Constraint Satisfaction Problem

Mahesh Devalla

November 7, 2016

1 Introduction

Constraint satisfaction problems are typically found in mathematics domains. There are some problems in real life where we need to satisfy some of the requirements(constraints) to solve a problem(question). For instance at Dartmouth we need to complete 13 credits to get a master's degree with some additional constraints imposed on courses. In the same way some of the problems in the mathematics or computer science we need to satisfy some constraints like routing algorithms, graph algorithms etc. We have to satisfy a set of objects which need some requirements to be called as solved.

In CSPs' the constraints should be homogeneous and should be finite, we can't pose a copious constraints and expect a problem to be solved. All these constraints should be defined over a set of variables, where we use these variables as the base list of constraints. These Constraint satisfaction problems are used in artificial intelligence widely because most of the problems in the artificial intelligence holds constraints. They can solve many problems even one problem may completely differ from the other. This is because the basic idea of solving a problem doesn't change from problem to problem. The only issue with these problems is they require good computing power and additionally requires good heuristics and search functions.

Some of the problems that can e solved as a constraint satisfaction problem are as follows:

- Map coloring problem.
- Crosswords.
- Eight queens puzzle.
- Sudoku.
- Futoshiki.
- Kakuro (Cross Sums).
- Numbrix etc.

In the general case, constraint problems re difficult to solve due to the difficulty in their representation methodologies. some of the real world examples are

- Automated planning.
- Resource allocation.

In this assignment I have provided constraints satisfaction problems for *graph coloring* and *circuit design problems* .

2 Graph coloring

Graph coloring is simply a way of labeling or naming graph components such as vertices, edges, and regions by imposing some constraints on each of them or all of them for ease of understanding or solving a problem.Generally in graphs no two adjacent vertices, adjacent edges, or adjacent regions are colored with same color. While coloring the graph constraints are placed on colors or order of coloring or the way the colors are assigned.

Some of the applications of graph coloring are

- Image segmentation.
- Clustering.
- Resource allocation.
- Networking.
- Data mining.
- Processes scheduling
- Image capturing.

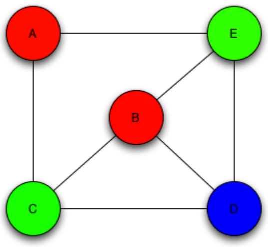

Figure 1: Graph Coloring

2.1 Chromatic number

The minimum number of colors required for a vertex to be colored is called chromatic number of a graph. As an instance in our example of Australia map we can't color this map with one or two colors. Even we can't divide this map into bipartite graph where we can color any big graph with just two colors.

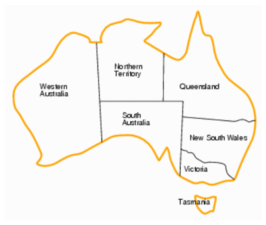

Figure 2: Australia Map

2.2 Bipartite Graph

A graph where vertices can be divided into two disjoint sets are called bipartite graphs. So that they can be colored with only two colors. In these types of graphs each edge connects a vertex to the other vertex in other set. Eventually these graphs don't contain any odd length cycles, so two colors would be sufficient. To the contrary we can't divide our Australia map into these graphs or else the computation could have been much easier.

