CS 440: Project 1
A Sudoku Puzzle Solving Agent
Due March 1

Scott Wallace

Abstract

This handout describes Project 1: A Sudoku Puzzle Solving Agent. The project will be executed in a team environment. Students will implement at least two solution strategies for the problem and then evaluate their strategies with a set of experiments. A written report will document both the implementation and the experimental evaluation.

Overview

This project is to be carried out in a 3-4 person teams. The goals are threefold: first, to further develop your AI/Search toolkit; second, to help develop your technical writing skills; and third, to give you some research savvy (this will come in handy regardless of whether you decide to head off to graduate school).

For this project, you will implement a Sudoku puzzle-solving agent using a set of strategies that you choose. At a minimum, you must implement at least two strategies from amongst the approaches covered in class. Strategies need not be completely different, but they should differ in more than a few lines of code (e.g., switching from depth-first to breadth-first alone, won't cut it).

If you are a firm believer that only one strategy from class is reasonable in this context, or if you wish to expand your report, you may also consider multiple *implementations* of a particular strategy. In this case again, you should ensure that your implementations differ in more than a few lines of code from one another. You can do this by changing data structures, or heuristics (and possibly other things as well). Be sure to document your changes so you can adequately report them.

The project has three basic parts: solution design/implementation, evaluation and write-up.

Sudoku

A standard Sudoku puzzle consists of a 9x9 grid which is sub-divided into 9 3x3 squares as show in Figure **??**. Each of the 81 cells contains either a number in the range 1-9, or is empty. The puzzle is solved by filling in each of the blanks with number 1-9 subject to the following constraints: each row must contain all the numbers 1-9, each column must contain all the numbers 1-9, and each of the 9 3x3 squares must contain the numbers 1-9.

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

A Sudoku Puzzle
Public domain image from: http://en.wikipedia.org/wiki/Sudoku

Figure 1: A 9x9 Sudoku Puzzle

Sudoku puzzles can be generalized for any n^2xn^2 grid. The standard puzzle described above is the $n = 3$ puzzle.

Solution Design and Implementation

You can implement your sudoku solving agent using a programming language of your choice. I suggest Python, but, if your teammates are willing, you may want to use another language all together.

Your implementation must consist of a program that is executable via the command line on one of the lab machine's operating systems. The program should accept sudoku puzzles from `stdin` and print results to `stdout`. Different strategies should be selected using a flag sent to the program. Flags should be single characters starting with *a*, in alphabetic order. Running the program with no flags should print a help message describing the strategy associated with each flag.

A puzzle is represented as a n^4 string of characters that

begins with the digit in the upper-left corner of the puzzle and proceeds from left to right and top to bottom. Thus the standard 9x9 puzzle contains $3^4 = 81$ characters. Each character is either a number 1-9 or a '.' to represent an empty cell. Larger puzzles (e.g., $n = 4$ or $n = 5$) can be described using alpha-numeric characters for digits. As with standard hexadecimal notation, assume $a = 10$ and character values increase through z and then continue with upper case letters $A \ldots Z$. This provides enough "digits" to generate an $n = 7$ puzzle. Your program should handle $n = 3$ puzzles. If you are able to solve larger puzzles, extra credit may be granted to your work.

To solve the Sudoku puzzles, you should select at least two strategies (you may do more; ambitious projects will earn extra credit). Each strategy should be implemented to solve puzzles as specified above. Be sure to document your code so we can take a look at your implementations. A good project will select strategies in a thoughtful manner, either to illustrate a point, or to explore particular performance issues.

Evaluation

In Computer Science, research papers typically revolve around a specific problem that is solved using a method proposed and implemented by the paper's authors. The authors must then demonstrate why their approach is interesting enough to warrant a paper. This is to say, they must evaluate their work in some way that is meaningful to the readers.

For this project, you must similarly provide an evaluation of your solution strategies for Sudoku. There is no single recipe for what constitutes a "good" evaluation; much will depend on exactly what strategies you choose to implement. The overall goal of the evaluation should be to characterize the strengths and weaknesses of each approach within the Sudoku puzzle domain.

A reasonable way to perform evaluations for this environment is with a set of experiments each of which ends with a set of data that can be used to generate a graph or table. All projects should contain at least two graphs/tables, and each graph/table should illustrate a different property of your solutions. Put another way, each graph or table should help to answer a question that someone else might have about your work. For example, most users of a Sudoku problem-solving agent would probably be interested to know how quickly the agent can solve puzzles using its built-in set of strategies. Users may care about average time and also about the worst case time. Likewise, if you swap out a data structure, you should consider what theoretical impact that might have and then try to produce a graph to illustrate the difference in practice. [1].

At a minimum, you should be sure to include two graphs illustrating different properties of your solutions. One graph/table should illustrate the amount of CPU time required to solve a fixed set of benchmark problems provided on the website. Ambitious projects may go beyond evaluating the solution strategies and also try to characterize the space of Sudoku puzzles (that is, the space beyond the small sample set that is provided).

[1] I'm willing to entertain other approaches for evaluating your work, but please run them by me first.

Write Up

To complete the project you must submit a write up that documents your approach along with the executable code.

The writeup should be 4-5 pages in AAAI two-column format. This is the standard format used in publications and will equal roughly 8 pages in microsoft word's default format. It is also the format that this document is written in.

The report should contain an *abstract* that briefly summarizes the paper in a single paragraph. Next, you should write a section with the title *introduction* that lays out the problem. Following this you may choose to write one section per strategy and evaluate each strategy within that section, or to write a single *implementation* section describing all of your strategies (possibly using subsections to signpost the strategies) and then a subsequent section entitled *evaluation* to describe the experiments performed. In either case, these middle sections should constitute the bulk of the writing and describe your implementations in enough detail for me to understand what you've done and think through the ramifications *without* needing to look at your code. Your paper should end with a section entitled *conclusions and future work* in which you should outline the "take away message" from your experiments as well as a set of future enhancements/implementations or experiments that would be interesting to pursue if you had unlimited time.

I strongly suggest that you do your writing in LaTeX. It's a bit of a drag to learn, but I'll provide some samples that will get you going. It's also much less likely that you'll lose work due to your text editor choking on a figure than if you use one of those "new-fangled" WYSIWYG word processors.

Turning it in

You need to turn in three things for this assignment:

1. The write up for your implementation and experiments as described above.
2. A executable file (or python source code) that can be run on the lab machines. Please make sure you specify what OS I should use.
3. A directory labeled `src` containing all of your source code along with compilation instructions if applicable.

Put all of these items into a folder with the last names of your group members (no spaces please), zip it up and have one person from your team submit to blackboard.