

# A MEMETIC ALGORITHM FOR AUTOMATED MUSIC COMPOSITION

by

DEREK WELLS

Advisor

DR. HALA EL AARAG

A senior research project submitted in partial fulfillment of the requirements  
for the degree of Bachelor of Science  
in the Department of Mathematics and Computer Science  
in the College of Arts and Science  
at Stetson University  
DeLand, Florida

Spring Term  
2011

## **ACKNOWLEDGMENTS**

The author would like to take this opportunity to thank Dr. El Aarag for all of her help in completing this project.

The author thanks the rest of the Computer Science Department for their continued support.

The author would also like to thank those who participated in the survey.

# TABLE OF CONTENTS

I. INTRODUCTION .....	9
II. RELATED WORK .....	10
II.I Applied Memetic Algorithms .....	11
II.II Algorithmic Music Composition .....	12
III. GENERAL MEMETIC ALGORITHMS .....	13
IV. MUSIC COMPOSITION .....	15
V. A MEMETIC ALGORITHM FOR COMPOSITION .....	16
V.I Initialization .....	16
V.II Stopping Conditions .....	18
V.III Local Search .....	18
V.III Crossover and Mutation .....	20
V.IV Population Regeneration .....	21
V.V Evaluation .....	21
VI. FITNESS FUNCTION .....	22
VII. PARENT SELECTION METHODS .....	24
VII.I Random .....	24
VII.II Elitism .....	25
VII.III Tournament .....	26
VII.IV Roulette .....	27
VIII. CROSSOVER TECHNIQUES .....	28
VIII.I Full Crossover .....	28
VIII.II Selected Crossover .....	29

IX. TESTING RESULTS .....	30
IX.I Selection Methods .....	31
IX.II Fitness Function Weighting .....	33
IX.III Percentage of Population as Parents .....	35
IX.IV Population Size .....	38
IX.V Crossover Techniques .....	40
IX.VI Genetic vs. Memetic .....	41
IX.VII Subjective Test .....	44
IX.VIII Example .....	48
X. CONCLUSION .....	49
XI. REFERENCES .....	51

## LIST OF FIGURES

Figure 1: Memetic Algorithm .....	16
Figure 2: Local search function .....	20
Figure 3: Pseudocode for Random parent selection .....	24
Figure 4: Pseudocode for Elitism parent selection .....	26
Figure 5: Pseudocode for Tournament parent selection .....	27
Figure 6: Pseudocode for Roulette parent selection .....	28
Figure 7: Parent one using Full Crossover.....	28
Figure 8: Parent two using Full Crossover .....	28
Figure 9: Offspring using Full Crossover .....	29
Figure 10: Parent one using Selected Crossover.....	29
Figure 11: Parent two using Selected Crossover .....	29
Figure 12: Offspring using Selected Crossover .....	30
Figure 13: Average number of generations of parent selection methods .....	31
Figure 14: Average time to completion for parent selection methods.....	32
Figure 15: Average ratio for acceptable offspring for parent selection methods .....	32
Figure 16: Average number of generations for fitness function weightings .....	34
Figure 17: Average time for fitness function weightings .....	34
Figure 18: Average ratio for fitness function weightings .....	35
Figure 19: Average generations for various percentages.....	36
Figure 20: Average time for various percentages .....	36
Figure 21: Average ratio for various percentages.....	37
Figure 22: Average generations for various population sizes.....	38

Figure 23: Average time for various population sizes .....	39
Figure 24: Average ratio for various population sizes.....	39
Figure 25: Average generations for crossover techniques.....	40
Figure 26: Average time for crossover techniques .....	40
Figure 27: Average ratio for crossover techniques.....	41
Figure 28: Average generations for genetic and memetic algorithms .....	42
Figure 29: Average time for genetic and memetic algorithms .....	42
Figure 30: Average ratio for genetic and memetic algorithms .....	43
Figure 31: Votes yea and nay for each composition.....	45
Figure 32: Difference of yeas and nays for each composition.....	45
Figure 33: Votes yea and nay for each crossover technique.....	46
Figure 34: Votes yea and nay for each parent selection method .....	46
Figure 35: One measure of pre-algorithm music .....	49
Figure 36: One measure of post-algorithm music.....	49

## LIST OF TABLES

Table 1: Note value representations.....	17
Table 2: A chromosome's note values and duration values .....	18
Table 3: A chromosome's converted note values and duration values.....	18
Table 4: Table of compositions used for subjective testing.....	44
Table 5: Chi-square test results.....	48

