CSCI 446 — Artificial Intelligence

Project #1
Search, Constraint Satisfaction, and Sudoku

1 Project Outline

Constrained Search

The purpose of this assignment is to give you an introduction to artificial intelligence from the perspective of search. You will develop a flexible solver that will be able to switch quickly between a set of given heuristics and use constraint satisfaction to navigate through the search. You need to develop a high level structure that will be able to take a problem and heuristic and then trace through the reasoning process to generate a valid result. You will also report on the computational complexity in terms of the number of state assignments and backups leading to your final result. Notice that the heuristics used by your solver will be adapted to the problem it will be applied to (namely Sudoku). That said, the driver needs to be versatile between problem instances and heuristics.

Sudoku

Sudoku is a class of puzzles taking place on a 9x9 grid. In standard games, this is further subdivided into separate 3x3 sections. The start of the puzzle consists of a subset of numbers already filled into spaces on the grid. The goal of each puzzle is to finish filling the grid with numbers. However, there are three basic rules of number placement that constrains the puzzle. First, each row must contain all numbers, one through nine, with no repeats. Second, each column must also contain all numbers, one through nine, with no repeats. Finally, each 3x3 grid is also subject to the same constraints and must have all numbers, one through nine. An example puzzle and solution is shown in Figure 1.

For this project, you need to be able to read in a Sudoku puzzle and then solve it. More specifically, you will implement and compare different heuristics in combination with a backtracking search designed to solve Sudoku puzzles. Three separate iterations of the backtracking search must be implemented, with differing heuristics for each. You will then test the performance of these methods against sample puzzles.

The input files for this assignment utilize the same format. Each line of the sudoku puzzle is contained on a separate, single-spaced line. All items within the line are separated by spaces. For this assignment, a '?' indicates an unknown value and any number will indicate a known value. An example input file is shown in Figure 2. Any created puzzles utilized in this assignment for experimentation should be non-trivial. Thus, a nearly solved puzzle with only a few missing items should not be created as an example.

New this year: In the past, we have requested that you provide a video that demonstrates the functioning of your code. In short, that has always presented problems for students due to the time limit on the videos and confusion over what should go into the video. This year, we are eliminating the video requirement. Instead, we are imposing a restriction and a new requirement. The restriction: All projects must be completed in Python. No exceptions will be granted. The new requirement: You must submit a notebook (e.g., Jupyter) that the teaching assistant will use to execute your code on a set of puzzle that ***will not*** be provided to you. You will be graded on the number of held-out puzzles your program is able to solve.

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

Figure 1: Sudoku Example

puzzle1.txt:

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

Figure 2: Example File

2 Input/Output Requirements

Since the TA is going to be running your code on puzzles you have not seen, it is critically important that a standard be applied for handling input and output. That's what this section describes. You should include how you will address the following as part of your Design Document.

2.1 Notebook Parameterization

Your notebook must define a # Parameters cell at the top with the following variables (exact names and cases):

```
# Parameters
GROUP_ID = 'YourGroupHere'
ALGORITHM = 'bt'
PUZZLE_TYPE = 'easy'
PUZZLE_PATH = 'puzzles/easy/puzzle01.txt'
```

Supported algorithm parameters:

- `bt` – Backtracking
- `fc` – Foreward Checking
- `ac3` – Arc Consistency (AC3)
- `sa` – Simulated Annealing
- `ga` – Genetic Algorithm

Supported `PUZZLE_TYPE` parameters:

- `easy`
- `medium`
- `hard`
- `extreme`

2.2 Input Format

The file given in `PUZZLE_PATH` will be a 9×9 Sudoku puzzle in row-major, comma-delimited text format.

2.3 Output Format

After solving, your notebook must save the completed puzzle in the same format as the input, with all '?' replaced by digits 1–9.

File Naming (must match exactly):

`[GROUP_ID]_[ALGORITHM]_[PUZZLE_TYPE]_[PUZZLE_NUMBER].txt`

Where:

- `GROUP_ID` – exactly as provided in the parameters.
- `ALGORITHM` – exactly one of: `bt, fc, ac3, sa, ga`.
- `PUZZLE_TYPE` – exactly one of: `easy, medium, hard, extreme`.
- `PUZZLE_NUMBER` – comes from the puzzle file name, without extension (e.g., `puzzle01`).

Example filename: `Group01_fc_easy_puzzle01.txt`

Example file content:

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

2.4 Implementation Rules

- The solver must read `PUZZLE_PATH` and `ALGORITHM` from the parameters, not from hardcoded values.
- The file must be saved in the notebook?s working directory.
- No CSV, JSON, or other formats – plain .txt only.

3 Deliverable Requirements

- Implement a puzzle importer. This will use a standard format (described above) for your puzzles.
- Sixteen different puzzles will be provided to you consisting of four easy, four medium, four hard, and four extremely hard. These should be read and processed separately based on the specified file name. You will be tested on four held-out puzzles, one in each category.
- Implement a single constraint solver capable of solving any 9×9 Sudoku puzzle, implementing the following variations:
  - simple backtracking (only run this on easy and medium puzzles),
  - backtracking with forward checking,
  - backtracking with arc consistency,
  - local search using simulated annealing with the minimum conflict heuristic,
  - local search using a genetic algorithm with a penalty function and tournament selection.
- Write a design document that conforms to the requirements in the document "Creating a Design Document," which you will find in the Course Information module of the course content in Canvas.

- Run experiments, keeping track of the main decisions being made for each algorithm. The decisions you count to give the performance metric should be described in both the design document and the paper. The count of these decisions will be used to gauge run time since CPU time and wall clock time are not reliable estimators. Warning: Do NOT record CPU time or wall clock time, or your grade will be penalized heavily.
- Write a paper that incorporates the following elements, summarizing the results of your experiments. Make sure you explain the experimental setup, any hyperparameter tuning process you might employ, and the final parameters used for each algorithm (if any).
    1. Title and author names.
    2. A brief, one paragraph abstract summarizing the results of the experiments.
    3. Problem statement, including hypothesis (how do you expect the different algorithms to perform?).
    4. Description of algorithms implemented.
    5. Description of your experimental approach.
    6. Presentation of the results of your experiments.
    7. A discussion of the behavior of your algorithms, combined with any conclusions you can draw.
    8. Summary.
    9. References of any external sources used in the completion of the project or paper.
- Submit your design document, your fully documented code, and your paper (which will include the results of the experiments) by the due dates indicated. Refer to the course calendar for all due dates.

4 Project Evaluation

Your grade will be broken down as follows.

- Design document – 30 points
- Code structure and documentation/commenting – 10 points
- Proper functioning of your code, arising from running on the held-out puzzles – 30 points
- Summary paper – 30 points