offset = n_special_nodes,
                         heads_tails = heads_tails, orientations = tree_arcs_orientations, parents = tree_nodes_parents,
                         level_ordering = tree_nodes_ordering, supplies_demands = supplies_demands,
                         tree_flows = arcs_flows, flow_imbalance = total_weight)

CALL InitNetworkGraphics(file = "Prim.ps", file_type = "PSCL", plot_title = (/ "Spanning
                              Tree" /), page_size = (/ 5000, 5000 /), label_format = "4E1", color_table = "RAIN",
                              colorbar_position = "Horizontal", axis_labels_format = (/ "NONE", "NONE", "NONE", "NONE" /))

CALL PlotNetwork2D (heads_tails = heads_tails, node_offset = n_special_nodes,
                     node_coords = node_coords, node_mask = nodes_mask,
                     arc_mask = arcs_mask, node_values = nodes_potentials, arc_values = ABS(arcs_flows),
                     resize_nodes = resize_nodes, resize_arcs = resize_arcs, color_nodes = color_nodes,
                     color_arcs = color_arcs, \& node_size_range = (/ -HUGE(1.0 \_wp) / 20, HUGE(1.0 \_wp) / 10 /), \&
                     node_colorbar_format = "5E1", arc_colorbar_format = "5E1", vector_type = 0, axis = (/ 0.0, 0.0,
                     REAL(lengths + 1) /))

CALL EndNetworkGraphics()
\end{verbatim}

This code is used in section 1.1.
2 Formatting rules for HPF/F90 files

These are just some auxiliary formatting rules and useful macros I use from time to time.

```plaintext
@m _SIZE(array, _kind,...)
   ifdef (#0, 0, INT(SIZE(array), KIND=_kind), INT(SIZE(array, #.), KIND=_kind))
@m _MAXLOC(array, _kind,...)
   ifdef (#0, 0, INT(MAXLOC(array), KIND=_kind), INT(MAXLOC(array, #.), KIND=_kind))
@m _MINLOC(array, _kind,...)
   ifdef (#0, 0, INT(MINLOC(array), KIND=_kind), INT(MINLOC(array, #.), KIND=_kind))
@m _BOUND(array, _kind,...) ifdef (#0, 0, INT(_BOUND(array, DIM=1), KIND=_kind),
   INT(_BOUND(array, #.), KIND=_kind))
@m _DEBUG(_name,...)
   ifdef (#0, 0, INT(_DEBUG(_name, #.), KIND=_name))
@m _generic_INTERFACE(_name,...)
   interface _name
   module procedure #.
   end interface _name
@m _DECLARE(_name,...)
   integer :: #.
@m _DECLARE(_name,...)
   integer (KIND = _name) :: #.
@m _DECLARE(_name,...)
   real (KIND = _name) :: #.
@m _DECLARE(_name,...)
   real (KIND = _name) :: #.
@m _DECLARE(_name,...)
   real (KIND = _name) :: #.
@m _FULL_EXTENT(_rank,...) ifdef (_rank,...)
   do (DIM, 2, _rank) { , : }
@m _VARSEQUENCE(_variable,...)
   _variable$#_start$do (DIM, $eval (_start + 1), _end) { , _variable@&DIM }
@m _NESTED_LOOP_START(_variable,...)
   $do (_DIM, _rank, 1, -1) { _variable@&DIM = _BOUND(_array, _kind, DIM = _DIM),
   _BOUND(_array, _kind, DIM = _DIM) }
@m _NESTED_LOOP_END(_rank,...) $do (_DIM, 1, _rank) { end do }
@m _DUMMY(....)
@m _DISPLAY_ARRAY(_message,...)
   if (SIZE(array) ≤ 20) then
      writeln(_message_print_unit, "(A)" _message
      writeln(_message_print_unit, "(2005.2)" _array
   end if
```