Genetic Algorithms and the N-Queens Problem

By
Eliott Chapuis

A thesis submitted in partial fulfillment of the requirements for graduation with

CARROLL SCHOOL OF MANAGEMENT HONORS

BOSTON COLLEGE
MAY 2016

Abstract

This paper demonstrates the various methods in which genetic algorithms may be used to solve the N-Queens problem, an exponentially complex constraint satisfaction problem. A variety of probability methods were examined, including linear and exponential fitness, as well as linear and natural rank. Additionally, the effects of partially mapped crossover, a crossover operator, were tested against each of the probability methods. The outcomes for each combination of probability method and crossover operator are shown and conclusions are drawn from the results. The linear rank method in combination with partially mapped crossover proved to be the most effective combination of assumptions, generating results the fastest for up to 100-Queens. Finally, further areas of research are proposed to expand upon this paper's findings and the effectiveness of genetic algorithms to the N-Queens problem.

Table of Contents

I. Introduction

This paper demonstrates the effectiveness of genetic algorithms on the N-Queens problem. In order to get a better understanding of what this means, it is first necessary to introduce both the problem itself as well as the advantages of using an evolutionary rather than a programmatic approach to constraint satisfaction problems (CSPs) such as the N-Queens Problem. By using the N-Queens problem, this paper aspires to generalize the effectiveness of genetic algorithms to other constraint satisfaction problems, although other problems are not formally explored.

Genetic Algorithms

True to their name, genetic algorithms, also called evolutionary algorithms or GAs, have their roots in biology. The basic principles of GAs were first introduced in 1975 by Holland in *Adaptation in Natural and Artificial Systems* [7]. With the creation of the first GAs, Holland took Darwinian evolution and DNA research and applied it to computational search tasks. GAs perform search tasks using chromosomal crossover and genetic mutation. Simply put, genetic algorithms model the process of DNA replication, where two parent chromosomes reproduce to create child chromosomes with a mix of genes from the parents. Additionally, after Darwin's evolutionary theory of "survival of the fittest", each chromosome is evaluated according to their structure and assigned a fitness value, where higher fitness values are more likely to be selected for reproduction [8]. Over the course of many crossovers, genetic algorithms yield child chromosomes with higher and

higher fitness values until some level of optimization is achieved. In this way, genetic algorithms can be used to solve a variety of complex problems, such as CSPs.

Although genetic algorithms are versatile, they are best suited for complex problems with many solutions. Specifically, genetic algorithms are better suited for complex problems because there is a large computational overhead associated with the initialization of genetic algorithms [7]. Moreover, they perform best when there are many possible solutions due to the randomness associated with genetic operators. With fewer solutions, it becomes increasingly difficult for genetic algorithms to converge to a desired value through genetic operators and mutation.

While faced with several limitations, the incentive to research genetic algorithms is that they have the potential to massively reduce computational time of complex problems [1]. Some fields where genetic algorithms have been used effectively include network routing [5] and airplane design [7]. A major advantage of genetic algorithms is that while the algorithm is well defined, the process in which is converges to a solution is inherently random. Therefore, solutions derived by genetic algorithms are often unique and insightful. This paper aims to better understand why these generalizations are true.

N-Queens Problem

Originally known as the 8-Queens problem published by the chess composer Max Bessel in 1848, the CSP was later adapted to support any number of queens, becoming the N-Queens problem [11]. The problem is simple: given a chessboard with N columns and N rows, find a way to organize N queens on the board without

having any queens conflicting another queen. A conflict between queens occurs when two queens are on the same diagonal or row. The elegance of the problem is demonstrated by its simplicity and its potential complexity. A board with N-Queens has N^2 positions to place queens in. Assuming that no two queens are in the same column, there are N^N possible arrangements of queens and N! arrangements if you assume each queen also falls on a unique row. With high values of N, it becomes impossible to go through every orientation of queens in order to find solutions. For this reason, the N-Queens problem presents an interesting and challenging problem that is suitable for genetic algorithms. Figure 1 demonstrates the significant increases in complexity with each additional increase in N, the number of queens.

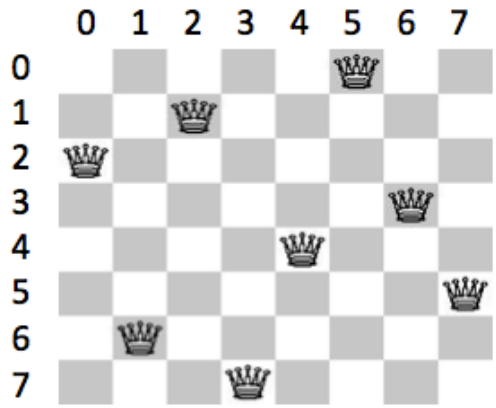
Figure 1: Number of Arrangements of Queens for Varying N

		Number of Arrangements of Queens		
N	Number of Solutions	All Combinations	Unique Cols.	Unique Rows & Cols.
1	1	1	1	1
2	0	12	4	2
3	0	504	27	6
4	2	43680	256	24
5	10	6375600	3125	120
6	4	1402410240	46656	720
7	40	4.32939E+11	823543	5040
8	92	1.78463E+14	16777216	40320
9	352	9.4671E+16	387420489	362880
10	724	6.28157E+19	10000000000	3628800
11	2680	5.09638E+22	2.85312E+11	39916800
12	14200	4.96338E+25	8.9161E+12	479001600
13	73712	5.71414E+28	3.02875E+14	6227020800
14	365596	HUGE NUMBER	1.1112E+16	87178291200
15	2279184	HUGE NUMBER	4.37894E+17	1.30767E+12

The added benefit of the N-Queens problem is that it is inherently very simple both to understand and to implement. Traditionally, the chromosomes used in genetic algorithms consist of a string of characters or numbers, and the N-Queens

problem lends itself well to this format [8]. Figure 2 gives a more concrete illustration of this description with a solution to the 8-Queens problem and its respective encoding.

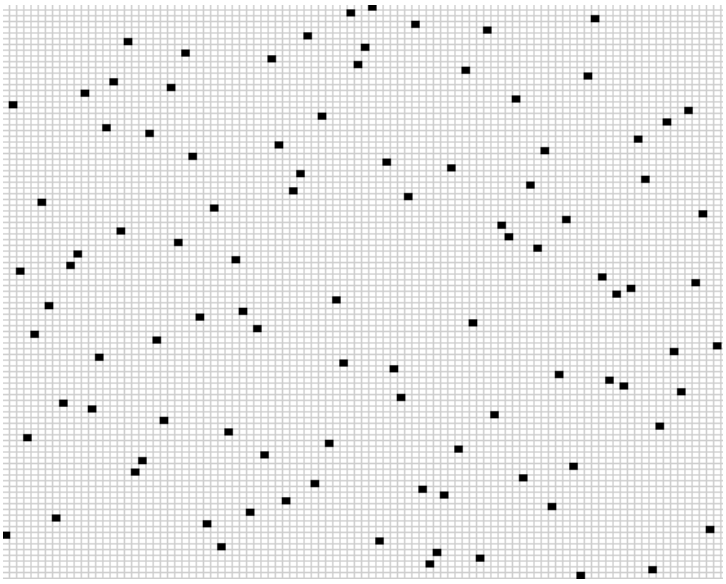
Figure 2: Example 8-Queens Solution



Coordinates: {(2,0), (6,1), (1,2), (7,3), (4,4), (0,5), (3,6), (5,7)}
Encoding (Columns implicit): {2, 6, 1, 7, 4, 0, 3, 5}

Each number in the encoding represents a row number, and the sequence in which the numbers appear represent the respective columns. In this way, the first queen in the first column is at position 2, the second queen in the second column is at position 6, and so on. For the 8-Queens problem, row numbers range from 0-7, inclusive, in order to map the state encoding to all eight rows of the chessboard. This method of encoding is the simplest to understand and is also highly scalable due to the fact that each additional queen can be placed in N possible position, but requires only a single additional integer for encoding. Therefore, the N-Queens problem is ideal for testing genetic algorithms, as the problem can easily be expanded to be more complex. An illustration of a solution for the 100-Queens problem is shown below, along with its respective encoding.

Figure 3: Example 100-Queens Solution



Encoding: {92, 17, 46, 75, 57, 34, 52, 89, 69, 45, 43, 15, 70, 61, 21, 13, 39, 6, 81, 79, 22, 58, 72, 14, 41, 8, 26, 54, 90, 35, 94, 74, 44, 53, 88, 56, 78, 9, 24, 86, 32, 29, 5, 83, 19, 76, 51, 62, 1, 10, 7, 0, 93, 27, 63, 68, 33, 3, 84, 97, 95, 85, 28, 77, 11, 55, 96, 4, 71, 38, 40, 16, 82, 31, 42, 25, 87, 64, 37, 80, 99, 12, 2, 47, 65, 50, 66, 49, 23, 30, 98, 73, 20, 60, 67, 18, 48, 36, 91, 59}

II. Related Research

Although this paper aims to present a new perspective on the functionality of a variety of genetic operators and probability methods, many of the key aspects of genetic algorithms have already been thoroughly researched. The origination of genetic algorithms was largely based upon their ability to generate solutions where no good heuristic is available or where the problem space is too complex to solve using traditional search methods [7]. The majority of research conducted around genetic algorithms has involved either an assessment of the viability of GAs for specific optimization problems and the advantages of using GAs over other search methods.

With regards to the discussion on the viability of GAs for specific search problems, the results are mixed. Depending upon the problem, genetic algorithms may not be appropriate, and [4] claims that non-heuristic based GAs perform poorly for the N-Queens problem. In other words, the structure of the algorithm is not sufficient for efficient solution discovery; a heuristic must be added in order to improve the speed at which solutions are generated. Otherwise, genetic algorithms would perform only slightly better than random search, and other search methods could calculate solutions faster. Fortunately, the heuristic functions available for the N-Queens are widely known; [2] offers an objective function that operates based on diagonal conflicts between queens and [8] illustrates a method involving rows and columns.

With a viable heuristic method used to generate descriptive fitness values, GAs can be shown to perform significantly better than other traditional methods. [3] demonstrates how the elements of genetic crossover alone contribute to a better than random solution convergence. Additionally, [10] compares the advantages and disadvantages of genetic algorithms when compared to backtracking. The study shows that, while GAs scale linearly with an increase in the number of queens, backtracking scales exponentially to the point where backtracking is no longer viable at higher values of N. This is one of the distinct advantages of GAs, as they scale well with higher complexity problems and are fairly versatile to a range of problems [3]. Again, however, these results are largely predetermined by the kind of problem evaluated as well as the heuristic functions used. For example, [6] finds that GAs work best for discontinuous problem spaces, but perform less well than

other traditional methods for local extrema discovery. In short, GAs, while somewhat versatile must be used carefully within the context of any problem, including the N-Queens problem, in order to effectively generate solutions.

III. Implementation

Given the complex nature of genetic algorithms (GAs), much care was given to the implementation used for this study. Java was the programming used, as its object-oriented nature complemented the structured design of the GA. While other programming languages and techniques may have improved computational performance, the purpose of this study is largely informative, and insight was favored over performance. Therefore, instead of using computational time as a metric for performance, crossover rates were used instead. Specifically, the methods that yielded solutions with the least number of crossovers were deemed most effective. Further detail on object design, genetic operators, and probability methods will be discussed in the following sections.

Object Design

For the purpose of maintaining a clear sense of order and organization within the algorithm itself, several key objects were designed. The base object, the State, represents the individual chromosomes being reproduced. Many states were grouped together to form a population, also known as a generation or epoch. Each test iterated over many generations until either a capped number of generations was reached or a solution was found. The methods in which the tests were

performed were dictated by a set of assumptions, which were manipulated to produce a variety of insightful results.

The core object of the GA that underlies all further discussion is the problem state, or the encoding of each hypothetical chessboard in the N-Queens problem. In more biological terminology, the primary objective of the state is to maintain a representation of each chromosome used for reproduction. The full representation of these chromosomes includes the state encoding, fitness value, and objective function value. As mentioned prior, the fitness value of the state determines the probability of the state being selected for reproduction, and it is heavily tied to the objective function value. A more thorough discussion of the distinction between these two values will take place in the discussion of probability methods.

The natural progression from a single state is a collection of states, or population. After the initialization of a randomly generated population, crossovers are continuously performed on one population until a new population of equal size is formed. Populations were the primary objects used for tracking genetic improvement, as they provided enough insightful detail without the confounding minutia given by the states. Using an entire population as an object allowed for the observance of higher-level statistics such as average fitness, population diversity, and crossover productivity. These characteristics are critical to the organization of the GA, as the entire population must be known before probabilities for selection can be assigned to the individual states.

Finally, collections of assumptions were tested against each other in order to determine the optimal set of operators and specifications for the GA. The primary

assumptions evaluated included a variety of probability functions, crossover operators, and testing specifications. The serialization of this object was necessary in order to facilitate the execution of tests amongst multiple servers. The design of this process is discussed further in the *Distributed Computing* section. Overall, the primary purpose of creating an Assumption object was to create a test specification that was both mutable and transferrable.

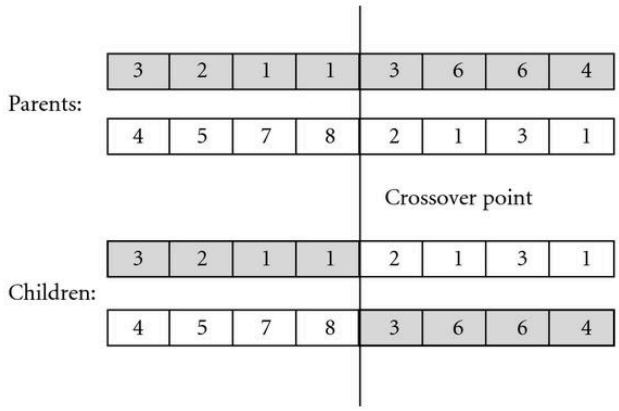
Crossover Operators

Crossover operators are specific protocols used when performing a crossover that allow for the resulting child chromosomes to adopt the best traits from their parents. Namely, crossover operators specify how the structural properties of the parent chromosomes are transferred to their child chromosomes. The primary crossover operators observed during this study were the number of crossover points used and partially mapped crossover (PMX).

The most basic and fundamental crossover operator is the specification stating the number of crossover points. The number of crossover points can have a profound influence on the productivity of a crossover, as each crossover point specifies a transition point in the chromosome where the child will adopt genetic information from one parent or another. In other words, a child may pull genetic data from a single parent up until a crossover point, then pull more genetic data from the other parent after that same crossover point. Traditionally, two crossover points are used for genetic algorithms, as this number of crossover points has been

shown to be effective in a number of problems, and is directly modeled after DNA replication [9]. An illustration of crossover points is shown below in Figure 4.