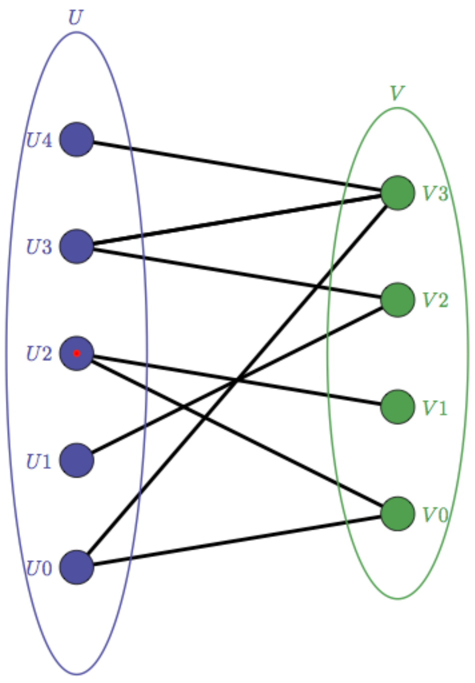

Figure 3: Bipartite Graph

2.3 Implementation of Map coloring

In this implementation of graph coloring, I created classes called *ConstraintSatisfaction*, *MapColorConstraints*, *MapColoring*, *Path* for ease of understanding and execution to separate from the circuit problem. Below are the implementations of the back track search code:

```
public ArrayList<Integer> Search() throws Exception {
    int nextassign;

    if (!baseList.contains(-1)) {
      return baseList;
    }

    if (isMRV()) {
      nextassign = getMRV(baseList);
    }
    else {
      int temp = baseList.indexOf(-1);
```

```java
      if (temp == -1) {
        System.out.println("This assignment is not valid");
        System.exit(-1);
      }
      nextassign = temp;
    }
    ArrayList<Integer> var = getOrgValues(nextassign, baseList);
    //Assigning only correct values
    for (int val : var) {
      baseList.set(nextassign, val);
      if (checkConstraints(baseList)) {
        if (isMAC_3()) {
          boolean flag = MAC3(nextassign, baseList);
          if (flag == true) {
            ArrayList<Integer> res = Search();
            if (res != null) {
              return res;
            }
          }
        } else {
          ArrayList<Integer> res = Search();
          ;
          if (res != null) {
            return res;
          }
        }
        baseList.set(nextassign, -1);
      }
    }
    return null;
  }
```

2.4 Implementation of inference technique (MAC-3)in Graph Coloring

```java
public boolean MAC3(int v, ArrayList<Integer> assignvals) {
    LinkedList<Path> path = new LinkedList<>();
    ArrayList<Path> passvals = getNextPath(v, assignvals);
    path.addAll(unUsedVals(passvals, assignvals));

    while (!path.isEmpty()) {
      Path temppath = path.pop();
      ArrayList<ArrayList<Integer>> list = new ArrayList<>();
      for (ArrayList<Integer> arr : listconsraints) {
        list.add((ArrayList<Integer>) arr.clone());
      }
      if (checknextvals(v, temppath.getFirstpath(), assignvals)) {
        if (listconsraints.get(v).size() == 0) {
          listconsraints = list;
          return false;
        }
        for (Path p : passvals) {
          if (!path.contains(p) && p != temppath) {
            path.push(p);
          }
        }
      }
    }
    return true;
  }
```

2.5 Implementation of MRV

Below is the implementation of MRV to just check the remaining values in the list.

```
public int getMRV(ArrayList<Integer> arr) {
  int mrv = 0;
  int mrvv = Integer.MAX_VALUE;
  for (int i = 0; i < arr.size(); i++) {
    int temp = getVal(i, arr);
    if (temp < mrvv) {
      mrv = i;
      mrvv = temp;
    }
  }
  return mrv;
}
```

2.6 Implementation of LCV

Below is the implementation of LCV to just check the remaining values in the list.

```
/*
 * A function for the implementation of LCV
 */
public ArrayList<Integer> getOrgValues(int v, ArrayList<Integer> assignval) {
  if (isLCV()) {
    ArrayList<Integer> listLCV = new ArrayList<Integer>();
    ArrayList<Integer> nextVals = nextVals(v);
    int prev = assignval.get(v);
    ArrayList<Integer> otrval = new ArrayList<>();
    for (int i : listconsraints.get(v)) {
      assignval.set(v, i);
      otrval.add(i, 0);
      for (int n_var : nextVals) {
        int tempval = getVal(n_var, assignval);
        otrval.set(i, otrval.get(i) + tempval);
      }
      assignval.set(v, prev);
    }
    for (int i = 0; i < otrval.size(); i++) {
      int tempmax = otrval.indexOf(Collections.min(otrval));
      listLCV.add(tempmax);
      otrval.set(tempmax, Integer.MAX_VALUE);
    }
    return listLCV;
  }

  else {
    return listconsraints.get(v);
  }
}
```

