Homework 2 Solutions
Michael Narayan

---

1. Problem 5.3

   When searching for a solution to a CSP it is often useful to assign a value that is least constraining to the most constrained variable. Every variable in a CSP must be assigned a value, and thus choosing a variable with few possible values earlier in the search space helps shrinks the search space. This is because the effect of the number of branches on the size of the search space increases exponentially the higher the variable is in the tree. On the other hand, it is usually useful to pick the value for a variable that is least constraining. A less constraining value, in the absence of other information, increases the chance that a consistent assignment will be found for the remaining variables.

2. Problem 5.4

   a. Constructing a crossword puzzle can be easily formulated as a tree search. The initial state would be the crossword grid with no spaces filled in. The operator would be to assign a word from the dictionary which fits in the next unfilled word space without overwriting any other word's letters. While individual letters could be assigned rather than words, the huge ratio between the number of permutations of English letters of a given length and English words of the same length means that this would probably make the search space needlessly large. Given that this search has no real 'optimal' solution we can be a bit more flexible in how we construct our heuristic. One one would be to prefer paths which limit the letters which can be put in blanks as little as possible. This heuristic could be created by simply summing up all the letters which can not be put in current blank spots based on the current state of the board.

   b. This problem can also be modeled as a CSP easily. Once again, the choice of variables representing words or letters is up to you. I again choose words for the same reasons as before. Here the domain of each variable is all the words of the given length that exist in the dictionary. The constraints are then between variables which represent spots that intersect and indicate that if one variable has a letter at one spot, then the other must have the same letter in the corresponding position.

3. Problem 5.5

   a. One possible way to model the rectilinear floor-planning problem as a CSP would be to find the lowest common denominator of the sides of all the various rectangles. Then construct a grid of squares with sides of this length inside the area to be filled and let each square be a variable. The domain of each variable would then be one of the available tiles, while the constraints would be groups of variables representing the possible permutations of tile placement, with each group requiring that if one variable had a particular value, all the rest would too.

   b. To model class scheduling as a CSP we can represent each of the classes to be taught as a variable. These variables then have domains of 3-tuples, with each tuple representing a professor, a time slot, and a classroom. The variables would then have constraints that no two could have the same professor and time slot nor the same classroom and time slot.

4. Problem 5.11

   We can transform a ternary constraint of three variables A, B, C into a set of binary constraints by introducing a new variable Y which has as a domain all the pairs which result from $A \times B$. We then create constraints on Y such that the first part of Y has the same value as A and the second the same as B. We create a binary constraint between C and Y which is the same as the original but replaces A with $first(Y)$ and B with $second(Y)$. For constraints with more than three variables we can do exactly the same thing. If a constraint has n variables, then create a new variable which is a $n-1$ tuple. Each element of the tuple will then correspond to one of the variables, and the introduced

variable can be combined with the remaining original variable to create a binary constraint which is identical to the original one. For unary constraints we can merely remove all values which violate the constraint from the domain of the variable, thus eliminating the need for the constraint since no values which can violate it can be chosen.

5. I chose to formulate the Sudoku puzzle as an 81 variable CSP. Each of the variables represented one of the squares on the board and had a domain of the digits 1-9. The constraints were that no two variables in the same row, column, or 3x3 grid could have the same value. In order to generate these constraints I merely assigned each position a number from 0-80 and then calculated what other variables were constrained by it.

   Under my implementation I found that the best consistency measure was AllDiff, followed by arc consistency, followed by forward propagation, and finally no consistency checking. In general, I found that AllDiff performed exponentially better; the harder the problem the greater the ratio was between it and other algorithms. When solving this type of problem it is important to attempt to balance the amount of time spend thinking against the amount of time spent doing. For this particular problem, a fairly high degree of thinking was beneficial, as evidenced by the performance of AllDiff. Too much thinking though can also be detrimental. In Sudoku there are a large number of dependencies between the different blocks, and thus establishing complete arc consistency can result in too much time spent thinking and not enough doing. While AllDiff ran the fastest, all of the algorithms accomplished the task in a reasonable amount of time. While at first glance the search space may seem prohibitively large, the number of interlocking constraints serves to narrow it down even with the most naive consistency checking.