RegWidthColoringStep(top) begin
  if top >= n then
    completeColorings := completeColorings + 1;
  regWidth := totalRegisterWidth(ColorStack);
  if minRegWidth > regWidth then
    optimalColors := colorStack; minRegWidth := regWidth;
  if not MeetOptimizationTimeConstraint() then exit;
  end if
  return;
end if
for c in minColor(top) to maxColor(top) do
  colorStack(top) := c;
  if RegWidthLowerBound(colorStack, asap, alap) >= minRegWidth then
    cutBranches := cutBranches + 1; continue;
  end if
  if top < n−1 then
    oper := order(top+1);
    minC := estimateMinConflictColor(ColorStack, top, oper, ConflictRelation);
    maxC := estimateMaxConflictColor(ColorStack, top, oper, ConflictRelation);
    minP := estimateMinNonConflictColor(ColorStack, top, oper, NonConflictRelation);
    maxP := estimateMaxNonConflictColor(ColorStack, top, oper, NonConflictRelation);
    minColor(top+1) := maximum(asap(order(top+1)), minC+1, minP);
    maxColor(top+1) := minimum(alap(order(top+1)), maxC−1, maxP);
    if minColor(top+1) > maxColor(top+1) then continue; end if
  end if
  coloringStep(top+1);
end for
end

Figure 12: The algorithm of register width minimization on set of operator colorings

and generates the following output:

1. pipelineCount, which is the number of generated pipelines
2. optimalColor, which is the optimal pipeline schedule as an array of operators with the corresponding pipeline stage
3. minRegWidth, which is the minimum total register width of the optimal pipeline schedule

The algorithm in figure 12 works as follows. The recursive function takes in an input parameter top, which indicates the top record in the stack of operators. Depending on the top value, the function can return the control, generate the next complete coloring solution and compare it with the best current one, choose the next correct color of the current operator and generate the next record in the stack for procedure recursive call. In the next top+1 record, the minimum and maximum colors of the next operator is determined. If the minimum color is larger than the maximum color, then recoloring of the current operator is performed. The computation of minimum and maximum colors for operators are performed for both the conflict and nonconflict graphs.

Figure 13 shows an algorithm to estimate minimum colors from a conflict graph. Among all operators that are recorded in the stack as predecessors and are in conflict relation with the given operator op, the operator with maximum color gives the value of minC that is returned by the algorithm as minimum color of op operator. The computations of maximum color from a conflict graph, minimum color from a nonconflict graph, and maximum color from a nonconflict graph are performed in a similar way.

Once all operators have been colored and a valid pipeline schedule is generated, the total register width is estimated to evaluate the efficiency of the schedule. The function totalRegisterWidth(colors) performs this, which takes in a pipeline schedule, and returns the total register width. The function sums the width of all required pipeline and transmission registers of a pipeline schedule. From all possible pipeline schedules, the smallest total register width is stored in the variable minRegWidth with the corresponding optimalColors as the best schedule.