Figure 4: Crossover Operation



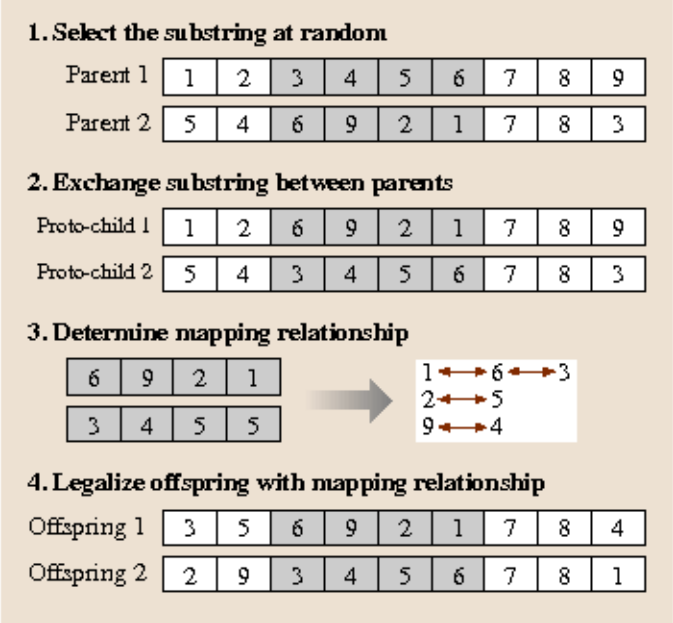
Source: http://america.pink/images/1/1/1/5/3/2/9/en/2-crossover-genetic-algorithm.jpg

The second crossover operator studied in this paper is known as partial mapped crossover, or PMX. PMX is a technique that improves the information passed on from parent to child by adjusting errors that appear due to crossover [2]. The algorithm for PMX is as follows:

1. Each section of the child chromosome is assigned to a parent, alternating parents at each crossover point. Each child has an implicit dominant parent and a non-dominant parent that is unique for each child.
2. Elements are added to the child sequentially. A horizontal conflict occurs (two queens fall on the same row), when the value of one element matches a value observed earlier in the child chromosome. That is, each conflict consists of two conflicting queens, which will be denoted as 'first' and 'second' based on the order in which they appear.
3. If the second conflict occurs on a non-dominant parent section, then the second conflict is maintained, as it represents newer information from crossover. The value of the dominant parent at the same point as the second conflict is then substituted for the value of the first conflict in the child.
4. If the second conflict occurs on a dominant parent section, then the reverse is performed. The second conflict is replaced by the dominant parent value at the first conflict and the first conflict is maintained.

To clarify the above algorithm, an illustration of this process is provided in Figure 5 below.

Figure 5: Partial Mapped Crossover (PMX)



Source: http://link.springer.com/referenceworkentry/10.1007%2F978-1-84628-288-1_42

Probability Methods

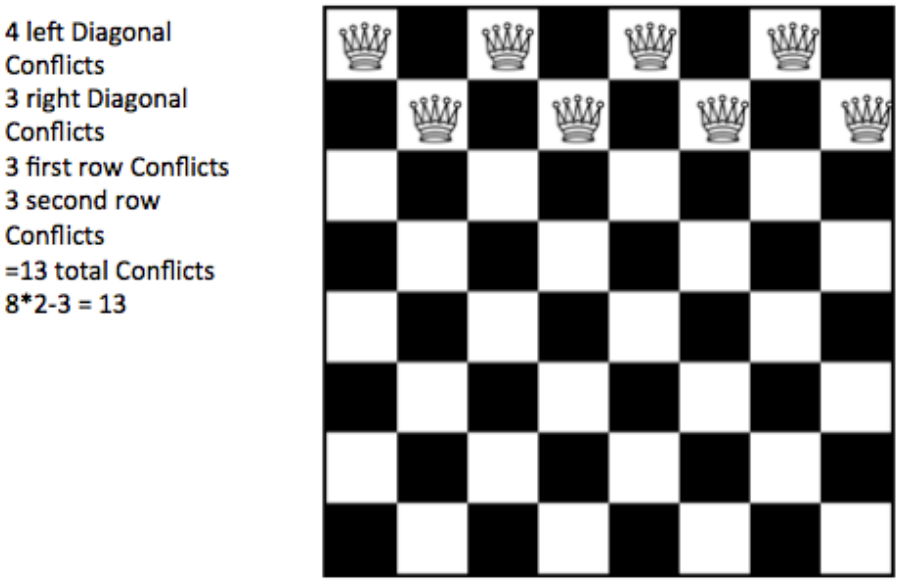
While crossover operators have a profound effect on the performance of the GA, the most significant agents acting on the convergence of GAs are the probability methods used. Probability methods denote the way in which states are selected from the population for reproduction [7]. While fitness values describe how desirable a given state is, probability methods translate fitness values into a probability distribution. Either changing the fitness value or the probability function that maps the fitness values to a probability can alter the probability of a given state. A probability distribution assigns all values within the population a value between 0

and 1 based on their relative fitness values, with the sum of all probabilities equal to 1.

A discussion of probability methods would not be complete without addressing the methodology behind assigning fitness values. The most intuitive method for assessing fitness is to count the number of conflicting queens. A queen is labeled as conflicting if it shares a row or diagonal with another queen (columns assumed to be unique). However, lower values (fewer conflicts) are more desirable than higher values, so this metric is inversely related to the desired probabilities. However, the number of conflicts is an important tool for visualization and goal recognition, as the goal state, or solution, for any number of queens will always have 0 conflicts. Therefore, this metric is denoted as the objective function.

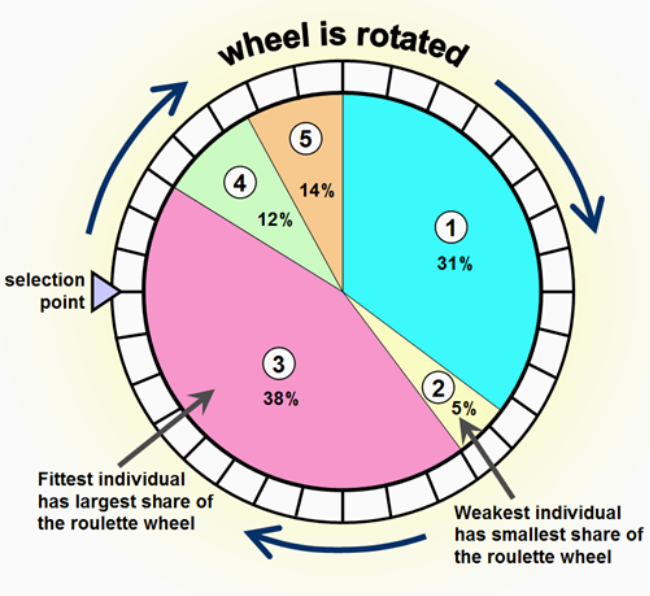
In order to get an appropriate fitness value that increased as the objective value decreased, the inverse objective value was used. This inverse value was calculated as the total number of potential conflicts minus the objective value. The number of total potential conflicts was determined to be 2*N-3, where N represents the number of queens. This limit is due to the method with which conflicts are summed up. Namely, each queen can add at most 3 conflicts: 1 horizontal, 1 right diagonal, and 1 left diagonal conflict. However, it is impossible to arrange the queens in such a way that each queen causes 3 new conflicts. Using a crisscross arrangement, the maximum number of potential conflicts is illustrated below.

Figure 6: Fitness Values – Inverse Objective Value



The overarching technique used to select states from the population was the roulette wheel or inverse cumulative distribution function method [7]. This method lines up all of the states within the population between 0 and 1 such that the probability of each state represents the width of its "pocket". A number is then pseudo-randomly generated between 0 and 1. The corresponding state of the pocket that is hit is then selected for reproduction. This method guarantees that any state within the population can be selected and that states with higher fitness values can be assigned higher probabilities of being selected. A simple demonstration is illustrated in Figure 7.

Figure 7: Roulette Wheel



Source: http://www.edc.ncl.ac.uk/assets/hilite_graphics/rhjan07g02.png

The first probability method explored dealt with altering the fitness value linearly by a scalar value, therefore referred to as the linear fitness method. By altering the fitness of each state in this way, the probabilities of each state are also altered. Because this method establishes a linear correlation between fitness value and probability, its performance heavily relies on the number of queens. Higher values of N result in higher fitness values with smaller differences between those fitness values. However, the linear fitness method proves highly useful, as it is a fairly stable method that represents the baseline impact of the crossover methods and assumptions being used. The formulas used for the linear fitness are shown below in Figure 8, with the probability function described echoing the procedure dictated by the roulette wheel method used.

Figure 8: Linear Fitness Equations

For a given state 'x' where 'f' represents fitness, 's' is the linear scaling variable, and 'o' is the objective function value.

$$f = s * o$$

$$P(x) = \frac{f_x}{\sum_1^n f_n}$$

The next method explored was exponential fitness, which takes a fitness value and raises it to a certain scalar value. In this way, the scalar value used has an impact on the probability of the states, with higher scalar values increasing the probability of higher fitness states. For example, you are given two states with fitness values of 1 and 2 that represent an entire population. With a scalar value of 1, the two states would have a probability of 1/3 and 2/3 respectively. However, with a scalar value of 2, the probability of the first state now becomes 1/5 and the probability of the second becomes 4/5. In this way, the exponential fitness method rewards higher fitness states more than the linear fitness model. A more formal definition of this method is shown below.

Figure 9: Exponential Fitness Equations

For a given state 'x' where 'f' represents fitness, 's' is the exponential scaling variable, and 'o' is the objective function value.

$$f = o^s$$

$$P(x) = \frac{f_x}{\sum_1^n f_n}$$

While the linear and exponential fitness methods alter the fitness values of the states, another method, the linear rank method, instead alters the probability function that maps the fitness values of the states based on their ranking relative to other states within the population. In other words, the states in a given population are sorted by their fitness values, with the higher fitness values at the "front" of the population and lower fitness values at the "back" of the population. States at the front of the population are then assigned higher probabilities for selection using a linear-based probability density function (PDF). However, although the PDF is continuous, the states in the population are discrete. Therefore, in order to map the PDF to discrete probability values, the cumulative distribution function (CDF) or the integral over small intervals is used. The formulas used to generate both the PDF and the resulting probabilities for each state can be seen below in Figure 10.

Figure 10: Linear Rank Equations
Where 'n' and 'r' are constants representing the population size and the rank of a specific state, respectively.

$$\boldsymbol{PDF = \frac{2}{n} - \frac{2}{n^2}x}$$

$$\boldsymbol{P(r) = \frac{2 \times n - 2 \times r - 1}{n^2}}$$

Similar to linear rank, another method, natural rank, instead uses a PDF function based on Euler's number '*e*'. This PDF function places a much heavier weight than the linear rank on lower ranks, effectively increasing the probability of selecting the highest fit individuals within the population. In order to manage the intensity with which the natural rank selects the highest ranked individuals, a

scaling constant is required. The probability functions associated with this method are shown below in Figure 11.

Figure 11: Natural Rank Equations

For a given population size 'n', a problem state rank 'r', and a scaling factor 's':

$$\boldsymbol{PDF = se^{-sx}}$$

$$\boldsymbol{P(r) = \ e^{-sr} - e^{-s(r+1)} + \frac{e^{-sn}}{n}}$$

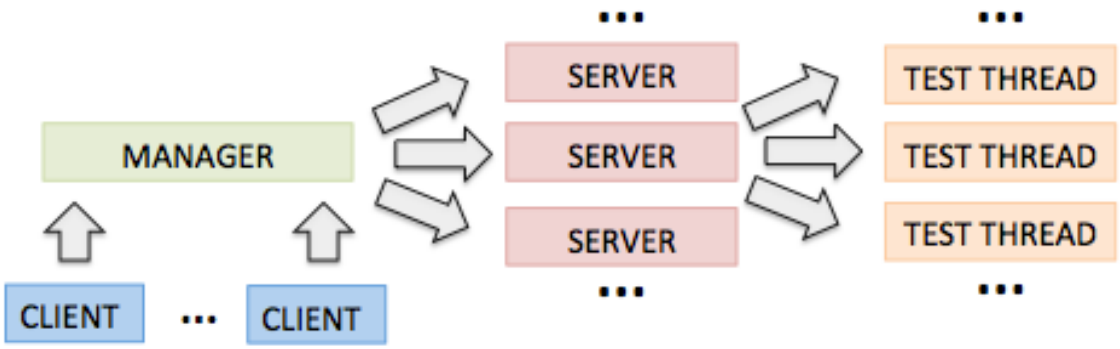
Distributed Computing

To conclude the discussion on the implementation of the GA, a critical aspect required to execute multiple tests was the design of a distributed system that could handle multiple requests and perform several tests at once. To allow for the distribution of work, both threading and the serialization of tests was performed. In this way, the execution of tests was passed along to multiple computers, allowing for many tests to be run simultaneously.

While simple in design, the purpose of the distributed computing setup was essential to the completion of the study. Due to the random nature of GAs, averages of many tests were used to generate smooth graphs. Using a centralized client-server model, a single client could send many requests to multiple servers. These computers then handled multiple requests each using threading. Testing information was returned to the client in the form of summary population statistics. An illustration of the design of this distributed system is show below in Figure 12, where each arrow represents a connection where tests and results could be passed

before. The directions of the arrows indicate the flow of the clients' test specifications being distributed to the servers.

Figure 12: Distributed Computing Design



IV. Results

In order to produce reliable, insightful results, rigorous testing was performed. Tests were organized into groups of 20 with the same assumptions, and the averages of their results are depicted in the graphic illustrations throughout this section of the paper. This method succeeded in smoothing out the inherent randomness associated with GAs, and it demonstrated the robustness of GAs by asserting that the results could be replicated. Solutions are not guaranteed for every test, so grouping tests together allows for the generalization of genetic convergence, the rate at which the algorithm reaches a solution. In short, there is a visible and concrete effect from each of the different assumptions used that can only be captured through generalization of testing with grouping.

