Overview:

Conventional mathematical optimization techniques rely on high level mathematical knowledge to provide exact answers to a problem with efficient use of computational resources. If the problem in question can be solved sufficiently with sub-optimal answers and computational resources are not a bottleneck there are methods called metaheuristics that can be used in lieu of traditional optimization techniques. The benefit of metaheuristics is their simplicity, since they require a less robust knowledge of mathematics. Instead they are reliant on logical operators such as if-statements to make decisions at discrete events in the program. In this paper we explore a combination of metaheuristic methods to solve the Slitherlink puzzle previously solved by mathematical optimization.

Methodology:

The Puzzles

The combination of metaheuristics used to solve two Slitherlink puzzles were a Tabu Search and a Genetic Algorithm. Phase 1 of our methodology starts off by importing each puzzle into the R programming language as a matrix. These matrices are shown below in Figure 1, Slitherlink 1 is on the left and Slitherlink 2 is on the right. The elements of these matrices are the demand values which define the puzzle, where NA corresponds to no demand required. Each demand value exists in a grid with four nodes to represent corners, and four arcs to represent the sides.

NA	3	NA	NA	2	NA	NA	NA	3	3	3	NA	3	NA	3	3	NA	3	NA	NA
NA	NA	1	2	NA	NA	1	2	2	NA	NA	NA	NA	NA	NA	NA	NA	NA	1	2
2	NA	2	NA	2	3	NA	2	2	NA	2	NA	NA	3	3	2	2	NA	NA	NA
2	2	0	NA	NA	3	NA	NA	2	NA	1	1	NA	NA	NA	NA	1	1	NA	2
NA	NA	2	NA	2	NA	3	NA	NA	NA	NA	NA	3	NA	2	NA	NA	3	NA	3
NA	NA	2	1	NA	NA	NA	NA	NA	NA	3	NA	NA	NA	NA	NA	2	2	3	NA
2	NA	NA	NA	2	NA	3	3	3	NA	NA	0	NA	1	NA	2	NA	NA	NA	NA
NA	2	1	NA	NA	NA	NA	NA	2	NA	1	2	3	NA	NA	3	2	NA	NA	3
NA	NA	NA	NA	NA	3	3	NA	NA	2	NA	NA	2	NA	2	NA	NA	2	NA	1
3	2	2	2	2	NA	NA	2	NA	NA	3	NA	3	NA	NA	2	1	1	NA	1

Figure 1: Slitherlink Puzzles - Matrix Inputs

The next series of steps in phase 1 is to give each grid a unique identification number, give each node a unique identification number, and give each arc a unique identification number. Then each grid must be mapped to a set of four arcs, and each arc must be mapped to a set of two nodes. This mapping will allow for easy verification of potential solutions. Finally three decision vectors are created to represent a potential solution, namely x, v, and f. Each of these vectors are binary and contain 220 elements which represents the 220 unique arcs in a 10 x 10 Slitherlink puzzle. Initially all values of x are 0, all values of v are 1, and all values of f are 0.

The purpose of vector x is so the Tabu Search and Genetic Algorithm can make choices on what solution to try next. If an element of x takes a value of 1 then that means an arc is being used, otherwise that arc is not being used. The purpose of vectors v and f are to maintain good solution characteristics when those characteristics are found. If an element of vector v takes a value of 1 then that means an arc is allowed to change in value, otherwise it is not allowed to change in value. If an element of vector f takes a value of 1 then that arc is fixed as being used, otherwise it is not fixed whatsoever. Therefore using x as a random variable, and using v and f as control variables, no matter what values that x takes, the good characteristics will be maintained by the following computation:

$$x = (x * v) + f$$

Application of Tabu Search:

Phase 2 of our methodology is to use Tabu Search to satisfy all of the demands in the puzzle. The first step in phase 2 is to extract the data frames from phase 1 which represent the mapping between each demand and four arcs, and each arc and two nodes. Table 1 below shows the demand to arc mapping, and Table 2 shows the arc to node mapping.

Table 1: Example Demand to Arc Mapping

arc1	arc2	arc3	arc4	demand
123	14	133	15	1

Table 2: Example Arc to Node Mapping

arc.id	node1	node2
123	24	35
14	24	25
133	25	36
15	35	36

The next step in phase 2 is that the Tabu Search will go through each row of the demand to arc mapping data frame and randomly pick “demand” many arcs from the set of arcs: arc1, arc2, arc3, arc4. Those arcs will then be given a value of 1 in the corresponding elements of x. Given that some demands share arcs, the random decision made in a later row can impact the decision made in a previous row, which will be evident when x is evaluated. Consider each row in the demand to arc mapping data frame, as a Tabu Search neighborhood. After the Tabu Search is done with randomly picking arcs in the demand to arc data frame, x will be compared to the current tabu list, if x is found to exist in this list then x is deleted and the Tabu Search is run again. If x is not found in the tabu list then summations for each neighborhood are computed in x and compared to their demand values. When there is not a match, that neighborhood must be revisited and x is stored in the tabu list, when there is a match that neighborhood doesn’t need to be revisited.

