ICMA395 Project Report

Kan-Anek Tantitayapong 6380612

Contents

1 Introduction

1.1 Magic Squares

A magic square of order n is an $n \times n$ matrix with n^2 entries such that each entry is distinct, and the sum of all the rows, columns and diagonals are the same. Furthermore, if only the rows and columns have the same sum, then it is called a semi-magic square. In most definitions, we require that each entry a_{ij} belongs to the set $\{1, 2, \ldots, n^2\}$. Below is an example of a magic square of order 3:

4	9	2
3	5	7
8	1	6

Now, for small n, an exhaustive search is reasonable. For example, to find a magic square of order 3, you simply look through all the $9! = 362880$ possible permutations. However, it is not hard to see why this is less feasible for large n. As the number of possible permutations is $(n^2)!$ which grows rapidly with n. Hence, heuristic methods are useful to search for even one feasible solution. Cruz and Grandchamp [2], in a 2012 paper attempted local search to solve this exact problem by starting with a random square. In short, they began by finding a semi-magic square, defining the cost function of a square by the difference of the desired sum and the row-wise and column-wise sums. For the project, I may attempt to find a magic square of high orders, and compare the algorithms taught in class with smaller ordered squares.

1.2 Variants

There are various other variants of the magic square problem, like the magic square of squares, which is where after squaring the entries, the sums are equal. Over the infinite field of integers, this has not been done yet, although nobody has yet to prove its impossibility either. There is also the notion of magic square of squares in finite fields. That is, $\mathbb{Z}_p$ where p is prime, where the result of the addition is evaluated modulo p. Cain [1], in a 2019 paper, focused on solving for 3×3 magic squares of squares in finite fields. With some algebra over the complex numbers, Cain introduced an algorithm which could search for the entries of a magic square of squares in an arbitrarily chosen finite field $\mathbb{Z}_q$. The results showed that some finite fields have multiple solutions (up to scaling), and that the higher the order of the field, the more abundant magic squares are. In the context of this project, it may be interesting to test the algorithms taught in class and see how they fair against the algebraic method, and perhaps try to find 4×4 or 5×5 magic squares in finite fields. It also may be insightful to try to go through all the algebraic steps then re-implement the algorithm mentioned in the paper.

2 Implementation and Results

2.1 Framework

The method of finding a magic square of order is the same method as in the paper by Cruz and Grandchamp [2]. We start by finding a semi-magic square. Then, we restrict ourselves to making moves that preserve row and column sums in hopes of achieving our goal. The method is more viable as the order n of the square increases, as the size of the search space grows more than exponentially with n.

2.1.1 Cost Functions

To begin our first phase of our search, we first need to find the target which we want each row and column to sum, S. We note that the sum of every entry of the square is $\frac{n^2(n^2+1)}{2}$, i.e. the sum of the numbers 1 up to n^2. This also represents the sum of n rows. Thus, it follows that the sum of each row, S (and consequently each column) is

$$S := \frac{n^2(n^2+1)}{2} \cdot \frac{1}{n} = \frac{n(n^2+1)}{2}$$

Let $\text{cost}_1(H)$ be the cost function concerned with finding a semi square of order n, evaluated on a square H. It is defined as

$$\text{cost}_1(H) = \sum_{i=1}^{n} |S - \sum_{j=1}^{n} H_{ij}| + \sum_{j=1}^{n} |S - \sum_{i=1}^{n} H_{ij}| \tag{1}$$

In words, it is the sum of the magnitude of the differences between S and the sum of each row and column. The optimal value of this cost function is 0, which is achieved when we have found a semi-magic square. This value 0 must necessarily be achieved if we want to find a magic square. Once we have a semi-magic square we can begin our second phase. Let $\text{cost}_2(H)$ be the cost function concerned with finding a magic square of order n, evaluated on a square H, given that $\text{cost}_1(H) = 0$. It is defined as

$$\text{cost}_2(H) = |S - \sum_{i=1}^{n} H_{ii}| + |S - \sum_{i=1}^{n} H_{i,n-i+1}| \tag{2}$$

It is simply the sum of the two diagonals of H. Similarly, the optimal value of this function is 0, which is when we have a magic square.

