Min/Cut---Max/Flow saturation in superconductors

Aleksandar Donev

July 2001

1 Main program

This program uses SSCNO and a scheme similar to the ine in the IV program to study the onset of dissipation in superconducting materials. It invokes the F77 Min-Cut routines from Graph_Algorithms_F77.

This documentation is not finished.

"WEAVE.f90" 1.1 ≡

PROGRAM Saturation
USE Precision // Kind parameters
USE Error_Handling
USE System_Monitors
USE Initialization_Termination
USE Network_Data_Structures
USE Lattice_Geometry
USE Network_Geometry
USE Simple_Graphics
USE Network_Graphics
USE Network_Spanning_Trees
USE Graph_Algorithms
USE Graph_Algorithms_F77
USE COST_TYPE@&_Cost_Parameters
USE COST_TYPE@&_Cost_Functions
USE Lattice_Network_Optimization
USE Conjugate_Gradient
USE Dual_Network_Solvers
USE DualLine_Minimizers
USE Dual_Newton_SSCNO
USE SSCNO_Interface

IMPLICIT NONE

INTEGER (KIND = i_op), DIMENSION (:,:), ALLOCATABLE :: special_heads_tails
INTEGER (KIND = i_byte), DIMENSION (:,:), ALLOCATABLE :: nodes_cuts

1
REAL (KIND = r_wp), DIMENSION (:), ALLOCATABLE :: arcs_cuts_capacities, cuts_capacities
INTEGER (KIND = i_byte), DIMENSION (:), ALLOCATABLE :: arcs_cuts
INTEGER (KIND = i_byte) :: head_cut, tail_cut, n_cuts = 1

INTEGER :: arc, node, sample, point, instance
CHARACTER (LEN = 60) :: title, plot_title(3) = ""
CHARACTER (LEN = 50) :: file_name = "", file_type = "CONS", file_extension = "dummy"
REAL (KIND = r_wp) :: applied_volages_sum, min_arc_value, max_arc_value
REAL (KIND = r_wp), DIMENSION (2) :: current_interval = (/ 0.0_wp, 1.0_wp /)

INTEGER :: n_samples = 1, n_points = 2, n_instances = 1
INTEGER :: alloc_status, source_type
REAL (KIND = r_wp), DIMENSION (4) :: axis
REAL (KIND = r_wp), DIMENSION (:), ALLOCATABLE :: total_currents, total_volages, averaged_currents,
                                           averaged_volages
REAL (KIND = r_wp), DIMENSION (::), ALLOCATABLE :: applied_volages, injected_currents
LOGICAL :: plot_samples = T, plot_potentials = T, change_current = T, change_volage = T, save_plot

@#if 1
   WRITE(*, *) "Enter change_current and change_volage (T or F)"
   READ(*, *) change_current, change_volage
   IF (change_current) THEN
      WRITE(*, *) "Enter the interval for the total injected current:"
      READ(*, *) current_interval
   END IF
   IF (change_volage) THEN
      WRITE(*, *) "Enter the interval for the total applied volage:"
      READ(*, *) voltage_interval
   END IF
@#endif

WRITE(*, *) "Enter n_points, n_cuts"
READ(*, *) n_points, n_cuts

@#if 1
   WRITE(*, *) "Plot every sample (T or F):"
   READ(*, *) plot_samples
   IF (plot_samples) THEN
      WRITE(*, *) "Plot the nodes potentials as well?:"
      READ(*, *) plot_potentials
   END IF
   WRITE(*, *) "Enter file output type (CONS, XWIN, PNG, POST, PSCL)"
   READ(*, *) file_type
SELECT CASE (file_type)
CASE ("POST", "PSCL")
   file_extension = "ps"
CASE ("PNG")
   file_extension = "png"
CASE DEFAULT
   file_extension = "dummy"
ENDSELECT
@#endif

