Project 6: Genetic Algorithm with WOC Numberlink

Drew Bender

Cameron Vincent

Caleb Klenda

Computer Science and Engineering

Speed School of Engineering

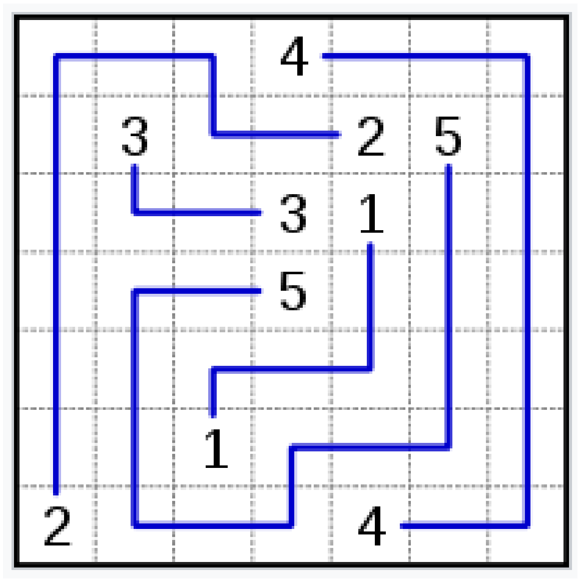
University of Louisville, USA

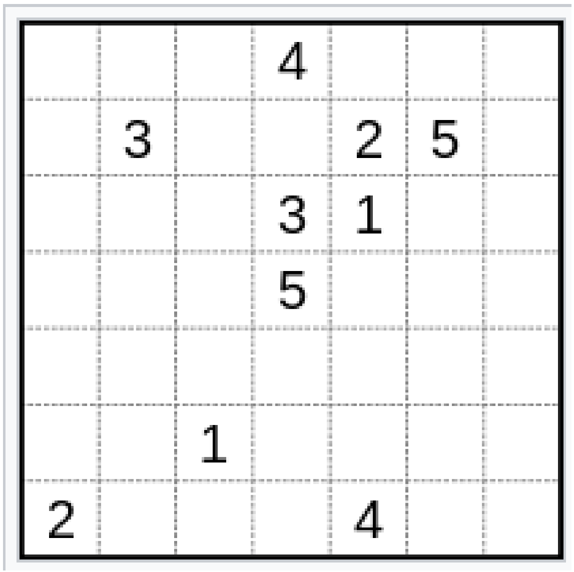
andrew.bender@louisville.edu

c0vinc04@louisville.edu

1. **Introduction** (What did you do in this project and why?)

The np-complete problem that was selected for the AI Final Project was the Numberlink game. The Numberlink game is a type of logic puzzle that involves a nxn grid with pairs of numbers placed on the grid. The goal of Numberlink is to match up all the pairs of numbers on the grid with continuous lines (or paths). The lines to connect each pair cannot split off or cross over each other, and the number from each pair has to mark the beginning and end of each line and cannot be in the middle. Typically, each Numberlink problem only has one solution and all of the squares are used in the end. This is an example of a Numberlink game with its solution to demonstrate the idea:





In this project we attempted to use a combination of genetic algorithms (GA) and wisdom of crowds (WoC) to find solutions to Numberlink problems.

2. **Approach** (Describe algorithm you are using for this project)

The first issue addressed in our approach was how the Numberlink game would be represented. We decided to display the data of the game as a 2d array. Each index of the array was given a numerical value, 0 meaning that the space on the board was empty, 1 meaning it was populated by a path associated with one, etc.

A GUI display was also created that took this 2D array as an input and created a color plot. This color plot assigned each value on the board with a particular color and filled in squares containing that number path with the particular color. The result was much easier to read and compare to the given answers for Numberlink.

To create the initial population, a new board is created. A random number on the board is picked and will attempt to draw a path to the number it shares on the board. This is done by picking random directions, until the final location is reached. This process is repeated until there remains no more room to draw paths on the game board. In the tests with a 7x7 grid, this resulted typically in 2-3 of the numbers having “valid’ paths.