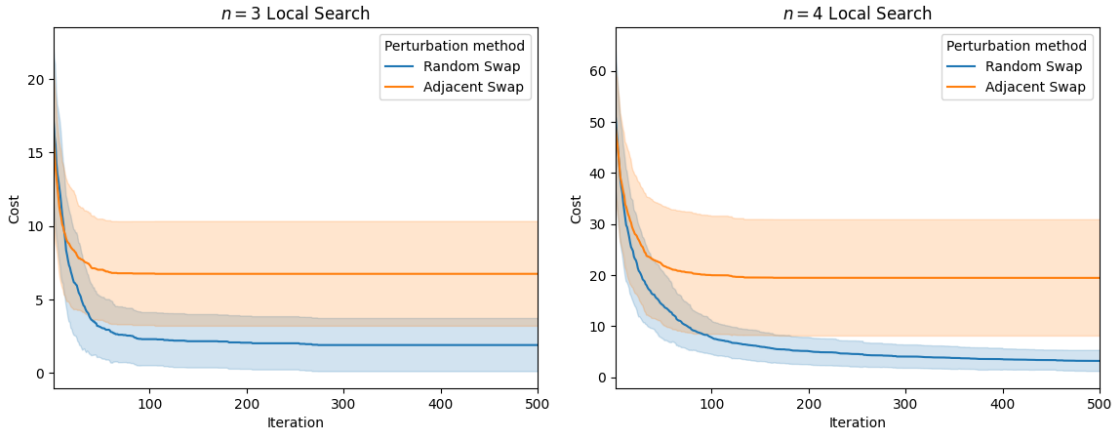
2.1.2 Perturbation Methods

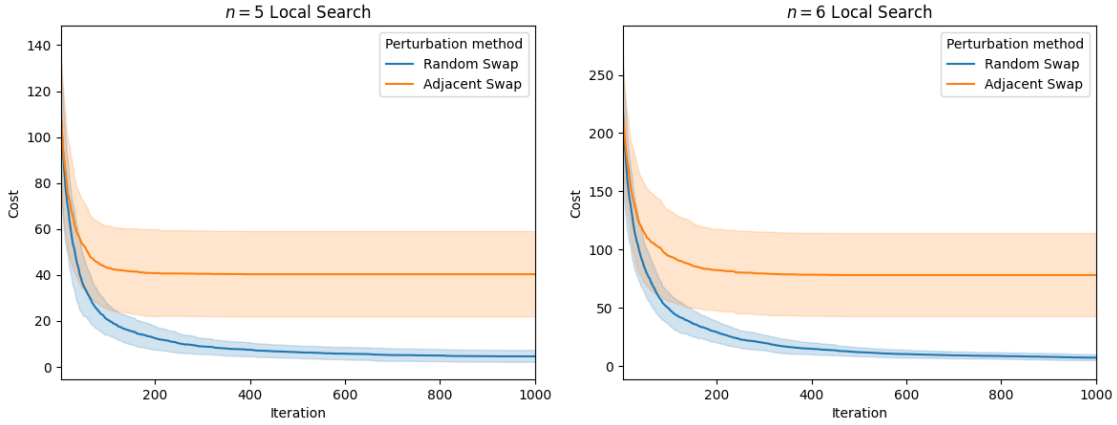
For algorithms of the local search family (Greedy, SA, TS), two perturbation methods are used during the first phase of searching, Random Swap and Adjacent Swap. As the name suggests, the former is swapping two random entries of the square, and the latter is swapping two random adjacent entries.

2.2 Local Search Family

For algorithms of the local search family, 100 trials of 10,000 iterations were performed for each, with $n \in \{3, 4, 5, 6\}$, and using the cost function $\text{cost}_1(H)$ (for now). The choice of 100 trials was to simplify the convergence percentage.

