Sudoku Student Manual

For CS: Introduction to Artificial Intelligence

Contributors: Junkyu Lee, Juston Lin, Zongheng Ma, Darren Sjafrudin, Cyrus Tabatabai, Abdullah (Ed) Younis, Ryozo Masukawa

Contact: Ryozo Masukawa (rmasukaw@uci.edu)

Last update: Jan 21, 2026 by Ryozo Masukawa

Table of Contents

Introduction

In this programming assignment, you will be tasked with implementing various approaches to solving Sudoku as a Constraint Satisfaction Problem. You will have the choice of programming in C++, Java, or Python. Much of the code has already been written for you, however, you will be asked to edit one or more files from whichever shell you decide to use. Your grade will depend on your agent's performance measure. At the end of the quarter, your agent will compete against your peers' agents in a class-wide tournament.

There are a total of 5 methods that students will implement:

Variable Selection Heuristics:

- Minimum Remaining Value (MRV)
- Minimum Remaining Value with Degree heuristic as a tie-breaker (MAD)

Value Selection Heuristics:

- Least Constraining Value (LCV)

Consistency Checks:

- Forward Checking (FC)
- Norvig's Check (NOR)

*For more information on Norvig's Check, see lecture slides for chapter 6.

Extra Heuristics:

- Students are free to implement their own heuristic if they wish to, and use that heuristic to participate in the tournament.

Sudoku Game Mechanics

Monster Sudoku (or Mega Sudoku) is a puzzle that follows the rules of Sudoku and is played on a NxN grid, but allows N to be any positive integer including N > 9. The numbers that fill each square are selected from the first N positive integers. (For display purposes they often are shown as 1, 2, ..., 9, A, B, ..., Z; so that each token takes up exactly one column when printed.) To learn more about Sudoku, click on the link below:

https://www.learn-sudoku.com/what-is-sudoku.html

Monster Sudoku puzzles are described by four parameters:

- N = the length of one side of the NxN grid, also the number of distinct tokens
- P = the number of rows in each block, also the number of block columns.
- Q = the number of columns in each block, also the number of block rows.
- M = the number of values filled in from the start.

(Note: Norvig [below] uses "box" where we use "block" --- the terms are equivalent.)

Additional definition of the game is given below:

Performance Measure:

- The performance measure of your agent will be a score calculated based on number of boards your agent has completed along with an acceptable number of backtracks. Full points are awarded to your agent only if it successfully solves the entire board and if the number of backtracks is within the acceptable range.
  - Easy: +1 , +0.5 for board completion and +0.5 for acceptable backtrack
  - Intermediate: +2 , +1 for board completion and +1 for acceptable backtrack
  - Hard: +3 , +1.5 for board completion and +1.5 for acceptable backtrack
  - Expert: +4 , +2 for board completion and +2 for acceptable backtrack

Acceptable Backtrack: If the ratio of your agent's backtrack against the teacher's backtrack is within a certain range, then it is an acceptable backtrack. Points will be deducted depending on how far you are from the acceptable range. If the agent is way off the acceptable range, then the agent will only be awarded 50% of the score for that board (awarded for completion).

Problems:

- Each difficulty has a different board dimension and number of values given initially:
    - Easy: P = Q = 3, N = 9 with 7 given values
    - Intermediate: P = 3, Q = 4, N = 12 with 11 given values
    - Hard: P = Q = 4, N = 16 with 20 given values
    - Expert: P = Q = 5, N = 25 with 30 given values

Tasks to Complete

Setup Your Environment

In this section, you will find help setting up your coding environment. This project will take advantage of UCI's openlab; any other coding environment is not supported.

PLEASE MAKE SURE YOUR CODE RUNS ON OPENLAB!

Install Required Applications

To connect to openlab, you will need to use SSH. SSH stands for Secure Shell. It is a program designed to allow users to log into another computer over a network, to execute commands on that computer and to move files to and from that computer. A Mac user can use the terminal application, whereas, a Windows user will need to install PuTTY. You can download PuTTY from here. Download the MSI installer for Windows, and run the installer for PuTTY.

Connect to Openlab

Connecting to openlab is as easy as SSHing into the open lab server. If you are on Windows and using PuTTY, type “openlab.ics.uci.edu” into the Host Name box; make sure the port is 22 and the SSH flag is ticked. Click open, and login using your ICS account info. If you are using Mac, open the terminal found under Application -> Utilities. Enter ‘ssh yourICSusername@openlab.ics.uci.edu’ and login using your into ICS account.