Additionally, all of the tests performed operated with the same baseline assumptions of a population size of 1000 and a mutation probability of 0.01. The

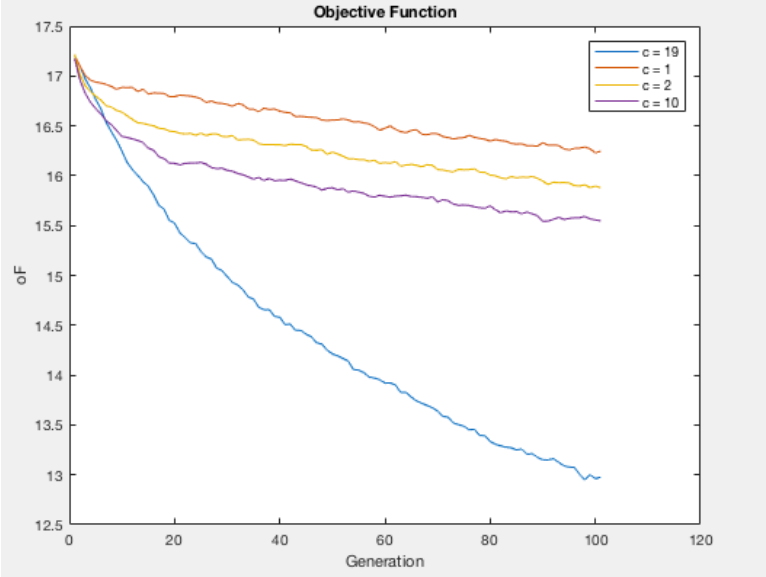
purpose of a large population size is to ensure that there is enough diversity in the pool of states being selected so that the crossover methods in the GA can be effective. A population size that is too large introduces too much randomness, whereas a population that is too small limits the potential for crossover productivity. The majority of tests performed – unless otherwise specified – were performed with 20-Queens, and it was determined that a population size of 1000 was an ideal population size. Moreover, the mutation rate was kept fairly low at 0.01 to allow for convergence without artificially adding too much randomness to the testing procedure. Without mutation, convergence would not be possible, but too much mutation results in the breaking up of near-perfect solutions, making it difficult to converge to a solution.

Linear Fitness

The first probability method that was tested was the linear fitness function. As a baseline protocol for state selection, this method allowed for the independent testing of a variety of other assumptions. It can be shown that the effects of many of these assumptions are true for not only the linear fitness method but other probability methods as well. However, the effects of these assumptions is best visualized with the linear fitness method, where convergence is rather unexciting to begin with.

The first major assumption that was researched was how many crossover points should be used. The results of testing can be seen below in Graph 1, where 'oF' denotes the average objective value function of the generation on the y-axis.

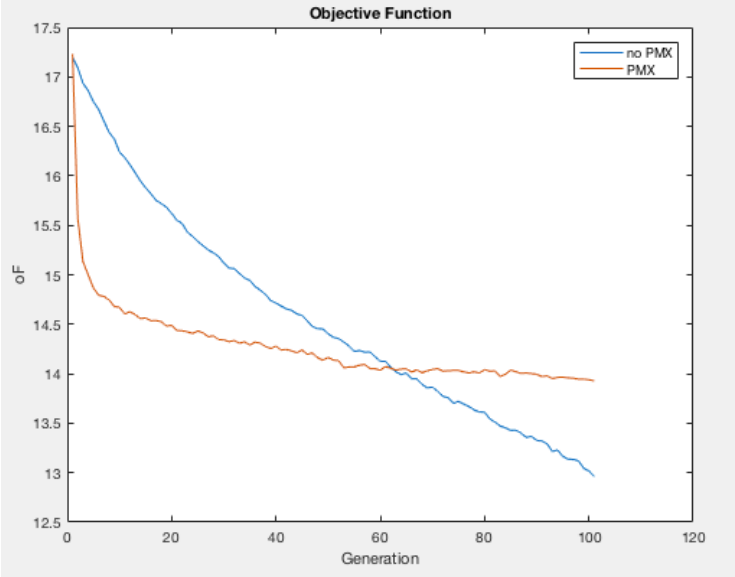
Graph 1: Linear Fitness Function under Varying Number of Crossover Points



The results show that, for the 20-queen problem, 19 crossover points is optimal, as tests run with 19 crossover points converge the quickest compared to other numbers of crossover points. Specifically, the method of using the maximum number of crossover points is known as uniform crossover, where every other element received through crossover in a child state comes from a different parent. Moreover, the difference between other numbers of crossover points (ie between 1,2,10 crossover points) is fairly unexciting, so throughout the remainder of the study, uniform crossover was assumed.

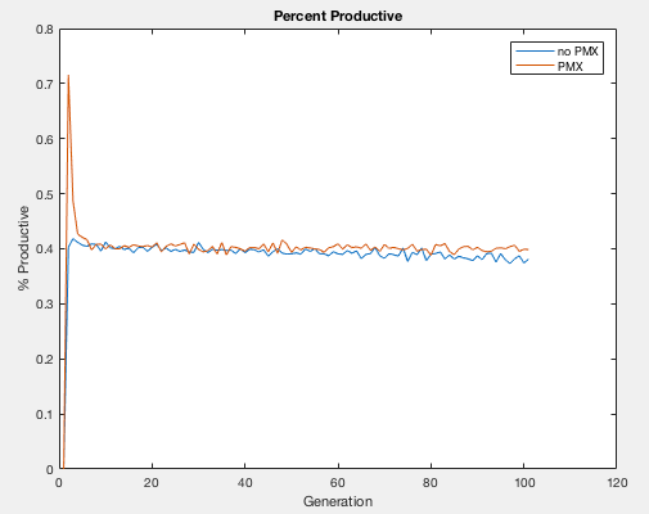
One of the major limitations of using a linear fitness method is that it doesn't converge very quickly, so supporting crossover operators must be used in order to speed up the probability method's convergence to a solution. The primary crossover method analyzed throughout this paper is PMX, as it can have a profound impact on the convergence of the GA. The impact of PMX on the linear fitness function is shown below in Graph 2a, which assumes that all other assumptions are held constant.

Graph 2a: Effects of PMX on Linear Fitness – Objective Function



As seen by this graph, PMX critically impacts the convergence of the linear fitness method in early generations, but convergence is reduced later on. The reason this occurs is because PMX generates child states that are very similar to one another. More specifically, the child states converge to a small collection of schema that are similar to one another in such a way that crossover productivity is minimized. This is illustrated by graph 2b below.

Graph 2b: Effects of PMX on Linear Fitness – Productivity



As illustrated in Graph 2b, productivity spikes for the Linear Fitness model with PMX initially then levels off to a near-random level of productivity afterwards. However, the interpretation of this graph is subject to the definition of productivity in the context of this problem. A crossover is considered productive when the sum of the objective function values of the children is less than the sum of the objective function values of both of the parents. In other words, a crossover is considered productive if the net change from parent to child is negative, or that overall, the children exhibited fewer queen conflicts than was observed in the parents.

Additionally, it can be shown that the percent productive rate of 0.4% is due only to mutation. This is true because there is an implicit 0.01% chance of mutation and the number of queens used for testing in this graph was 20. The probability of mutation in a given state is therefore 0.2%. However, each crossover yields two children, so this probability is doubled to 0.4%. It's important to note, however, that this is only possible when no advantage is gained through crossover, and the true probability of a productive crossover is actually slightly lower than 0.4%, as there are some cases where mutation results in no change or adversely affects productivity. To summarize, the probability of random productivity is given by the following formula:

Figure 13: Probability of a Randomly Productive Crossover

For a given number of queens 'n' and a mutation rate 'm':

$$P(productive) \cong 2nm$$

With this understanding, it is clear that the beneficial effects of PMX are somewhat limited. While productivity is boosted initially due to the effects of the

algorithm, it levels off quickly once the population achieves a high level of homogeneity.

Exponential Fitness

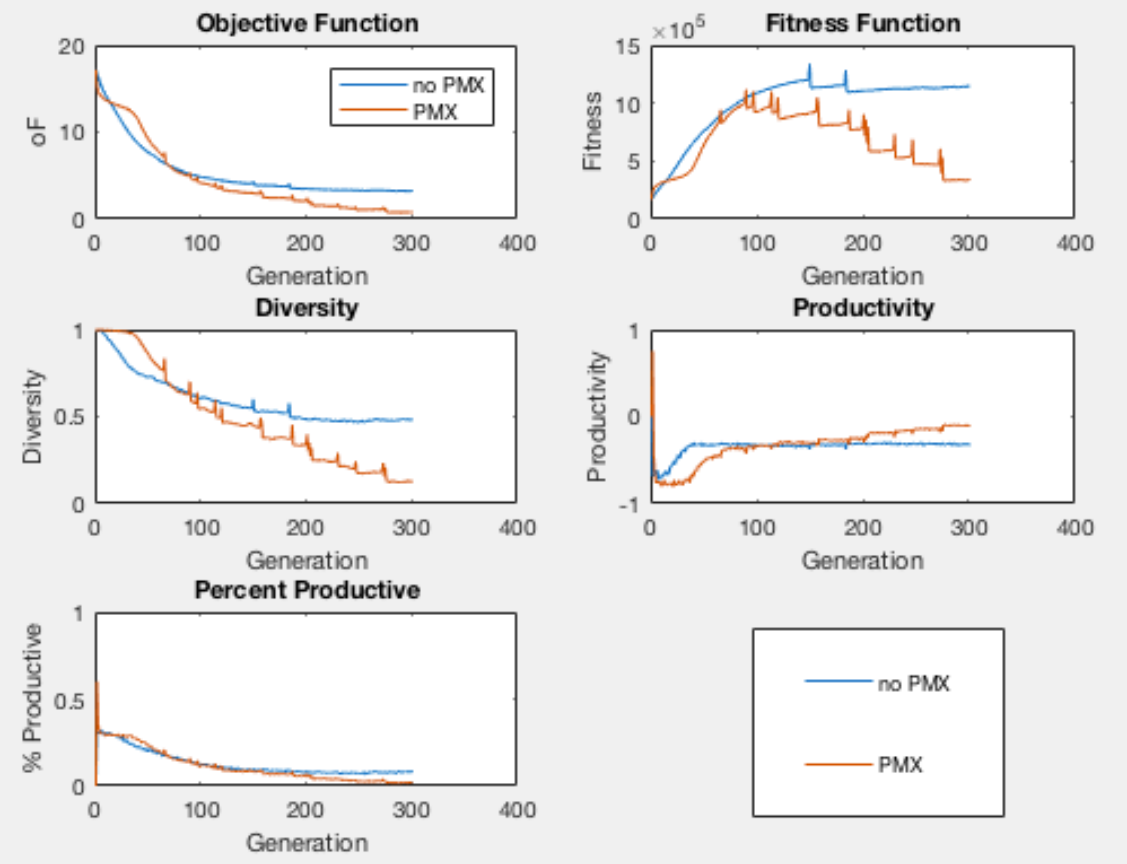
The second probability method that was analyzed was the exponential fitness function. The primary assumption tested was whether or not PMX had a different effect on solution convergence with the exponential fitness function than it did with the linear fitness function. Other assumptions that were tested using the exponential fitness function can be viewed in Appendix A.

The effects of PMX on the exponential probability function were tested in order to determine whether or not the exponential fitness function would experience a different reaction to the crossover method than the linear fitness function. The results produced were drastically different from the effects observed with the linear fitness function and are shown below in Graph 3a.

The results of PMX testing on the exponential fitness function produce remarkable results. Before discussing these results, a quick note on the new graph displaying 'Productivity'. The distinction between 'Percent Productive' and 'Productivity' is that the percent productive is the proportion of the population that yielded child offspring with higher fitness values than the parents, whereas productivity, in its bare form, represents the average improvement in the objective function over the entire population. While these results are included, they are not discussed in great detail. The main takeaway is that, more often than not, productivity will be negative because it is much easier to lose advantageous genetic

data than to gain the few missing pieces needed to improve. In this way, a negative productivity level is expected, with near-zero productivity indicates that many tests have found solutions. The productivity graph for the PMX tests approaches zero, indicating that the majority of tests found a solution.

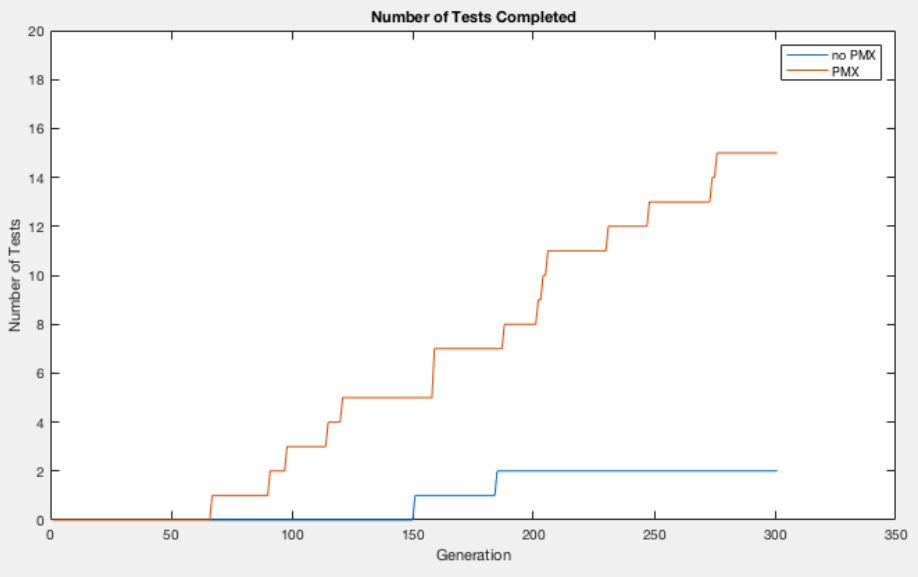
Graph 3a: Effects of PMX on Exponential Fitness



Several other discoveries can also be made from these results. First, without PMX, the GA begins to level off after 100 generations, and solutions are found at random, which is probabilistically unlikely. Second, there is a positive correlation between the diversity of the population and the productivity of the population. Namely, as the number of unique states within the population decreases, the percent of crossovers that are productive also decreases. And finally, each spike

shown in the fitness function graph and subsequent decrease in the average fitness of the population portrays a population that has yielded a solution. In other words, as described earlier in the Results section, each line on the graphs represent the average result of 20 different tests performed. However, when a test finds a solution, that test terminates and produces a zero value for subsequent generations. In this way, convergence can be illustrated accurately as being the point where the line hits zero, or all tests in a given group find a solution. This is critical information needed to interpret the results, as the spikes shown in the Fitness graph are due to each test discovering a solution. Simply, the exponential fitness function yields significantly more solutions at a faster rate with PMX than it does without PMX. This is more clearly depicted in Graph 3b below.

