Nicki Polyakov
Carolyn Ryan
Barr Iserloth

CS 361 Final Project Proposal

Problem Definition

Our EA will evolve to make a viable crossword puzzle. We will start with an input of a 10x10 empty grid, where some squares in the grid will represent spaces where we can place letters, and some squares in the grid will contain black squares (places where we cannot place letters). Our output would be a grid filled in with words and black squares. A good solution will include a high number of valid words (both horizontal and vertical), and a low number of invalid words. Our evaluation method will penalize for any invalid words. We think the scope of this project is appropriate - it is doable in a few weeks but isn't too easy.

Infrastructure

For our EA code, our starting point will be using Python and building off of the genetic algorithm code found at https://github.com/groger/crossword-solver. This code uses an EA to create a crossword puzzle, so it gives us a good starting point, but we would still need to expand on that code. This uses populations based off of a modifiable default starting grid which delineates where the black squares in the grid are, and generates multiple candidate genomes of words, which will be explained in significantly more depth in our Approach section. The code we found currently creates 8 by 10 boards, so we would want to write code that could create a 10 by 10 board. We would also write code that would create a simulation for the evaluation method where we could print the resulting board.

Approach

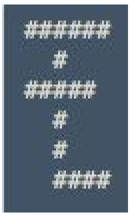
For this project, we will be using python code and will use a github repository of a crossword-solver as our starting point (https://github.com/groger/crossword-solver).

Representation

Our representation of an individual solution in our population will be a list of the words, first the horizontal words then followed by the vertical words. This will ensure that our genotype is more similar to the phenotype than if we used individual characters to fill out the grid. We will also have all individuals in the same run of our algorithm be working to fill out the same crossword grid, that is all words in the same place in the genotype will have the same length and same position in our crossword. With this representation, we must think of how conflicts will be handled. A conflict in this case is a location in the grid where an individual character is part of both a horizontal and vertical word. We will use our original grid to id these conflict locations.

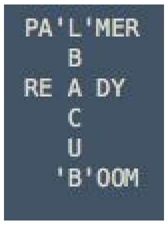
Instead of ensuring through the representation that conflicts are handled, we will instead add conflict handling to our fitness calculations.

Example Genotype and Phenotype



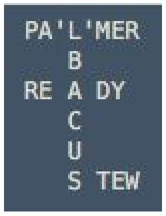
```
######
  #
#####
  #
  #
  ####
```

An example of a small crossword would be to fit into the grid (right) where '#' represents an empty slot in a crossword grid (i.e. one that should be filled in with a character). As such, the genotype for this grid would have a length of 21. The first 6 characters in the string will be for the first horizontal word, the next 5 for the second horizontal word, and the next 4 for the final horizontal word. The remaining 6 will be for the vertical word. From the grid, we will identify conflict/crossover characters. In this case these are ((3, 16), (9, 18), (12, 21)). These are the locations in the string where the character(s) for the crossword entry will be. So a perfect solution will have the character at index 2 be the same as the character at index 15, etc. A solution for this grid with fitness 2 would be "palmerreadyboomabacus." We can look at index 2 (character = 'l') and index 15 (character = 'a') and see this is one conflict. We can look at the other conflict locations and see one is valid (ie a conflict doesn't exist) and one is invalid (a conflict exists). This can be seen in the grid to the left, where the conflicts are indicated using single quotation marks around the conflict locations.

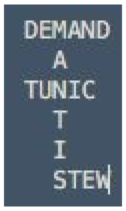


```
PA'L'MER
   B
RE A DY
   C
   U
  'B'OOM
```

A mutation on this individual could turn the word 'boom' into another word 'stew.' This is more likely with our mutation checking through validity (so a mutation will try to come up with a word that begins with the same character as in the other conflict zone). Here this would look to see the first index in our word should be 's' to correspond to the 's' in "abacus." Then the new genotype for this crossword solution would be 'palmerreadystewabacus' and have a fitness of 1 (phenotype to the right). A solution to this grid with fitness of zero is "demandtunicstewmantis." Which corresponds to the filled in crossword (right).



```
PA'L'MER
   B
RE A DY
   C
   U
   S TEW
```



```
DEMAND
  A
TUNIC
  T
  I
  STEW
```

Fitness Calculations

Similar to the fitness calculations in the papers we read for our literature review, we will have a fitness function where lower fitness scores correspond to the more fit individuals. We will have a mapping of fitness such that a fitness of 0 is a perfectly filled in crossword for our desired grid. The fitness function will look at the potential conflict locations in the grid. For every conflict location the fitness calculation method should look at that location in the horizontal word and in the vertical word and if the characters do not match, it should increase the fitness of the crossword. Thus, if the fitness of any crossword is zero, it will not have any conflicts at these points of intersection.