The creation of children and changing populations over generations was handled with the **Reproduce()** fmethod. **Reproduce()** takes in the current generations of individuals and the number of number pairs for the problem as parameters and the bulk of this function is making calls to the **Crossover()** and **Mutate()** methods until a new generation is made up of different individuals is created. The first method that will be covered is **Crossover()** which takes in two randomly selected individuals from the current generation, with higher selection odds given to more fit individuals, and the number of number pairs as parameters. The actual function takes the two selected individuals/parents and randomly selects the value of each space from one of them to populate the space for the child. As an example, if space [0][0] contained 3 in one parent and 4 in the other, then the child could possibly have either 3 or 4 as its value. Once two children were created this way the method then returned the children. This approach to crossover was not successful and was flawed once looking back since it was almost equivalent to being random and did not take valid paths into account or attempt to preserve previous valid paths. Some ideas on how this could potentially be improved will be touched upon in the *Discussion* section.

The **Mutate()** method that is called from **Reproduce()** takes an individual and the number of number pairs as parameters. This method is also only called when a random value between 0 and 1 is less than or equal to the *mutRate* attribute value assigned to the **Genetic()** class item and each child created has the potential to mutate after every crossover. **Mutate()** randomly selects one of the number paths in an individual and then attempts to create another path that is valid using the **createPath()** method. After **createPath()** returns the individual with a new path there is a check to see if the new path is valid (legally connects a number pair), If the new path is valid then the individual uses the newly created path in place of the old, otherwise, the individual stays the same as it originally was. The **createPath()** method takes in the number pair being modified, their starting positions, and a grid with only the starting positions filled. The functionality of **createPath()** is to pick random directions to create a valid path and is assisted by the method **checkDirection()** to confirm nothing goes out of bounds of the indices or to an illegal/occupied space. If a valid path is found then a check value is set to True, otherwise, it is set to False. This implementation of a mutation method did not end up being beneficial in most cases since it did not take into consideration preserving other potential valid paths and could destroy them in the process of mutating a path. Some ideas on how a mutation could be improved will be discussed in the *Discussion* section.

After a set of runs on a dataset, a collection of the best individuals from the final generations were passed to the **WisdomOfCrowds()** function. tHis first defines a 3 dimensional array, the first two dimensions correspond to the grid of the Numberlink problem and the values within are an array representing the frequency of each number in the wisemen's solution. Thus, the maximum value of each inner array was the most picked number for that square across all of the collected solutions. The function translates this back into a 2d array, choosing the squares based on highest frequency, and returns that as its answer.

3. **Results** (How well did the algorithm perform?)

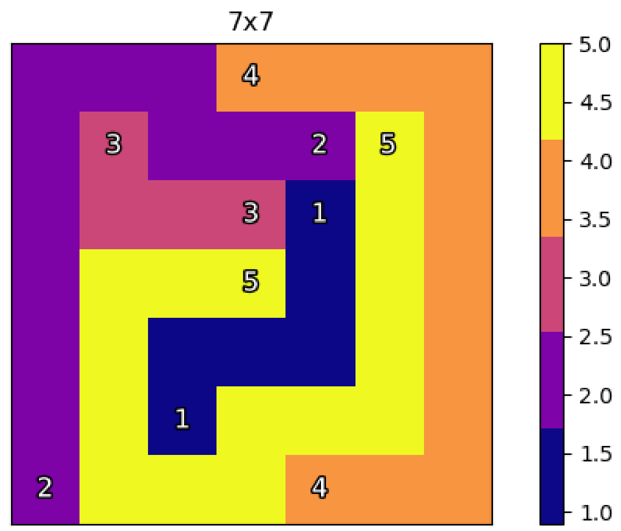
Overall, the algorithm struggled and was not able to solve the provided problems in the genetic nor wisdom of crowds steps.

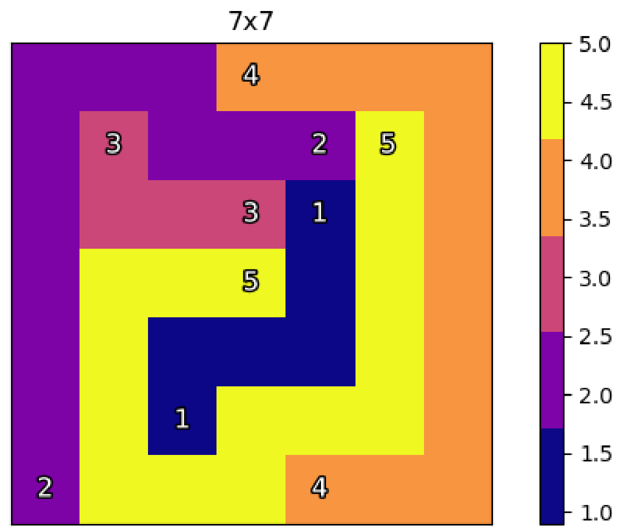
3.1 **Data** (Describe the data you used.)