Graph 3b: Effects of PMX on Exponential Fitness: Number of Successful Tests out of 20



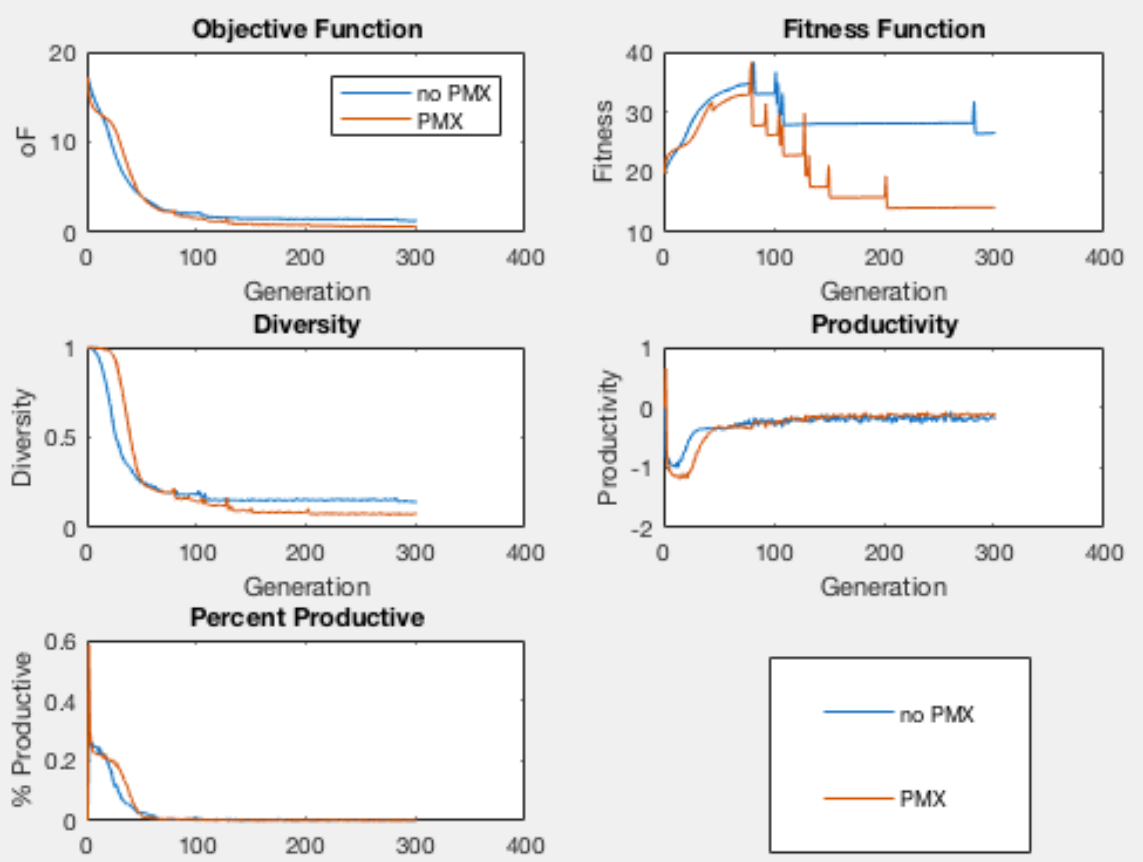
Linear Rank

While both the linear and exponential probability functions are fitness-based, the linear rank probability method is purely probabilistic, relying on the objective function values of states relative to other states within the population. In this way, linear rank allows for the probabilities, which directly impact the states' likelihood of being chosen for reproduction, to be changed instead of fitness, which indirectly affects the probability of being chosen. While many variables were tested, the most significant factor tested was once again PMX. Additional results can be viewed in Appendix A. In general, PMX enhances the convergence of probabilistic methods and provides for interesting results, which is why it has been illustrated the most frequently in this paper.

As PMX had the largest impact on both the linear and exponential probability methods, it was only natural that the effects of PMX be tested on the linear rank method as well. Graph 4a illustrates the results obtained from this testing. Interestingly, PMX has less of an impact on linear rank than it does on the exponential and linear methods. This is likely due to the fact that the linear rank method is much more selective than the other methods. To quantify this, the linear rank method assumes that the top 50% of states will be selected 75% of the time, whereas the linear fitness method has no guarantee; the top 50% of the states are selected between 50-60% of the time, depending upon the average fitness of the population. Because of this, the linear rank method converges very quickly, and diversity drops drastically. Due to the significant drop in diversity, the

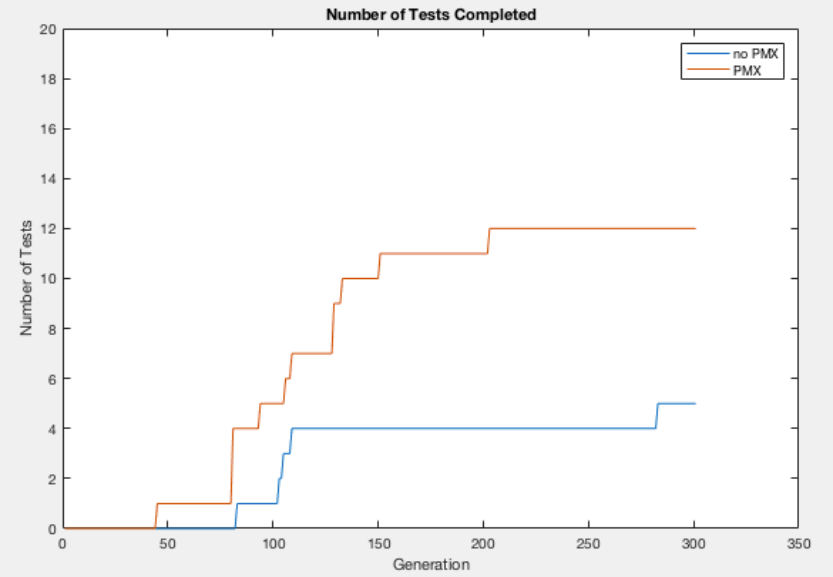
homogenizing effects of the linear rank method outmatch the standardizing effects of PMX.

Graph 4a: Effects of PMX on Linear Rank



One of the most notable findings from this testing was the fact that PMX can play a pivotal role when the population reaches a near-solution phase. This realization makes logical sense, as PMX applies algorithmic intuition to an evolutionary process. Additionally, it is clear from the spikes in fitness values that linear rank yields several solutions for 20-queens within 300 generations. The number of completed tests is shown below in Graph 4b, and the graph demonstrates that the use of PMX for linear rank presents a clear advantage in terms of finding solutions.

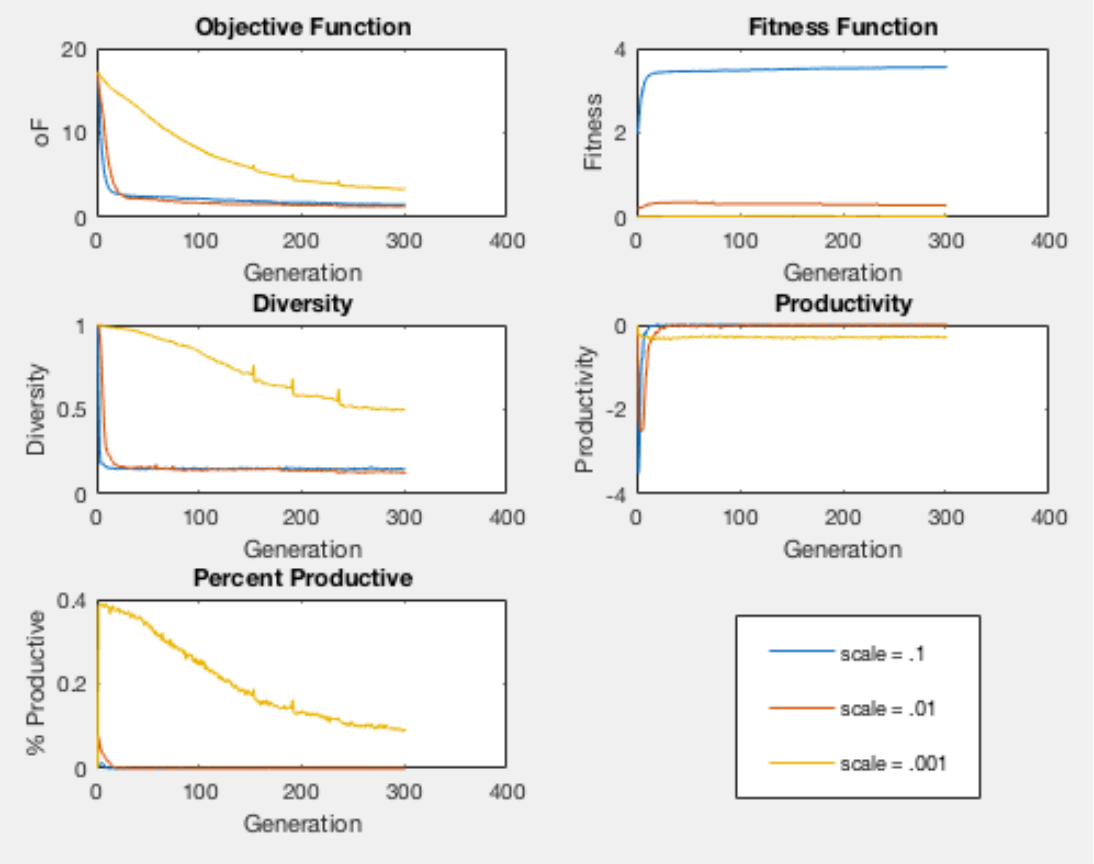
Graph 4b: Effects of PMX on Linear Rank: Number of Successful Tests out of 20



Natural Rank

Finally, the last probability method tested was natural rank. As discussed earlier, natural rank attempts to use a more fluid curve for determining state probability using Euler's number. It became quickly evident that the base function used would yield curves much too steep, and only the first few states from each population would be probabilistically likely to be chosen. For this reason, a scaling factor was used to lengthen the curve, and various inputs were tested. The results can be seen below in Graph 5a. While scaling factors of .1 and .01 yield extremely quick convergence rates, their convergence comes at a huge cost of productivity. This phenomenon is referred to as the stability of the algorithm. In other words, using higher scaling factors yields less stable probability methods, as the results observed are grossly limited by the initialization of the population.

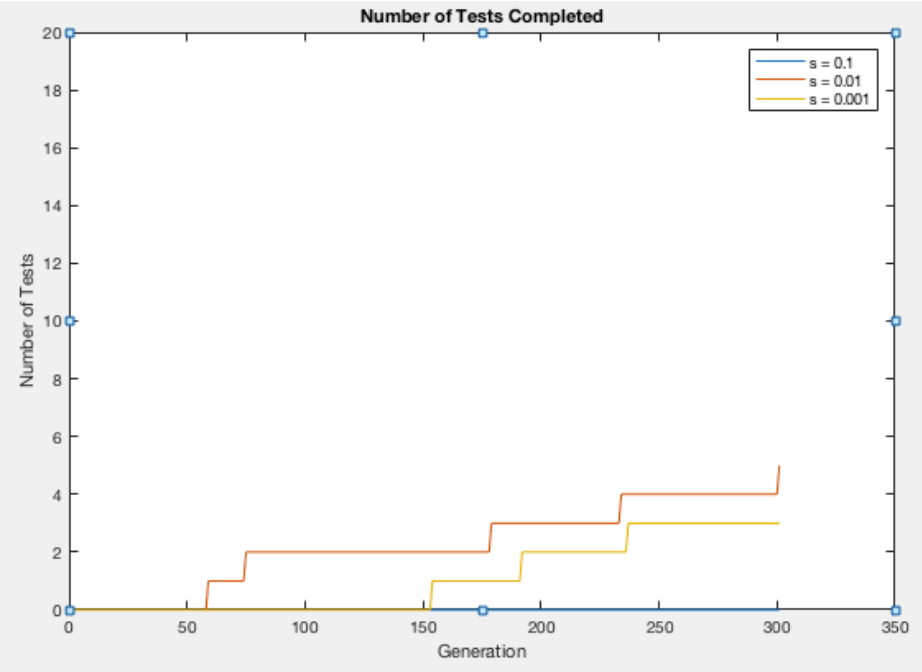
Graph 5a: Effects of Scaling on Natural Rank



Using a scaling factor of 0.001 or lower will yield a smoother curve that has a more consistent productivity rate. This is desirable because it means that, although the algorithm will converge more slowly initially, it is more likely to reach a solution than using a lower scaling factor. This is due to the fact that the algorithm itself plays a role throughout the evolutionary process whereas a higher scaling factor limits the potential for the algorithm to be effective by cutting down the diversity of the population. However, there is a middle ground where the natural rank probability method is most effective. During testing, it was found that the scaling factor of 0.01 worked best, although the results were still worse than the

exponential fitness and linear rank methods. This is best demonstrated by the number of solutions found illustrated below in Graph 5b.

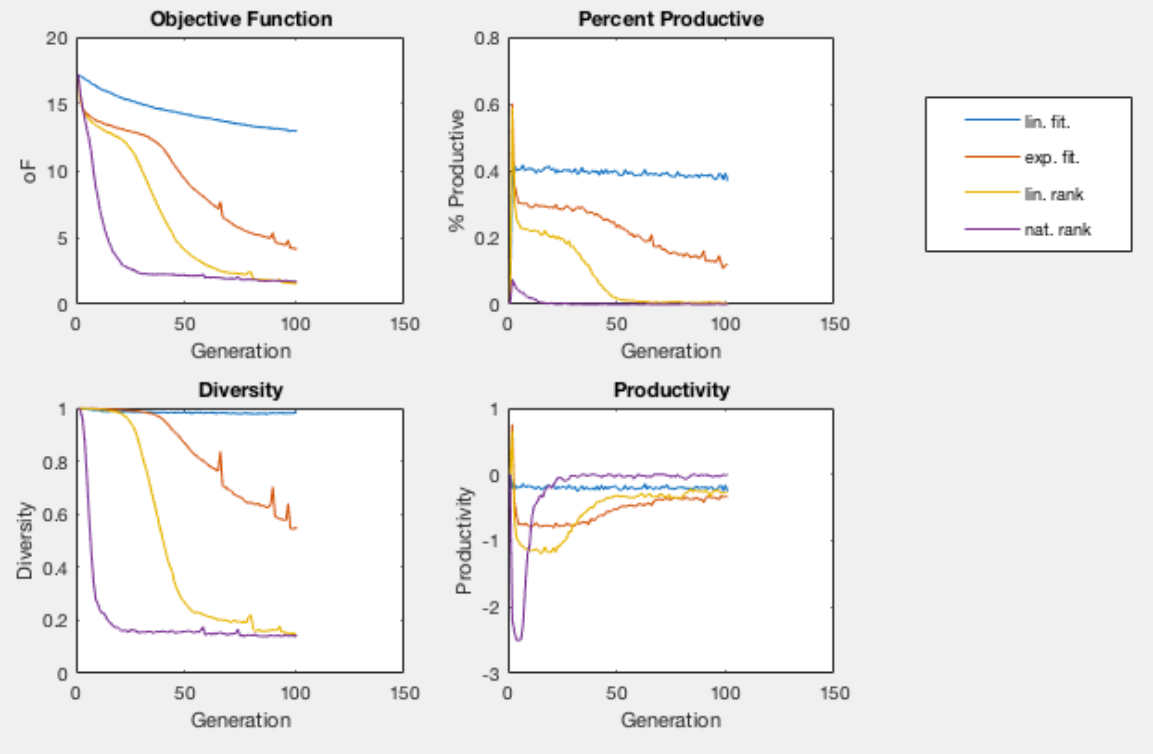
Graph 5b: Effects of Scaling on Natural Rank: Number of Successful Tests out of 20