CALL StartProgram
ALLOCATE (total_currents, n_points, total_voltages, n_points)
ALLOCATE (averaged_currents, n_points, averaged_voltages, n_points)
ALLOCATE (applied_voltages, n_points, n_samples, injected_currents, n_points, n_samples)
CALL InitializeLatticeNetworkProblem
CALL InitializeSSCNO
CALL ResetTimer(100)
CALL StartTimer (100)
DO point = 1, n_points  // Preassign the desired set of total current flows or applied voltages
   total currents, point = current interval + REAL (point - 1) * (current_interval_2 - current_interval_1) / REAL (n_points - 1)
   total voltages, point = voltage_interval + REAL (point - 1) * (voltage_interval_2 - voltage_interval_1) / REAL (n_points - 1)
END DO
injected currents = 0.0_wp  // Initialize
applied voltages = 0.0_wp  // Zero all averages at start!
// All injection here is done along the x axis!
SourceType: DO source_type = 1, 2
   IF (source_type == 1 ∧ (~change_current)) CYCLE SourceType
   IF (source_type == 2 ∧ (~change_voltage)) CYCLE SourceType
SampleLoop: DO sample = 1, n_samples
   WRITE (message_print_unit, *) "->Doing sample #", sample
   IF (source_type == 1) THEN  // Change the current
      sources, status_1 = 'R'
      sinks, status_1 = 'R'
   ELSE IF (source_type == 2) THEN  // Change the potential
      sources, status_1 = 'F'
      sinks, status_1 = 'F'
   END IF
   CALL CreateLatticeNetworkProblem(create_tree = T)
   CALL CreateSSCNO
   _AllocateNodalArray(nodes_cuts, i_byte, -ALLOCATED, "Breakdown")
   _AllocateArcArray(arcs_cuts, i_byte, -ALLOCATED, "Breakdown")
   _AllocateArcArray(arcs_cuts_capacities, 1.0_wp, -ALLOCATED, "Breakdown")
   _AllocateArray(cuts_capacities, 1, n_cuts, 1.0_wp, -ALLOCATED, "Breakdown")
   ALLOCATE (special_heads, tails, -n_special_arcs, i, star = alloc_status)  // A temporary
   CALL RecordAllocation(n_elements = SIZE(special_heads, tails, i_wp), mold = li_wp,
      caller = "Saturation", alloc_status = alloc_status)
InstanceLoop: DO instance = 1, n_instances
   CALL AssignCostParameters ()  // An instance means a set of random cost parameters!
   special_heads, tails = heads, tails, 1, i, -n_special_arcs  // Save these values
   heads, tails = heads, tails, arcs, capacities, arcs, cost, parameters, 1, 1, n_arcs
   source, sink = (sources, sinks) / nodes, cuts = nodes, cuts, arcs, cuts, ars, cuts, n_arcs, n_cuts, cuts, capacities
   heads, tails = heads, tails, -n_special_arcs, (-1) = special_heads, tails  // Restore these values back
WHERE (cuts_capacities < 0.5_rwp) cuts_capacities = 0.5_rwp
    // Sometimes it breaks down due to integer arithmetic overflow!

nodes_cuts1:n_nodes = nodes_cuts1:n_nodes - MINVAL(nodes_cuts1:n_nodes) + 1  // Shift to 1
DO  arc = 1, n_arcs
    IF (arcs_cuts_arc > 0) THEN
        arcs_cuts_capacities_arc = cuts_capacities(arcs_cuts_arc)
        arcs_cuts_arc = n_cuts - arcs_cuts_arc + 1  // Reverse the order for plotting
    ELSE
        arcs_cuts_capacities_arc = 0.5_rwp
    END IF
END DO
arcs_cuts_capacities -n_special_arcs = 0.5_rwp  // Just in case

PlotCuts("Screen", "CONS", "5F0", arcs_cuts)
PlotCuts("Screen", "CONS", "5F1", arcs_cuts_capacities)