When all neighborhoods no longer need to be revisited, the chosen arcs are then used to extract all of the nodes that were used by extracting the node1 and node2 pairs from the arc to node data frame. The final evaluation is to see if a node was used more than two times, if so then this indicates that an intersection exists and therefore the closed loop requirement of the puzzle cannot be met. When an intersection at a node is found, all neighborhoods that share that node must be revisited by the Tabu Search, and the current x is stored in the tabu list.

Overall, the Tabu Search iterates and when all demands haven’t been met or when any node has been used more than twice. When both of these conditions are satisfied, then the Tabu Search has completed its objective. When the Tabu Search is done, the final x vector is used to update vectors v and f such that all arcs which are used to satisfy all demands cannot be changed.

Application of Genetic Algorithm:

Phase 3 of our methodology is to used a Genetic Algorithm to satisfy the single closed loop requirement of the puzzle. The first step in phase 3 is to extract the decision vectors from phase 2 and used them to create a population of potential solutions. A potential solution is created by iterating through vector v and if an entry has a value of 1, then the corresponding entry in x is randomly chosen to have a value of 1 or 0, otherwise the corresponding value in x is unchanged. This is done N many times to create a population of N potential solutions, where N is a user input. After a population of solutions is created then each solution is evaluated for node violations in the same manner as in the Tabu Search by utilizing the arcs to node data frame. A node violation this time is defined as a node not being used twice, if a node is used once then this is a dead end, if a node is used more than twice then this is an intersection. There is a user input for node violations, so if a solution doesn’t meet the node violation requirement then it is deleted and a new one is generated and evaluated. This population generation procedure iterates until N potential solutions that all meet the node violation requirement have been created.

The next step in phase 3 is to create N competitions of F solutions. F is a user input for the competition size. Every solution in each competition is evaluated for two statistics, the number of closed loop tours used and the number of node violations. If there are node violations present then closed loop tours cannot exist and the value for the number of closed loop tours is infinity. The winner of each competition is the solution with the minimum number of closed loop tours, if all solutions in a competition have a value of infinity for this statistic then winner is the solution with the minimum number of node violations. If any of the winners used 1 tour then the Genetic Algorithm met its objective and is done. If a single tour solution wasn't found then the winner which used the minimal number of tours or node violations is saved as the best solution for this iteration.

The next step in phase 3 is to pick a certain percentage of the population, as defined by the user, to undergo cross sectioning. Solutions are randomly picked to be in the cross section group and then this group is randomly split into pairs. Then for each pair, a random number between 73 and 147 is generated, C times. The range of 73 to 147 represents the center 33% of entries in the decision vector x. The random value chosen in this range represents the position of the cut that will be made to cross section each pair. The value of C is a user input to determine the number of times cross sectioning must occur per pair. For every cut, either all entries to the left of the cut will be swapped between the pair, or all of the entries to the right of the cut will be swapped between the pair. If the cut was an odd number, such as the first cut, the third cut, and on so, then the entries to the left are swapped, otherwise the entries to the right are swapped. After all pairs have been cross sectioned then they are reunited with the rest of the population which did not undergo cross sectioning.

The final step in phase 3 is mutation. A certain percentage of the population, as defined by the user, undergoes mutation. Solutions are randomly picked to be in the mutation group, and then each solution is mutated M times by switching an entry from 1 to 0 or 0 to 1 if it is allowed to vary according to vector v. M is defined by the user. After all solutions in the mutation group have been mutated M times then this group is reunited with the rest of the population which did not undergo cross sectioning.

Overall the Genetic Algorithm iterates through competition, cross sectioning, and mutation events until a single tour solution has been found or the user defined time limit has been reached. If the time limit is reached before a single tour solution was found then the best solution from all solutions that were saved for each iteration is the best available solution. This best available solution is determined by having the minimal number of tours or node violations. The Genetic Algorithm is also designed to allow for multiple parameters to be tested, so before the Algorithm starts, it saves the original given population, that way if the time limit for one combination has been reached, then the next combination of parameters can be run with the same starting population.

Results:

Performance of Tabu Search:

The performance of the Tabu Search for each puzzle will be measured based on the total number of iterations and the total time in seconds required to satisfy the demands, as well as the final tour length. The final tour length with regards to the Tabu Search represents the total number of arcs that were used to just satisfy the demands.