Method Comparison

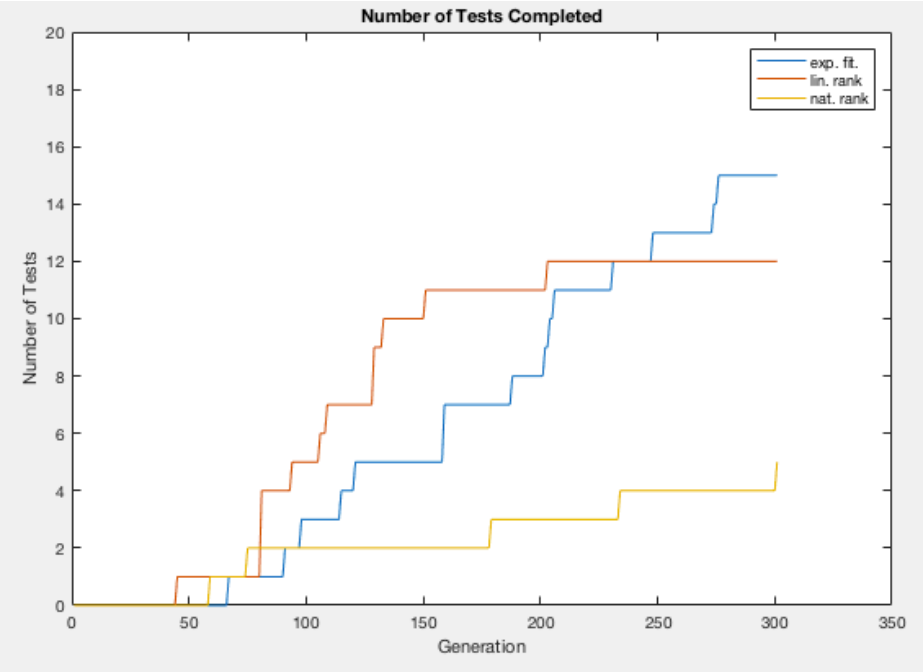
Although the results illustrate which assumptions work best for the individual probability methods, the natural progression would be to ask which probability methods perform best. Of the methods used, the exponential fitness model performed the best. A comparison of the different methods used can be seen below in Graph 6a.

Graph 6a: Comparison of Probability Methods: Overview



Clearly, the exponential fitness, linear rank, and natural rank functions far outperform the linear fitness method. These methods experience a steeper convergence and are therefore attain solutions much faster. In order to better quantify the difference between these algorithms, the number of generations observed was expanded to 300, and the number of solutions each probability method discovered can be seen below in Graph 6b.

Graph 6b: Comparison of Probability Methods: Number of Successful Tests out of 20

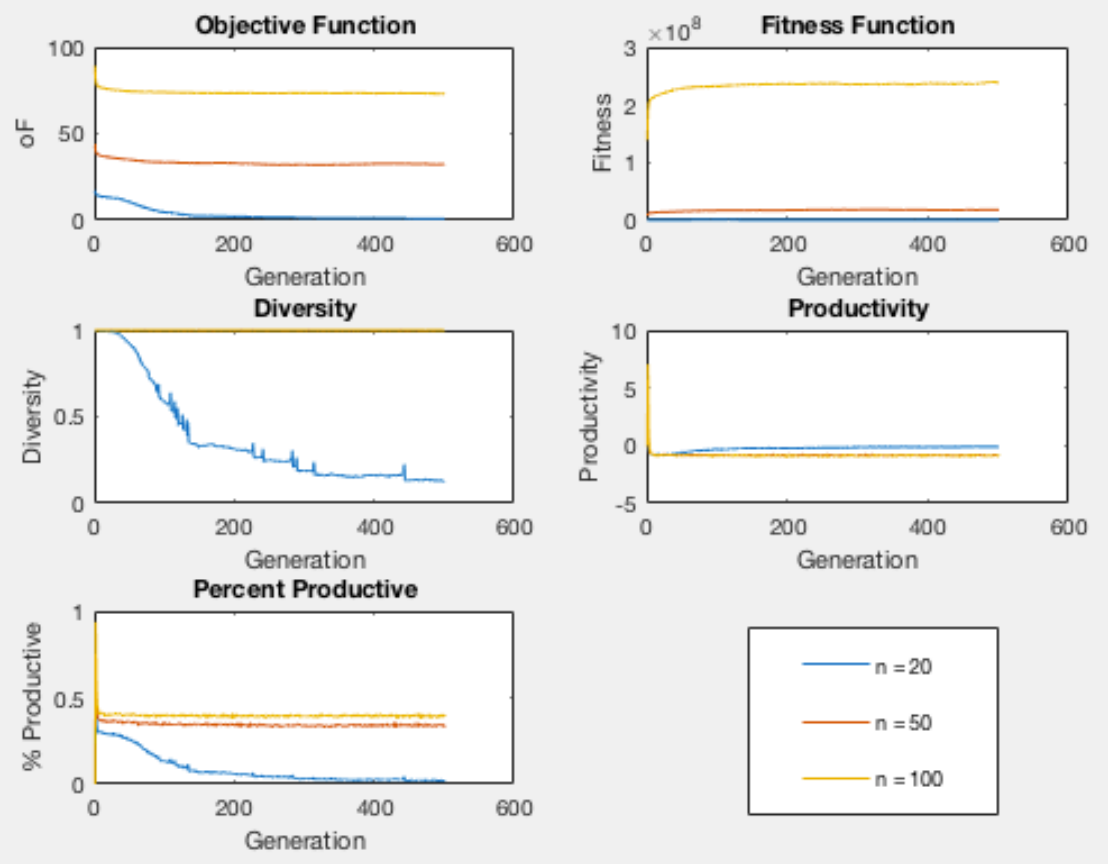


From these findings, it was determined that the exponential fitness method, although it did not have the quickest convergence, found a good balance of algorithmic evolution and productivity. Not only did it yield the highest number of solutions per group, but it also had the most stable level of productivity. This is desirable in a GA, as the consistency of solution discovery is highly valued. However, as both the exponential fitness and linear rank methods performed similarly well, further testing was conducted for both.

The basis for the majority of testing throughout this paper has been focused on the various assumptions applicable to the N-Queens problem and genetic algorithms in general. Once the set of optimal assumptions was discovered, the next logical step was to test whether or not the performance on these methods was maintained for larger numbers of queens. The effects of increasing the number of

queens can be seen in Graph 6c for the exponential fitness function and Graph 6d for the linear rank function.

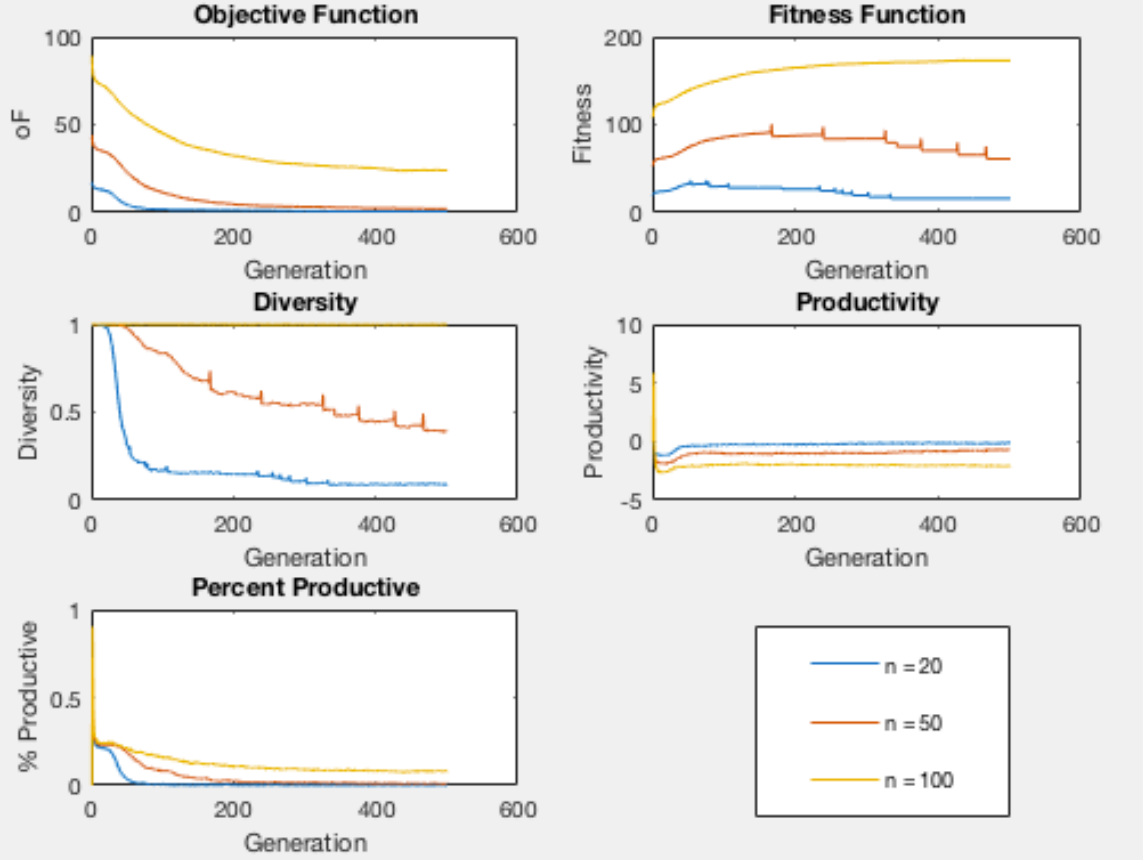
Graph 6c: Effects of Increasing Number of Queens: Exponential Fitness



The most important takeaway from these graphs is the fact that, while there is some convergence for N = 20, there is relatively little convergence with the exponential fitness function for higher numbers of queens. This indicates that the exponential fitness function is not very scalable. This is due in large part to the complexity of the problem at hand and the associated probabilities. As discussed earlier in this paper, because the differences between the fitness values are less relevant with higher N, the benefits of the exponential fitness function are lost. In

order to improve this function, a displacement variable would need to be added and tested further.

Graph 6d: Effects of Increasing Number of Queens: Linear Rank



In comparison to the exponential fitness graphs, the linear rank function converges much faster. This is largely due to the fact that the exponential fitness function is effective primarily when the differences in fitness are a significant portion of the overall fitness. On the other hand, linear rank directly deals with selection probability; therefore each state is evaluated relative to the complexity of the problem as whole. In other words, states that are ranked higher due to their lower objective functions are rewarded with a greater probability for selection in the same way for N=20, N=50, and N=100. To better quantify the difference in

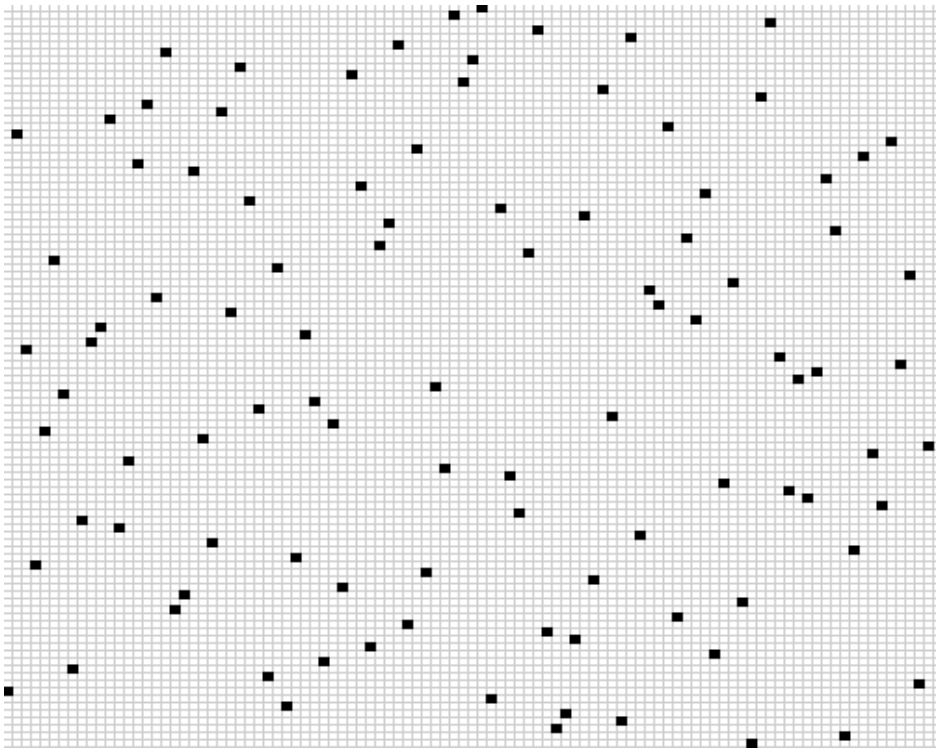
performance between the linear rank and exponential fitness methods, the percentages of tests completed after a number of generations can be seen below in Figure 14.

Figure 14: Percent of Tests Completed out of 20 after Varying Numbers of Generations

	Exp. Fit.	Lin. Rank	Exp. Fit.	Lin. Rank
Number of Generations	N = 20	N = 20	N = 50	N = 50
100	15%	15%	0%	0%
200	45%	25%	0%	5%
300	65%	45%	0%	10%
400	70%	55%	0%	25%
500	75%	55%	0%	35%

This table illustrates how exponential fitness has significantly better performance for 20-queens, whereas it yields no solutions for the 50-Queens problem. On the other hand, the linear rank function reaches 35% test completion for the 50-Queens problem after 500 generations. This indicates that, after 500 generations 35% of the tests performed have generated a solution to the problem. Additionally, although not illustrated in this table, a solution for the 100-Queens problem was discovered using the linear rank method after over 4000 generations (2 million crossovers). In short, the linear rank function is more effective for higher values of N, whereas the exponential fitness function operates more effectively for lower values of N, as discussed earlier. Figure 15 has been included below to illustrate the 100-Queens solution, one of the cornerstones of this research endeavor.

