Assignment 3 CSP

Due Nov 21 by 12p.m. **Points** 100

Table of Contents

Warning

We know that solutions to this or related problems may exist online. *Do not use these solutions, as this would be plagiarism.* To earn marks on this assignment, you must develop your own solutions.

Also, please consider the following important points:

- *Do NOT add any non-standard imports*. All imports already in the starter code must remain. When in doubt, post a question on Piazza.
- *Do NOT modify any provided code*. Modifying the provided code may cause you to *fail* all the tests.
- Submit your program on MarkUs and run the provided tests *well before* the deadline. The fact that your code runs on your own system but not on MarkUs is *NOT* a legitimate reason for a regrade request.
- The provided tests on MarkUs are an *extremely basic* sanity check. Passing the provided tests only means that your program runs without errors; it does *NOT* mean youwill receive full marks on the assignment.

The Kropki Sudoku Puzzle

The goal of this assignment will be to implement a program to solve the Kropki Sudoku puzzles, as shown in Figure 1.

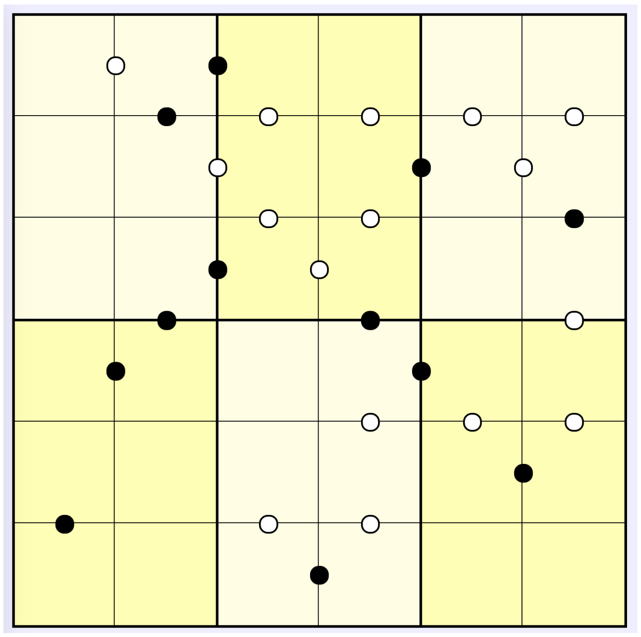

Figure 1: An example of a 6x6 Kropki Sudoku Puzzle. You can play this puzzle **here** (https://www.puzzlemix.com/playgrid.php?id=94743&type=kropki&share=1) .

A Kropki puzzle is a regular Sudoku puzzle with extra constraints. We will consider Kropki puzzles of sizes 6x6 or 9x9.

- A 6x6 puzzle has 6 subgrids. Each subgrid has 3 rows and 2 columns.
- A 9x9 puzzle has 9 subgrids. Each subgrid has 3 rows and 3 columns.

Consider an NxN Kropki puzzle. Our goal is to fill in the puzzle with numbers 1 to N such that the numbers satisfy all the constraints.

First, a Kropki puzzle must satisfy the constraints of a regular Sudoku puzzle.

- Each row must contain different numbers from 1 to N.
- Each column must contain different numbers from 1 to N.
- Each sub-grid must contain different numbers from 1 to N.

Moreover, a Kropki puzzle must satisfy additional “adjacency” constraints.

- A puzzle may have black or white circular markers between two adjacent squares.
- If there is a black circle between squares A and B, it means that the number in one of the two squares is twice the number in the other square. In other words, A=2B or B=2A.
- If there is a white circle between squares A and B, it means that the two numbers differ by one. That is A - B = 1 or B - A = 1.
- If there is no circular marker between two squares, then the two numbers do not differ by one and one is not twice the other.
- When 1 and 2 are in adjacent squares, either a white or black circle may be given (since both are true).

Your Tasks

You will implement a program to solve a Kropki Sudoku Puzzle using backtracking search and constraint propagation algorithms. Specifically, you will implement the following components.

- A CSP model for a Kropki Sudoku Puzzle. This includes creating the variables, the constraints, and the CSP.
- Two constraint propagation algorithms: Forward Checking and the AC-3 algorithm.
- The MRV heuristic for choosing the next variable.

Starter Code

To support you in completing this assignment, we have provided several files to get you started.

board.py: The Board object allows you to visualize a Kropki Sudoku puzzle. You can use the read_from_file function in csprun.py to create a board from an input file. The Board object also has a readable string representation for debugging.

cspbase.py: This file contains several classes you can use to create a CSP.

- The Variable class keeps track of the name of a variable and its domain. Read the comments carefully to understand the difference between the permanent domain and the current domain.
- The Constraint class allows you to create a constraint by adding the satisfying tuples, check whether a list of values satisfies a constraint, and get certain variables from the constraint.
- The CSP class keeps track of the variables and constraints inside the CSP.
- Finally, the BT class provides functions to perform the backtracking search.

cspmodel_starter.py

- You must implement the kropki_model function to create and return a CSP object and a list of its variables given a Board.
- We have broken this down for you into several helper functions.

csprun.py

- Implements read_from_file, which creates a board from the input file.

propagators_starter.py

- This file contains the implementation of three propagators: BT (for backtracking search), FC (for forward checking, and AC3 (for the AC-3 algorithm).
- You must implement prop_FC for forward checking and prop_AC3 for the AC-3 algorithm.
- You also need to implement ord_mrv for the MRV heuristic.

