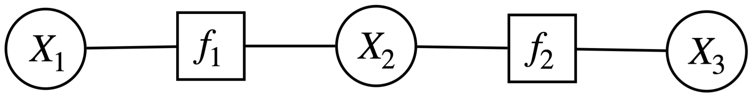
CSC 665: Artificial Intelligence

Homework 3

By turning in this assignment, I agree to abide by SFSU's academic integrity code and declare that all of my solutions are my own work.

1 Small CSP

Consider a small CSP with 3 variables and 2 binary factors, where each variable is binary-valued, i.e. $D_1 = D_2 = D_3 = \{0, 1\}$.



a. (4 points) If f_1, f_2 are XOR functions (that is, $f_1(x) = x_1 \oplus x_2$ and $f_2(x) = x_2 \oplus x_3$), what are all the consistent assignments for this CSP?

b. (3 points) If f_1 is an XOR function and f_2 is a sum function (that is, $f_2(x) = x_2 + x_3$), what are all the consistent assignments for this CSP?

c. (3 points) With the same factors as in (b), what is the maximum weight assignment for the CSP?

2 Light bulbs

(20 points) Suppose you have m light bulbs, where each light bulb $i = 1, \ldots, m$ is initially off. You also have n buttons which control the lights. For each button $j = 1, \ldots, n$, we know the subset $T_j \subseteq \{1, \ldots, m\}$ of light bulbs that it controls. When button j is pressed, it toggles the state of each light bulb in T_j. For example if $3 \in T_2$ and light bulb 3 is off, then after pressing button 2, light bulb 3 will be on.

Your goal is to turn on all the light bulbs by pressing a subset of the buttons. Construct a CSP to solve this problem. Your CSP should have n variables and m constraints. Each constraint can be a function of up to n variables. In your answer, be sure to include

- the variables $X_1, \ldots, X_n$, and what they represent
- the variables' domains $D_1, \ldots, D_n$, and what each value in the domain represents
- an expression for the constraints $f_1, \ldots, f_m$
- the set of variables each constraint takes as input

3 Art galleries

Some artists have finished new paintings and are trying to display them. Luckily, some art galleries are looking for new paintings to display. It is your job to match artists and galleries, taking into account the preferences of the artists, the preferences of the galleries, and the capacity of each gallery.

Here is the formal art gallery matching problem setup:

1. There are m artists $A_1, \dots, A_m$ who each have a single painting they would like displayed.
2. There are n galleries $G_1, \dots, G_n$ that have space to display paintings.
3. Each artist A_i specifies non-negative preferences $P_1^{(i)}, \dots, P_n^{(i)} \geq 0$ for each of of the n galleries. A large preference value of $P_j^{(i)}$ means that artist A_i really wants their painting to be displayed in gallery G_j, and a preference value of $P_j^{(i)} = 0$ means that artist A_i does not want their painting to be displayed in gallery G_j.
4. Each gallery G_j specifies non-negative preferences $Q_1^{(j)}, \dots, Q_m^{(j)} \geq 0$ for each of the m artists. A large preference value of $Q_i^{(j)}$ means that gallery G_j really wants to display artist A_i's painting, and a preference value of $Q_i^{(j)} = 0$ means that gallery G_j does not want to display artist A_i's painting.
5. Each gallery G_j can have a maximum of one painting displayed. (Note that this means a gallery can have 0 paintings displayed.)
6. Each artist must be matched to exactly one gallery for which they have specified a positive preference (assume each artist has at least one such preference) and for which the chosen gallery specifies a positive preference for the artist (assume each gallery has at least one such preference).

We can model the art gallery matching process as a CSP. Our CSP should find the assignment with the maximum weight as determined by the product of the preference values of both the artists and galleries. There are two possible formulations of this CSP: one with m variables (one for each artist $A_1, \dots, A_m$), and one with n variables (one for each gallery $G_1, \dots, G_n$).