Figure 15: 100-Queens Solution



Encoding: {92, 17, 46, 75, 57, 34, 52, 89, 69, 45, 43, 15, 70, 61, 21, 13, 39, 6, 81, 79, 22, 58, 72, 14, 41, 8, 26, 54, 90, 35, 94, 74, 44, 53, 88, 56, 78, 9, 24, 86, 32, 29, 5, 83, 19, 76, 51, 62, 1, 10, 7, 0, 93, 27, 63, 68, 33, 3, 84, 97, 95, 85, 28, 77, 11, 55, 96, 4, 71, 38, 40, 16, 82, 31, 42, 25, 87, 64, 37, 80, 99, 12, 2, 47, 65, 50, 66, 49, 23, 30, 98, 73, 20, 60, 67, 18, 48, 36, 91, 59}

V. Conclusion

To sum up the results observed, the linear rank method proved to be the most tenacious of the probability methods explored, and PMX played a pivotal role in discovering solutions. Additionally, with regards to the assumptions used, uniform crossover combined with a low mutation rate of 0.01 and a large population of 1000 proved most effective. Lastly, a solution to the 100-Queens problem was discovered after over 4000 generations using the linear rank method, which, to put into perspective, took roughly 100 times as long as the 8-Queens problem and with significantly less consistency.

While the results demonstrated throughout this paper make strides towards a better understanding of genetic algorithms and how they can be used to solve the N-Queens problem, there are additional methods that could have been explored further. The first area that would require further study would be to determine the impact of combining multiple probability methods together, balancing each methods' strengths and weaknesses to optimize crossover productivity and minimize diversity decay. Alternatively, the use of more dynamic variables could also be tested, where mutation rate or some scaling variable may fluctuate according to a specific metric such as productivity or diversity. Finally, additional crossover operators should be explored. One such crossover operator might be to identify and retain identified schema throughout a problem state and replicate that schema for higher numbers of queens. In other words, it might be possible to solve the 100-Queens problem by first solving the 10-Queens problem then mapping that solution onto non-conflicting locations for the 100-Queens chessboard. An illustration of this notion can be seen below in Figure 16.

Figure 16: Potential Groupings in the 100-Queens Solution

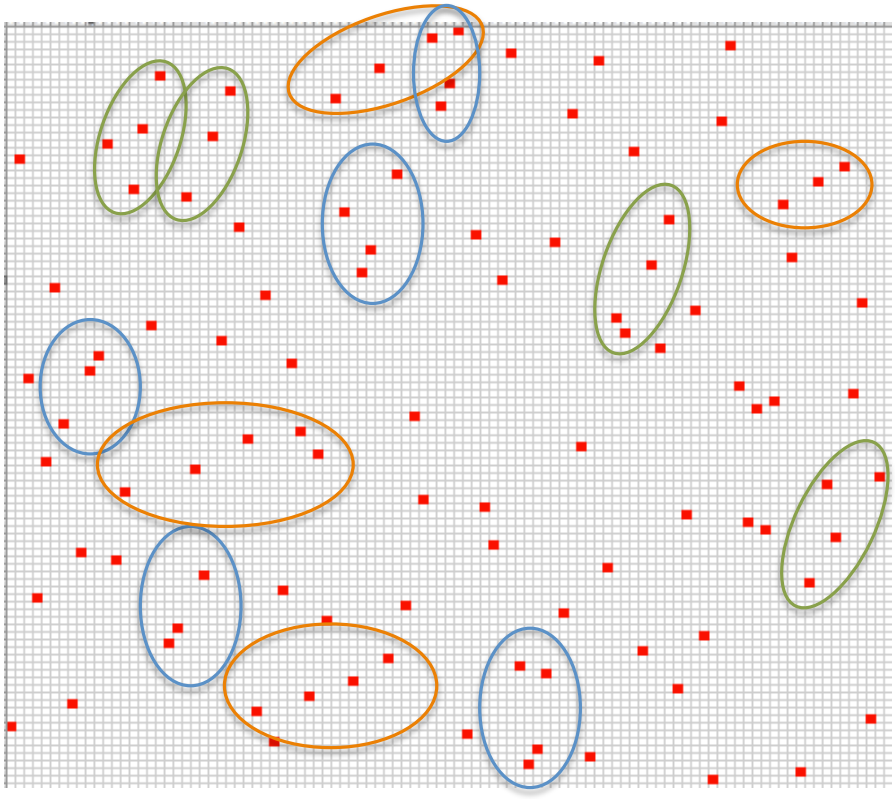


Figure 16 demonstrates how specific patterns appear within the 100-Queens solution that may be able to be replicated in further tests. Certain groupings or pairings of different groups in specific combinations may offer an advantageous to solution discovery. However, it is clear from this illustration that the groupings are far from concrete, and extensive testing and analytical reasoning would be needed to determine exactly how best to implement such a procedure within the genetic algorithm. In short, a variety of other innovative methods and operators could be added to the genetic algorithm structure to make it more effective. With the right intuition and careful testing, it is firmly believed that genetic algorithms can outperform nearly every other algorithmic method for complex solution discovery.

Works Cited

1. Booker, L. “Improving search in genetic algorithms.” *Genetic Algorithms and Stimulated Annealing,* L. Davis (Eds). 1987, pp. 61-73.

2. Crawford, Kelly D. “Solving the n-queens problem using genetic algorithms”. *ACM/SIGAPP 1992 Proceedings*. New York, NY. 1992, pp. 1039-1047.

3. Dejong, K. “The analysis and behavior of a class of genetic adaptive systems”. PhD thesis, University of Michigan. 1975.

4. Eiben, A.E., and P. E. Raue, Z. Ruttkay, "Solving constraint satisfaction problems using genetic algorithms." *Evolutionary Computation 1994 IEEE World Congress on Computational Intelligence*. Orlando, FL, 1994, pp. 542-547, vol.2.

5. Gihan, Nagib, and Wahied Ali. “Network Routing Protocol using Genetic Algorithms.” *International Journal of Electrical and Computer Sciences*. pp. 36-40, vol. 10.

6. Homaifar, A., and J. Turner, S. Ali, "The N-queens problem and genetic algorithms," *Southeastcon 1992 Proceedings, IEEE*. Birmingham, AL, 1992, pp. 262-267, vol.1.

7. Man, K.F. and K.S Tang, S. Kwong. *Genetic Algorithms: Concepts and Designs*. London, U.K. Springer-Verlag London Limited, 1999, Print.

8. Russell, Stuart, and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ. Prentice Hall, 1995, Print.

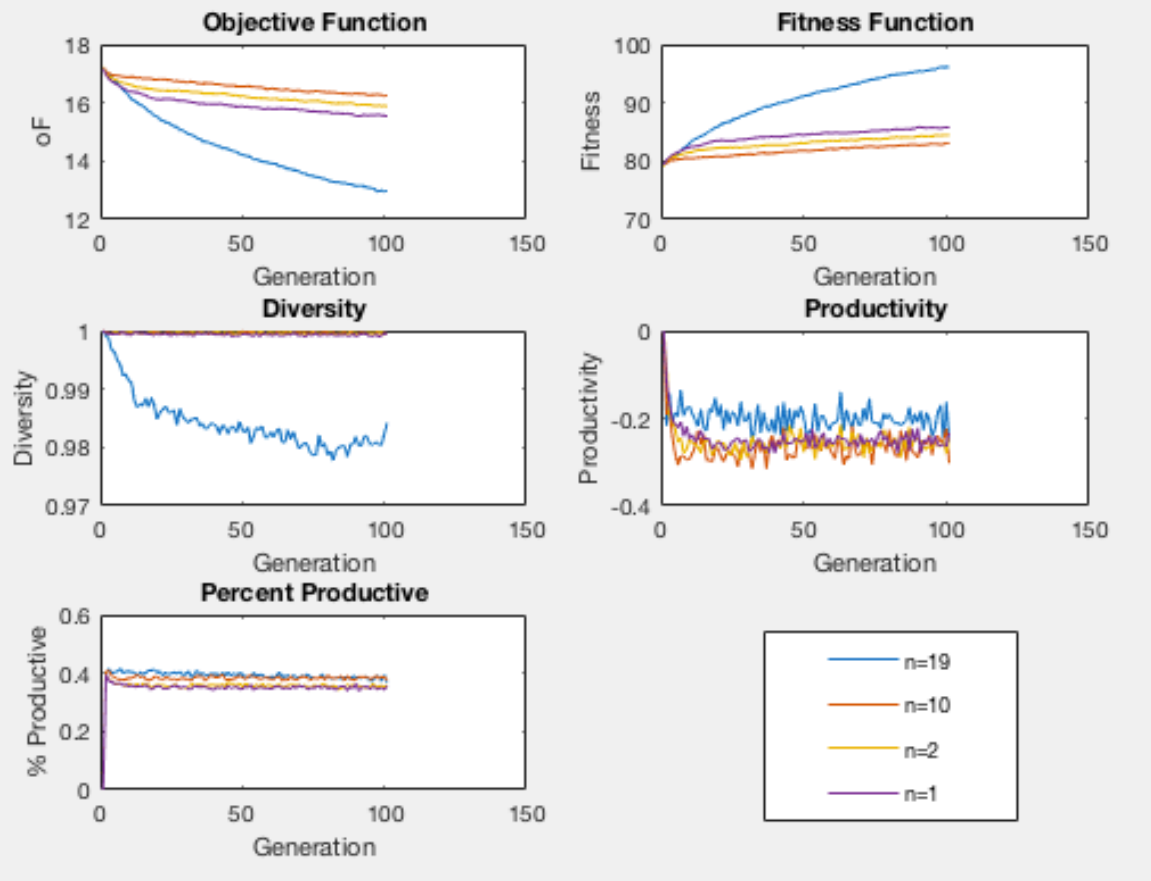
9. Spears, W.M. and K. Dejong. “An analysis of Multi-point crossover.” *Foundations of Genetic Algorithms*, G.J.E. Rawlins (Eds), pp. 301-305.

10. Thada, Vikas, and Shivali Dhaka. “Performance Analysis of N-Queen Problem using Backtracking and Genetic Algorithm Techniques”. *International Journal of Computer Applications*. 2014, vol. 102.

11. Watkins, John J. *Across the Board: The Mathematics of Chess Problems*. Princeton, MA. Princeton University Press. 2004.

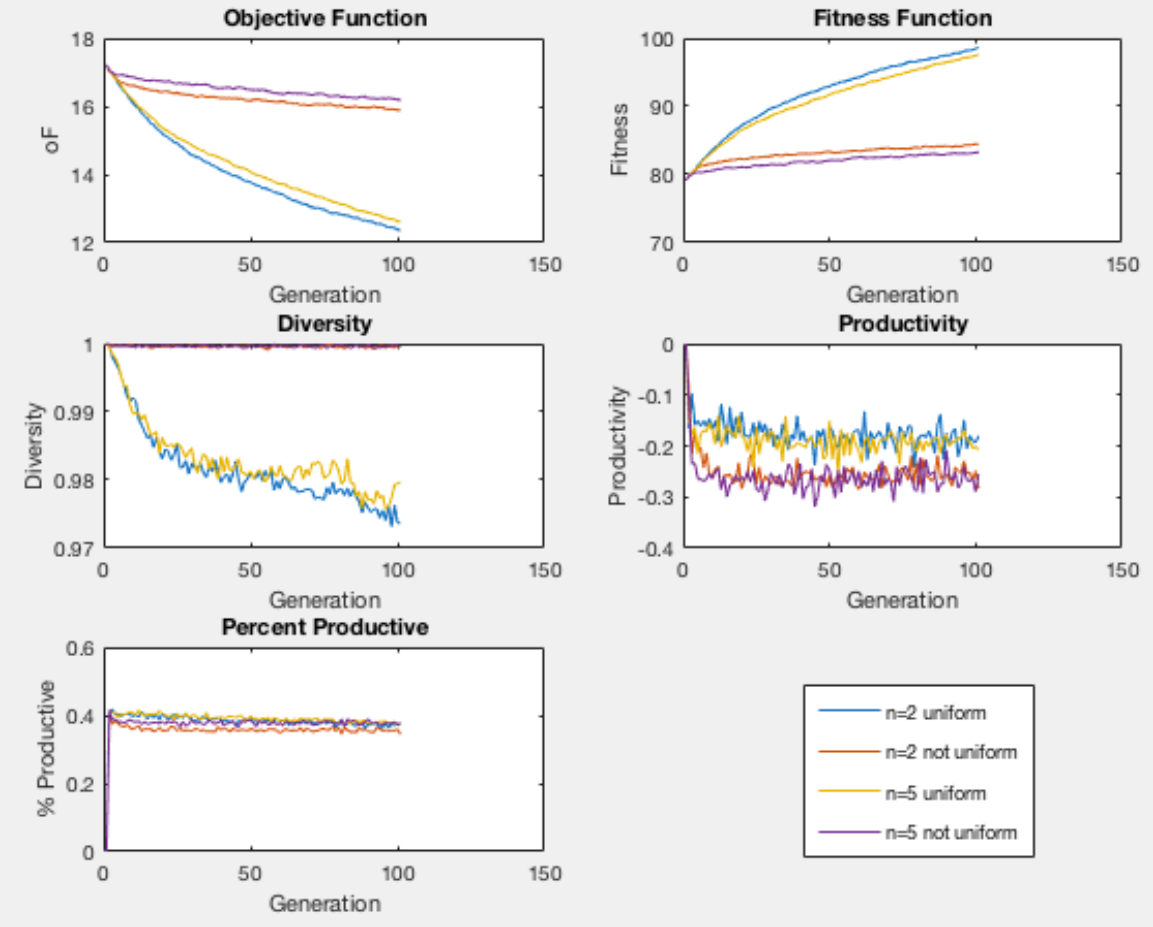
APPENDIX A – Additional Results and Graphs