2.2.1 Greedy Local Search

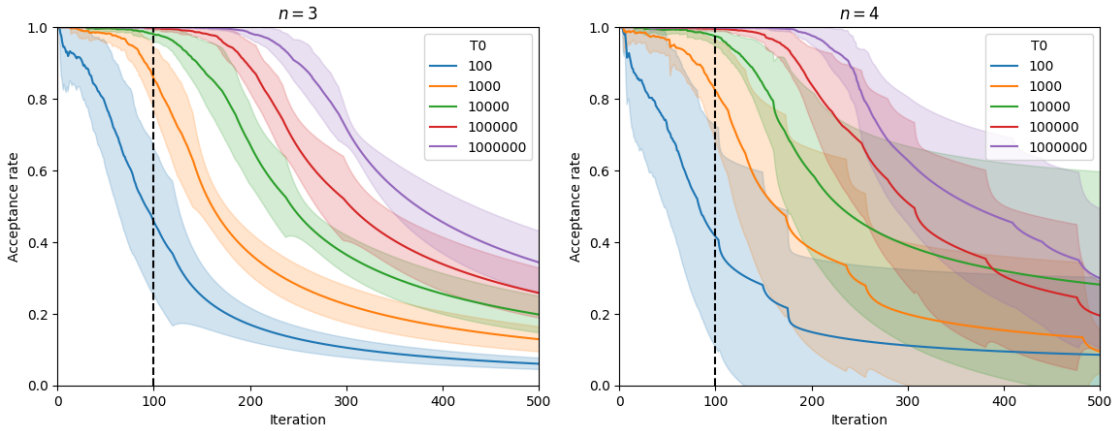


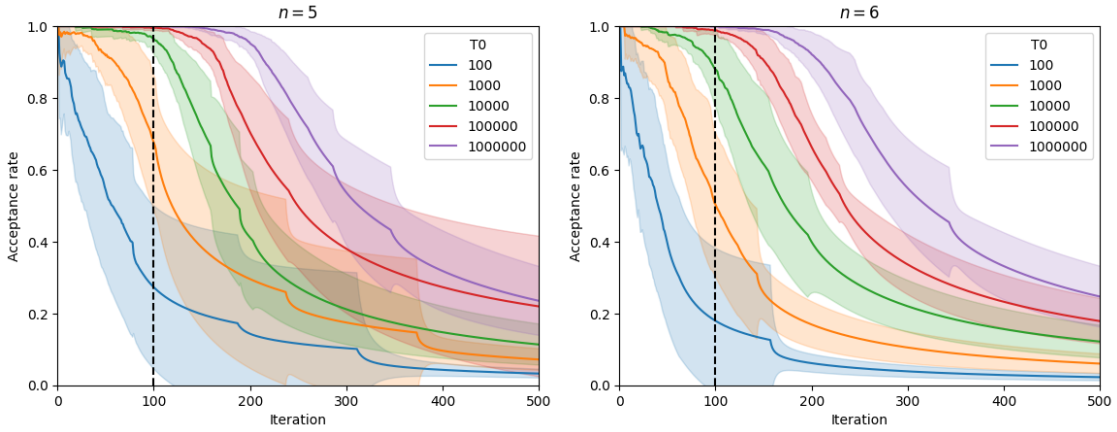


We see that in general Random Swap reduces the cost faster than Adjacent Swap. This may be because it is more unrestricted and allows the algorithm to explore more of the search space. For $n = 3$, the Local Search algorithm had a 44% convergence rate. For orders 4, 5, and 6, the convergence rate was 13%, 5%, and 1% respectively. Note that in this case, convergence means a semi-magic square was found. All convergences occurred when Random Swap was used. Also, in some cases, the corresponding semi-magic square happened to be a complete magic square. This occurred for $n \in \{3, 4, 5, 6\}$. Moving the semi-magic squares found into the second phase did not help turn into magic squares, meaning the only valid squares we are able to find were found by chance during the first phase. This may be because the process for finding 'effective' moves is not efficient enough to get to the solution in 10,000 iterations.