- **Formulation 1: Artists as variables**
  a. (1 point) What is the domain for each of the m variables $A_1, \dots, A_m$?
  b. (9 points) What are the factors? State the arity of each (e.g. unary, binary, k-ary) and write them as functions from variable assignments to non-negative real numbers. *Hint: You should have three different types of factors, and all the factors should be unary or binary.*
- **Formulation 2: Galleries as variables**
  c. (3 points) What is the domain for each of the n variables $G_1, \dots, G_n$?
  d. (12 points) What are the factors? State the arity of each (e.g. unary, binary, k-ary) and write them as functions from variable assignments to non-negative real numbers. *Hint: You should have four different types of factors. Unary and binary factors alone will not be sufficient in this case.*

Now suppose we have $m = 3$ artists and $n = 3$ galleries. The artists' preferences are as follows (artists are columns, galleries are rows):

j	$P_j^{(1)}$	$P_j^{(2)}$	$P_j^{(3)}$
1	0	1	3
2	3	2	2
3	0	4	1

The galleries' preferences are as follows (galleries are columns, artists are rows):

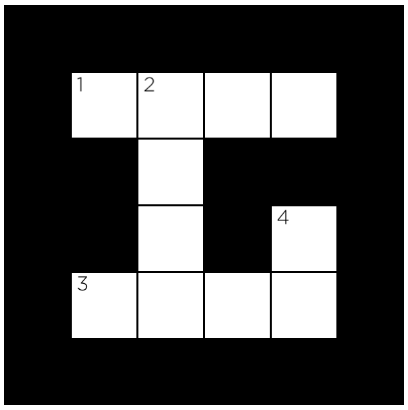
i	$Q_i^{(1)}$	$Q_i^{(2)}$	$Q_i^{(3)}$
1	2	1	0
2	4	4	5
3	0	5	1

Assume that we are modeling the problem using the **first formulation of the problem (artists as variables).**

e. (10 points) Enforce arc-consistency among the variables from the CSP you designed in parts (a) and (b). Write out the final domain of each variable after arc-consistency has been enforced.

4 Crossword

How might you go about generating a crossword puzzle? Given the structure of a crossword puzzle (i.e., which squares of the grid are meant to be filled in with letters), and a list of words to use, the problem becomes one of choosing which words should go in each vertical or horizontal sequence of squares. We can model this sort of problem as a constraint satisfaction problem. Each sequence of squares is one variable, for which we need to decide on its value (which word in the domain of possible words will fill in that sequence). Consider the following crossword puzzle structure.



In this structure, we have four variables, representing the four words we need to fill into this crossword puzzle (each indicated by a number in the above image). Each variable is defined by four values: the row it begins on (its i value), the column it begins on (its j value), the direction of the word (either *down* or *across*), and the length of the word. Variable 1, for example, would be a variable represented by a row of 1 (assuming 0-indexed counting from the top), a column of 1 (also assuming 0-indexed counting from the left), a direction of *across*, and a length of 4.

As with many constraint satisfaction problems, these variables have both unary and binary constraints. The unary constraint on a variable is given by its length. For Variable 1, for instance, the value BYTE would satisfy the unary constraint, but the value BIT would not (it has the wrong number of letters).

The binary constraints on a variable are given by its overlap with neighboring variables. Variable 1 has a single neighbor: Variable 2. Variable 2 has two neighbors: Variable 1 and Variable 3. For each pair of neighboring variables, those variables share an overlap: a single square that is common to them both. We can represent that overlap as the character index in each variable's word that must be the same character. For example, the overlap between Variable 1 and Variable 2 might be represented as the pair (1, 0), meaning that Variable 1's character at index 1 necessarily must be the same as Variable 2's character at index 0 (assuming 0-indexing, again). The overlap between Variable 2 and Variable 3 would therefore be represented as the pair (3, 1): character 3 of Variable 2's value must be the same as character 1 of Variable 3's value.

For this problem, we'll add the additional constraint that all words must be different: the same word should not be repeated multiple times in the puzzle.

The challenge ahead, then, is to write a program to find a satisfying assignment: a different word (from a given vocabulary list) for each variable such that all of the unary and binary constraints are met.

There are two Python files in this project: `crossword.py` and `generate.py`. The first has been entirely written for you, the second has some functions that are left for you to implement.

