CS420 – Project 1
Sudoku Solver/Generator
Due: October 21th, 2005

Introduction

For this assignment you will create two programs: one will be a Sudoku problem solver and the second will be a Sudoku problem generator. The command-line format for these programs are given below. To complete this assignment you will use Constraint Satisfaction techniques discussed in class. The solver will use a variety of heuristics to solve the problem and will generate the appropriate performance statistics so that your implementation can be evaluated. The generator will accept a solution and create a Sudoku puzzle that has the most numbers removed and is still capable of producing only a single solution. Once complete you will submit your finished programs and a minimum 10-page report describing your efforts. The format of the report will follow the the ACM Author Submission Guidelines. A link to the guidelines can be found on Professor Pfaffmann's website, along with useful Sudoku links.

Program Specifications

The programs for project 1 are to be written in Python, and are not restricted to functional programming techniques. The programs should follow good software engineering practices. Additionally, all students will strive to reproduce the same program interface, unifying the command-line processing and file formats.

File formats will consist of 9 lines, each containing 9 items. Each item will contain a number from 1 thru 9 or a dash ('-'), where the dash represents an empty location on the puzzle. Each item on a line will be separated from the next by a single blank character. If a data file is not properly formatted, then your program should reject it. This should indicate that the file format is correctly implemented and the provided puzzle is consistent with the game constraints.

Two data file examples are given below:

```
- - 6 - - - - - 1
- 7 - - 6 - - 5 -
8 - - 1 - 3 2 - -
- - 5 - 4 - 8 - -
- 4 - 7 - 2 - 9 -
- - 8 - 1 - 7 - -
- - 1 2 - 5 - - 3
- 6 - - 7 - - 8 -
2 - - - - - 4 - -
```

```
5 3 6 8 2 7 9 4 1
1 7 2 9 6 4 3 5 8
8 9 4 1 5 3 2 6 7
7 1 5 3 4 9 8 2 6
6 4 3 7 8 2 1 9 5
9 2 8 5 1 6 7 3 4
4 8 1 2 9 5 6 7 3
3 6 9 4 7 1 5 8 2
2 5 7 6 3 8 4 1 9
```

Below are the requirements for each of the two programs. Deviation from these requirements will result in point loss.

Sudoku Solver Requirements

Your solver must have the following features:

- The program will be called `sudoku_solver_yourid`, where yourid indicates that your id should be appended to the name.
- The program must use the following command-line format:
  - `sudoku_solver_yourid <command_options> <input_file.sdk> <output_file.sdk>`
  - The command-line options will specify the different heuristic that will be used:
    - `-b` : using just backtracking;
    - `-bh` : using backtracking and a variable ordering heuristic;
    - `-f` : using forward checking;
    - `-fh` : using forward checking and a variable ordering heuristic;
    - `-cp` : using constraint propagation.
  - The input file must use the extension "`.sdk`" and will contain the Sudoku puzzle.
  - The output file must use the extension "`.sdk`" and will contain the Sudoku solution.
- The program must follow good programming practices, such as, well formated code, meaningful variable names, comments, etc.
- The program must be executable from the shell.
- The program must implement the different solution techniques specified above in the command-line options.
- The program, once complete, will print the runtime statistics to standard-out before exiting. These statistics will be nicely formatted.

Sudoku Generator Requirements

Your solver must have the following features:

- The program will be called `sudoku_generator_yourid`, where yourid indicates that your id should be appended to the name.
- The program must process based on the following command-line format:
  - `sudoku_solver_yourid <solution_file.sdk> <problem_file.sdk>`
  - The solution file must use the extension "`.sdk`" and will contain the solution for the "minimum unique puzzle" the generator is searching for.
  - The problem file must use the extension "`.sdk`" and will contain the "minimum unique puzzle" your program must generate.
- The program must follow good programming practices, such as, well formated code, meaningful variable names, comments, etc.
- The program must be executable from the shell.
- The program, once complete, will print the runtime statistics to standard-out before exiting. These statistics will be nicely formatted.

Sudoku – The Problem

Links to Sudoku explanations can be found on the professor's website. Sources for example problems will be found by the students themselves, which can be done by looking in puzzle books or newspapers. Be sure that the puzzles you use have singular solutions. Also, it helps to work the problem to get a feel for how the different constraints can be manipulated. A week before submission, the Professor will distribute a problem and a solution to verify that your reading correctly formatted input files.

Your solver program should attempt a solution in the five following ways:

1. Using basic Backtracking, which will give the lower bound on your program behavior and gives you a baseline to compare other techniques to.
2. Using basic Backtracking with a variable/value selection heuristic of your choice.
3. Using Forward Checking.
4. Using Forward Checking and your previously chosen variable/value selection heuristic.
5. Using Constraint Propagation.

Note, you are being given the opportunity to choose from a variety of techniques. This is done purposefully to push you to think about which may be the better alternative. Additionally, each submitted program will be run and compared to see which is most efficient on an unknown set of problems, the top performer in each category will win a prize for its designer.

This competition will be evaluated based on the mean number of consistency checks over 5 problem instances, similar to figure 5.5 in the textbook. Since this is a competition, the number of consistency checks that your program performs during its execution must be displayed as the program exits. You will also find it useful to determine other metrics of program performance during the execution of your program, which should also be displayed when your program exits. Since there may be programs that perform the same, ties will be broken based on raw processing time (or wall-time).

The solver portion of the assignment is fairly straight-forward, while the generator portion is not, because there is not an immediate solution (other than brute force). So the question becomes, is there an elegant way of using the constraints to remove numbers from the grid to find the minimum sized puzzle that will still generate a single solution. Note, it is easy to see that here will be multiple suboptimal results, the trick becomes finding the global optimum (and whether or not there are several equivalent global optimums).

Submission

Your submission should consist of the two programs, the puzzles that you used in your experiments, and a 10-page (minimum) paper in PDF format. These items should be placed in a directory called "`CS420_Project1_yourid`", which is then tarred and gzipped (producing a file called "`CS420_Project1_yourid.tgz`").

The paper requires you to summarize what you did, describe your approach, present your results, and discuss your conclusions about the viability of your approach. Be aware the professor is looking for a conference quality submission, in both organization and information content. Thus, the paper should follow the ACM submission guidelines, with an additional requirement of 1 inch margins. A link to these guidelines can be found on the instructor's website. The paper must contain the following elements in the given order.

- **Abstract** – The paper should start with a 500-word abstract summarizing your work.
- **Introduction** – Introduce the problem and your approach to a casual reader with a technical background.
- **Formal Description** – Formally describe the problem and how you decided to solve the problem.
- **System Performance** – Describe your methodology for determining system performance and quantitatively discuss its performance.
- **Conclusion** – Re-introduce your problem/approach, qualitatively discuss the performance of your system, and draw-out any conclusions or lessons that may have been learned.