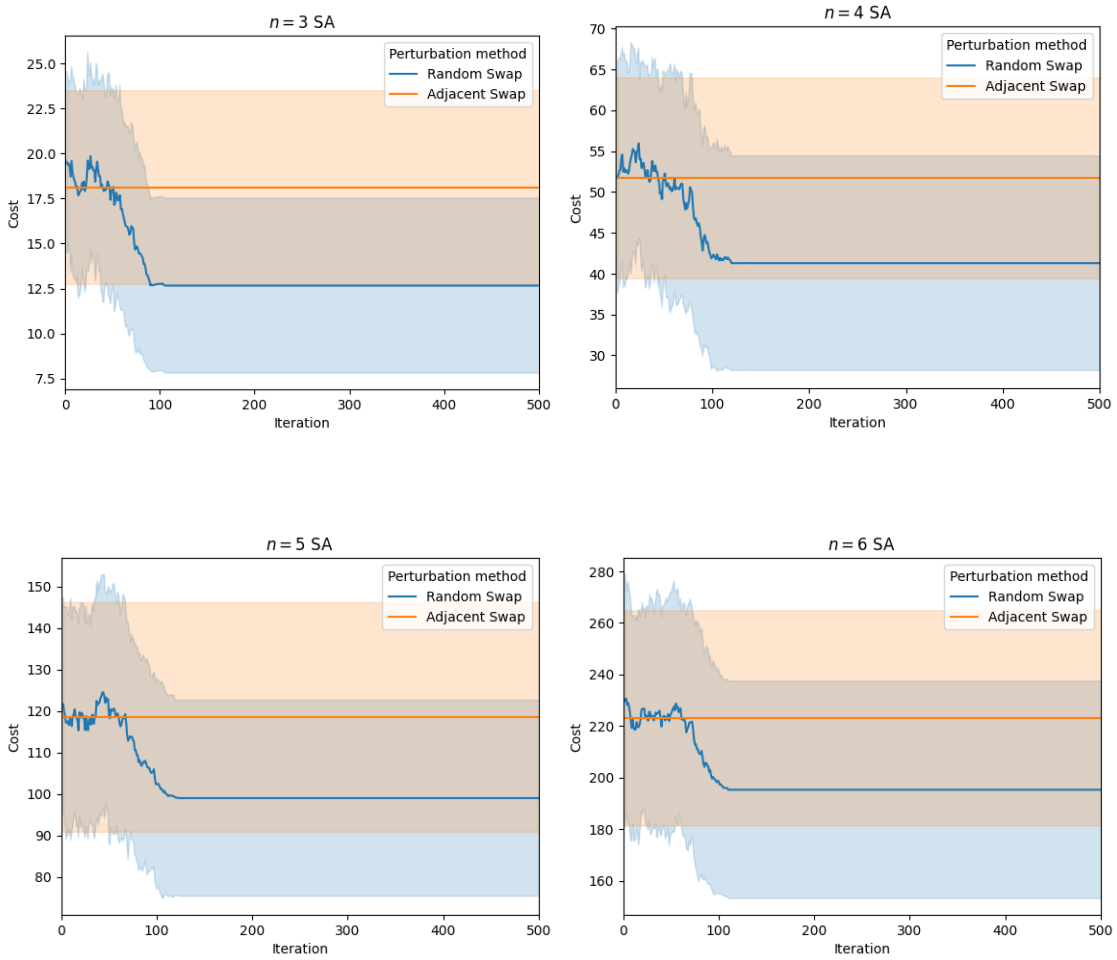
2.2.2 Simulated Annealing

For the fixed cooling rate $\alpha = 0.95$, we attempted to find an appropriate starting temperature. We want the acceptance rate (the probability of accepting worse solutions) to be around 50% to 70% at the beginning of our search. 20 trials of 10,000 iterations (Random Swap) were performed for $T_0 \in \{100, 1000, 10000, 100000, 1000000\}$ and for each $n \in \{3, 4, 5, 6\}$. Note that we choose define the beginning of the search to be the first 1% of the search. For 10,000 iterations, this would be the first 100 iterations.





