

Computer and Information Science 352  
Main Campus

January 24, 2022  
Queen's University

Homework Assignment #1: Constraint Satisfaction  
**Due: February 13, 2022 by 11:59 PM**

**Late Policy:** 25% per day after the use of 3 grace days.

**Total Marks:** This assignment represents 12% of the course grade.

**Handing in this Assignment**

You must submit your assignment electronically. Download the assignment files from the A1 assignment on onQ. Modify `propagators.py`, `heuristics.py`, and `cagey_csp.py` appropriately so that they solve the problems specified in this document. **Submit your modified `propagators.py`, `heuristics.py`, and `cagey_csp.py` files, and only those files, in a single .zip.**

How to submit: Submissions should include just the three files above in a zip file, and should be uploaded once for your group. It is your responsibility to include all necessary files in your submission. For the purposes of determining the lateness penalty, the submission time is considered to be the time of your latest submission.

**Extra Information**

Clarification Page: Important corrections (hopefully few or none) and clarifications to the assignment will be posted on the Assignment 1 description in onQ. You are responsible for monitoring the announcements in onQ for any updates.

Questions: Questions about the assignment should be asked on during office hours. If you have a question of a personal nature, please email the A1 TA, Mackenzie Sharp, at [mcs4@queensu.ca](mailto:mcs4@queensu.ca) with **CISC352: A1** in the subject line of your message.

---

**Last Updated:** January 31, 2022

## Corrections Made

- (Jan 31) Changed the description of “list of lists” to just a “list” of Variables returned by the CSP model functions.

## Evaluation Details

We will test your code electronically. **You will be supplied with a testing script that will run a subset of the tests (tests.py).** If your code fails all of the tests performed by the script (using Python version 3.5.2 or higher), you will receive zero marks. **It's up to you to create test cases to further test your code—that's part of the assignment!**

When your code is submitted, **we will run a more extensive set of tests.** You have to pass all of these more elaborate tests to obtain full marks on the assignment.

**We will set a timeout of 60 seconds per board during marking.** This 60 second time limit includes the time to create a model and find a solution. Solutions that time-out are considered incorrect. Ensure that your implementations perform well under this limit, and be aware that your computer may be quicker than our cluster (where we will be testing). We will not conduct end to end tests on boards with constraints whose domain is larger than six.

Your code will **not be** evaluated for partial correctness on a particular test: it either works or it doesn't. It is your responsibility to hand in something that passes at least some of the tests in the provided testing script.

- *Make certain that your code runs using Python3 (version 3.5.2 or above) using only standard imports..* Your code will be tested using this version and you will receive zero marks if it does not run using this version.
- *Do not add any non-standard imports from within the Python file you submit (the imports that are already in the template files must remain).* Once again, non-standard imports will cause your code to fail the testing and you will receive zero marks.
- *Do not change the supplied starter code.* Your code will be tested using the original starter code, and if it relies on changes you made to the starter code, you will receive zero marks.
- All existing submissions will be tested by the official deadline. Any instance of a non-functional submission will be reported to its respective team, giving them the 3-day grace period to remedy the situation.

## Introduction

There are two parts to this assignment.

1. **Propagators.** You will implement two constraint propagators—a Forward Checking constraint propagator, and a Generalized Arc Consistency (GAC) constraint propagator—and two heuristics—Minimum-Remaining-Value (MRV) and Degree (DH).
2. **Models.** You will implement three different CSP models: two grid-only Cagey models, and one full Cagey puzzle model (adding *cage* constraints to grid).

## What is supplied

- **cspbase.py.** Class definitions for the python objects Constraint, Variable, and BT.
- **propagators.py.** Starter code for the implementation of your two propagators. You will modify this file with the addition of two new procedures `prop_FC` and `prop_GAC`.

- **heuristics.py**. Starter code for the implementation of the variable ordering heuristics, MRV and DH. You will modify this file with the addition of the new procedures `ord_mrv` and `ord_dh`.
- **cagey\_csp.py**. Starter code for the CSP models. You will modify three procedures in this file: `binary_ne_grid`, `nary_ad_grid`, and `cagey_csp_model`.
- **tests.py**. Sample test cases. Run the tests with “python3 tests.py”.
- **csp\_sample\_run.py**. Example CSP problems to demonstrate usage of the API.

