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 fill in the blank functions from whichever shell you decide to use.

Please read this document before attempting to do any work.

Your grade will depend on your agent's performance measure. Optionally, 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 to 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). **Has a specific tie-breaking mechanism for this assignment; see appendix.**

Consistency Checks:

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

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

Extra Heuristics (Tournament AI):

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

Sudoku Game Mechanics

To learn more about Sudoku, click here: https://www.learn-sudoku.com/what-is-sudoku.html

We will be working with Monster Sudoku instead of the regular Sudoku. Monster Sudoku (or Mega Sudoku) is a puzzle that follows the rules of Sudoku and is played on a NxN grid, N being any positive integer including N > 9. The numbers that fill each square are selected from the first N positive integers. For display purposes, they are shown as 1, 2, ..., 9, A, B, ..., Z so that each token takes up exactly one column when printed.

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 uses "box" where we use "block" -- the terms are equivalent.

Additional information of the assignment is given below.

Performance Measure

The performance measure of your agent is based on how well it does on Gradescope's automated tests. Each function will have its own set of automated tests in Gradescope. Your grade is based on the number of tests passed. However, not all tests are visible; test your program thoroughly.

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. Think of openlab as a remote Linux computer you connect to when you are working on your program.

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. A Windows user will need to install PuTTY, while Mac/Linux users do not have to install anything and can use the terminal directly.

If you are new to Linux or the terminal, there are simpler, GUI alternatives to use the openlab and move files between openlab and your computer. Windows has WinSCP, and Mac has Cyberduck.

Connect to openlab

First, make sure you are connected to the UCI network. If you are outside UCI, you can use the VPN provided by UCI from this link: https://www.oit.uci.edu/help/vpn/?target=software-vpn.

Install the VPN and use it whenever you need to connect to openlab outside UCI. If you are not connected to the UCI network or the VPN, you cannot connect to openlab.

SSH into the openlab server to connect to openlab. 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.

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.

Keep in mind that your ICS account password is different from the one you use when logging into Canvas. If you forget your ICS account password, refer to this page for more information: https://www.ics.uci.edu/computing/account/

The same instructions are used to connect to openlab using WinSCP or Cyberduck (make sure the file protocol is SFTP, which stands for SSH File Transfer Protocol). The Host Name box should be `openlab.ics.uci.edu` , port is 22, username/password is your 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 https://gitlab.ics.uci.edu/ai-projects/Sudoku_Student.git
```

If there are updates to the project, execute `git pull` in the `Sudoku_Student` directory.

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

For helpful commands you (most likely) need when working in openlab, refer to the Useful Linux Commands section on the appendix. **If you are new to Linux, please read it first before working on openlab.**

Form your team

Form a team of 1-2 people on Gradescope. The procedure (naming rules, etc.) is described in the Team Formation section, under Deadlines.

Program your AI

Once you have your environment and team set up, 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. For WinSCP users, you can edit the file in openlab directly by opening the file using a text editor installed on your PC (right-click the file you want to edit then select the appropriate options). You can also use vim; refer to the Appendix: Helpful Tips section.

If you don't know how or where to start, also refer to the Appendix: Helpful Tips section below (especially the Dependecies section). Afterwards, read all the code provided to you in the shell, including the comments, and understand how the whole system works. For Part1 AI, MRV should be the simpler function to try tackling first.

Remember that LCV has a tie-breaking mechanism. Refer to the Appendix for the details.

There are also Coding Clinic sessions you may want to attend. These are "office hours" with the people who worked on the shell, though you cannot ask them to write the code for you.

PLEASE MAKE SURE YOUR CODE RUNS ON OPENLAB! All grading is done inside openlab, so please test your code and make sure it runs on openlab before submitting. Do not test your AI anywhere other than inside openlab, as there are no guarantees it would work inside openlab. The procedures for compiling and testing are described below.

Compile Your AI

Inside the directory with the `Makefile` file, execute the command: `make`

A `bin` folder should have appeared with the compiled product in there. It is recommended to do this before any coding to ensure that the shell works fine (the shell will compile without any student code).

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 at the end of this document 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 Python 3.5.2. On openlab, run the command `module load python/3.5.2` to load Python 3.5.2.

Write Your Project Report

Write a report according to the Professor's instructions. Make sure your report is in pdf format. You only need to write the report once, after all functions have been written. Submit by the last deadline.

Submit Your Project

Part1 AI, Part2 AI

Download your BTSolver file and submit it to the appropriate assignment on Gradescope.

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 the special heuristics 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.

Note: C++ is significantly faster than Python in most cases. Take this into account if you are planning to participate in the end-of-quarter tournament.

Deadlines

For this project, you will have a team formation deadline and three AI deadlines throughout the quarter, each of which build on top of each other. Refer to the syllabus on Canvas for the exact dates of the deadlines.

- Team Formation submission
- Part1 AI, Part2 AI
- Final Report
- Tournament Bonus / Final AI (Part2 AI)

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

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:

```
team_name
member_1_name, member_1_netid, member_1_id_number
member_2_name, member_2_netid, member_2_id_number
```

For example,

YetAnotherAITeam123
John Smith, jsmith, 12345678
Jane Doe, jdoe, 11235813

You will lose points (up to 100%) if you don't follow the stated rules above. Type this in a .txt file and submit it to Gradescope.

Three Deadlines

Part1 AI

Implement MRV, LCV, and FC.

Part2 AI

Implement MAD and NOR.

Each of the methods in Part1 and Part2 should be able to pass all provided Gradescope tests. Some tests are hidden; please test your functions thoroughly for edge cases.

Final Report

If you write each section in clear, logical, technical prose, you will get 100% credit. 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 by Part2 deadline. You only need to write the report once.

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 by 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 default, provided 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 program execute the binaries in the bin folder (assuming you are in the `Sudoku Student` directory):

- 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 compare it to this format.

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 by 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.

Appendix: Helpful Tips

LCV Tie-breaking Mechanism

To standardize answers, your implementation of LCV will use the following tie-breaking mechanism: if the list returned by LCV contains two or more values which has the same influence on the board, order them based on the value itself. For example, suppose the list of values returned is [2, 1, 7, 5, 3] with 7 and 5 affecting the same amount of variables (and no other ties). The correct list should be [2, 1, 5, 7, 3] because 5 comes before 7.
Remember this when you implement LCV as any other tie-breaking mechanism will not be accepted.

Useful Linux Commands

- `man` - `man` is essentially the help command for Linux. When Linux was made, the creators made a help page that corresponds to each command. The man command is the way to access these help pages. Ex. `man man`, `man ls`, `man cd`
- `ls` - Stands for list structures, This command will list all structures (i.e. files and folders) under the current directory if no additional arguments are passed, otherwise it will list all structures under a given directory. Ex. `ls` (lists structures under your current directory), `ls /home` (will list all of the structures under the home directory)
- `mkdir` - Stands for make directory. This command will make a directory with whatever name you choose. After executing the command you can run ls and see the new directory that has been made. Ex. `mkdir new_dir` creates a directory called `new_dir`.
- cd - Stands for change directory. Will change your directory to the given path. Examples:
  - `cd` (A cd with no argument will change you to the '~' directory. This is considered a default directory in Linux, for openlab it is usually /home/).
  - `cd .` (This will change the current directory to your current directory, since `.` stands for the current directory. This does nothing)

- `cd ..` (This will change you to the parent directory, i.e. one directory level above where you currently are. If you are currently in /home//my_dir and you run cd .. you will be put into /home/. Also keep in mind that you can string the .. together, so if you ran cd ../../ you would then end up in /home)
- `cd /home/<ucinetid>/Sudoku_Student` (This will change the current directory to `/home/<ucinetid>/Sudoku_Student`. Remember that the terminal and PuTTY has the tab autocomplete feature, so typing `Sudoku` and pressing TAB should autocomplete into `Sudoku_Student`. If there are multiple directories that start with `Sudoku`, cycle through them using TAB.)

- `rmdir` - Stands for remove directory, Important to note that this only works on empty directories. Ex. `rmdir <directory_to_be_removed_that_is_also_empty> /`
- `pwd` - Stands for print working directory. This will print out the absolute path up to the directory you are in. Ex. `pwd`
- `vim` - Stands for Vi improved. Vi is an older version of vim. Once you enter the vim interface type `i` to enter the insert mode. From there you can type to make changes to the file. Once you are done hit the Esc button to exit insert mode. Now press Shift+Z+Z to write your changes to the file. To quit vim, type `:q` (make sure you already hit Esc first). You can use other text editors to edit the file directly in openlab if you are using WinSCP/Cyberduck by opening the file using a text editor.
- `cat` - Stands for concatenate files. This command will take all of the contents of a given file and output them to your terminal screen. Ex. `cat myfile.py`
- `rm` - Stands for remove. Can be used to remove both directories and files. can also recursively destroy directories recursively with the -r flag. Usually when you run rm it will prompt you if you actually want to remove a certain file. If you do not want these prompts then use the -f flag Ex. `rm -r <directory_to_be_recursively_destroyed>`, `rm -f <file_to_be_removed>`, `rm -rf <directory_be_destroyed_no_questions_asked>`. Note: please be CAREFUL with rm -rf it can very easily destroy your whole project with the wrong input. The command is irreversible, and there is no going back once you run it.

Common Mistakes

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

- Leaving `print`, `cout`, or other debug statements in the source code. Please delete these because it will cause problems 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, so please test it for consistency.
- Non-standard LCV tie-breaking mechanism. See above for details.

Dependencies

Below is how the project is structured. Before reading the source code, please read the the paragraph below.

BTSolver -> Constraint Networks -> Constraints -> Variables -> Domain.

For the entirety of the project, you will be working in and modifying the `BTSolver` class. `BTSolver` uses the `ConstraintNetwork` class, which can be thought of as the combination of all individual constraint graphs. `ConstraintNetwork` contains a list of `Constraint` instances which is the actual individual constraint graphs. A `Constraint` contains a list of `Variable` instances which are the "nodes" in the specific constraint graph. A `Variable` contains an instance of `Domain`. A `Domain` of a `Variable` is the possible values that the current variable can take, without making the constraint inconsistent.

Please also read and understand how the `solve` function works.

Data types

Some functions in the `BTSolver` class may have you return some data types which might be unfamiliar to you. Here are some of them:

Pair

Comparable to a 2-tuple in Python, a pair consists of exactly two elements. In C++, you can use `make_pair(element1, element2)` to make a pair of `element1` and `element2`, e.g. `pair<string, bool> example_pair = make_pair("hello",true)`. To access the first and second elements inside a pair, use `first` and `second` like this: `example_pair.first` and `example_pair.second`.

http://www.cplusplus.com/reference/utility/pair/pair/ In Java, the pair is implemented as a Map.Entry object. This is not an actual map entry, but merely a 'key-value' pair. Treat the 'key' as the `first`, and the `value` as the `second`. Do not confuse this with an actual entry to a Map.

Map

Comparable to a dictionary in Python or a HashMap in Java, a Map takes some data as the key and 'maps' it to some other data as the value. You can then access the data stored as value using the key you used. Think of it like an array but you use actual data instead of numbers to access it instead. In C++, you can use maps by first initializing it (`map<string,bool> example_map;`) and access the elements inside the map (`example_map["hello"] = false`). http://www.cplusplus.com/reference/map/map/map/