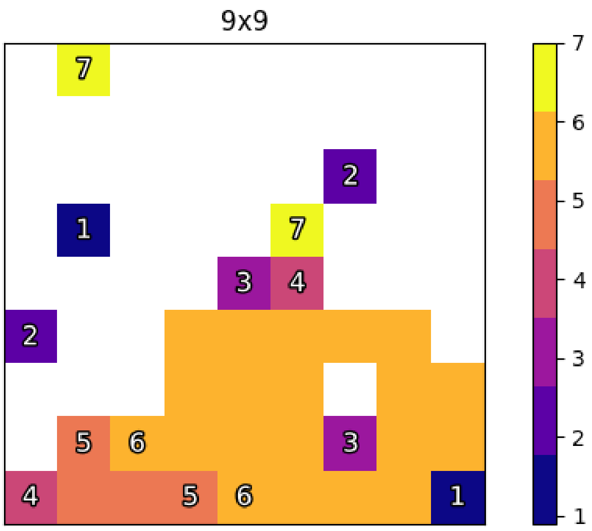
Data was pulled from online solutions to Numberlink and translated by hand into a text file that could be read from. The file represented the coordinates of a 2d array whose contents were the squares of the Numberlink game. Only grids with solutions were chosen so that the algorithm was guaranteed to run on problems with solutions (no attempt was made to handle unsolvable grids.)

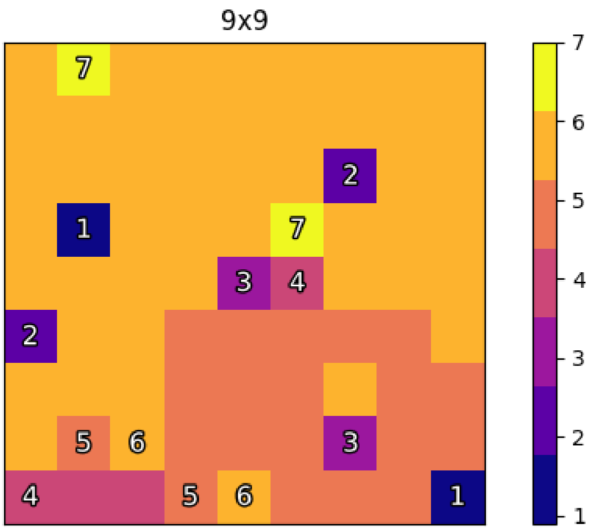
The output is the grid with squares colored corresponding to their number. Ideally, a path is formed between each number. For each dataset, the first figure is the best result from a set of runs of the genetic algorithm and the second is the aggregate solution using the wisdom of crowds technique.

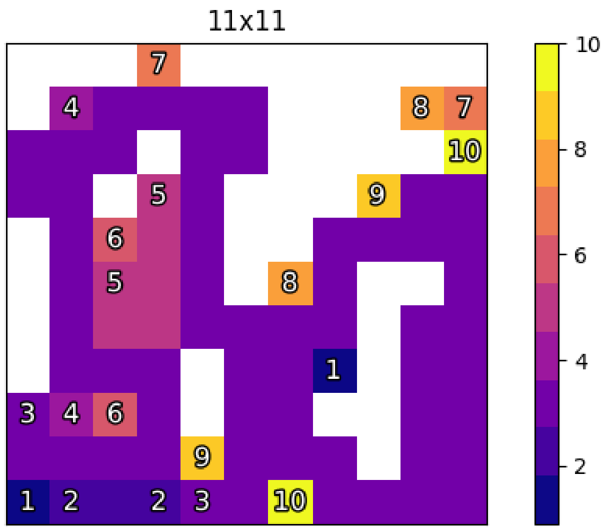
3.2 **Results** (Numerical results and any figures or tables.)