The summary statistics on the total number of iterations, total time, and final tour length of the Tabu Search are shown below in Tables 1 and 2 for Slitherlink 1 and 2 respectively. These statistics were computed by running 25 Tabu Searches for each puzzle. The Tabu Search was able to solve the demand requirements of Slitherlink 2 in under half the time as compared to Slitherlink 1, on average. The tour length is not significantly different when comparing the puzzles. The best performance for each puzzle, based on minimum iterations, is plotted on the following page.

Table 3: Slitherlink 1 Tabu Search Tabular Performance

Measure	#. Iterations	Time (secs)	Tour Length
Min.	253	2.13	72
1st Qu.	435	4.05	73
Median	731	6.26	74
Mean	913.8	8.19	74.24
3rd Qu.	1148	10	75
Max.	3579	37.72	78

Table 4: Slitherlink 2 Tabu Search Tabular Performance

Measure	#. Iterations	Time (secs)	Tour Length
Min.	63	0.53	74
1st Qu.	151	1.11	75
Median	287	2.25	77
Mean	386.9	3.20	76.72
3rd Qu.	505	3.97	78
Max.	2162	20.70	79

The number of neighborhoods that needed to be re-visited at the end of each iteration of the Tabu Search when solving the demand requirements of Slitherlink 1 is shown below in Figure 2. After iteration 1, there are approximately 11 identified neighborhoods of demand not fulfilled. Throughout this graph there are random spikes increasing the number of neighborhoods to visit, meaning that the Tabu Search was converging on a poor solution. These spikes are due to demands not being satisfied, or node intersections existing. At iteration 150 the tabu lists are sufficient enough to decreases the variability in the number of neighborhoods that need to be visited. At iteration 200 the Tabu Search is starting to converge around 1 neighborhood visitations. Finally, at iteration 253 the Tabu Search has found a solution which meets demand requirements without node intersections.

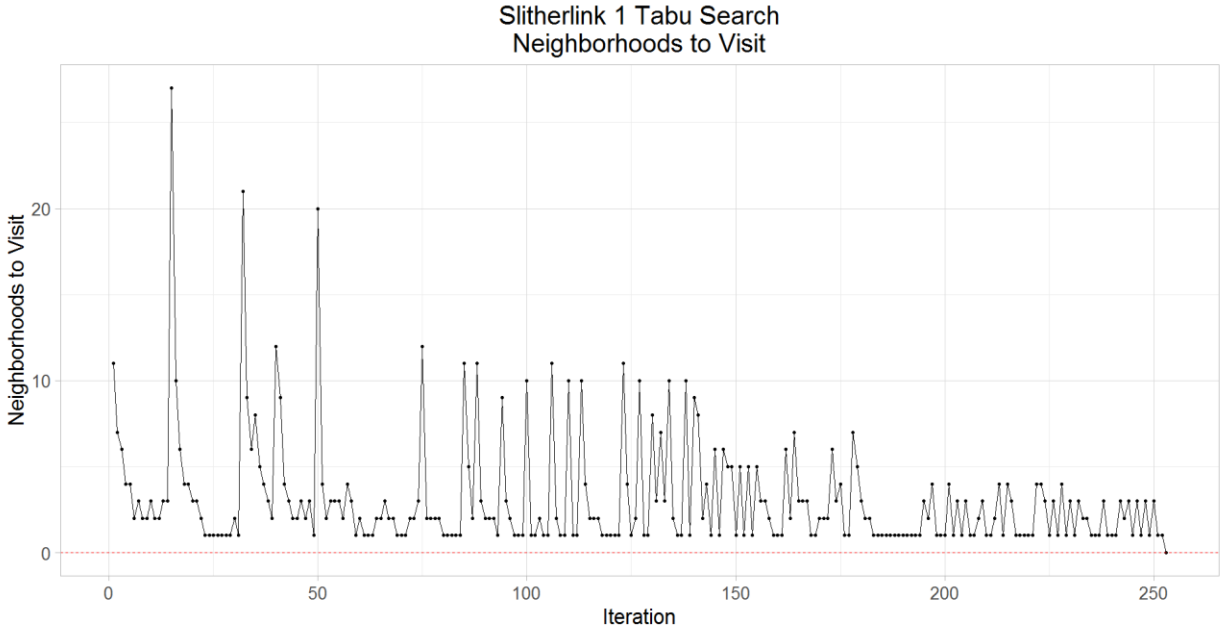

Figure 2: Slitherlink 1 Tabu Search Graphical Performance - Best Performance

The number of neighborhoods that needed to be re-visited at the end of each iteration of the Tabu Search when solving the demand requirements of Slitherlink 2 is shown below in Figure 3. This graphic gives a closer view on how truly random the spikes are. When comparing this graphic to Figure 2, it seems that the demand requirements of Slitherlink 1 are more difficult to satisfy than Slitherlink 2, which is supported by all summary statistics in Tables 3 and 4 previously.

