

CS472 Foundations of Artificial Intelligence
Fall 1999
Assignment 4 / Randomization (Programming)
Due Friday, November 19 at the beginning of class

Your task is to create a Java Applet that demonstrates various randomization techniques in the context of search. The application domain is the “Quasigroup Completion Problem”. A simple description of the problem follows.

For an example see www.cs.cornell.edu/Info/People/gomes/QUASIdemo.html.

Quasigroup Problem — Given an $N \times N$ grid and N colors, color each cell of the grid in such a way that the same color does not occur more than once in each row and each column. Another way of stating this problem is that each row and each column of the grid must contain all the N different colors.

Quasigroup Completion Problem — given a partially filled $N \times N$ grid, without repetition of the same color per row or per column, color the remaining cells of the grid to obtain a complete quasigroup.

This problem can be formulated as a Constraint Satisfaction problem: each cell of the matrix corresponds to a variable and its domain is the set of possible colors for that cell. The constraints are such that any two cells in the same column or row cannot be assigned the same color.

1. (15 pts.) Create an applet to generate a 15×15 grid, where each cell can be given one of 15 colors and implement a mechanism that allows a user to manually color each cell of the grid. Make sure you don't allow the same color to appear more than once per row and per column.
2. (15 pts.) Implement depth-first search with forward checking and propagation. In forward checking, each time a variable is instantiated (*i.e.*, given a value) you remove inconsistent values from the domains of the remaining uninstantiated variables. You branch on the variable with the smallest domain first. Note that when removing inconsistent values from the domain of a variable, if the domain of that variable is reduced to one single value, additional propagation is triggered in order to remove inconsistencies between the domains of the remaining variables and the domain of the variable whose domain became a single value. (So, when a domain is reduced to one value, the variable needs to be assigned the remaining color, and inconsistent colors can be removed from the domains of the other variables.)
3. (10 pts.) Randomization. Randomize your depth-first search. This means that when selecting a variable from the set of variables with the smallest domain, in case of ties, select a variable randomly (from those that are tied for smallest domain). Also, from the remaining colors, select one randomly. You need to keep track of which colors and variables have been tried when backing up, so that you don't repeat choices, and your algorithm remains complete.

Incorporate a `cutoff` value in your algorithm. When the number of backtrack reaches the cutoff value, the procedure restarts its search.

4. (10 pts.) Experiment. Find a pattern that is relatively hard to complete (but can be completed) for the two strategies. Report the number of backtracks it takes to complete the pattern (for randomized strategy report the average number of backtracks over 10 runs). What is a good cutoff value?
5. (10 pts.) Use conjugate constraints to improve on randomized depth first search. Consider the following additional (redundant) conjugate constraints: each color has to occur at least once per column and row. One way to get extra pruning from this is to ensure that after the propagation, each color still occurs in the union of the remaining domains (after propagation) of the variables for each row and for each column. Incorporate this form of pruning into your randomized depth-first search to create a new strategy.
6. (10 pts.) Experiment. Use strategy with conjugate constraints on the same pattern used in the experiment above and report the average number of backtracks over 10 runs. (Note a “run” is a series of one or more restarts that leads to a solution.) Tune the cutoff value to optimize the overall performance.
7. (10 pts.) Implement a local search strategy. Try to find a good strategy. Describe your strategy in your writeup.
8. (10 pts.) Experiment. Use local search on the same pattern used in the experiment above and report the average number of local changes needed to complete the pattern using local search (average over 10 runs).
9. (10 pts.) Experiment. Consider the following pattern to be completed: along one of the main diagonals all the cells have the color red except one that has the color blue. Report how the different strategies perform on this pattern. Explain the difference in performance.

Your applet should also include the following features:
(See also URL mentioned in intro.)

1. A button for each search strategy.
2. A “stop” button, which allows you to interrupt the search at any time.
3. A button for manual preplacement.
4. Have an option to store your preplaced pattern.
5. A “clear” button, which clears the grid, except for the preplaced pattern.
6. A “clear-all” button, which clears all squares.
7. A display of the number of backtracks for depth-first search, the average number of backtracks until solution for the randomized searches, and the number of local changes until solution for local search.

8. Have options to set various parameters, such as the “cutoff” value and “delay” value (for slowing down the graphics display).
9. Colors are displayed during the search (to show progress).

In grading this assignment, in addition to the standard issues of program correctness, we will check the following:

1. Your applet should run in a standard browser. This is very important: all other checks depend on it.
2. Your applet should find correct solutions (i.e., fully colored grid with no repeated colors in rows or columns).
3. Your search routines should not change the manually assigned pattern.
4. When entering colors manually, you check for obvious conflicts (and disallow those).
5. In the depth-first search, your applet should select the most constrained variable first. (You can check this yourself by entering a pattern which causes one square to be highly constrained. Your applet should start by coloring that square. This also requires your propagation to work correctly.)
6. Your randomized depth-first search should give different runs on each restart.
7. The strategies except for local search should be complete. You can check this yourself by giving it a pattern that cannot be completed. In that case, your applet should be able to determine that the pattern cannot be completed. (Note: This may take too long to check for pure depth first search.)
8. We’ll also check the various features listed above, and your written report.

Hand in a description of your applet with detailed answers to the questions given above. Also add an on-line version of this description (html) to your code. Provide comments in your code, especially to describe your main data structures.

You may work in teams of two.

I will announce a mechanism to submit your final code.