Input and Output File Format

An example of an input file is given below.

```
6
-------------
|. .|.o.*2 6|
|  o|  *|* *|
|. .o. .*. .|
|* *|   |   |
|.o.|. .|.o.|
-o-o-----o-o-
|.o.|. .|.o.|
|  o|   |o *|
|. .|.o.*. .|
|o  |*  |* *|
|. .*. .|. .|
-------------
```

The input/output file has the following format.

- The first line describes the dimension (N) of the puzzle, which can be 6 or 9.
- The remaining N lines will be a square grid representing the initial layout of the puzzle. Each square has (N+1) possible values.
    - “.” (dot) represents no value for that square.
    - A number from 1 to N represents the given value for that square.
- The following characters represent the black or white dot constraints.
    - "*" (an asterisk) represents a black circular marker.
    - "o" (the letter o) represents a white circular marker.

Below is the solution to the puzzle above.

```
6
-------------
|1 5|3o4*2 6|
|  o|  *|* *|
|4 6o5 2*1 3|
|* *|   |   |
|2o3|1 6|4o5|
-o-o-----o-o-
|3o2|6 1|5o4|
|  o|   |o *|
|5 1|4o3*6 2|
|o  |*  |* *|
|6 4*2 5|3 1|
-------------
```

Running Your Program

Once you have implemented and tested the individual functions, you can run your program to solve a Kropki Sudoku puzzle.

An example of a command is below.

```
python3 csprun.py --inputfile <input file> --outputfile <output file> --propagator <propagator> --he
uristic
```

A summary of the command line parameters is provided below.

Command Line Parameters

Parameter	Type	Required?	Description
`--inputfile`	str	required	The input puzzle file.
`--outputfile`	str	required	The output file that contains the solution to the puzzle.
`--propagator`	str	required	The propagator to be used. Choices are BT, FC and GAC. The default value is BT.
`--heuristic`	boolean	optional	If this flag is present, the solver chooses the next variable using the MRV heuristic.

Grading Scheme

We have set up three projects on MarkUs: A3-Week1, A3-Week2 and A3. The first two projects (A3-Week1 and A3-Week2) contain the extra unit tests for the optional early deadlines. They are not worth any marks.

To earn marks for this assignment, you must submit cspmodel.py and propagators.py to the A3 *project on MarkUs*.

The table below contains detailed information regarding the available tests and the grading scheme. We have also included a few important notes below the table. Please read them carefully!

MarkUs Test Availability and Grading Scheme

Category	**Test Descriptions**	**A3 - Week 1**	**A3 - Week 2**	**A3**	**Marks**
		Oct 26 - Nov 2	**Nov 2 - Nov 14**	**Nov 14 - Nov 21**	
Kropki Sudoku Model	create_initial_domain	1 public test		2 hidden tests	2
	create_variables	1 public test		2 hidden tests	2
	satisfying_tuples_difference_constraints	1 public test		2 hidden tests	2

Category	Test Descriptions	A3 - Week 1 Oct 26 - Nov 2	A3 - Week 2 Nov 2 - Nov 14	A3 Nov 14 - Nov 21	Marks
	satisfying_tuples_white_dots	1 public test		2 hidden tests	2
	satisfying_tuples_black_dots	1 public test		2 hidden tests	2
	binary difference constraints			8 hidden tests	15
	the full CSP model			6 hidden tests	15
Forward Checking			1 public test	1 public test + 9 hidden tests	25
AC-3			1 public test	1 public test + 9 hidden tests	25
MRV heuristic				2 hidden tests	10

The Optional Early Feedback Deadlines

To encourage you to start working on the assignments early and tackle them incrementally, we have introduced *two optional early feedback deadlines* for each assignment. During the three weeks, we will provide some extra unit tests during the first two weeks. If you submit during either week, you can get additional feedback from these extra unit tests. Note that these extra unit tests will be *exclusively* available during their respective weeks and will no longer be available in the third week. These optional early feedback deadlines are not worth any marks.

We created these optional early feedback deadlines for the following reasons.

- *Encourage you to start the assignments early*: If you start early, you can take advantage of these extra unit tests, which won't be available later on.
- *Help you tackle the assignment incrementally*: We break down the assignment into smaller tasks so that the task for each week is more manageable.
- *Provide extra support*: The extra unit tests make it easier to debug and test your program by identifying problems in your program.
- *Avoid adding extra stress*: The early deadlines are not worth any marks. Therefore, if you are busy with other obligations and cannot take advantage of the early deadlines, you are not missing out on any marks, and you can still potentially get full marks on the assignments.

We have set up three projects on MarkUs: A3-Week1, A3-Week2 and A3. The first two projects (A3-Week1 and A3-Week2) contain the extra unit tests for the optional early deadlines. Please check the table above for the public tests available. Remember that the public tests in A3-Week1 and A3-Week2 are different from those in the A3 project on MarkUs.