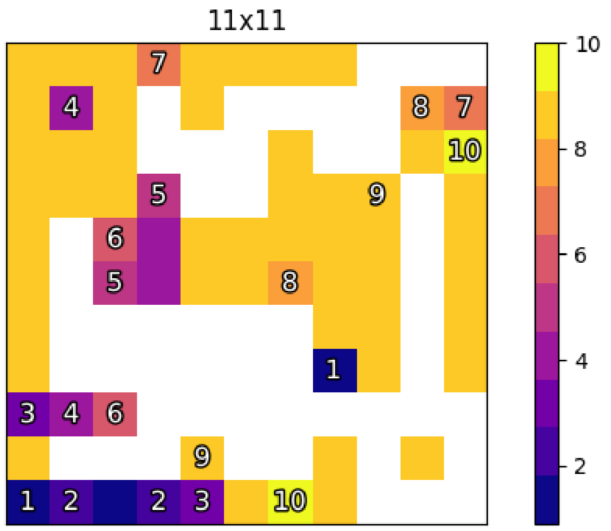

7x7
4
3
2
5
3
1
5
1
2
4
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0

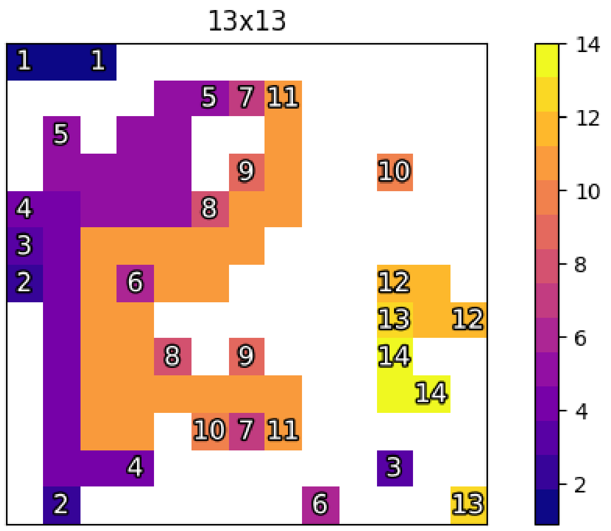

7x7
4
3
2
5
3
1
5
1
2
4
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0

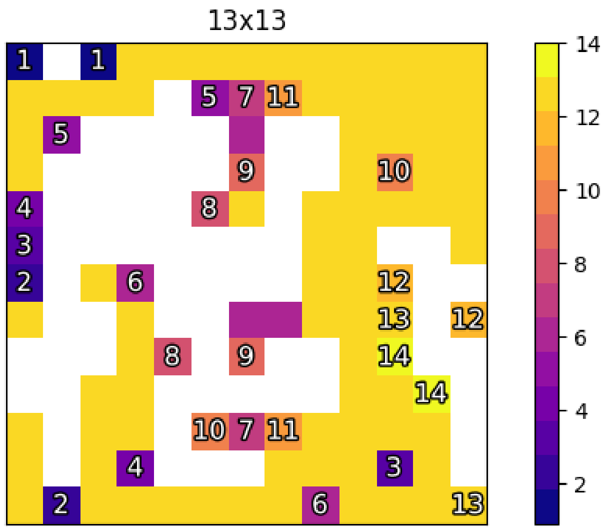

9x9
7
2
1
7
3
4
2
5
6
3
4
5
6
1
1
2
3
4
5
6
7

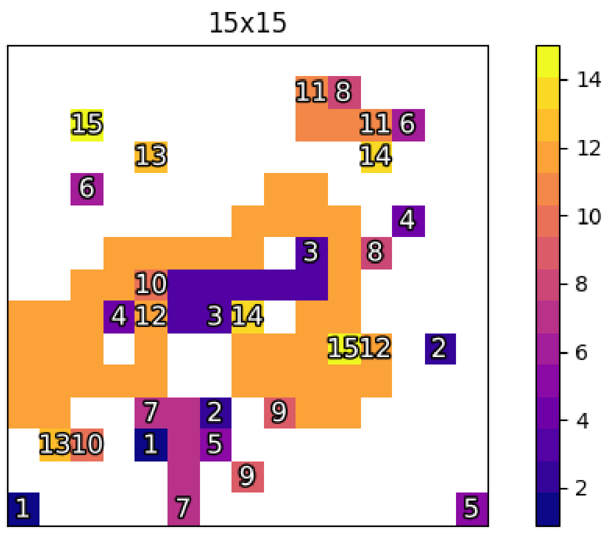

9x9
7
2
1
7
3
4
2
5
6
3
4
5
6
1
1
2
3
4
5
6
7

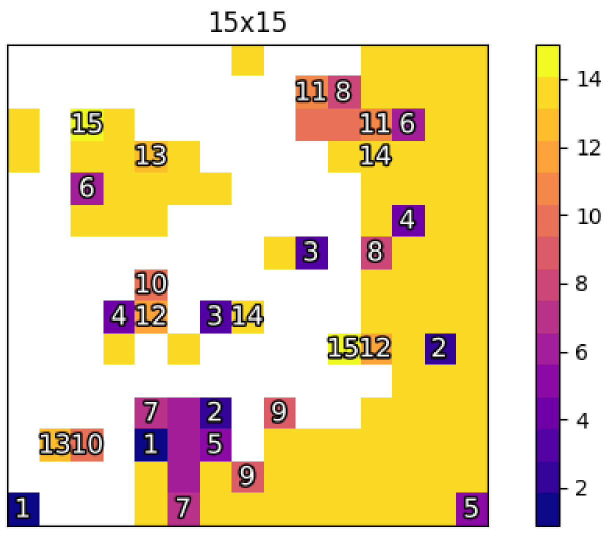

11x11
7
4
8
7
10
5
9
6
5
8
1
3
4
6
9
1
2
2
3
10
2
4
6
8
10


11x11
7
4
8
7
10
5
9
6
5
8
1
3
4
6
9
1
2
2
3
10
2
4
6
8
10


13x13
1
1
5
7
11
5
9
10
4
8
3
2
6
12
13
12
8
9
14
14
10
7
11
4
3
2
6
13
14
12
10
8
6
4
2


13x13
1
1
5
7
11
5
9
10
4
8
3
2
6
12
13
12
8
9
14
14
10
7
11
4
3
2
6
13
14
12
10
8
6
4
2





4. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

In hindsight, perhaps the most challenging issue of a genetic implementation of Numberlink is how to define fitness. In our previous experience with TSP, fitness could very clearly be defined as a shorter and shorter path. But in Numberlink, a short path in many cases does not mean it is included in the final solution. This often leads to trends where a path that is complete would then be defined as “fit” but it actually differed greatly from the final solution. An alternative is to simply define fitness in terms of connected numbers, however, this also introduces a problem: homogenization of fitness scores leading to a difficult to rank population. In the end, a hybrid approach was chosen that combined several factors, but was still lacking in properly representing an individual's closeness to a good solution. This issue of defining fitness contributed greatly to the issues we encountered with this problem.

Overall our initial methodology was somewhat flawed. We started by focusing on values of individual squares, originally having our mutation change individual squares, and having the crossover focus and just grabbing a portion of each parent, without much care into preserving specific paths. We quickly discover that in this game a partial path is fairly meaningless. A path that is 90% complete and a non-existent path are really the same amounts of incorrect in regards to the final solution. As