3 Results

Below are the states of Australia after the coloring and satisfying the constraints.

```
Hashtable representing the states of Australia are:
```

```
{[0, 1]=[[1, 0], [1, 2], [0, 1], [0, 2], [2, 0], [2, 1]], [5, 4]=[[1, 0], [1, 2], [0, 1],
    [0, 2], [2, 0], [2, 1]], [5, 3]=[[1, 0], [1, 2], [0, 1], [0, 2], [2, 0], [2, 1]], [2,
    3]=[[1, 0], [1, 2], [0, 1], [0, 2], [2, 0], [2, 1]], [5, 2]=[[1, 0], [1, 2], [0, 1], [0,
     2], [2, 0], [2, 1]], [5, 1]=[[1, 0], [1, 2], [0, 1], [0, 2], [2, 0], [2, 1]], [5,
    0]=[[1, 0], [1, 2], [0, 1], [0, 2], [2, 0], [2, 1]], [1, 2]=[[1, 0], [1, 2], [0, 1], [0,
     2], [2, 0], [2, 1]], [3, 4]=[[1, 0], [1, 2], [0, 1], [0, 2], [2, 0], [2, 1]]}



The colors represented in the Australia map after satisfying the constraints are:
[WA:GREEN, NT:RED, Q:GREEN, NSW:RED, V:GREEN, SA:BLUE, T:GREEN]
```

4 Circuit Board

These circuit layout problems are also can be considered as Constraint Satisfaction Problems, i.e. they can be solved by using the same techniques of CSP. In this We assign values to the variables that are having some constraints like in our assignment example. As a instance if V is a set of variables, D is a set of domains, one for each variable, and C is a subset of the Cartesian product of the domains, specifying illegal combinations of variables.

This CSP can be solved by selecting a tuple specifying the illegal and legal values of the subsets of the set of variables. Here we can represent the as constraint networks aligned with each other.

Generally, sometimes these algorithms run into dead-end when an assignment violates the existing constraint. This prevents from the extending the design of the circuit layout. So we require a backtrack algorithm to come back to the previous state and should find out a correct state that is feasible. The implementation and efficiency is dependent upon the design that are optimized. This can be done by using good heuristic functions and some methods like assign heavily constrained object first and later the smaller.

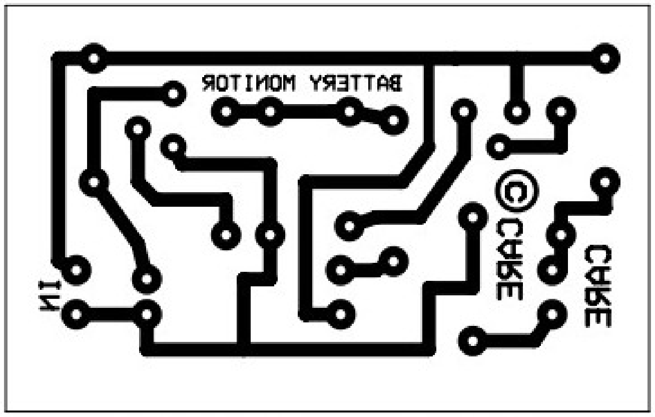

Figure 4: Circuit Layout

4.1 Methodology of Circuit Design in CSP

These are the methods generally followed in this problems:

4.1.1 Selection of Objects

- Selection of object that is most(heavily) constrained.
- Selection of object that has less remaining values.
- Max-Cardinality: Selection of variable that includes in the most constraints with the assigned variables.
- Min-Width: Selection of variables that includes in the most constraints with the unassigned variables.
- Max-Degree: Selection of variables that are included in the most constraints with all the other variables.

4.1.2 Value selection:

- Relaxed Model: Selection of the values that produces maximum number of solutions to a much simplified version of the problem. Many different solutions are created.