A.1 Linear Fitness: Comparison over Varying Number of Crossover Points



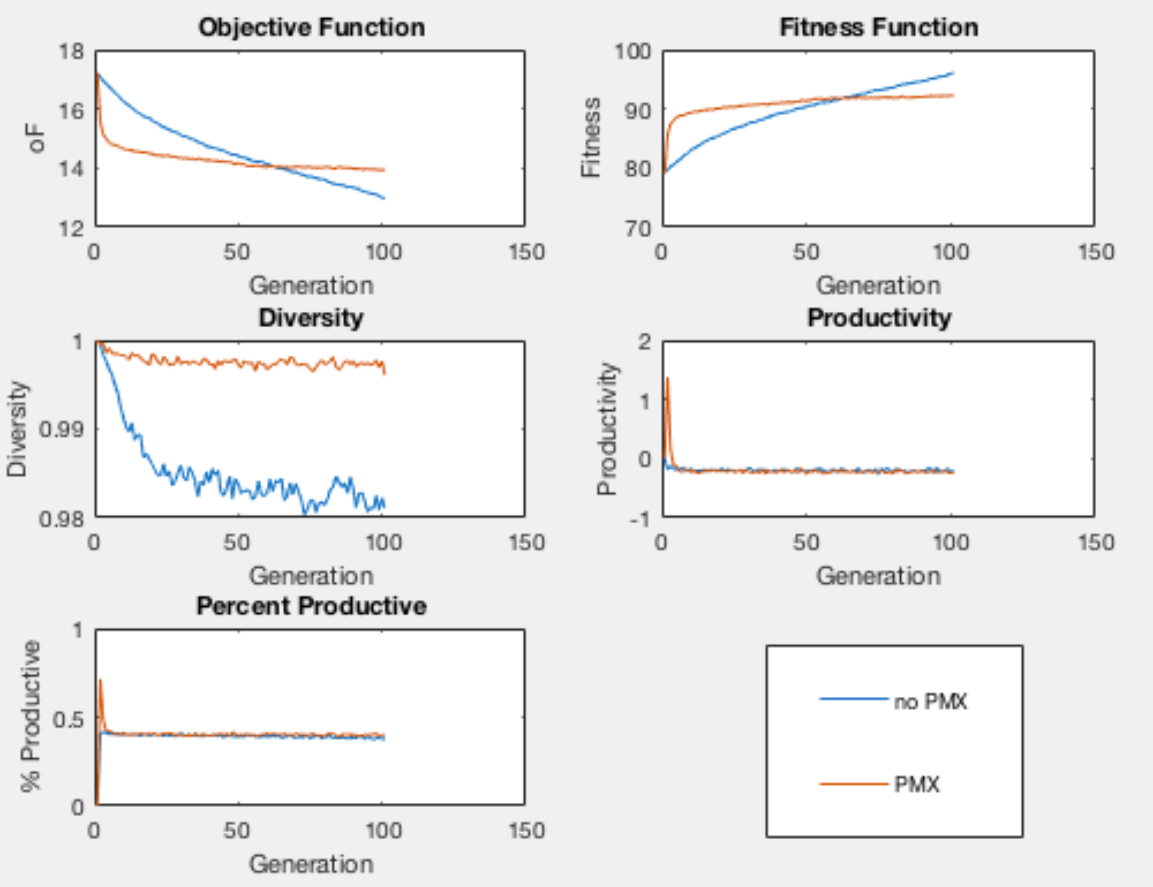
Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: linear
Crossover Points: **VARIES**
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (false, false, false)

A.2 Linear Fitness – Effects of Evenly Distributed Crossover Points



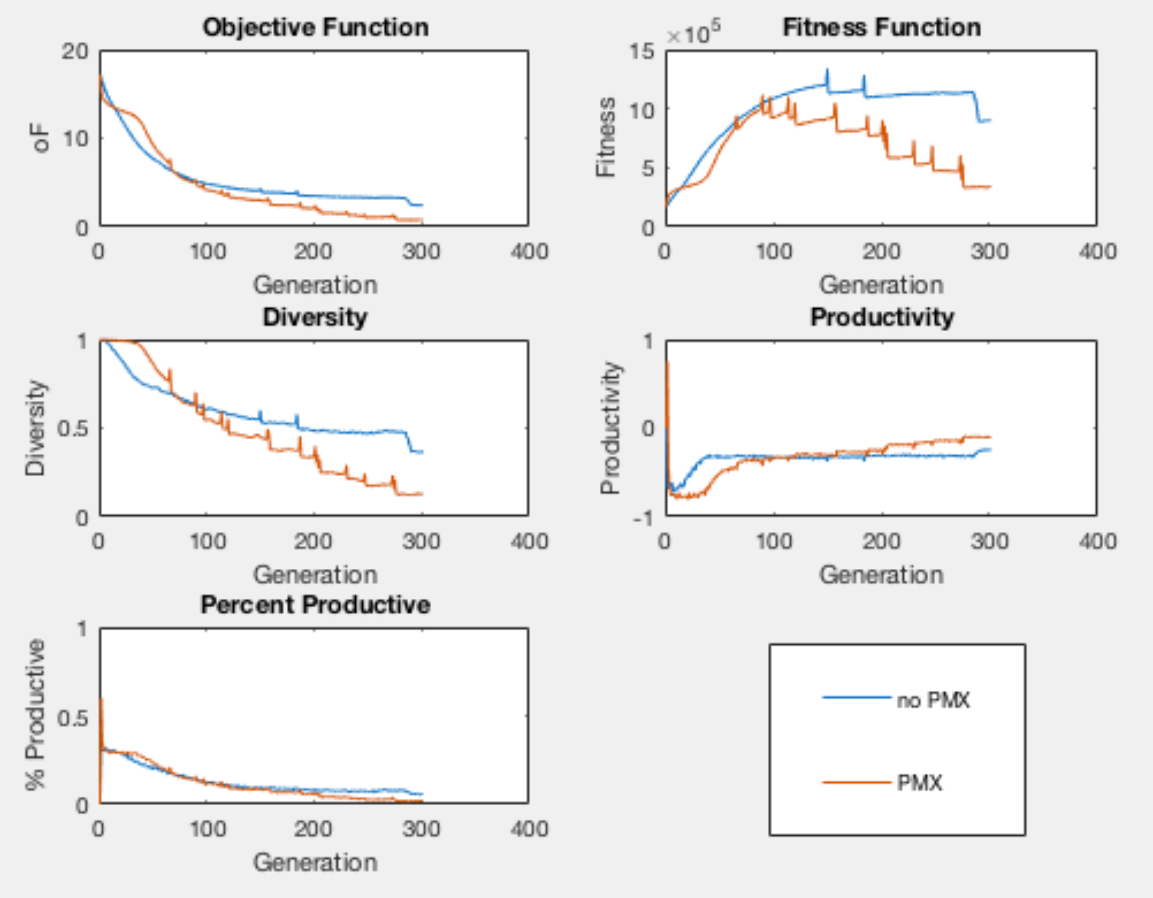
Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: linear
Crossover Points: **VARIES**
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (false, false, **VARIES**)

A.3 Linear Fitness – Effects of Partial-Mapped Crossover (PMX)



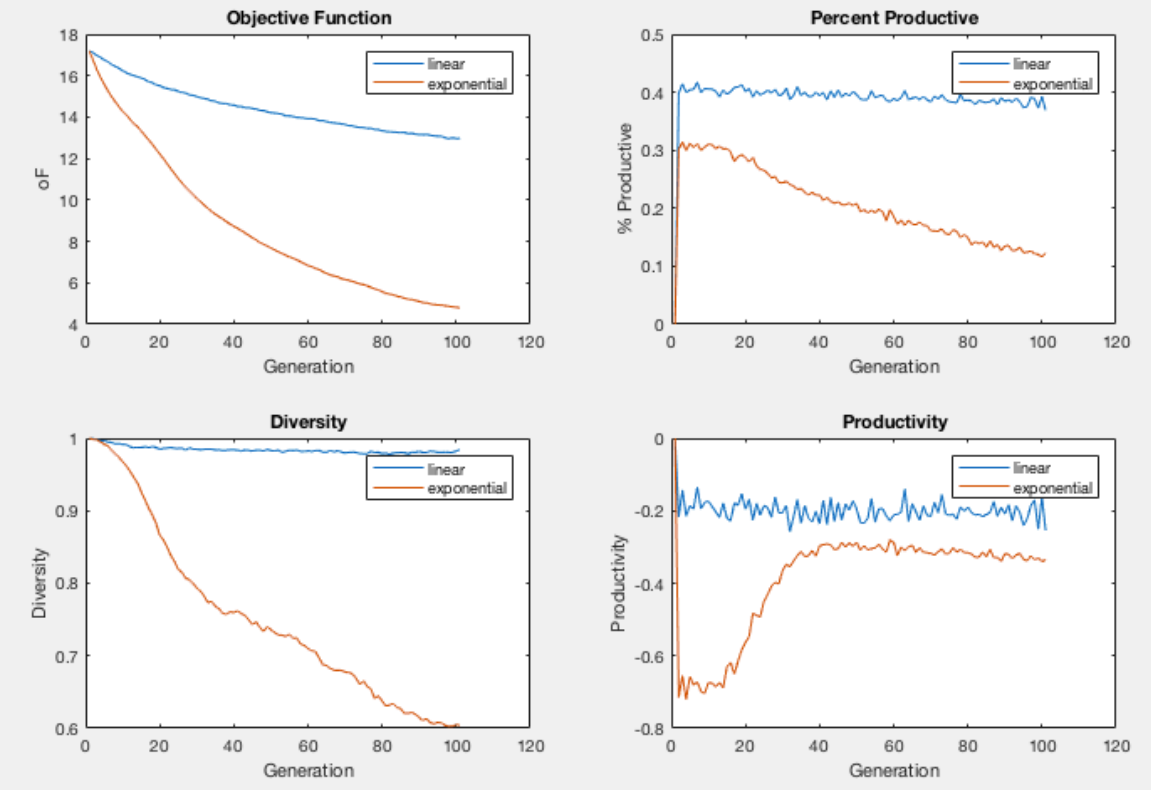
Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: linear
Crossover Points: 19
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (**VARIES**, false, false)

A.4 Exponential Fitness – Effects of PMX



Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: power
Crossover Points: 19
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (**VARIES**, false, false)

A.5 Comparison of Exponential and Linear Probability Functions



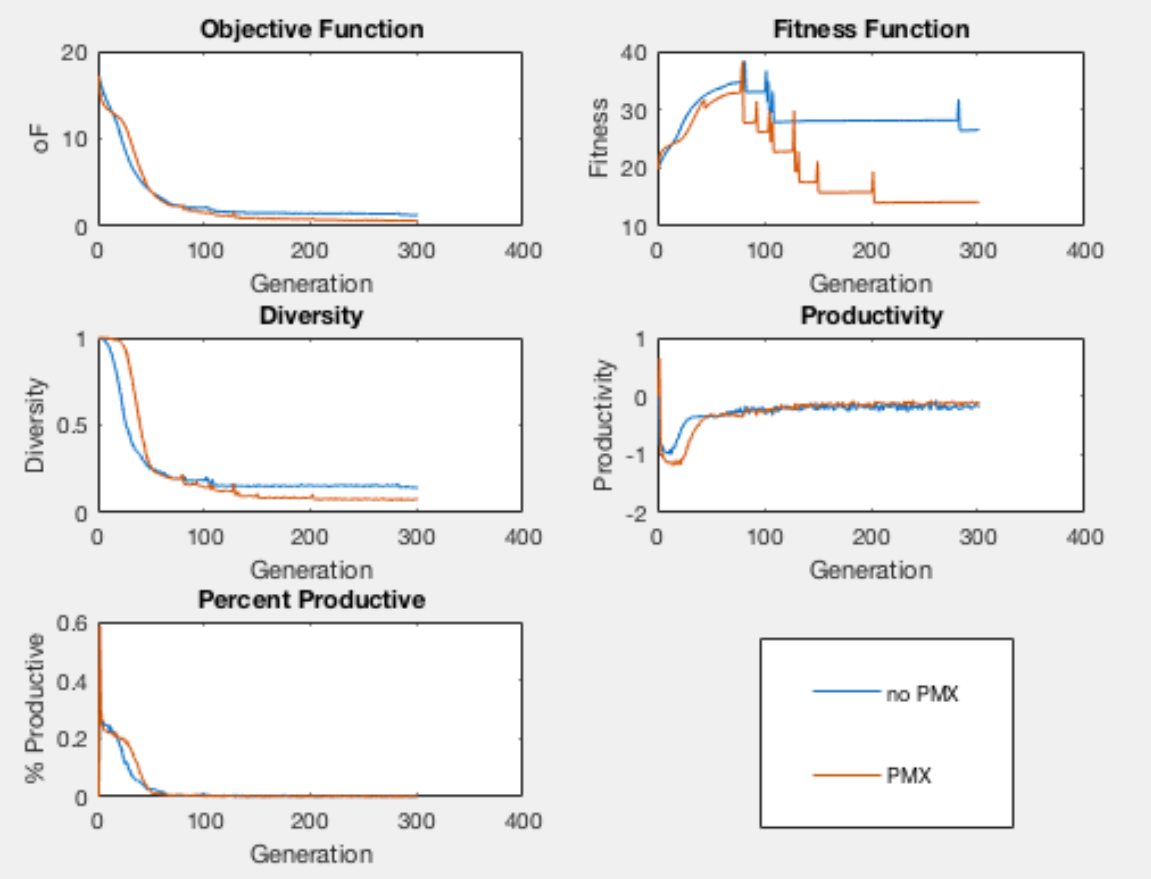
Linear Fitness

Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: linear
Crossover Points: 19
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (false, false, false)

Exponential Fitness

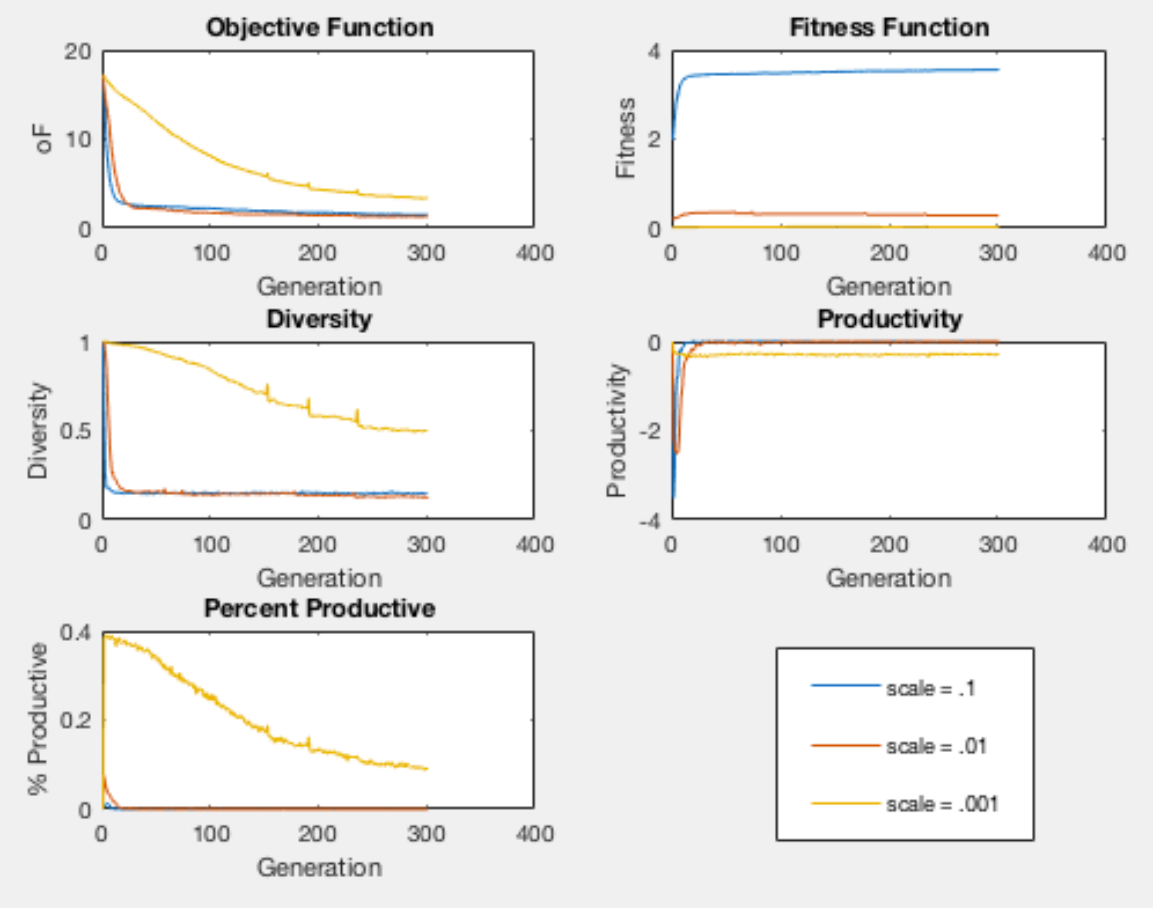
Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: power
Crossover Points: 19
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (false, false, false)