we began developing a solution, we realized to instead prioritize the creation of paths, and those specific paths surviving throughout the generations. This results in better end results, but it is very hard to improve diversity or create new paths that weren’t in the first generation. This leads us to believe that the genetic algorithm approach to this game is not an ideal approach. We were unable to get a 100% complete solution with our algorithm for all except the 7x7 board. Even in that case, it took several attempts to arrive at a correct solution. Although we do believe it is possible to solve this game using a genetic algorithm consistently, it appears to be much more challenging than with other approaches. Ultimately, even with the small datasets used, our approach of a genetic algorithm performed extremely slowly and has little benefit over other approaches. When combined with the Wisdom of Crowds, it is possible a better methodology could produce better results but would require the introduction of some new greedy algorithm which likely would outperform the genetic algorithm in the first place. Fundamentally, the Wisdom of Crowds struggled to perform due to the underlying issues from the genetic algorithm, but it did do what it was supposed to in forming an aggregate solution.

One approach that has shown promising results for others is to use a Zero-Suppressed Binary Decision Tree. This was an approach described in *Finding all solutions and instances of Numberlink and Slitherlink by ZDDs.* Overall rather than generating random potential solutions like our genetic algorithm, sequential begins to eliminate potential solutions through a decision tree until it eventually reaches a solution. Their method described includes no randomness, which our method using Genetic and WOC relied much too heavily on.

5. **References** (If you used any sources in addition to lectures please include them here.)

Yoshinaka, R., Saitoh, T., Kawahara, J., Tsuruma, K., Iwashita, H., & Minato, S.-ichi. (2012). Finding all solutions and instances of Numberlink and Slitherlink by zdds. *Algorithms*, *5*(2), 176–213. https://doi.org/10.3390/a5020176

Website used to gather solutions: https://flowfreesolutions.com/