First, let's take a look at `crossword.py`. This file defines two classes, `Variable` and `Crossword`.

Notice that to create a `Variable`, we must specify four values: its row `i`, its column `j`, its direction (one of `Variable.ACROSS` or `Variable.DOWN`), and its length.

The `Crossword` class requires two values to create a new crossword puzzle: a `structure_file` that defines the structure of the puzzle (the _ character is used to represent blank cells, any other character represents cells that won't be filled in) and a `words_file` that defines a list of words (one on each line) to use for the vocabulary of the puzzle. Three examples of each of these files can be found in the `data/` directory of the project.

Note in particular, that for any crossword object `crossword`, we store the following values:

- `crossword.height` is an integer representing the height of the crossword puzzle.
- `crossword.width` is an integer representing the width of the crossword puzzle.
- `crossword.structure` is a 2D list representing the structure of the puzzle. For any valid row `i` and column `j`, `crossword.structure[i][j]` will be `True` if the cell is blank (a character must be filled there) and will be `False` otherwise (no character is to be filled in that cell).
- `crossword.words` is a set of all of the words to draw from when constructing the crossword puzzle.
- `crossword.variables` is a set of all of the variables in the puzzle (each is a `Variable` object).
- `crossword.overlaps` is a dictionary mapping a pair of variables to their overlap. For any two distinct variables `v1` and `v2`, `crossword.overlaps[v1, v2]` will be `None` if the two variables have no overlap, and will be a pair of integers `(i, j)` if the variables do overlap. The pair `(i, j)` should be interpreted to mean that the ith character of `v1`'s value must be the same as the jth character of `v2`'s value.

`Crossword` objects also support a method `neighbors` that returns all of the variables that overlap with a given variable. That is to say, `crossword.neighbors(v1)` will return a set of all of the variables that are neighbors to the variable `v1`.

Next, take a look at `generate.py`. Here, we define a class `CrosswordCreator` that we'll use to solve the crossword puzzle. When a `CrosswordCreator` object is created, it gets a `crossword` property that should be a `Crossword` object (and therefore has all of the properties described above). Each `CrosswordCreator` object also gets a `domains` property: a dictionary that maps variables to a set of possible words the variable might take on as a value.

We've also defined some functions for you to help with testing your code: `print` will print to the terminal a representation of your crossword puzzle for a given assignment (every assignment, in this function and elsewhere, is a dictionary mapping variables to their corresponding words). `save`, meanwhile, will generate an image file corresponding to a given assignment (you'll need to `pip3 install Pillow` if you haven't already to use this function). `letter_grid` is a helper function used by both `print` and `save` that generates a 2D list of all characters in their appropriate positions for a given assignment: you likely won't need to call this function yourself, but you're welcome to if you'd like to.

a. (5 points) Implement `is_complete`, which checks to see if a given assignment is complete.

b. (10 points) Implement `is_consistent`, which checks to see if a given assignment is consistent.

c. (5 points) Implement `select_unassigned_var`, which returns a single variable in the crossword puzzle that is not yet assigned at random.

d. (15 points) Implement `backtrack`, which accepts a partial assignment assignment as input and, using backtracking search, returns a complete consistent assignment of variables to values (if it is possible to do so).

You should not modify anything else in `generate.py`, although you may write additional functions and/or import other Python standard library modules. You should not modify anything in `crossword.py`.

To run your program, you can run a command like

```
python generate.py data/structure1.txt data/words1.txt
```

If an assignment is possible, you should see the resulting assignment printed. You may also add an additional command-line argument for an image file, as by running

```
python generate.py data/structure1.txt data/words1.txt output.png
```

to generate an image representation of the resulting crossword puzzle as well.

Submission

Submission is done on Canvas. You should submit two files: one containing your solutions to the written problems, and one for the coding problem.

- Submit your written solutions in a single PDF file with your name at the top. Make sure to clearly indicate the number and letter of the problem corresponding to each solution. It is okay to hand-write your solutions and then scan them into a PDF, but *only if your handwriting is legible.*
- Submit your coding solution in the provided `generate.py` file. Do not modify the name of this file after downloading it from the course website.