A.6 Linear Ranking - Effects of PMX



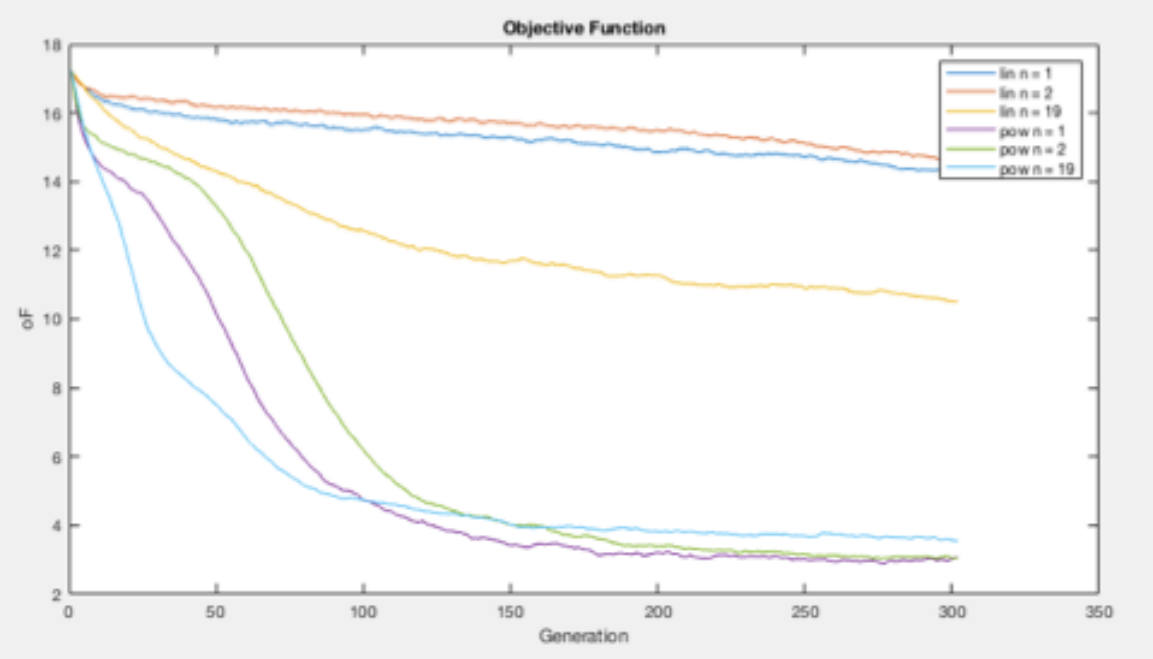
Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: lRank
Crossover Points: 19
Scaling: 1.0
Displacement: 0.0
(PMX, Unique, Uniform): (**VARIES**, false, false)

A.7 Natural Rank – Comparison over Varying Scaling Factors



Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: eRank
Crossover Points: 19
Scaling: **VARIES**
Displacement: 0.0
(PMX, Unique, Uniform): (false, false, false)

A.8 Comparison of Exponential and Linear Probability



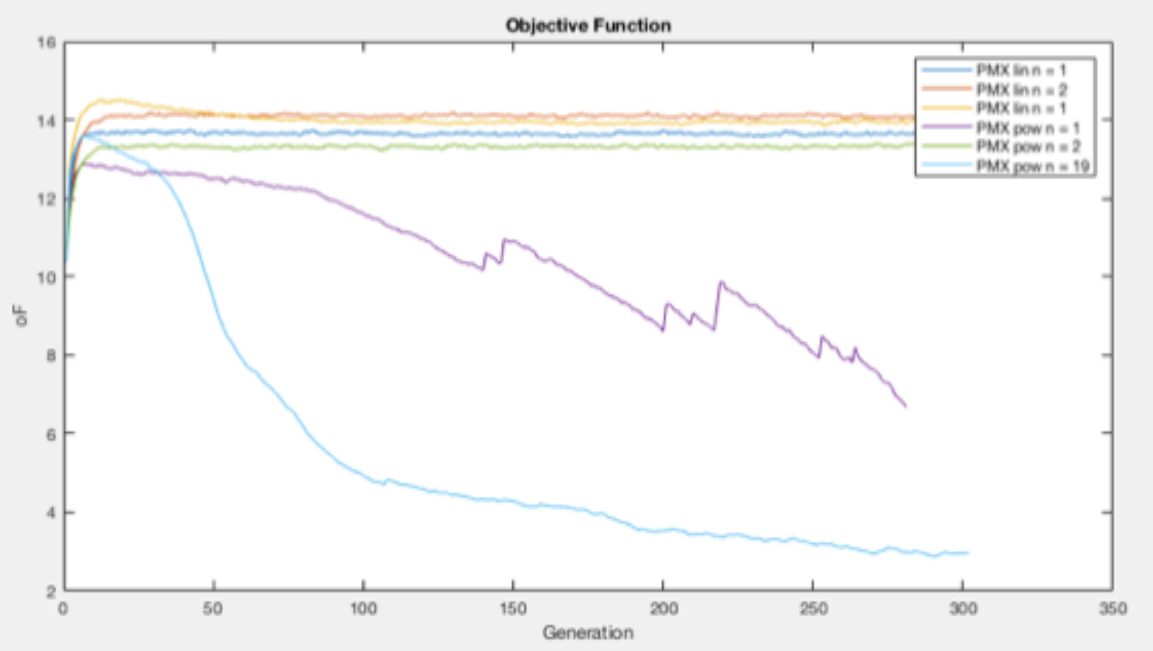
Linear Fitness

Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: linear
Crossover Points: **VARIES**
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (false, false, false)

Exponential Fitness

Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: power
Crossover Points: **VARIES**
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (false, false, false)

A.9 Comparison of Exponential and Linear Probability using PMX



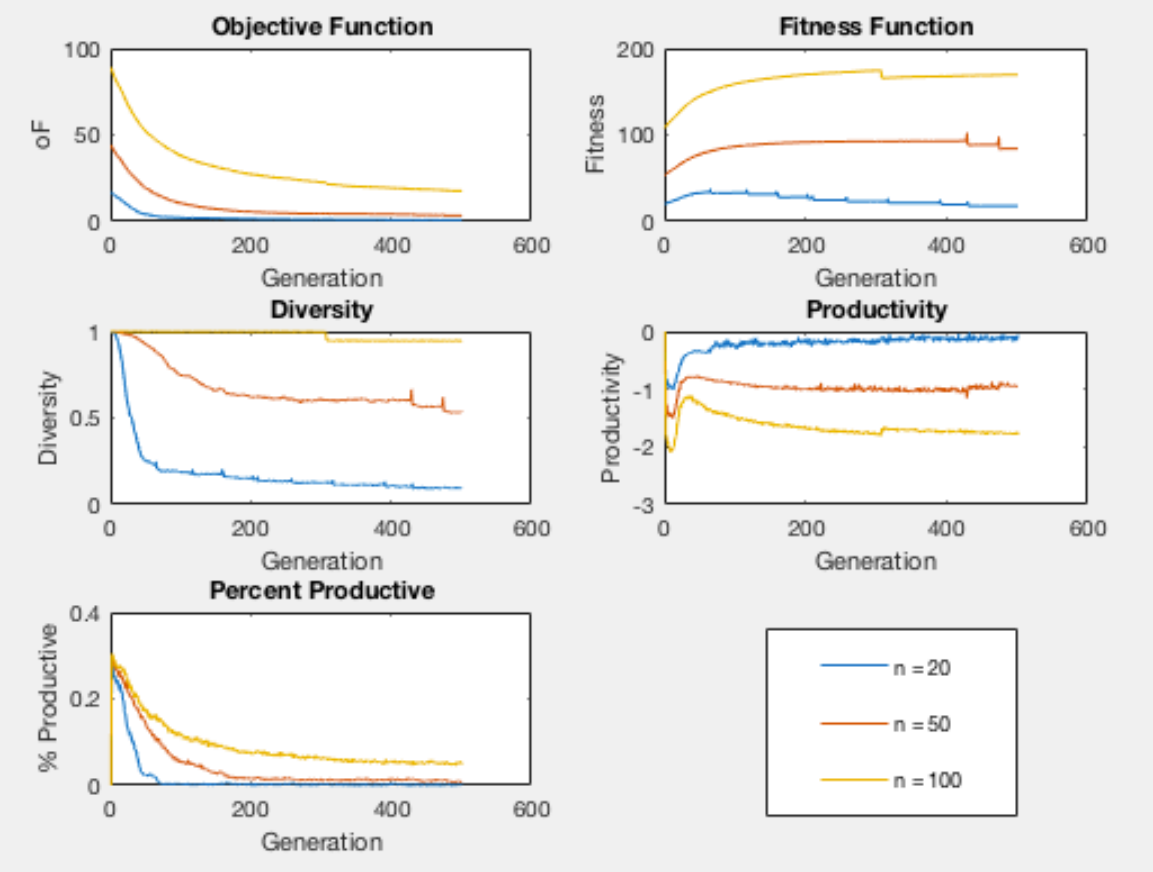
Linear Fitness

Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: linear
Crossover Points: **VARIES**
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (true, false, false)

Exponential Fitness

Assumptions:
Number of Queens: 20
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: power
Crossover Points: **VARIES**
Scaling: 4.0
Displacement: 0.0
(PMX, Unique, Uniform): (true, false, false)

A.10 Linear Rank – Comparison over Varying Number of Queens



Assumptions:
Number of Queens: **VARIES**
Population Size: 1000
Mutation Probability: 0.01
Crossover Method: lRank
Crossover Points: **VARIES**
Scaling: 1.0
Displacement: 0.1
(PMX, Unique, Uniform): (false, false, false)

APPENDIX B – Sample Solutions

B.1 Sample Solutions for N Queens = 20

1. [18, 16, 7, 12, 3, 6, 14, 9, 0, 15, 4, 1, 10, 8, 17, 11, 13, 2, 5, 19]
2. [17, 4, 8, 13, 5, 19, 2, 12, 6, 3, 1, 10, 16, 0, 9, 15, 18, 14, 11, 7]
3. [13, 10, 12, 7, 4, 19, 9, 18, 1, 17, 6, 16, 0, 8, 3, 14, 2, 15, 5, 11]
4. [6, 13, 5, 14, 9, 18, 15, 1, 8, 16, 2, 0, 3, 11, 4, 19, 12, 10, 17, 7]
5. [16, 13, 15, 8, 2, 14, 9, 3, 10, 19, 17, 1, 18, 0, 11, 7, 5, 12, 6, 4]

B.2 Sample Solutions for N Queens = 50

1. [12, 33, 15, 8, 46, 19, 30, 1, 24, 6, 25, 37, 48, 14, 39, 5, 2, 26, 22, 38, 41, 31, 20, 7, 4, 45, 0, 27, 36, 17, 47, 13, 23, 35, 40, 3, 9, 44, 49, 32, 11, 28, 34, 42, 29, 21, 16, 43, 10, 18]
2. [28, 18, 46, 26, 35, 44, 15, 2, 14, 31, 48, 1, 30, 11, 29, 12, 41, 43, 45, 13, 16, 23, 34, 8, 37, 0, 33, 7, 19, 6, 38, 36, 21, 17, 3, 27, 39, 47, 42, 25, 5, 40, 10, 22, 32, 9, 24, 4, 49, 20]
3. [21, 29, 39, 30, 22, 27, 37, 4, 10, 1, 5, 16, 13, 49, 20, 14, 41, 46, 28, 9, 35, 47, 2, 43, 32, 36, 45, 23, 12, 15, 17, 6, 44, 40, 48, 3, 0, 8, 26, 42, 38, 34, 18, 24, 33, 7, 19, 25, 11, 31]
4. [7, 39, 32, 21, 9, 11, 8, 23, 42, 18, 29, 46, 6, 34, 27, 0, 13, 15, 44, 14, 35, 43, 36, 33, 48, 45, 2, 38, 20, 25, 19, 22, 5, 41, 37, 16, 10, 49, 4, 40, 28, 3, 24, 30, 1, 31, 17, 47, 12, 26]
5. [26, 22, 3, 41, 20, 35, 33, 29, 12, 21, 24, 31, 46, 2, 14, 40, 1, 45, 49, 0, 44, 16, 27, 10, 23, 7, 43, 4, 47, 36, 13, 17, 6, 48, 25, 38, 15, 39, 28, 32, 34, 9, 30, 18, 42, 37, 11, 5, 8, 19]

B.3 Sample Solution for N Queens = 100

[92, 17, 46, 75, 57, 34, 52, 89, 69, 45, 43, 15, 70, 61, 21, 13, 39, 6, 81, 79, 22, 58, 72, 14, 41, 8, 26, 54, 90, 35, 94, 74, 44, 53, 88, 56, 78, 9, 24, 86, 32, 29, 5, 83, 19, 76, 51, 62, 1, 10, 7, 0, 93, 27, 63, 68, 33, 3, 84, 97, 95, 85, 28, 77, 11, 55, 96, 4, 71, 38, 40, 16, 82, 31, 42, 25, 87, 64, 37, 80, 99, 12, 2, 47, 65, 50, 66, 49, 23, 30, 98, 73, 20, 60, 67, 18, 48, 36, 91, 59]