Computer Science 384 October 11, 2016
St. George Campus University of Toronto

Homework Assignment #2: Constraint Satisfaction

Due: October 25, 2016 by 11:59 PM

Last Updated: October 12

Silent Policy: *A silent policy will take effect 24 hours before this assignment is due, i.e. no question about this assignment will be answered, whether it is asked on the discussion board, via email or in person.*

Late Policy: 10% per day after the use of 3 grace days.

Total Marks: This part of the assignment represents 10% of the course grade.

Handing in this Assignment

What to hand in on paper: Nothing.

What to hand in electronically: You must submit your assignment electronically. Download `orderings.py` and `hitori_csp.py` and other related files from:

`http://www.teach.cs.utoronto.ca/~csc384h/fall/Assignments/A2`.

Modify `orderings.py` and `hitori_csp.py` so that they solve the problems specified in this document. **Submit your modified files** `orderings.py` **and** `hitori_csp.py`**.** using MarkUs. Your login to MarkUs is your teach.cs username and password. You can submit a new version of any file at any time, though the lateness penalty applies if you submit after the deadline and after grace days have expired.

If you submit after your grace days have expired, please submit your assignment files to csc384f16-late at cs.toronto.edu with the subject heading '[CSC384] Late Assignment Submission'. For the purposes of determining the lateness penalty, the submission time will be considered to be the time of your latest submission.

We will test your code electronically. You will be supplied with a testing scirpt that will run a **subset** of the tests. IF your code fails on all of the tests performed by the script (using Python version 3.5.2), you will receive a failing grade on the assignment.

When your code is submitted, we will run a more extensive set of tests which will include the tests run in the provided testing script and a number of other tests. You have to pass all of these more elaborate tests to obtain full marks on the assignment.

Your code will not be evaluated for partial correctness; 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 on teach.cs using python3 (version 3.5.2) using only standard imports.* This version is installed as "python3" on teach.cs. 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.

Clarification Page: Important corrections (hopefully few or none) and clarifications to the assignment will be posted on the Assignment 2 Clarification page, linked from the CSC384 A2 web page, also found at: `http://www.teach.cs.toronto.edu/~csc384h/fall/Assignments/A2/a2_faq.html`.

You are responsible for monitoring the A2 Clarification page.

Help Sessions: There will be help sessions for this assignment. Dates and times for these sessions will be posted to the course website and to Piazza ASAP.

Questions: Questions about the assignment should be asked on Piazza:
`https://piazza.com/utoronto.ca/fall2016/csc384/home`.

If you have a question of a personal nature, please email the A2 TA, `jdrummond` at `cs` dot `toronto` dot `edu`, or the instructor, Sonya Allin, at `sonyaa` at `teach` dot `cs` dot `utoronto` dot `ca` placing 384 and A2 in the subject line of your message.

Introduction

There are three parts to this assignment

1. The encoding of two different CSP models to solve the logic puzzle, Hitori, as described below. In one model you will use only binary not-equal constraints, while in the other model you will use *n*-ary constraints.
2. The implementation of two variable ordering heuristics, and one value ordering heuristic – Minimum Remaining Values (MRV), Degree Hueristic (DH), and Least Constraining Value (LCV);
3. The implementation of a variable or value ordering heuristic of your design, which can be a combination of MRV, DH, or LCV.

What is supplied:

- **cspbase.py** – class definitions for the python objects Constraint, Variable, and BT.
- **propagators.py** – class containing a backtracking propagator.
- **orderings.py** – starter code for the implementation of two variable ordering heuristics (MRV and DH), one value ordering heuristic (LCV), and the ordering heuristic of your design. You will modify this file with the addition of the new procedures `ord_mrv`, `ord_dh`, and `ord_lcv`, to realize MRV, DH, and LCV, respectively. You will also add the procedure `ord_custom` to realize your ordering heuristic.
- **hitori_csp.py** – starter code for the two Hitori CSP models.
- **Sample test cases** for testing your code. Test cases for Q1 and Q2 will be released on the A2 web page on **Wednesday, October 12**.