## **ABSTRACT**

Music has been an integral part of society for hundreds of years. Since the advent and rise of computers and computer technologies, musicians have utilized these advances in their craft, be it in electronic music, or the notation of musical scores. The field of memetic computing is a new and developing one. Research is being done in applying memetic algorithms to real world problems. This research will offer a novel approach for a memetic algorithm to produce quality musical compositions. At the time of writing this paper, there were no proposals found in any literature for the use of memetic algorithms to compose music. This paper presents the results of the development of four parent selection methods, a fitness function, and two crossover techniques. Also present are objective and subjective testing results of the effectiveness of the algorithm using the different parent selection methods, fitness function weighting, and crossover techniques. The test results also show how the memetic algorithm improves over the genetic algorithm.

## I. INTRODUCTION

Music composition systems have been around for many years, especially in the form of expert systems. As computer technology advances, composition systems have become increasingly more powerful and complex. For computer-assisted music composition, the majority of the software that falls under this umbrella acts mostly as a tool to assist with visualizing composition, such as PatchWork or OpenMusic, and not as software that composes music directly [ASS99]. There are expert systems and genetic algorithms in existence that self-create works of music that are based on the analysis of a user's input or the creation of the music from scratch [ARI 05]. An increasingly popular method of composition using algorithms is that of an interactive genetic algorithm [NAK09][UNE01]. This involves using a genetic algorithm to produce sections of music, which are then represented back to the user in either graphical or aural form, allowing the user to evaluate the 'goodness' of the piece, until a predetermined number of 'good' sections are available. Genetic algorithms are synonymous with evolutionary algorithms; these are algorithms which are developed and designed using natural evolution and biology as inspiration, and have been used with great success in many real world optimization applications [NGU07].

While the concept of memetics itself has been in existence for years, memetic computing is a relatively new field of study for computer scientists. 'Meme's have been defined in many various ways, coined originally by Dawkins [DAW76], and essentially can be summarized as units of social information upon which we construct our culture [REA01][NGU07]. Memetic algorithms represent the next step in working with genetic algorithms. A memetic algorithm can be seen as an amalgamation of an evolutionary algorithm and individualized improvement procedures [NGU07]. The purpose behind memetic algorithms is to improve efficiency in

optimal-solution convergence; studies have shown that memetic algorithms do this more efficiently than heuristic or genetic algorithm approaches [NGU09].

This paper presents an effective memetic algorithm for music composition that builds upon existing expert system and genetic algorithm based approaches to the problem. The following paper is organized as follows. Section II presents related works in the field of applied memetic algorithms and music composition. Described are real-world applications of memetic algorithms and pre-memetic algorithmic music composition systems. Section III presents a definition of a memetic algorithm and discusses the various aspects involved in their design. Section IV explains some basic rules of composition in Western music theory. Section V presents the proposed memetic algorithm for music generation. Section VI offers the development of a fitness function for use with the algorithm. Section VII describes four different parent selection methods appropriate for use with the memetic algorithm. Section VIII describes two crossover techniques employed with the algorithm. Section IX shows the results of extensive testing of various components of the algorithm, including subjective testing of the created music. Section X offers a conclusion to the paper.

## **II. RELATED WORK**

In this section, review on the research done into papers and literature concerning applied memetic algorithms and music composition systems built upon artificial intelligence methods and other algorithms can be found.

## II.1 Applied Memetic Algorithms

In the last few years, research in the field of memetic algorithms has led to the development of algorithms for solving various problems. Several authors [TIN10, FER10, KON07] have focused on creating memetic algorithms for improving and optimizing the lifetime of wireless sensor networks. In [TIN10], the authors deem that the use of heuristic algorithms is too inefficient due to the trade-off between solution quality and run time. The paper also discusses the use of a genetic algorithm, which solves for near optimal solutions in acceptable time, but suffers from the dependency of needing information beforehand that is implausible to obtain. Their solution is a memetic algorithm which uses a Darwinian evolutionary algorithm combined with a Lamarckian localized improvement algorithm to find the optimal solution. The Lamarckian theory states that a parent can pass knowledge or characteristics that they obtain during their life to their offspring. Authors Lui [LUI08] and Bontoux, et. al [BON10], work with memetic algorithms in an attempt to solve large scale traveling-salesman problems. Lui [LUI08] utilizes the ‘nearest neighbor’ algorithm that is already commonly used to solve large scale traveling salesman problems, and combines it with localized search and population recombination procedures. The main focus of [BON10] is the localized search crossover procedure, with a comparison of results of their proposed crossover algorithm against four crossover techniques used in memetic algorithms. Samanlioglu, et. al [SAM08]., took the traveling-salesman problem a step further and proposed a memetic algorithm for solving multi-objective traveling salesman problems; effectively to optimize multiple objectives, such as time, distance traveled, cost, and others. The authors propose an algorithm consisting of a random-keys genetic algorithm with a 2-opt local search feature. Further research in memetic algorithms was