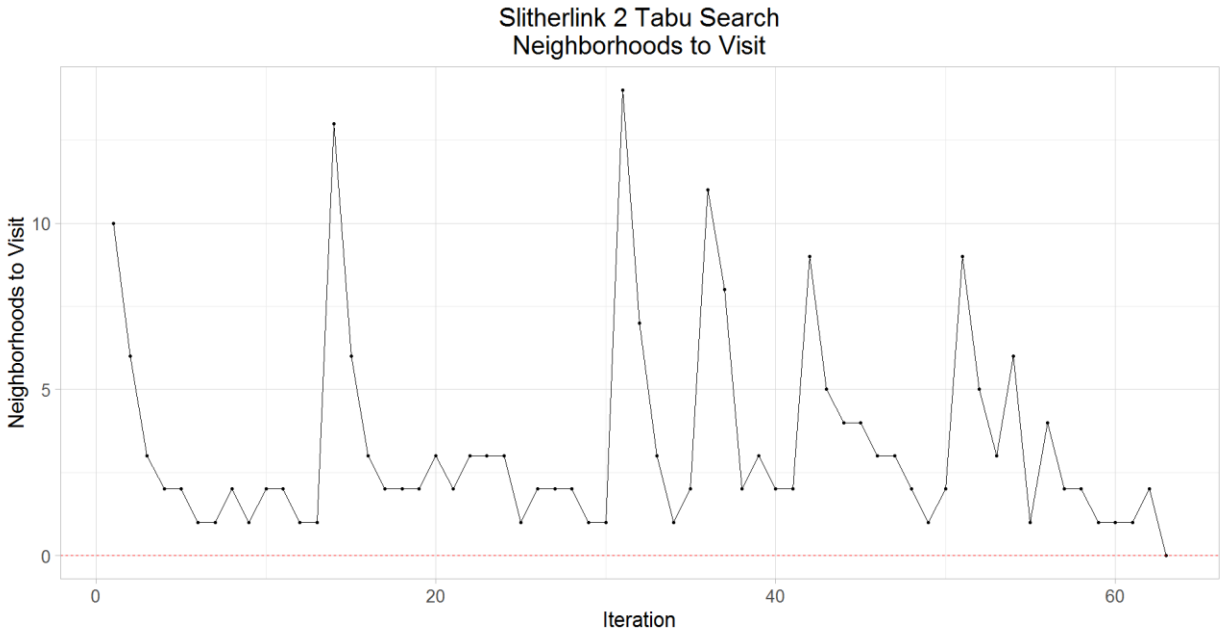

Figure 3: Slitherlink 2 Tabu Search Graphical Performance - Best Performance

When 16 different Tabu Search solutions for puzzle 1 were passed to the Genetic Algorithm and run for 12 hours, the Genetic Algorithm couldn't achieve any number closed loop tours. The cause of this issue was likely due to that there are many ways to satisfy the demands without node intersections that still prevent closed loop tours from being possible. Suggestions for improving the Tabu Search to allow for closed loop tours to be possible are discussed in the Future Improvements section. Therefore to truly see how well the Genetic Algorithm performs, the way arcs were used around demands in the solutions from the previous optimization approach were passed to the Genetic Algorithm to represent what would happen when the Tabu Search completes its objective better.

Performance of Genetic Algorithm:

The performance of the Genetic Algorithm for each puzzle will be measured based on the total number of iterations to solve, the total time in seconds to solve, the final tour length, the number of subtours found before the final solution, the number of tours in the final solution, and the estimated MIP % gap. The user input parameters for the Genetic Algorithm that were used will be shown first, and then the summary of the performance will be discussed.

The node violation requirements that were used for each puzzle to create the initial population of potential solutions are shown below in Table 5. The starting violations are the number of node violations present when just the demands are satisfied. The max violations is the node violation requirement that was used as the input parameter for the creation of the initial population.

Table 5: Genetic Algorithm Population Requirements

Puzzle	Starting Violations	Max Violations
Slitherlink 1	46	39
Slitherlink 2	38	38

There were 8 different combination of parameters prepared for the Genetic Algorithm as shown below in Table 6. The only parameters that varied were the number of competing solutions which was previously denoted as F, the cross section percentage, and the cross section cuts which was previously denoted as C. Populations size was previously denoted as N, and mutations was previously denoted as M.