Based on the plots are our definition of the beginning of the search, we choose $T_0 = 100$ for $n = 3$, $100 < T_0 < 1000$ for $n = 4$, $T_0 = 1000$ for $n = 5$ and $n = 6$. Below are the performance plots for SA.



We see that the algorithm tends to get stuck at some local minimum. The convergence rate is 0% for every $n \in \{3, 4, 5, 6\}$. This is very surprising, and indicates that SA is not a good choice for solving this problem.

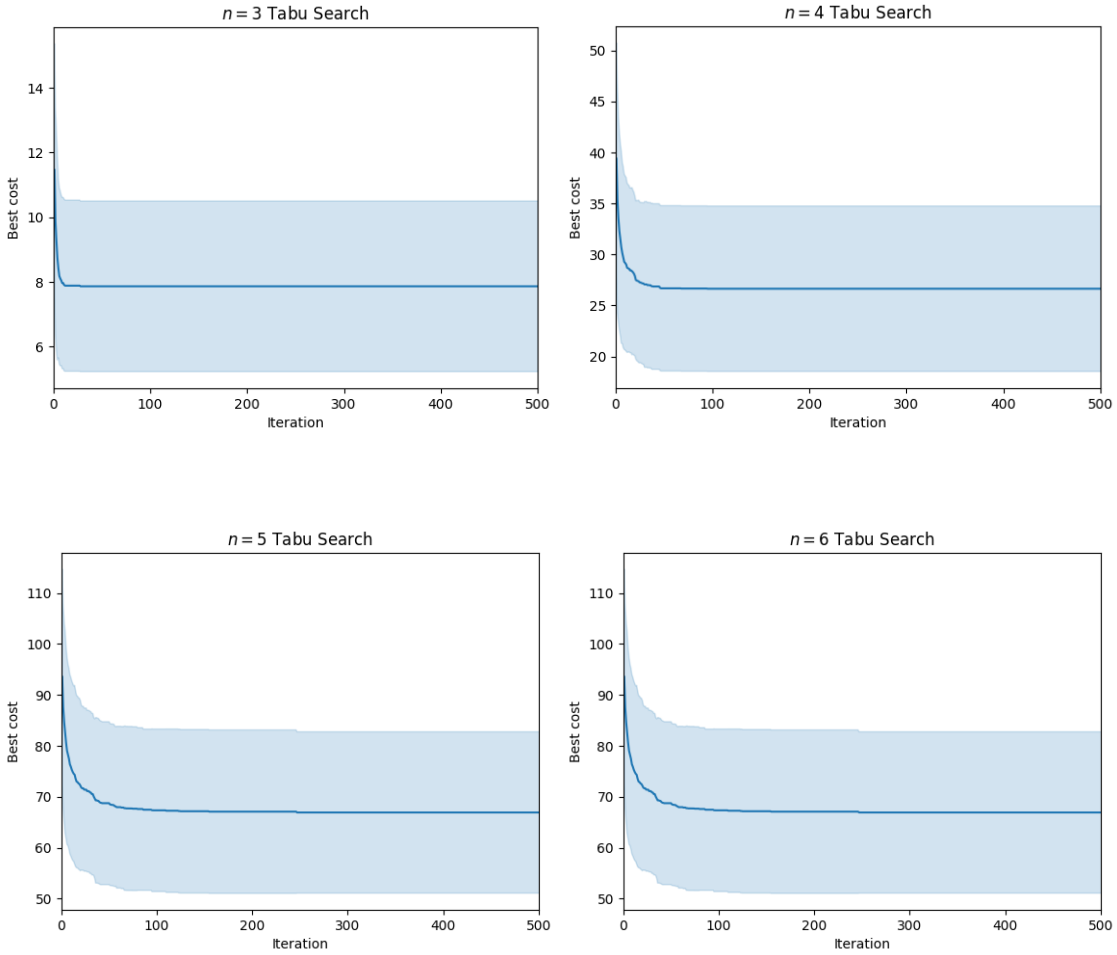
Now, it is most likely the case that our choice of T_0 can be improved upon and that we may need to increase the number of iterations.