Download the shells on Openlab

To download the shells on Openlab, you will use Git. On openlab, whether through PuTTY or terminal, execute the following git clone command:

```
Git clone address....
```

Extra Information about Openlab:

http://www.ics.uci.edu/~lab/students/#unix

https://www.ics.uci.edu/computing/linux/hosts.php

Extra Information about UNIX:

https://cgi.math.princeton.edu/compudocwiki/index.phptitle=Documentation_and_Information:Getting_started_with_Linux

Program Your AI

Once you have your environment setup, you can start to program your agent. In the ‘src’ folder of your shell you will find the source code of the project. **You are only allowed to make changes to the BTSolver class.**

Compile Your AI

This is simple. From the folder with the "Makefile", execute the command: **make**
A bin folder should have appeared with the compiled product in there. I recommend everyone to make sure they can compile their code before they start coding.

Test Your AI

To run your program after you have compiled it, navigate to the bin folder. You should find the compiled program inside. Refer to the Shell Manual Appendix for help running it. To generate large amounts of boards to use with the folder option, refer to the Board Generator. If you are using the Python Shell make sure you are using the same python version in openlab (Python 3.10.12 for Winter 2026). However, this python version is tentative and TA, please log into the openlab.ics.uci.edu and check the latest python version (https://wiki.ics.uci.edu/doku.php/instructional_support:openlab).

Write Your Project Report

Write a report according to your Professor’s instructions. Make sure your report is in pdf format and place it inside the ‘doc’ folder.

Submit Your Project

At this point you should have your most up-to-date source code in the ‘src’ folder, your report in pdf format in the ‘doc’ folder, and your compiled project in the ‘bin’ folder. Navigate to your shell’s root directory and execute the command make submission. It will ask you for some information and create a zip file inside the folder. Submit this zip file to EEE or Canvas.

Understanding the Tournament

After you submit your project and the deadline passes, your final AI be entered into a tournament with your classmates. The tournament checks to make sure you followed all the instructions correctly, then runs your agent across several hundreds boards of four different difficulty level consisting of several boards each using all the five heuristics, or any other special heuristic the AI has. Every agent is run on the same boards to ensure fairness. Your agent's total score is calculated and a scoreboard is constructed that will be made available. Your agent will be timed-out if it hangs for longer than one hour for a board. After the scoreboard is constructed, scores are checked for any illegal submissions. These include two agents with the same score.

Deadlines

For this project, you will have a total of 5 deadlines throughout the quarter, each of which build on top of each other. The deadline breakdown is as follows:

1. Team Formation
2. Minimal AI
3. Draft AI
4. Final AI (Tournament is run with this code)
5. Project Report

For Minimal AI, every agent will be tested on the same set of boards to ensure fairness respectively. Similarly for Draft/Final AI. This set will contain N Easy, N Intermediate, and N Hard boards. Should your agent exceed one hour while running on one board, you will lose points for that deadline.

Scoring Explanation

The scoring will depend on:

1) Team Formation submission

2) Coding submissions: Minimal, Draft, Final AI

3) Final Report

4) Tournament Bonus

*** For every submission, you will lose 10% for each late day after the deadline.**

1) Team Formation

The submission of Team Formation must follow strict rules:

- Team names must use only numbers and alphabetic characters. Any symbols including whitespace, is not allowed. Capital letters are allowed.
- The submission text must follow this format:

TeamName: <enter team name>

Member1: <enter member 1 name only>

Member2: <enter member 2 name only>, leave blank if none

You will lose points (up to 100%) if you don't follow the stated rules above.

2) Coding Deadlines

Minimal AI: Implement Forward Checking (FC). Your code can compile and run on Openlab. Agents that are able to successfully run FC on Openlab will receive full credit.

Draft AI: Implement Minimum Remaining Value heuristic (MRV), Least Constraining Value heuristic (LCV) and Forward Checking (FC).

Each of the methods should be able to complete 70 percent of the boards from 'Easy' with an average score of 0.8 per board, 50 percent of the boards from 'Intermediate' with an average score of 1.6 per board, and 30 percent of the boards from 'Hard' with an average score of 2.4 per board. Agents that are able to fulfill this requirement will receive full credit.

Final AI: Implement Norvig's Check (NOR), FC+MRV+LCV, FC+MAD+LCV, NOR+MRV+LCV and NOR+MAD+LCV.
Each of the methods should be able to complete 70 percent of the boards from 'Easy' with an average score of 0.8 per board, 50 percent of the boards from 'Intermediate' with an average score of 1.6 per board, and 30 percent of the boards from 'Hard' with an average score of 2.4 per board. Agents that are able to fulfill this requirement will receive full credit.

3) Final Report

If you write each section in clear, logical, technical prose, you will get 100% credit.
Again, you will lose 10% credit for each late day after deadline. A report template is given at the bottom of the manual, and a report template should already be uploaded on canvas.

Submit the report in pdf form on canvas when completed.

