COMPSCI 270	**Instructor:** Kris Hauser
Homework 3	**Due:** 03/06/2017

1 Instructions

Read this assignment carefully, complete the programming assignment and answer all of the written questions. Place all of your code in the `hw3.py` file and your written answers in the `hw3_answers.py` file in the appropriate locations. Submit both files on Sakai.

1.1 Homework Policies

Homework and lab assignments **must be completed individually**. Students are permitted and even encouraged to discuss assignments. However, any attempt to duplicate work that is not your own – for example, in the form of detailed written notes, copied code, or seeking answers from online sources – is strictly prohibited and will be considered cheating.

Homework assignments are due by the end of class on the due date unless otherwise specified. **No extensions will be granted** except for extenuating circumstances. Extensions are granted at the instructor's discretion, and valid extenuating circumstances include, for example, a debilitating illness (with STINF), death in the family, or travel for varsity athletics. Extensions will not be granted for personal or conference travel, job interviews, or a heavy course load.

2 Overview

This assignment combines constraint satisfaction problems and the basics of probabilistic inference. Running "python hw3.py" will run the functions `p1`() through `p4`() corresponding to each of the problems in this assignment. Press enter to cycle through the problems. Change only enough framework code to make your program run.

3 Problems

3.1 Question 1

Programming You will build a CSP for the Street Puzzle covered in class. Read the documentation of the CSP framework at the top of `hw3.py`. You may wish to use pydoc to help you understand the CSP API a bit better.

The current implementation only constructs a few of the constraints of the Street Puzzle CSP. Finish this construction by adding the remaining constraints shown on slide 17 of Lecture 8. (Keep in mind that you need to make sure all of the variables in a particular class take on different values, e.g., N1 != N3)

To test your construction, we provide the `solution` variable which is a satisfying assignment, and `invalid1` and `invalid2` are two invalid assignments. The printouts will test whether these satisfy all of the constraints, and should read `True`, `False`, `False`.

Written

3.1.1 There are typically two methods for making sure all of the variables in a particular class take on different values: either add an inequality constraint Xi!=Xj for all pairs of related variables (Xi,Xj), or add a single AllDiff constraint that returns True only when all of its arguments are different. Give at least one benefit and one drawback of each approach. Which implementation did you choose in your CSP, and why?

3.1.2 `p1`() will print out the number of constraints involving a variable or sets of variables using the `incident`() method. Let the "puzzle constraints" be defined as the constraints related to statements listed in the puzzle, and the "all-diff constraints" be defined as the constraints thateach variable is different. How many puzzle constraints and all-diff constraints involve N1? A1? D1? Both N1 and D1 simultaneously? Both N1 and D3 simultaneously? Both N1 and N2 simultaneously? All of N1, N2, and N3, simultaneously?

3.2 Question 2

Programming The n-Queens problem is formulated for you, and a plain backtracking algorithm is given in `CSPBacktrackingSolver` when `doForwardChecking`=`False` is provided in the constructor. This works fine for the 8-queens problem, but takes a minute or two when n grows to around 20, and is intractable for n=50.

1. Forward checking is not yet implemented. Implement it in the `CSPBacktrackingSolver.forwardChecking` method. Most of the bookkeeping is done for you, and you will need to implement the constraint testing.
2. No variable ordering heuristics are implemented. Implement the most-constrained-variable and most-constraining-variable heuristics in the `CSPBacktrackingSolver.pickVariable` method.
3. **Bonus**. Implement the least-constraining-value heuristic in `orderValues` and AC3 in `constraintPropagation`.

Written

3.2.1 How many fewer states does the search examine with forward checking for n=4, 8, 12, and 16? How much less time does search take? (You will need to add state counting and timing functionality.)

3.2.2 How many fewer nodes does the search examine with your variable ordering heuristics for n=4, 8, 20, and 50? How much less time does search take?

3.2.3 (optional, for Bonus above) Same question as b.

3.2.4 How high can you make n before your best method seems to run too long (more than a couple of minutes or so)?

3.3 Question 3

Written

3.3.1 Given the following joint probability table:

A, B	$P(A, B)$
F,F	0.5
F,T	0.3
T,F	0.1
T,T	0.1

Show the symbolic steps needed to compute $P(A)$ through marginalizing B out of $P(A, B)$. Enter the values from the table into your equation to compute the value of $P(A)$.

3.3.2 Given the following joint probability table:

A, B, C	$P(A, B, C)$
F,F,F	0.2
F,F,T	0.3
F,T,F	0.06
F,T,T	0.24
T,F,F	0.02
T,F,T	0.08
T,T,F	0.06
T,T,T	0.04

Show the symbolic steps needed to compute $P(A, B)$ through marginalization (Careful: this is a table, not a single value). Confirm, by entering the values in $P(A, B, C)$, that this table is equal to the table given in part a. Show the symbolic steps needed to compute $P(A)$ through marginalizing B and C out of $P(A, B, C)$. Confirm that the value of $P(A)$ calculated in this way is equivalent to the value you achieved in 3.3.1.

3.3.3 Show the symbolic steps needed to compute $P(A, B|C = T)$ from $P(A, B, C)$ through conditioning (again this is a table). Enter the values from the table in 3.3.2 to calculate it.

3.3.4 Show the symbolic steps needed to compute $P(A|B = T, C = T)$ from $P(A, B, C)$ through conditioning. Enter the values from the table in 3.3.2 to calculate it.

Show the symbolic steps needed to compute $P(A|B = T, C = T)$ from the table $P(A, B|C = T)$ through conditioning. Enter the values from the table you calculated in 3.3.3 to calculate it

3.4 Question 4

Programming For this question, a probability table P(A,B,C) is given by a Python dictionary mapping tuples to probability values. For example, a table `Ptable` may encode a probability distribution via `Ptable[(a,b,c)]` $= P(A = a, B = b, C = c)$. Probabilities should sum to one. You will implement the basic marginalization and conditioning operations on this representation.

(Note: we'll be dealing with 1, 2, and 3-tuples in this assignment, so you can specialize your code if you'd like, but there are more elegant solutions for general n-tuples)

- Implement the `marginalize` function. This takes a probability distribution and the index into the tuple over which the distribution is marginalized, and marginalizes over that index. Specifically, the return value is a probability distribution over the variables remaining after removing the one specified by the index.
- Implement the `condition1` function to compute a conditional distribution, given an index and value of the conditioned variable. This variant should compute the denominator by marginalization.
- Implement the `condition2` function to compute a conditional distribution, given an index and value of the conditioned variable. This variant should not compute a denominator, and instead should just call `normalize`() on the resulting distribution (this function is provided for you). Confirm that this result is the same as `condition1`. Which variant is faster?
- Confirm that the printouts in `p4`() match your answers in Question 3