School of Computing Science
Simon Fraser University

CMPT 310 - Introduction to Artificial Intelligence

Term:	**Fall 98-3**
Instructor:	**Bill Havens**
Office:	**10828 APSC**
email:	**havens@cs.sfu.ca**

Assignment #2: Graph Colouring

Reading Assignment

1. Read chapters 4, 6 and 7 in Russell & Norvig.

Homework

In this assignment, you will use constraint satisfaction techniques to solve map colouring problems. As described in class, map colouring is a convenient abstraction of real-world problems but intuitively easy to understand. We will use some map colouring instances made available as part of the DIMACS problem set. The problem instances are stored in the CSIL directory:

```
/gfs1/CMPT/310/src/Colouring
```

A map colouring problem consists of a set of nodes (variables), a set of possible colours for the nodes (domains) and a set of arcs between adjacent nodes int the graph (constraints). A solution is an assignment of a single colour to each node such that no two adjacent nodes have the same colour. The maps we used in lecture were planar since they could be drawn on a flat surface as regions (nodes) to be labelled (coloured). In general, maps can be nonplanar (and likely more difficult to colour).

The graphs I want you to colour are from Donald Knuth's Stanford GraphBase. Given a work of literature, a graph is created where each node represents a character. Two nodes are connected by an edge if the corresponding characters encounter each other in the book. Knuth creates the graphs for five classic works: Tolstoy's *Anna Karenina* (anna), Dicken's *David Copperfield* (david), Homer's *Iliad* (homer), Twain's *Huckleberry Finn* (huck), and Hugo's *Les Miserables* (jean).

The syntax for the DIMACS problem instances is as follows. Basically, there are some descriptive comments in the file, followed by a problem description specifying the number of nodes and edges, followed by the edge descriptions. Here is the problem description language in BNF form:

```
c comment line
c comment line
  ...
c comment line
p edge <numNodes> <numEdges>
e <nodeNum1> <nodeNum2>
```

```
e <nodeNum1> <nodeNum2>
 ...
e <nodeNum1> <nodeNum2>
```

where the line with an 'e' in column one indicate edges in the graph.

Map colouring is NP-hard in general so search is required. Various techniques have been employed. Many heuristics are also known. Constraint propagation is also effective. Please try the following techniques in C++, Lisp or whatever language you feel appropriate (note that Prolog is not appropriate here since I want you to implement the search algorithm yourself):

2. Implement a simple depth-first backtrack search algorithm for solving map colouring problems. Given a problem instance in a file, read in the problem, setup the constraint graph and search for a solution. I suggest that you make an explicit representation for nodes, domains and arcs since you will need them in the next part. Instrument your code so that you can see its backtracking performance, count the number of backtracks and print out any solutions found. Define a parameter "domain-size" for your search algorithm which specifies the number of colours initially in each variable domain. Add a parameter "max-fail" to the search algorithm which makes it stop after some specified number of backtracks. Add another parameter "all-solutions" which causes the algorithm to try to find all solutions instead of just the first one.

   Try your algorithm on the five instances from Knuth. Set domain-size to the optimal (minimal) number of colours for each problem which are as follows:

• anna	11
• homer	11
• david	13
• huck	11
• jean	10

   Instantiate the variables in strict descending order (n, n-1, . . ., 1)[1]. Look for only the first solution until you have debugged your code. What do you think would be a good value for "max-fail"? Remember that these problems are $O[k^n]$ where n = number of nodes and k=domain size. If you cannot get a solution, try increasing the domain size slightly from the optimal value. The "homer" file is quite large (561 nodes) so save it for last.

3. Use arc consistency techniques to improve the search for map colouring. Whenever you instantiate a node to a particular colour from its domain, you have reduced that domain to a singleton (as per the AB-Consistent algorithm given in class). This may mean that neighbouring nodes (connected via an arc) are now arc-inconsistent with this node. Likewise, if the domains of any of these nodes is reduced, then arc-consistency needs to be propagated to their neighbours recursively.

   Instantiate the variables in strict descending order as before. How much better does adding arc-consistency work than simple backtracking? Remember that disequality is a weak constraint. Show your results as a table comparing the two methods on each problem.

1. Ascending order is known to be very easy for these datasets.

4. Now use the first-fail principle with arc consistenc to assign variables in heuristic order. The first-fail heuristic picks the uninstantiated variable with the smallest remaining domain to instantiate next. If two or more variables have the same minimal domain, then such ties are broken randomly. How much better does this work for arc-consistency? Again show your results as a table for each problem.

Due Date: Tuesday, October 13 before 3pm in the class assignment box (note change of due date)