Table 6: Genetic Algorithm Parameter Combinations

Run ID	Population Size	Time Limit (secs)	Competing Solutions	Cross Section %	Cross Section Cuts	Mutation %	Mutations
1	2000	5400	2	33.33%	1	5%	3
2	2000	5400	5	33.33%	1	5%	3
3	2000	5400	2	66.67%	1	5%	3
4	2000	5400	5	66.67%	1	5%	3
5	2000	5400	2	33.33%	3	5%	3
6	2000	5400	5	33.33%	3	5%	3
7	2000	5400	2	66.67%	3	5%	3
8	2000	5400	5	66.67%	3	5%	3

The statistics of the Genetic Algorithm are shown below in Tables 7 and 8 for Slitherlink 1 and 2 respectively. These statistics were computed by running up to 8 Genetic Algorithms for each puzzle, based on Table 6 above. The Genetic Algorithm was able to solve Slitherlink 2 in under 8 minutes and nearly solved Slitherlink 1 after 9 ¼ hours. The extreme difference in performance when solving each puzzle shows how the choice of input parameters significantly impacts the solution time. The total number of subtours found in Slitherlink 1 shows how difficult is was for the Genetic Algorithm to deviate away from multiple subtour solutions. The estimated MIP % Gap was not computed for Slitherlink 1 because a solution was not found. The estimated MIP % Gap for Slitherlink 2 is 0% because this solution is identical to the solution found by the optimization approach which also had a 0% MIP % Gap. The node violations and number of tours used by the best solution at each iteration of the Genetic Algorithm for all runs is plotted on the next page.

Table 7: Slitherlink 1 Genetic Algorithm Tabular Performance

#. Iterations	Time (secs)	Tour Length	#. Subtours Found	Final #. Tours	MIP % Gap
3719	32915.88	120	76	2	NA

Table 8: Slitherlink 2 Genetic Algorithm Tabular Performance

#. Iterations	Time (secs)	Tour Length	#. Subtours Found	Final #. Tours	MIP % Gap
56	308.72	110	0	1	0%

The following 8 plots in Figure 4 below represent the 8 runs indicated in Table 6. The data points are a light shade of red when no tours were used, and a light shade of blue when tours were used. In terms of Figure 4, the light shade of blue represents a solution with two tours. Run 7 was the only set of parameters that allowed tours to be found within the specified time limit, therefore the vast majority of the time and iterations indicated in Table 7 is due to runs 1 - 6. If this were to be run again, the parameter combinations in run 7 would be used to solve the puzzle.