4) Tournament Bonus

Your AI's score will be compared to other students and ranked based on the total score obtained. The tournament bonus is between 1-10, where the top 10% will get a 10, the second 10% will get a 9, and so on. Generally, a 10 means a 10% bonus to your Final AI submission. Only students who submitted the Final AI on the deadline will be allowed to enter the tournament.

Appendix: Shell Manual

Operating

Once you have compiled your program, you can test to see that it is working. The BackTracking Solver will have some basic backtracking logic already implemented at the start. This logic should suffice to solve P = 3, Q = 3, matrices (More on P and Q below).

To run your progam just execute the binaries in the bin folder:

```
C++: bin/Sudoku
Java: java -jar bin/Sudoku.jar
Python: python3 bin/Main.pyc
```

By default, a random 3x3 matrix will be generated and displayed. The solver will attempt to solve it, and the solution will be displayed once found. If no solution is found, a text message description appears. The number of assignments and backtracks made will also be displayed; this can be used to test your heuristic implementations.

You add tokens on the command line to make your program run in different ways. For example, if you would like to use MRV to select a variable and LCV to select a value you would execute:

```
C++: bin/Sudoku MRV LCV
Java: java -jar bin/Sudoku.jar MRV LCV
Python: python3 bin/Main.pyc MRV LCV
```

The token order doesn't matter. The following tokens are valid:

MRV: Minimum Remaining Value Variable Selector

MAD: MRV and DEG tie breaker

LCV: Least Constraining Value Value Selector

FC: Forward Checking Constraint Propagation

NOR: Norvig's Sudoku Constraint Propagation

TOURN: Custom Heuristic for tournament (..)

You can also specify a path to a Sudoku file or folder containing many Sudoku files. Sudoku files are outputs of the board generator also included in the student repository (More on that below).

This is an example of something the system will execute when grading your projects for part 1:

```
C++: bin/Sudoku MRV LCV FC path/to/board/files
Java: java -jar bin/Sudoku.jar MRV LCV FC path/to/board/files
Python: python3 bin/Main.pyc MRV LCV FC path/to/board/files
```

Board files, Understanding and Generation

You will have to generate and use Sudoku Board files throughout this project. This is made easy with the Board Generator. This should be found in the shell root folder. Look for "Sudoku_Generator". In there you can execute the **make** command to simply generate a set of custom boards. You can also use the board generator by the following synopsis:

```
python3 board_generator.py <File Prefix> <# of boards> <P> <Q> <M>
```

This will generate your desired boards. The file format is very simple, so you can custom the boards easily. The file format is:

```
P Q
# # # ...
# # # ...
# # # ...
.
.
.
```

Where each # represents the value at that place on the board. If this is confusing, generate a file, and look at it.

N P Q and M

- N = the length of one side of the NxN grid, also the number of distinct tokens
- P = the number of rows in each block (Norvig's box is a synonym for block as used here)
- Q = the number of columns in each block
- M = the number of filled-in values at the start

Each block is a rectangle, P rows X Q columns. The set of blocks that align horizontally are called a block row (= a row of blocks). Similarly, the set of blocks that align vertically are called a block column (= a column of blocks).

N = P*Q, so P = N/Q and Q = N/P. Thus, there are P block columns and Q block rows. Please distinguish between rows/columns per block and block rows/block columns per grid. You can experiment by generating different board configurations with these parameters to see how they work.

M = 0 is an empty sudoku board, and M = 81 is a sudoku board with 81 values filled in. Note that higher values of M result in longer board generation times. There is no guarantee that a randomly generated board is always solvable.

Common Mistakes

Below are the common mistakes that students often make when submitting:

- Leaving 'print' or 'cout' statements in the source codes. Please delete these because it will create problem for the grader in both time and script.
- Giving additional information or text in the Team Formation submission. Please only fill the required info.
- Not testing in openlab. This seldom happens but it does: it runs in your computer environment, but it doesn't run in openlab (and openlab is where the grading will take place), so please test it for consistency.
- Only submitting the source code(s)/ Not submitting a zip file. Please use the make/ make all command provided, as it will make a zip file with your team name (<team_name>.zip).

Sudoku Final AI Report

Team name____

Member #1 (name/id)____

Member #2 (name/id)____

I. Minimal AI

I.A. Briefly describe your Minimal AI algorithm. What did you do that was fun, clever, or creative?

I.B Describe your Minimal AI algorithm's performance:

Board Size	Sample Size	Score	Worlds Complete
9x9			
12x12			
16x16			
25x25			
Total Summary			

II. Final AI

II.A. Briefly describe your Final AI algorithm, focusing mainly on the changes since Minimal AI:

II.B Describe your Final AI algorithm's performance:

Board Size	Sample Size	Score	Worlds Complete
9x9			
12x12			
16x16			
25x25			
Total Summary			

III. In about 1/4 page of text or less, provide suggestions for improving this project (*this section does* ***NOT*** *count as past of your two-page total limit.*)