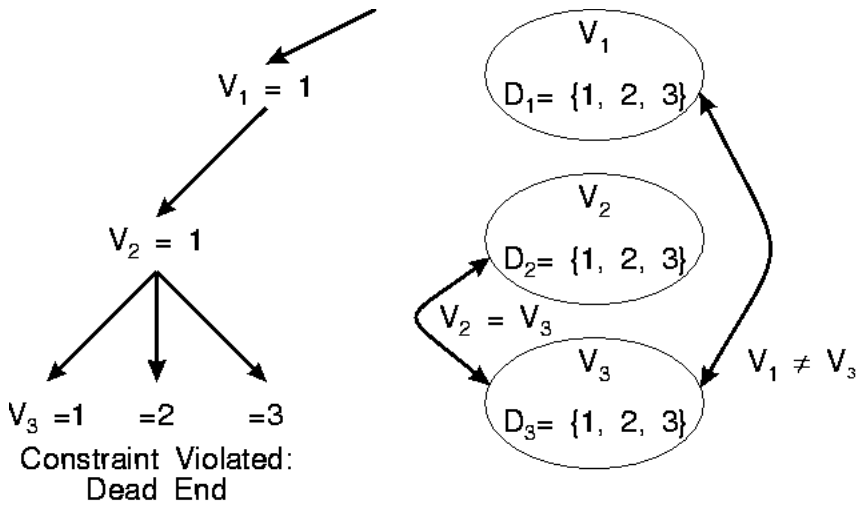

Figure 5: Circuit Design Implementation

4.2 Implementation of Circuit Board Problem

In this implementation of Circuit Board Layout, I created classes called *CircuitBoard*, *CircuitBoardConstraints*, *ConstraintSatisfaction* for the ease of understanding and execution. I used alphabets A,B,C as suggested in the assignment to overlay on the circuit board and "*" to just represent the empty space. Below are the implementations of the back track search code:

```
 public Map<Integer, Integer> solve() throws Exception {
   noOfCons = 0;
   Map<Integer, Integer> solution = SolvingConstraint(new HashMap<>(), 0);
   System.out.println("Number of constraints checked in the Circuit Board are:" + noOfCons)
   ;
   return solution;
 }

   public Map<Integer, Integer> SolvingConstraint(Map<Integer, Integer> map, int limit)
   throws Exception {
   Set<Integer> listOfCons;
   Map<Integer, Integer> temppath = path.get(limit);
   Map<Integer, Set<Integer>> tempunlist = unlist.get(limit);
   Integer ucons = randCons(map);
   if (ucons == -1)
     return map;
   listOfCons = baselist.get(ucons);
   int[] templistOfCons = new int[listOfCons.size()];
   int i = 0;
   for (int j : listOfCons) {
     templistOfCons[i] = j;
     i++;
   }

   int listcnt = templistOfCons.length;
   for (i = 0; i < listcnt; i++) {
     int j = templistOfCons[i];
     map.put(ucons, j);
     temppath.put(ucons, j);
     if (MAC(ucons, j, map, tempunlist, temppath)) {
       unlist.add(new HashMap<Integer, Set<Integer>>());
       path.add(new HashMap<Integer, Integer>());
       Map<Integer, Integer> mapcircuit = SolvingConstraint(map, limit++);
       if (mapcircuit != null) {
         return mapcircuit;
       }
     }
     Set<Integer> redudlist = new HashSet<Integer>(temppath.keySet());
     for (int red : redudlist) {
       map.remove(red);
     }

     Set<Integer> unlistset = new HashSet<Integer>(tempunlist.keySet());
     for (int unset : unlistset) {
       Set<Integer> cordx = tempunlist.get(unset);
       for (Integer tempcordx : cordx) {
         this.baselist.get(unset).add(tempcordx);
       }
       tempunlist.remove(unset);
     }
     tempunlist = new HashMap<Integer, Set<Integer>>();
     temppath = new HashMap<Integer, Integer>();
   }

   if (limit >= 1) {
     unlist.remove(unlist.size() - 1);
     path.remove(unlist.size() - 1);
   }
   return null;
 }
```