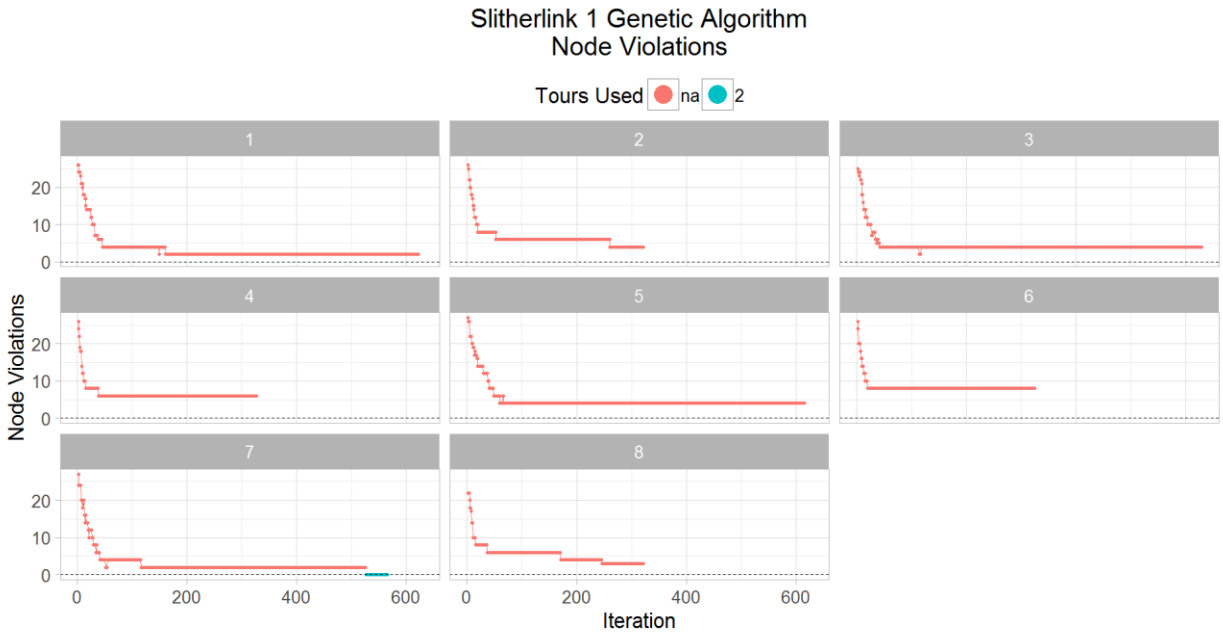

Figure 4: Slitherlink 1 Genetic Algorithm Graphical Performance

The first run was able to solve Slitherlink 2 so the other 7 runs were not tested. This graphic shows clearly how the Genetic Algorithm behaves, where improvements are made very quickly at the start and then begin to trail off with slow improvements stepping down towards a solution. When comparing the Figure 4 and 5 its becomes clear that small scale experiments should be run first to quickly estimate the better set of parameters, before letting the Genetic Algorithm solve for a solution.

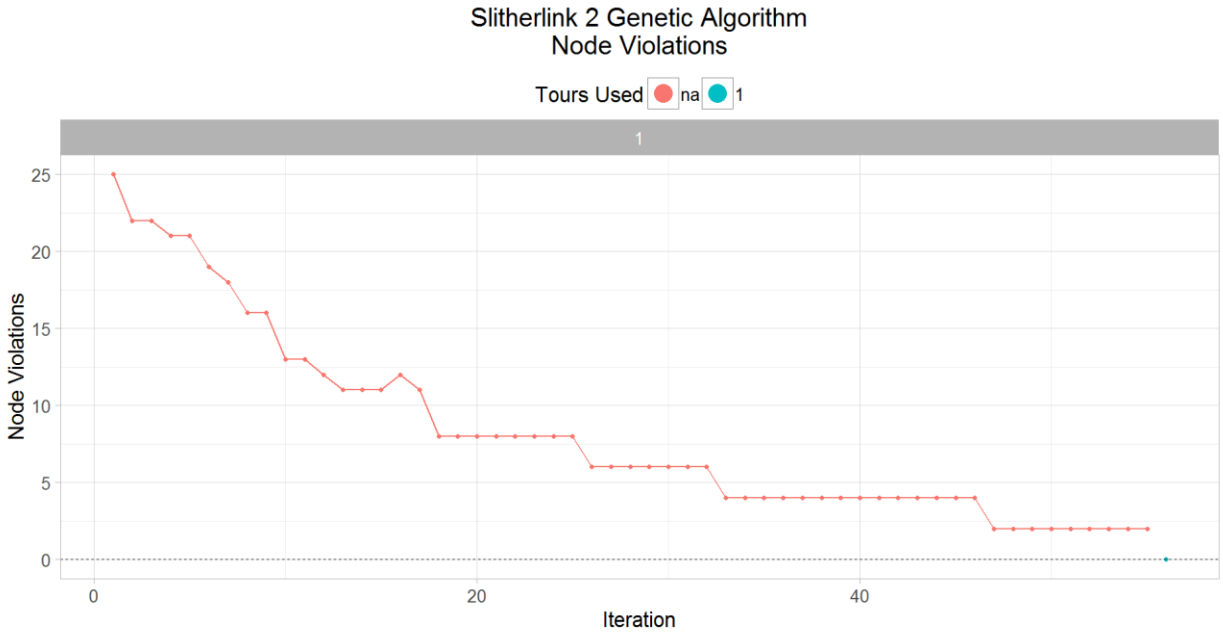

Figure 5: Slitherlink 2 Genetic Algorithm Graphical Performance

Comparison of Metaheuristics & Integer Programming:

The comparison of metaheuristics and integer programming will be measured based on the total number of iterations to solve, the total time in seconds to solve, the final tour length, the number of subtours found before the final solution, the number of tours in the final solution, and the estimated MIP % gap.

The comparison of methods for solving Slitherlink 1 and 2 is shown below in Table 9 and 10 respectively. These tables show a common pattern between Slitherlink 1 and 2 where Slitherlink 1 takes more iterations and time to solve due to getting caught in more subtours. The integer program outperforms the metaheuristics in time to solve significantly, but the integer program is running on a commercial solver and the metaheuristics is running on a student designed solver. The metaheuristics show a promising characteristic, particularly for Slitherlink 2, with significantly less iterations required than the integer program. If the metaheuristics solver were to be designed at the same level of quality as the integer program's solver, then the metaheuristics may outperform the integer program given the correct starting parameters.