IF (file_type ≠ "CONS" AND file_type ≠ "XWIN") THEN
    PlotCuts("MC_MT_cuts." || TRIM(file_extension), TRIM(file_type), "5F0", arcs_cuts)
    PlotCuts("MC_MT_capacities." || TRIM(file_extension), TRIM(file_type), "5F1",
              arcs_cuts_capacities)
END IF

PointLoop: DO  point = 1, n_points
    WRITE (message_print_unit, *) "---->Doing instance, point #", instance, point
    IF (source_type ≡ 1) THEN  // Change the current
        sources_flow_capacity1 = total_currents_point
        sinks_flow_capacity1 = -total_currents_point
    ELSE IF (source_type ≡ 2) THEN  // Change the potential
        sources_flow_capacity1 = 0.5_rwp * total_voltages_point
        sinks_flow_capacity1 = -0.5_rwp * total_voltages_point
    END IF

CALL AssignSuppliesDemands()
CALL CalSSCNO (ElementalCosts = COST_TYPE & ElementalCosts,
               initialize_guess = (point ≡ 1))

nodes_potentials = nodes_potentials - MINVAL(nodes_potentials)  // A shift to the origin

IF (source_type ≡ 1) THEN  // Record the potential drop
    applied_voltages_point, sample = applied_voltages_point, sample +
                                (nodes_potentials sources(1) -
                                 nodes_potentials sinks(1))
ELSE IF (source_type ≡ 2) THEN  // Record the current
    injected_currents point, sample = injected_currents point, sample +
                                        (supplies_dems sources(1) -
                                         supplies_dems sinks(1))
END IF

IF (plot_samples) THEN
    IF (source_type ≡ 2) THEN
        PlotNetwork("Dummy", "CONS", "Absolute values of voltages", arcs_voltages,
                    nodes_potentials, T, T)
    END IF

    IF (file_type ≠ "CONS" AND file_type ≠ "XWIN") THEN
        WRITE (*, *) "Save this plot?"
        READ (*, *) save_plot
        IF (save_plot) THEN
            WRITE (*, *)
            WRITE (filename, "((E10.2)) total_currents_point"
                      file_name = "V_" || TRIM(ADJUSTL(filename)) || "," || TRIM(file_extension)
PlotNetwork(TRIM(file_name), TRIM(file_type), "Absolute values of arc voltages", arcs_voltages, nodes_potentials, \( T, \tau \))

END IF
END IF
ELSE
END IF
ELSE
END IF
END DO PointLoop
END DO InstanceLoop

applied_voltages = applied_voltages / n_instances // Average
injected_currents = injected_currents / n_instances // Average

Call DestroySSCNO

Call DestroyLatticeNetworkProblem(destroy_tree = \( T \))
DealocateArray(n_nodes, 1, byte, ALLOCATED, "Breakdown")
DealocateArray(n_arcs, 1, byte, ALLOCATED, "Breakdown")
DealocateArray(n_arcs_capacities, 1, byte, ALLOCATED, "Breakdown")
DealocateArray(cuts_capacities, 1, byte, ALLOCATED, "Breakdown")

Call RecordAllocation(n_elements = SIZE(special_heads_tails, i_wp), mold = i_wp,
caller = "Saturation")
DEALLOCATE(special_heads_tails, stat = alloc_status) // A temporary
END DO SampleLoop
END DO SourceType

Call StopTimer(100)
WRITE(*, *) "Solving", n_points * n_samples * n_instances, " NO problems took a total of (s) :"!, ReadTimer(100)
WRITE(*, *) "----------------------------------------"

Call ProfileSSCNO // Detailed timing statistics
WRITE(*, *) "----------------------------------------"

@if 1
Call InitGraphics(file = "V1.1", | TRIM(file_extension), file_type = TRIM(file_type),
n_plots = n_samples + 1, tick_labels = "GF1", legend_position = "UL", plot_title = (/ "IV curve for a 2D network" /), xlabel = "$I\$", ylabel = "$V\$")

