CSC384 Assignment 3: CSP (Due Jul 23, 2018)

1 Introduction

In this project, you will implement some new constraints and backtracking search algorithms.

Note that this code base is unrelated to the Berkeley pacman code base. So you will not need any of the files from A1 nor A2. (We know Pacman is loved by many of you. It will make a glorious comeback in A4.)

The code for this project contains the following files, available as a zip archive.

Files you'll edit:

`backtracking.py`	Where all of the code related to backtracking search is located. You will implement forward checking and gac search in this file.
`csp_problems.py`	Where all of the code related implementing different CSP problems is located. You will implement a new version of the nQueens CSP and a CSP to solve the sport tournament scheduling problem in this file.
`constraints.py`	Where all of the code related implementing various constraints is located. You will implement a new summation constraint in this file.

Files you can ignore:

`csp.py`	File containing the definitions of Variables, Constraints, and CSP classes.
`util.py`	Some basic utility functions.
`nqueens.py`	Solve nQueens problems.
`sudoku.py`	Solve sudoku problems.
`autograder.py`	Program for evaluating your solutions. As always your solution might also be evaluated with additional tests besides those performed by the autograder.

Files to Edit and Submit: You will fill in portions of `backtraking.py`, `csp.py`, and `csp_problems.py` during the assignment. You may also add other functions and code to these file so as to create a modular implementation. You will submit these file with your modifications. Please *do not change* the other files in this distribution.

Evaluation: Your code will be autograded for technical correctness. The tests in `autograder.py` will be run as well as some additional tests. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder.

Getting Help: You are not alone! If you find yourself stuck on something, contact us for help. The piazza discussion forum will be monitored and questions answered, and you can also ask questions about the assignment during office hours. If you can't make our office hours, let us know and we will arrange a different appointment. We want the assignment to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

Piazza Discussion: Please be careful not to post spoilers.

What to Submit

You will be using MarkUs to submit your assignment. MarkUs accounts for the course will be soon be set up. You will submit four files:

Your modified `backtracking.py`

Your modified `csp_problems.py`

Your modified `constraints.py`

A signed copy of the following acknowledgment

Note: In the various parts below we ask a number of questions. You do not have to hand in answers to these questions, rather these questions are designed to help you understand what is going on with search.

Autograder

`autograder.py` is not the same as the Berkeley autograder. You can only run the command

```
python autograder.py -q qn
```

where `qn` is one of `q1, q2, q3, q4, or q5`. Or you can run the grader on all questions together with the command

```
python autograder.py
```

Question 1 (4 points): Implementing a Table Constraint

`backtracking.py` already contains an implementation of BT (plain backtracking search) while `csp_problems.py` contains an implementation of the nQueens problem. Try running

```
python nqueens.py 8
```

to solve the 8 queens problem using BT. If you run

```
python nqueens.py -c 8
```

the program will find all solutions to the 8-Queens problem. Try

```
python nqueens.py --help
```

to see the other arguments you can use. (However, you haven't implemented FC nor GAC yet, so you can't use these algorithms yet.) Try some different small numbers with the '-c' option, to see how the number of solutions grows with the number of Queens. Also observe that even numbered queens are generally faster to solve, and the time to find a single solution for 'BT' grows quite quickly. Observe the number of nodes explored. Later once you have FC and GAC implemented you will see that they explore fewer nodes.

For this question look at `constraint.py`. There you will find the class `QueensTableConstraint` that you have to implement for this question. This class creates a table constraint to capture the nQueens constraint. Once you have that implemented you can run

```
python nqueens.py -t 8
```

to solve the nQueens CSP using your table constraint implementation. Check a number of sizes and '-c' options: you should get the same solutions returned irrespective of whether or not you use '-t'. That is, your table constraint should yield the same behavior as the original `QueensConstraint`.

Question 2 (5 points): Forward Checking

In `backtracking.py` you will find the unfinished function `FC`. You have to complete this function. Note that the essential subroutine `FCCheck` has already been implemented for you. Note that your implementation must deal correctly with finding one or all solutions. Check how this is done in the already implemented `BT` algorithm ... just be sure that you restore all pruned values even if `FC` is terminating after one solution.

After implementing `FC` you will be able to run

```
python nqueens.py -a FC 8
```

to solve 8-Queens with forward checking. Solve some different sizes and check how the number of nodes explored differs from when `BT` is used.

Also try solving sudoku using the command

```
python sudoku.py 1
```

Which will solve board #1 using Forward Checking. Try other boards (1 to 7). Also try

```
python sudoku.py -a 'BT' 1
```

to see how `BT` performs compared to `FC`. Finally try

```
python sudoku.py -a 'FC' -c 1
```

To find all solutions using `FC`. Check if any of the boards 1-7 have more than one solution.

Question 3 (7 points): GAC Enforce and GAC

In `backtracking.py` you will find unfinished `GacEnforce` and `GAC` routines. Complete these functions.

After finishing these routines you will be able to run

```
python nqueens.py -a GAC 8
```

Try different numbers of Queens and see how the number of nodes explored differs from when you run `FC`.

Does `GAC` also take less time than `FC` on `sudoku`? What about on nqueens?

Now try running

```
python sudoku.py -e 1
```

which will not do any backtracking search, it will only run `GacEnforce`.

Try running only `GacEnforce` on each board to see which ones are solved by only doing `GacEnforce`.

Question 4 (2 points): AllDiff for Sudoku

In `csp_problems.py` you will find the function `sudokuCSP`. This function takes a model parameter that is either `'neq'` or `'alldiff'`. When `model == 'neq'` the returned CSP contains many binary not-equals constraints. But when `model == 'alldiff'` the model should contain 27 `allDifferent` constraints.

Complete the implementation of `sudokuCSP` so it properly handles the case when `model == 'alldiff'` using `allDifferent` constraints instead of binary not-equals.

Note that this question is very easy as you can use the class `AllDiffConstraint(Constraint)` that is already implemented in `constraints.py`. However, you must successfully complete Question 3 to get any marks on this question.

Question 5 (4 points): NValues Constraint

The `NValues` Constraint is a constraint over a set of variables that places a lower and an upper bound on the number of those variables taking on a specified value. In `constraints.py` you will find an incomplete implementation of class `NValuesConstraint`. In particular, the function `hasSupport` has not yet been implemented. Complete this implementation.

Working successfully in a pair

You may work in pairs for this assignment.

If you are working with a partner, make sure that you are actually working together. Your goal should be for the two of you to help each other learn the material and to avoid getting stuck with frustrating errors. If you split up the assignment and work separately, you are not getting practice on all aspects of the assignment.

Sometimes a student who is working with a partner drops the course or becomes ill in the middle of an assignment. If this happens, the other partner is still responsible for completing the assignment on time. If he or she has been actively engaged in the entire assignment, this should not be a problem; the assignments are designed so that an individual student can complete them. However, if the remaining partner has not been actively involved or does not have copies of all of the work, they will have serious difficulty completing the assignment. Make sure you don't find yourself in this situation: Be active in all parts of the assignment, and make sure that at the end of each meeting, both partners have a copy of all of the work.