4.3 Implementation of inference technique (MAC-3)in circuit-board layout problem

Below is the implementation of inference technique for circuit board layout problem.

```
public boolean MAC(int v1, int v2, Map<Integer, Integer> cspcircuit, Map<Integer, Set<
    Integer>> unlist,
      Map<Integer, Integer> listOfCons) {
    int constraint, consval;
    Set<Integer> templist;
```

```
Set<CircuitBoardConstraints> circuit;
Iterator<Integer> itr;
CircuitBoardConstraints tempConstraint;
for (int i : getBaselist().keySet()) {
  if (baselist.get(i).contains(v2)) {
    baselist.get(i).remove(v2);
    if (unlist.containsKey(i)) {
      unlist.get(i).add(v2);
    } else {
      unlist.put(i, new HashSet<Integer>());
      unlist.get(i).add(v2);
    }
  }
}

for (int i : getBaselist().get(v1)) {
  if (i != v2) {
    if (unlist.containsKey(v1)) {
      unlist.get(v1).add(i);
    } else {
      unlist.put(v1, new HashSet<Integer>());
      unlist.get(v1).add(i);
    }
  }
}
getBaselist().put(v1, new HashSet<Integer>());
getBaselist().get(v1).add(v2);

for (CircuitBoardConstraints tempcons : constraints.keySet()) {
  noOfCons++;
  if (tempcons.getLength() == v1) {
    int tempVar = tempcons.getBreadth();
    if (!cspcircuit.containsKey(tempVar)) {
      circuit = constraints.get(tempcons);

      itr = getBaselist().get(tempVar).iterator();

      while (itr.hasNext()) {
        consval = itr.next();
        tempConstraint = new CircuitBoardConstraints(v2, consval);

        if (!circuit.contains(tempConstraint)) {
          itr.remove();
          if (unlist.containsKey(tempVar)) {
            unlist.get(tempVar).add(consval);
          } else {
            unlist.put(tempVar, new HashSet<Integer>());
            unlist.get(tempVar).add(consval);
          }
        }

      }
    }
  } else if (tempcons.getBreadth() == v1) {
    constraint = tempcons.getLength();

    if (!cspcircuit.containsKey(constraint)) {
      circuit = constraints.get(tempcons);

      itr = getBaselist().get(constraint).iterator();

      while (itr.hasNext()) {
        consval = itr.next();
        tempConstraint = new CircuitBoardConstraints(consval, v2);
```

```
            if (!circuit.contains(tempConstraint)) {
              itr.remove();

              if (unlist.containsKey(constraint)) {
                unlist.get(constraint).add(consval);
              } else {
                unlist.put(constraint, new HashSet<Integer>());
                unlist.get(constraint).add(consval);
              }
            }
          }
        }
      }
    }

    for (int temp : baselist.keySet()) {
      if (!cspcircuit.containsKey(temp)) {
        templist = baselist.get(temp);

        if (templist.size() == 0) {
          return false;
        }

        if (templist.size() == 1) {
          itr = templist.iterator();
          Integer nextValue = itr.next();

          cspcircuit.put(temp, nextValue);
          listOfCons.put(temp, nextValue);

          if (!MAC(temp, nextValue, cspcircuit, unlist, listOfCons))
            return false;
        }
      }
    }
    return true;
  }
```

4.4 Results

Below are the results i.e. circuit layout board problem after satisfying the constraints.

```
Number of constraints checked in the Circuit Board are:24
Circuit designed with the given constraints are:
AAABBBBBCC
AAABBBBBCC
EEEEEEE*CC
```

5 References

- *https://courses.cs.washington.edu/courses/cse573/06au/slides/chapter05-6pp.pdf*
- *https://courses.cs.washington.edu/courses/cse573/06au/slides/chapter05-6pp.pdf*
- *http://pami.uwaterloo.ca/ basir/ECE457/week5.pdf*
- *http://aima.cs.berkeley.edu/newchap05.pdf*

- $http://www.cs.dartmouth.edu/\ devin/cs76/05_constraint/constraint.html$
- $https://www.ics.uci.edu/\ welling/teaching/271fall09/CSPpaper.pdf$
- $https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-spring-2005/lecture-notes/ch3_csp_games1.pdf$
- $http://embedded.eecs.berkeley.edu/Alumni/wray/complex/complex/node16.html$
- $http://www.cs.cmu.edu/\ cga/ai-course/constraint.pdf$

References