IF (change_voltage \& change_current) THEN
axis = (/ MIN(MINVAL(injected_currents), MINVAL(total_currents)),
MAX(MAXVAL(injected_currents), MAXVAL(total_currents)), MIN(MINVAL(applied_voltages),
MINVAL(total_voltages)), MAX(MAXVAL(applied_voltages), MAXVAL(total_voltages)) /)
ELSE IF (change_voltage) THEN
axis = (/ MIN(MINVAL(injected_currents), MAXVAL(injected_currents), MINVAL(total_voltages),
MAXVAL(total_voltages) /)
ELSE IF (change_current) THEN
axis = (/ MIN(MINVAL(total_currents), MAXVAL(total_currents), MINVAL(applied_voltages),
MAXVAL(applied_voltages) /)
END IF

IF (change_current) THEN
averaged_voltages = SUM(applied_voltages, DIM = 2) / n_samples // Final average IV curve

Call Plot2D(x = total_currents, y = averaged_voltages, plot_spec = "L.B", axis = axis)
DO sample = 1, n_samples
   CALL Plot2D (x = total_currents, y = applied_voltages, sample, plot_spec = "SCR", axis = axis)
END DO
END IF
IF (change_voltage) THEN
   averaged_currents = SUM(injected_currents, DIM = 2) / n_samples  // Final average IV curve
   CALL Plot2D (y = total_voltages, x = averaged_currents, plot_spec = "L:R", axis = axis)
   DO sample = 1, n_samples
      CALL Plot2D (y = total_voltages, x = injected_currents, sample, plot_spec = "SSB", axis = axis)
   END DO
END IF
CALL EndGraphics ()
@#endif
CALL EndProgram
DEALLOCATE (total_currents, total_voltages, averaged_currents, averaged_voltages)
DEALLOCATE (applied_voltages, injected_currents)
END PROGRAM Saturation

"WEAVE.f90" 1.2 =
@m _PlotCuts(_file_name, _file_type, fmt, arc_array)
   arcs_mask = T_{I_{wp}}
   arcs_mask:0 = F_{I_{wp}}
   DO arc = 1, n_arcs  // We do not plot the periodic boundary arcs
      arcs_mask:arc = (arcs_mask:arc \& (arcs_status:arc \neq periodic_be:arc))
   END DO
   nodes_mask = T_{I_{wp}}
   nodes_mask:0 = F_{I_{wp}}
   CALL InitNetworkGraphics (file = _file_name, file_type = _file_type, xLabel = "Current flow \$\rightarrow\$", & page_size = (/5000, 5000/), label_format = "NOC", color_table = "RAIN", colorbar_position = "Horizontal", axis_labels_format = (/"NONE", "NONE", "NONE", "NONE")
   CALL PlotNetwork2D (heads_tails = heads_tails, 1:n_arcs, node_offset = -1, node_coords = node_coords, 1:n_nodes, node_mask = nodes_mask, 1:n_nodes, arc_mask = arcs_mask, 1:n_arcs, arc_values = REAL (arc_array, 1:n_arcs, r_wp), node_values = REAL (nodes_mask, 1:n_nodes), r_wp), resize_nodes = T, color_nodes = T, resize_arcs = T, color_arcs = T, & node_size_range = (-HUGE(1.0_wp) / 50, HUGE(1.0_wp) / 20), & node_colorbar_format = "5F0", arc_colorbar_format = fmt, vector_type = 0, axis = (/0.0, 0.0, REAL(lengths + 1)/))
   // The special arcs need to be plotted separately now
   nodes_mask:0:n_nodes = F_{I_{wp}}
   nodes_mask:0:n_special_arcs = T_{I_{wp}} arcs_mask = T_{n_special_arcs : (-1)}
   =TDaacr = n_special_arcs, -1  // We do not plot the periodic boundary arcs arcs_mask:arc = (arcs_mask:arc \& (arcs_status:arc \neq periodic_be:arc))

6