2.2.3 Tabu Search

The tabu list stores the previous k pairs of elements that were swapped. The aspiration criterion is the default, that is, we override a tabu move if the resulting solution is better than the best solution so far. Note here that the perturbation method is exclusively Random Swap, as we've established that it outperforms Adjacent Swap. The long term memory component is the information about the frequency of moves. More specifically, if the swap (i, j) was made to perturb square H to square H^*, the cost function can be expressed as follows:

$$\epsilon(H^*) = \begin{cases} \text{cost}_1(H^*), & \text{if } \text{cost}_1(H^*) \leq \text{cost}_1(H) \\ \text{cost}_1(H^*) + 5 \times \text{freq}(i, j), & \text{otherwise} \end{cases}$$

where $\text{freq}(i, j)$ is the number of times the swap (i, j) was made so far. The algorithm uses this long term component to diversify the search every 50 iterations. Below are the performance plots for $k = 4$ and $|V^*| = 10$.



The convergence percentage for is 0% for every $n \in \{3, 4, 5, 6\}$, which is again surprising since we expect the

modified local searches to perform better than the simple greedy type. Now, note that the only outputs that we care about is the ones obtained from greedy local search, as SA and TS did not even manage to get a semi-magic square. Local search actually managed to find semi-magic squares for n up to 9. We will try to improve on this changing the way we perform phase 2 of our search (see **2.3.2**).