## Extended Cagey Formal Description

The Cagey puzzle<sup>1</sup> has the following formal description:

- Cagey consists of an  $n \times n$  grid where each cell of the grid can be assigned a number 1 to  $n$ . No digit appears more than once in any row or column. Grids range in size from  $3 \times 3$  to  $9 \times 9$ .
- Cagey grids are divided into heavily outlined groups of cells called *cages*. These *cages* come with a *target* and an *operation*. The numbers in the cells of each *cage* must produce the *target* value when combined using the *operation*. In this extended variant of the Cagey Puzzle, the operation is not given, and must be determined.
- For any given *cage*, the *operation* is one of addition, subtraction, multiplication or division. Values in a *cage* can be combined in any order: the first number in a *cage* may be used to divide the second, for example, or vice versa. Note that the four operators are “left associative” e.g.,  $16/4/4$  is interpreted as  $(16/4)/4 = 1$  rather than  $16/(4/4) = 16$ .
- A puzzle is *solved* if all empty cells are filled in with an integer from 1 to  $n$  and all above constraints are satisfied.
- For this assignment we are focusing on an extended version of the Cagey Puzzle where the operations in a given cell are not explicitly given, and must be determined as part of the puzzle.
- An example of a  $6 \times 6$  grid following the extended ruleset is shown in Figure 1. Note that your solution will be tested on  $n \times n$  grids where  $n$  can be from 3 to 9.

---

<sup>1</sup>Some might recognize it as <http://thinkmath.edc.org/resource/introducing-kenken-puzzles>

5	3		30	16	
216		1		7	
		3			4
3	1	11	11		
				108	
20		6			

5	3 <sup>×</sup>		30 <sup>×</sup>	16 <sup>×</sup>	
5	1	3	6	2	4
216 <sup>×</sup>		1	5	7 <sup>×</sup>	
3	6	1		4	2
6	2	3 <sup>×</sup>	4	1	3 <sup>×</sup>
3 <sup>×</sup>	2	1 <sup>×</sup>	11 <sup>+</sup>	11 <sup>+</sup>	4 <sup>×</sup>
2	3	6	4	5	1
1	4	5	2	108 <sup>×</sup>	3
20 <sup>×</sup>	4	5	6 <sup>×</sup>	2	3
4	5	2	3	1	6

Figure 1: An example of a  $6 \times 6$  Cagey grid with its start state (left) and solution (right).

## Question 1: Propagators (worth 7/12 marks)

You will implement Python functions to realize two constraint propagators—a Forward Checking (FC) constraint propagator and a Generalized Arc Consistency (GAC) constraint propagator. These propagators are briefly described below. The files `cspbase.py`, `propagators.py`, and `heuristics.py` provide the **complete input/output specification** of the two functions you are to implement.

**Brief implementation description:** The propagator functions take as input a CSP object `csp` and (optionally) a Variable `newVar` representing a newly instantiated Variable, and return a tuple of (`bool`, `list`) where `bool` is `False` if and only if a dead-end is found, and `list` is a list of (`Variable`, `value`) tuples that have been pruned by the propagator. `ord_mrv` and `ord_dh` take a CSP object `csp` as input, and return a Variable object `var`. In all cases, the CSP object is used to access variables and constraints of the problem, via methods found in `cspbase.py`.

### You must implement:

`prop_FC` (worth 2/12 marks)

A propagator function that propagates according to the FC algorithm that check constraints that have *exactly one uninstantiated variable in their scope*, and prune appropriately. If `newVar` is `None`, forward check all constraints. Otherwise only check constraints containing `newVar`.

`prop_GAC` (worth 3/12 marks)

A propagator function that propagates according to the GAC algorithm, as covered in lecture. If `newVar` is `None`, run GAC on all constraints. Otherwise, only check constraints containing `newVar`.

