Crossword Solver - AI HW 2

Morgan Ciliv

February 2017

1 Looking back

1.1 Successor functions

1.1.1 For factored representation

The successor function would be the values that are contained in the structure of the representation's neighboring representations.

1.1.2 For unfactored representation

For the unfactored representation it is simply the set of neighbors, but no values to base it off on.

1.2 Local Search

For example, let's take a local search algorithm such as Hill Climbing

1.2.1 For factored representation

In the factored representation, we could use hill climbing based off of the numerical values. It would be similar to the way in which we do the programming assignment below because you are looking for a maximum or a minimum in the data set and moving that direction. However, with hill climbing we could add multiple different routes because the greedy algorithm approach isn't always the best. Instead, you could choose multiple different ones and if time runs out go back and try the next best guess.

1.2.2 For unfactored representation

For this representation, the hill climbing would be very simple and you would just maximize the steps you take and if time runs out go back and try another route.

However, since it's unfactored it might not work as well since there is less information to base the maximization of the funciton on.

2 Crossword Solver Report - Programming

2.1 Implementation of choose_entry

I decided to implement this function by looping through each of the entry options, determine the number of possible values each option could take on, and keep track of the option with the least number of values that could be taken on. I then return that corresponding entry. This implementation was the most straightforward to me. When going through each of the array elements, I decided to just keep track of the index of the corresponding entry, so that I simply select the corresponding entry and return it at the end.

2.2 Implementation of choose_value

In my implementation of choose_value, I reasoned that the function needed a nested for-loop because we are choosing among the possible values that an entry can take identifying how many options there are left in the board which I decided to calculate by looping through each of the other corresponding entries given that choice. I decided to sum up all of those entries possible values because it was analogous to identifying how many branches are left in the search tree. From the looping, I grabbed the index of the value that kept the greatest amount of options open, then return the corresponding chosen value.

2.3 Changing algorithm by choosing subset of values

Taking a just one value at random from the set was the fastest. This makes sense because in a crossword puzzle like there are many constraints. Though there are many words in the dictionary, finding words with certain letters at certain spots, greatly narrows down the options. Moreover, there are many more constraints as the cross-word is filled out; therefore, there are very few options as the puzzle is solved. This is why when watching the solver solve the crossword with a subset of 1 chosen, the solver finished solving the puzzle very quickly. Overall, the size of the domains decrease as the puzzle is solved.

3 Looking forwards

I would design the new algorithm as follows by slightly changing the randomized subset algorithm:

Given that algorithm, the new algorithm would replace the randomization by picking "off_set" number of the highest scoring values.

To do this more specifically, I would add an extra loop after the variable declarations replacing the randomization loop which identifies the "off_set" number of highest scoring values by keeping track of the smallest of the 5 and if a value is greater than that, placing it into the new subset vector (new_value_options)