Table 9: Slitherlink 1 Method Comparison

Method	#. Runs	#. Iterations	#. B&B Nodes	MIP % Gap	Tour Length	Time (secs)	#. Subtours Found	Final #. Tours
IP with cuts	4	1940	32	0	120	0.66	11	1
IP without cuts	6	14299	224	0	112	3.16	15	1
Tabu Search	1	913.8	NA	NA	74.24	8.19	NA	NA
Genetic Algorithm	7	3719	NA	NA	120	32915.88	76	2

Table 10: Slitherlink 2 Method Comparison

Method	#. Runs	#. Iterations	#. B&B Nodes	MIP % Gap	Tour Length	Time (secs)	#. Subtours Found	Final #. Tours
IP with cuts	2	688	0	0	110	0.36	3	1
IP without cuts	2	11324	513	3.64%	110	0.42	2	1
Tabu Search	1	386.9	NA	NA	76.72	3.20	NA	NA
Genetic Algorithm	1	56	NA	0%	110	308.72	0	1

The solution for Slitherlink 1 found by the metaheuristics is shown on the left side of Figure 6 and the integer program solution is on the right. These solutions are 98.33% similar with a difference of just two arcs. The subtours that the metaheuristic got stuck in before reaching the time limit were subtours found when building the subtours constraint for the integer program. Despite the metaheuristics not solving this puzzle, the solution is still useful for a user that can easily see that turning off the (104, 115) and (105, 116) arcs, and turning on the (104, 105) and (115, 116) arcs would yield a solution, which is also identical to the integer program. The solution for Slitherlink 2 was identical for both methods, and is shown below in Figure 7.