`ord_mrv` (worth 1/12 marks)

A variable ordering heuristic that chooses the next variable to be assigned according to the Minimum-Remaining-Value (MRV) heuristic. `ord_mrv` returns the variable with the most constrained current domain (i.e., the variable with the fewest legal values remaining).

`ord_dh` (worth 1/12 marks)

A variable ordering heuristic that chooses the next variable to be assigned according to the Degree heuristic (DH). `ord_dh` returns the variable that is involved in the largest number of constraints involving other unassigned variables.

## Question 2: Models (worth 4/12 marks)

You will implement three different CSP models using three different constraint types. The three different constraint types are (1) binary not-equal; (2)  $n$ -ary all-different; and (3) *cage*. The three models are (a) binary grid-only Cagey; (b)  $n$ -ary grid-only Cagey; and (c) full Cagey. The CSP models you will build are described below. The file `cagey_csp.py` provides the **complete input/output specification**.

Brief implementation description: The three models take as input a valid Cagey grid, which is a formatted list of elements, where the first element  $N$ , gives the size of each dimension of the board. The second element is a list defining all *cages*. Each element in the *cages* list is a tuple representation of the grid *cages*. In this tuple, the first element is an integer which defines the *expected value* (natural number) of the *cage*. The second element in this tuple is a list containing indicies for all grid-cells enclosed by the *cage*. These indices are formatted as coordinate tuples  $(m,n)$ , where  $m$  represents the row ( $1 \dots N$ ), and  $n$  represents the column ( $1 \dots N$ ).

The third and final element of this tuple is an operation identifier character. For a given cage, the enclosed values must be able to be arranged in some order such that the corresponding mathematical operation produces the *expected value* when applied.

Possible characters are:

- '+' - Addition
- '-' - Subtraction
- '\*' - Multiplication
- '/' - Division
- '?' - Unknown

The *unknown* operator can refer to any of the other operations, which can produce the expected values given the right grid-valuations.

For example, the model  $(3, [(6, [(1, 1), (1, 2), (1, 3)], '+'), (2, [(2, 1), (2, 2), (2, 3)], '/'), \dots])$  corresponds to a 3x3 board<sup>2</sup> where

1. cells (1,1), (1,2) and (1,3) must sum to 6, and
2. the result of dividing some permutation of cells (2,1), (2,2), and (3,1) must be 2. That is,  $(C_{21}/C_{22})/C_{23} = 2$  or  $(C_{21}/C_{23})/C_{22} = 2$ , or  $(C_{22}/C_{21})/C_{23} = 2$ , etc...

All models need to return a CSP object, and a list of Variable objects representing the board (i.e., the cell values and operand for a cage). The returned list is used to access the solution. The grid-only models do not need to encode the *cage* constraints.

---

<sup>2</sup>Note that cell indexing starts from 1, e.g. 1,1 is the cell in the upper left corner.

**You must implement:**

`binary_ne_grid` (worth 1/12 marks)

A model of a Cagey grid (without *cage* constraints) built using only binary not-equal constraints for both the row and column constraints.

`nary_ad_grid` (worth 1/12 marks)

A model of a Cagey grid (without *cage* constraints) built using only *n*-ary all-different constraints for both the row and column constraints.

`cagey_csp_model` (worth 2/12 marks)

A model built using your choice of (1) binary binary not-equal, or (2) *n*-ary all-different constraints for the grid, together with (3) *cage* constraints. That is, you will choose one of the previous two grid models and expand it to include *cage* constraints.

**Notes:** The CSP models you will construct can be space expensive, especially for constraints over many variables, (e.g., for *cage* constraints and those contained in the first `binary_ne_grid` grid CSP model). Also be mindful of the **time** complexity of your methods for identifying satisfying tuples, especially when coding the `cagey_csp_model`.

**Code Quality** (worth 1/12 marks)

The overall code quality (readability, proper use of files, etc) will be worth one mark. Improper use of external libraries will result in far more than just 1 grade being deducted – you shouldn't need anything beyond standard Python libraries, and you can ask on the Forum if you'd like to use anything else.

HAVE FUN and GOOD LUCK!