CMPT-827 - Intelligent Systems
School of Computing Science
Simon Fraser University

Bill Havens
TASC1-9425
August 27, 2007

Assignment #1: CSP Solving

1 Overview

This assignment is designed to give students practical experience in using modern constraint programming (CP) algorithms for solving constraint satisfaction problems (CSPs). You will employ both constructive backtrack methods and stochastic local search methods to solve a set of randomly generated CSPs. You will experiment with variations of these techniques and compare their performance on the test problems. By so doing, you will gain knowledge about the difficulty of CSPs and the power of modern methods to solve them.

We are given a database of random CSP problem instances on the website. All of these problem instances are solveable meaning that at least one solution exists for each instance. Your job is to find one of these solutions. Each problem instance is represented by a set of parameters recognized by a random CSP generator written in Java which is available on the website. Running the problem generator on a problem instance will create a constraint network of CSP variables and constraints. You then run your various search algorithms on this constraint network for each such problem instance. You will keep track of interesting performance parameters of your algorithms in order to compare your results later.

You will experiment with at least three different search methods on these random CSP instances. You can try more methods if you have time and are interested. You will try a constructive (backtrack) algorithm, a basic hillclimbing local search algorithm with random restart and a more sophisticated local search algorithm (such as *tabu* search or *simulated annealing*). We will cover these search methods in lecture during the first part of the course.

You will provide a short report on your experimental results which plots the runtime performance of the three algorithms on the given dataset. We will compare the performance against the probabilistic difficulty of each problem instance. Details of these experimental techniques will be provided in lecture.

2 Random CSP Generator

There has been considerable research in the CP community in constructing hard random CSPs. The size and difficulty of these problem instances can be varied parametrically using standard random CSP generators. We have such a generator written in Java which is available on the class website under the link: `CSP Solving Resources`. A short tutorial on using the generator is provided. As well, there exists a calculator written in *MSExcel* which allows you to vary the expected difficulty of each problem instance by varying its parameters. Details of these parameters will be discussed in class.

Also on this web page is a database of 4500 solveable random CSP problem instances.[1] These problem instances have 20 variables each with a domain size of 10 values. They vary from easy to hard with 150 instances of each problem difficulty. By finding a single solution for each problem instance, you will be able to plot the relative performance of your algorithms for each level of problem difficulty.

3 Search Methods

You will implement three different search algorithms and apply each algorithm to the random CSP dataset. The algorithms are the following but many variations are possible:

1. **Constructive Backtrack Search** - Implement a popular backtracking technique for solving CSPs as described in class. Simple chronological backtracking will be inefficient but easy to implement. A better choice will be backjumping which is not much more complicated. Add forward checking of unassigned (future) variables to your search algorithm. This will significantly reduce the size of the search space. An alternative is to use full arc-consistency among future variables. Use minimum domain size of future variables (*first-fail principle*) as an effective variable ordering heuristic. Does this variable ordering work significantly better than assigning variables in lexicographic order? Use lexicographic as the value ordering heuristic. Can you think of any alternatives which might work better?

2. **Hillclimbing Local Search** - Implement basic hillclimbing local search for CSPs using *minconflicts* as the evaluation function. Initialize the search with a random assignment of values to variables. Since all local search algorithms get stuck in local minima, use random restart to escape local minima. Use max-conflicts as the variable ordering heuristic and min-conflicts as the value ordering heuristic. Break ties randomly for both variable and value ordering heuristics.

3. **Advanced Local Search** - Extend the basic hillclimbing method above by adding either *tabu* search or *simulated annealing* search. Both are good at escaping from local minima. Does this work better than random restart? You may also try other local search methods if you like.

[1] file: `RandomCSP20_10_0.4(0.20-0.49).txt`.

4 Experimental Evaluation

Run your three algorithms on the dataset provided. For each instance, keep track of the number of iterations and the run-time required to find a first solution. Some problems will have more than one solution but we only want to find the first one. For backtrack search, we take an iteration to be the assignment of a variable to a value (either a future variable or a variable being backtracked). For local search, an iteration is simply the assignment of a value to a variable.[2]

Now plot the runtime performance of all three algorithms against the parameter p2 in the random CSP dataset. You will have 150 execution performance values for each value of p2 which will smooth out the uncertainty of each problem instance.[3] You should see a peak in problem difficulty for certain values of p2 that is manifest in the performance of all three algorithms. Does this peak correspond to $\kappa = 1.0$ using the provided κ-calculator?

5 Report

Provide a short report on your experiments which covers the following issues. The report should be in good technical style with appropriate references to papers relevant to your approach. The length of the report should be about 10 pages single spaced but use as much space as necessary to describe your results and present your diagrams. A report template will be provided on the website.

1. Which methods did you implement? Describe the algorithms used including the details of the constraint propagation, backtracking, variable and value ordering, restart techniques, *tabu* or simulated annealing parameters. Give references into the appropriate literature. What variations did you implement if any?
2. Provide run-time performance graphs of the algorithms. Describe the results in good technical English. How well did the three algorithms work on the dataset?
3. Discuss what you have learned about solving CSPs. Are some methods better than others? What variations or other algorithms would you expect to work well on random CSPs?

[2] The initial randomc assignment of variables counts as a single iteration.

[3] If the results are not sufficiently smooth for the randomized local search methods then we can run each instance multiple times and use an average performance value.