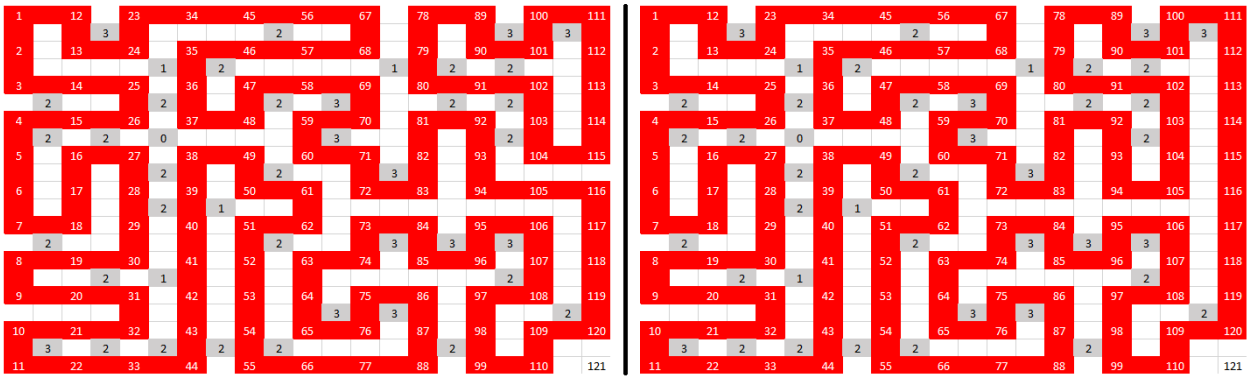

Figure 6: Slitherlink 1 Solution Comparison

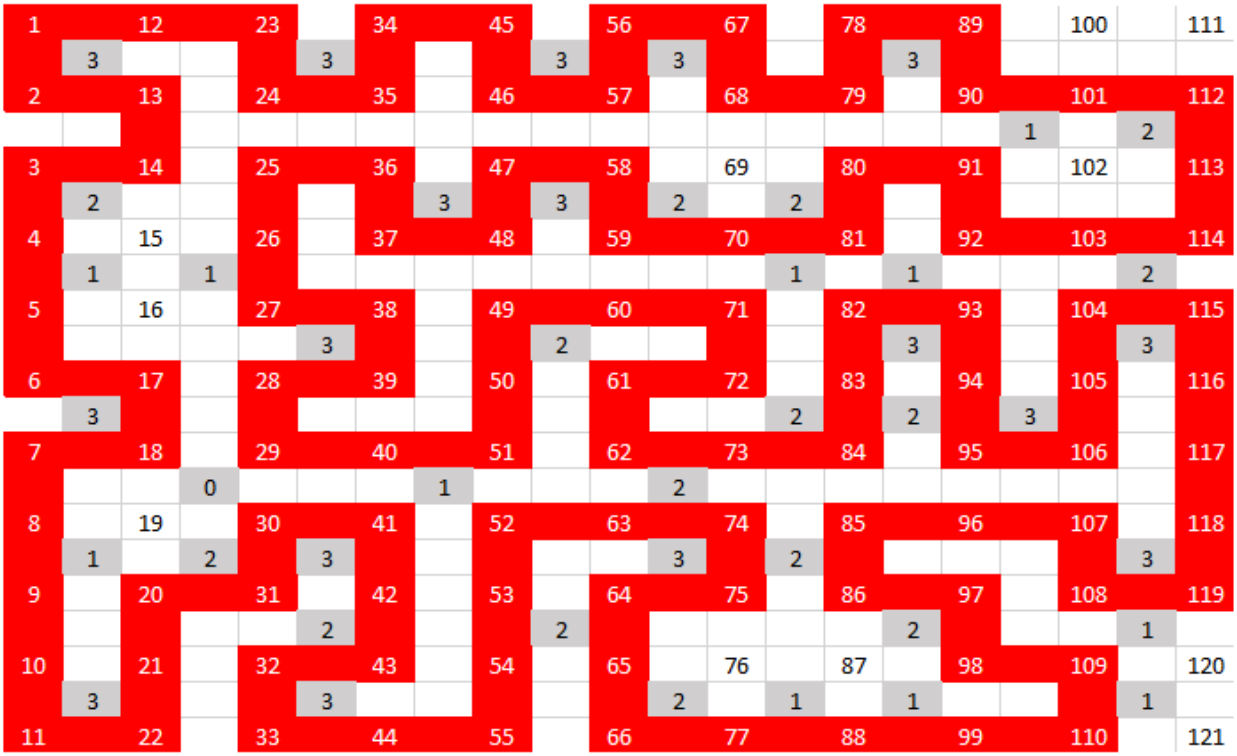

Figure 7: Slitherlink 2 Solution Comparison

Conclusion:

Metaheuristic Limitations:

There is high customization required to properly apply a metaheuristic to a problem. Concessions were made to simplify the code which lead to weakness in the Tabu Search. The Tabu Search was built to make decisions about nodes and arcs that fulfill demands, without consideration for how it would impact arcs and nodes that are not around demands. Reiterating an early point, these demands are satisfied at random by searching a grid of four arcs. It is likely that the majority of demands fulfilled by the Tabu Search and passed to the Genetic Algorithm cannot be solved for a single closed loop in their current form. These decisions by the Tabu Search become fixed in the Genetic Algorithm, so they cannot be changed. The Genetic Algorithm requires fine tuning for it to meet its objective and this tuning is very different across different instances of the same problem type.

Future Improvements:

The focus on improvement for the Tabu Search would be to add more end conditions that must be met, besides demand satisfaction and node intersection prevention. The rules that would be needed are the rules shown in Figure 8. These rules may have a positive impact on the Tabu Search considering the impact they had when applied to the integer program. Implementing the additional rules is expected to increase iteration count as well as computational time since the Tabu Search is required to satisfy more end conditions before converging onto a proper solution.

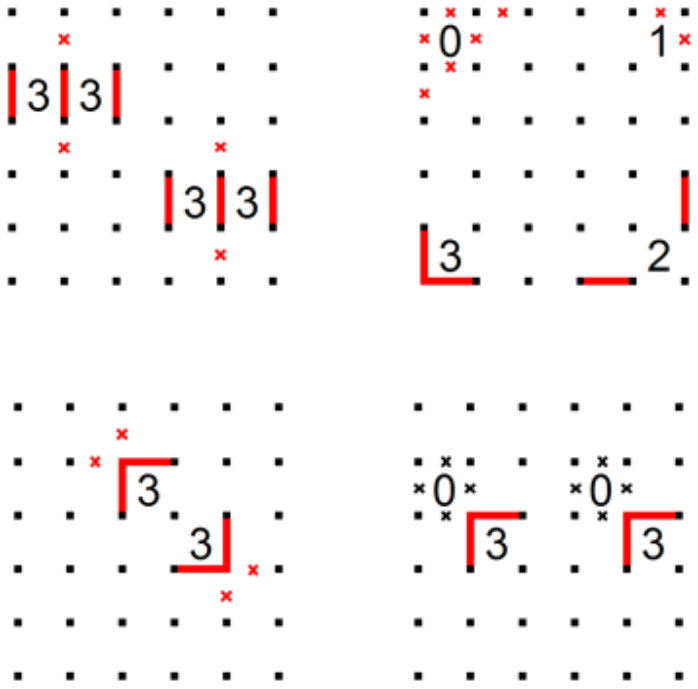

Figure 8: Slitherlink Conditions for Tabu

The focus on improvement for the Genetic Algorithm would be to design an experiment with the goal of finding good input parameters without having to run many iterations. An initial design could be choosing a limit on the number of iterations the genetic algorithm is allowed to run. Vary the cross section and mutation parameters such that the number of unique designs doesn't exceed 10. Compare the results at the end of the iteration limit. The combination that has a consistent number of minimal node violations will be chosen to run in the full model.