Selection Method

We will experiment with selection methodology and try multiple different ways of selecting the next generation. We will start with fitness proportional and tournament style selection methods. We already have some Java code for these methods of selection that we can use as a jumping off point for our project. We will also try to implement a selection method that was used in one of the papers we read. This method made N (population size) individuals using the crossover and mutation as normal and then evaluated their fitness. These N individuals were then added to the population, making the new (temporary) population have size 2N. Then, they chose the top N fit individuals to continue on to the next generation (individuals with the lowest fitness calculated scores). As such, we ensure that we keep previously good solutions unless we produce better solutions.

Mutation Method

A small percentage of the population in each generation will be selected at random and be subject to mutation. We will work with two main mutation methods. In the first method, a random word in the chosen crossword will be selected and replaced with another random word of the same length. In the second method, a random word in the chosen crossword will be selected and replaced with another word of the same length. This time, we will only allow the original word to be replaced with words that do not invalidate intersecting letters. So, if we had the word "computer" intersecting with the word "code" at the letter "o," we would only be able to replace "computer" with words that have "o" as their second letter. We are also considering only applying this to valid words. So, if "computer" intersected with the word "pwqr" at "p," we might not require "p" to remain in the new word since "pwqr" is an invalid word.

Crossover Method

For our crossover method, we will take two existing crosswords, and select words of equal length from each crossword. Next, we will swap those words. So, if crossword 1 contained the word "house" and crossword 2 contained the word "crazy," we could swap those two words. We also might add validity checks to these crosswords. We would have access to a list of squares in the grid in which an intersection occurs. If we change a horizontal word, we would want to check whether that invalidated the vertical

We might not want to change the letters at these intersections, since it creates the possibility of invalidating a valid word. Thus, if we end up changing a letter at one of these intersections, we could consider not using that crossword in the next generation (and using one of its parents instead).

Measuring Success

We want to create several base cases that we can test our EA against so that we can test whether our system works well or not. At the lower bound, we would want a solution that randomly places words with an appropriate length in the horizontal slots, or randomly places words with an appropriate length in the vertical slots. We can place random letters in the remaining slots. We think this will have a pretty low chance of success because it is very likely that many of the vertical words will be invalid if we only pay attention to horizontal words, and many horizontal words will be invalid if we only pay attention to vertical words. On the upper bound, we can create crossword puzzles by hand that contain only valid words. This would mean that the fitness of the puzzle is 0, a perfect fitness. We will consider our system to be somewhat successful is if it able to beat the lower bound base case, and we will consider it very successful if it is able to match or come close to the upper bound. We will consider a fitness of 1 or better to be successful, since that would mean that only up to 2 words in the crossword are invalid.

Goal Setting

Minimal Version of Project

The minimal successful version of our project would be to implement basic improvements to the crossover and mutation methods that we will be building on, and also display the generational chromosomes and print the final board. We want to be able to generate a 10x10 crossword with at least a few intersecting words, and be able to define multiple board states (in terms of black cells) for which our genetic algorithm can find perfect (or close to) solutions routinely within a given number of runs.

Good Version of Project

Something that is beyond the most basic functionality but is still very doable for implementation is output files much like the ones from the Tartarus project we did. We can output files containing the best individuals from each generation, as well as other information that we consider necessary. We can also work on generating graphs containing best fitness, average fitness and other potentially useful statistics as well. These output files would ideally be graphical representations of our crosswords rather than merely the string of the genome.

Great Version of Project

Interactive interface like the Prisoner's Dilemma simulation, where users can edit values for crossover/mutation/population size and so on. This could potentially include editable board states to test on, as well as built-in visuals for the user to view as the generations occur. We could also try to implement multiple populations to more quickly converge to an optimal 0-fitness solution, as opposed to using only mutation and crossover to make progress towards an optimal solution. The multiple populations could be used to see if we can aggregate solutions to be able to come to the perfect solution in fewer generations.