Hitori Formal Description

The Hitori puzzle[1] has the following formal description:

[1] `https://en.wikipedia.org/wiki/Hitori`

- The puzzle is played on a grid *board* with dimensions n rows and n columns. The board is always a square.
- The board has n^2 spaces on it, where each space may take a value between 1 and n inclusive, or black ■ (denoted a '0' in the code). You will use a list of lists in order to represent assigned values to the variables on this grid.
- The start state of the puzzle has all spaces filled in with numbers, within the range 1 to n inclusive. Some numbers may be repeated in individual columns or rows.
- A puzzle is *solved* if:
  - Every space on the board is given one value between 1 and n inclusive, or black ■.
  - No two spaces on the board that are either vertically or horizontally adjacent both contain black squares ■.
  - No row contains more than one of the same number.
  - No column contains more than one of the same number.

Question 1: Variable and Value Ordering Heuristics **(worth 50/100 marks)**

You will implement python functions to realize two defined variable ordering heuristics – MRV and DH, and one value ordering heuristic – LCV. These heuristics are briefly described below. You will also implement one ordering heuristic of your own design, which may be a combination of MRV, DH and LCV. The file `orderings.py` provides the **complete input/output specification** of the functions you are to implement.

Implementations of MRV, DH, and your heuristic will all be graded on a 10 point scale, while LCV will be graded on a 20 point scale.

Brief implementation description: A `Variable Ordering Function` takes as input a CSP object `csp`, and returns a Variable object `var`. The CSP object is used to access variables and constraints of the problem, via methods found in `cspbase.py`. A `Value Ordering Function` takes as input a CSP object `csp` and a Variable object `var`, and returns a `list` of Value objects `[Val, Val, Val, ...]`. You must implement:

`ord_mrv` **(10 points)** A variable ordering heuristic that chooses the next variable to be assigned in backtracking search, according to the minimum remaining values (MRV) heuristic. MRV returns the variable with the most constrained current domain (i.e., the variable with the fewest legal values).

`ord_dh` **(10 points)** A variable ordering heuristic that returns the next variable to be assigned in backtracking search, according to the degree heuristic (DH). DH returns the variable that imposes the most constraints on remaining unassigned variables.

`val_lcv` **(20 points)** A value ordering heuristic that, given a variable `var`, returns a list of current domain values in order from best to worst, evaluated according to the Least Constraining Value (LCV) heuristic. (In other words, the list will go from least constraining value in the 0th index, to most constraining value in the $j-1$th index, if the variable has j current domain values.) The best value, according to LCV, is the one that rules out the fewest domain values in other variables that share at least one constraint with `var`. More formally, let V be the set of variables that share a constraint with `var`. Then, the best value is:

$$\text{argmax}_{\texttt{val}\in\texttt{curr\_domain(var)}} \min_{\texttt{v}\in V} \text{len}(\texttt{curr\_domain(v)}|\texttt{var} == \texttt{val})$$

`ord_custom` **(10 points)** A variable ordering heuristic that returns the next variable to be assigned in backtracking search, according to a heuristic of your own design. This heuristic can be a combination of MRV, DH and/or LCV.

Question 2: Hitori Models (worth 50/100 marks)

You will implement two different CSP encodings to solve the logic puzzle, Hitori. In one model you will use only binary not-equal constraints, while in the other model you will use 9-ary constraints (that are similar to all-different constraints). These CSP models are described below. The file `hitori_csp.py` also provides the **complete input/output specification** for the two CSP encodings you are to implement.

The correct implementation of each encoding is worth 25/100 marks.

Brief implementation description: A `Hitori Model` takes as input a Hitori board (represented as a list of lists), and returns a `CSP object`, consisting of a variable corresponding to each cell of the board. The variable domain of any given cell is $\{■, i\}$, where ■ is denoted as 0 in the code, and i is the value the initial puzzle contains in that cell. You must implement:

`hitori_csp_model_1` A model built using only binary not equal constraints, capturing the row/column rule for repeated numbers and no orthogonally adjacent ■ rule.

`hitori_csp_model_2` A model built using n-ary constraints to capture both the repeated numbers rule and ■ rule.

Caveat: The Hitori CSP models you will construct can be space expensive, especially for constraints over many variables, (e.g., those contained in the second Hitori CSP model). *HINT: Also be mindful of the **time** complexity of your methods for identifying satisfying tuples, especially when coding the second Hitori CSP model.*

HAVE FUN and GOOD LUCK!