2.3 Genetic Algorithm

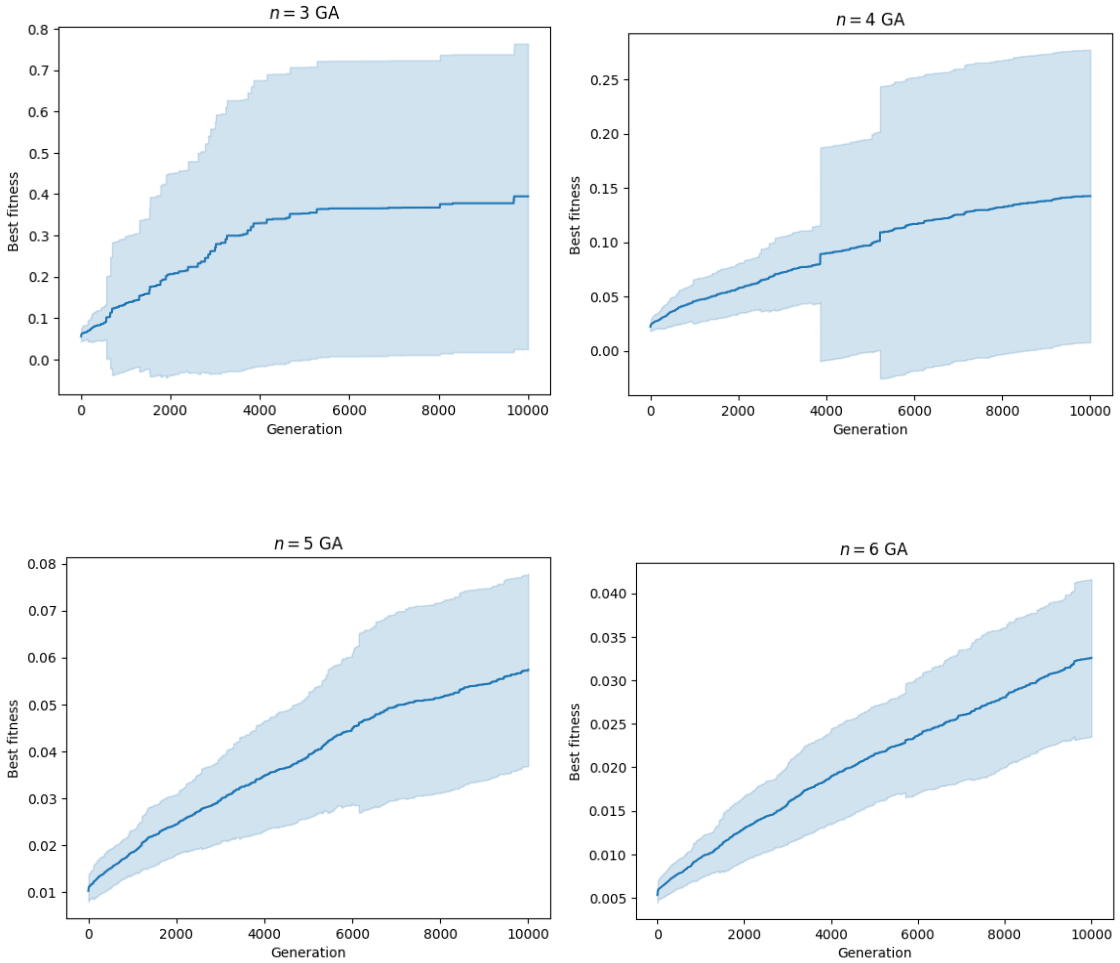
The representation of an individual is a string of n^2 numbers (e.g. '78245613'). The fixed parameters were $p_c = 1.0$ and $p_\mu = 0.03$ with a population size of 10 and PMX for crossover. The fitness function is

$$f(H) = \frac{1}{1 + \text{cost}(H)}$$

where $\text{cost}(H) = \text{cost}_1(H) + \text{cost}_2(H)$. We have combined the first phase and the second phase to see if GA would be able to produce magic squares in just on run.

2.3.1 Results

Below are the performance plots of 100 trials and 10,000 generations of GA for $n \in \{3, 4, 5, 6\}$:



The convergence percentages were 26%, 2%, 0% and 0% for $n = 3, 4, 5,$ and 6 respectively. This time, convergence means finding a complete magic square (when the fitness function achieves a value of 1). GA with combined cost functions also didn't seem to do the job.

2.3.2 Using GA for Phase 2

What we will do in this section is we are going to take semi-magic squares (of the same order) achieved from the previous parts, append them into a generation, then have that generation by the starting generation of GA with fitness function

$$f(H) = \frac{1}{1 + \text{cost}_2(H)}$$

with the same fixed parameters as mentioned above. Recall that $\text{cost}_2(H)$ is the cost function where we are only concerned with the diagonal sums. Unfortunately, this did not result in magic squares of order higher than 6, potentially because the crossover disrupts the schema corresponding to a semi-magic square and simply prioritizes the diagonal sums as time goes on. However, it did manage to increase the rate at which the fitness function increases.

3 Conclusion

For the problem of finding magic squares, the algorithms were only able to achieve up to a square of order $n = 6$. This may be due to the astronomically large search spaces and our iteration limit. Interestingly, all the valid squares came from the local greedy search. This may be because it is in a sense less restrictive than say Tabu Search or Simulated Annealing, and its true randomness benefits the search in that we may get lucky given a 'good' starting state. All in all, a heuristic approach may not be the most optimal way to solve this problem. An algebraic approach of constructing a square may be more practical.

References

[1] O. Cain. Gaussian integers, rings, finite fields, and the magic square of squares, 2019.

[2] E. Cruz and E. Grandchamp. Heuristic method to find magic squares. pages 119–123, 12 2012.