done by Lu and Hao [LU10] in focusing on developing an algorithm for solving graph coloring problems by using an algorithm which builds upon a popular meta-heuristic algorithm currently employed in solving graph-coloring and applying a localized improvement feature.

## **II.II Algorithmic Music Composition**

Algorithmic composition for music has been around for a number of years, in the form of expert systems, heuristic algorithms, and, more recently, genetic algorithms. Pope [POP95] explains a history of fifteen years of computer-assisted composition in a paper written 15 years ago. In that time, music composition systems have moved even further. The papers of Dion [DIO06] and Chiu et al. [CHI06] propose music composition systems that work on the premise of deconstructing a user's input set of music examples and creating new works from discovered patterns and common musical structures. Nakamura et al [NAK09] proposes a system to produce music and lyrics based on the input by the user, and also utilize an interactive genetic algorithm in which each section of music produced is displayed to the user to receive input on its fitness, an idea proposed by Unehara and Onisawa [UNE01] as the optimal way to evaluate generated works. Ariza [ARI05] attempts to codify the lexicon and main descriptors of computer-assisted algorithmic composition systems. Marques et al [MAR00] propose the use of genetic and evolutionary algorithms for music composition, which are essential to the idea of developing a memetic algorithm for the same purpose. They also describe in detail the encoding process in which the chromosomes are created within the genetic algorithm. A number would be randomly generated between -128 and 127, which one can represent in a single byte, and a series of these numbers would be placed together. Note values are represented by the values from -48 to 49,

with 0 being the middle C, sustains are represented with values -99 to -49 and 48 to 99, and pauses are the values from -128 to -100 and 100 to 127. In [SHE08], the authors use a similar approach, but focus on generating a melody. The paper proposes following the MIDI standard of assigning middle C the value of 60. The authors decide to use a 2 octave range, with values ranging from 48 to 72. They also determine values for six musical durations: thirty-second notes (.125), semiquavers (.25), quavers (.5), crotchets (1), minims (2), and semibreves (4). In their algorithm, each chromosome is divided into two partitions; one for pitch, and one for duration.

### **III. GENERAL MEMETIC ALGORITHMS**

Memetic algorithms are a fairly recent development in the field of computer science. A memetic algorithm is a combination of a genetic algorithm with a localized improvement feature. Studies have shown that memetic algorithms tend to find high-quality solutions more efficiently than straightforward genetic algorithms or heuristic algorithms [NGU09]. Digalakis and Margaritis [DIG04] performed a comparison on various local search techniques for memetic algorithms, and came to the conclusion that population size is perhaps the most important parameter to control for efficiency in the algorithm. Another important factor in developing memetic algorithms is the balance in frequency and intensity of the evolution of the population and the individualized learning [NGU07]. In all problem specific instances of memetic algorithms researched, there can be found four main components; the initial generation of a population by use of a genetic algorithm, a localized search for evaluation and improvement of individuals in the population, the crossover and mutation operations of parents to produce unique offspring, and the regeneration of the population to introduce the offspring and remove weak individuals [TIN10, LU10, FER10, LIU08, SAM08, BON10, KON07, DIG04, NGU07,

NGU09]. The steps of a memetic algorithm are described in [TIN10]: The typically genetic algorithm [FER 10] produces an initial population whose individuals are termed ‘chromosomes’. The local search chooses two of these chromosomes to act as parents, usually by selecting on the basis of ‘fitness’ values. The crossover operation then creates offspring by using the data from the two parents and passing the best qualities on. These offspring are sometimes mutated and enhanced to improve their fitness for propagation. A survival procedure determines whether the offspring are fitter to survive than the existing members of the population; if so, the offspring are entered into the population as a replacement for the least fit individuals. This cycle continues until an optimal, or acceptable, solution is found [TIN10]. In [LIU08], the author gives more detail on parent selection methods: fitness-proportionate, elitism, and tournament selection. Fitness-proportionate simply weights more ‘fit’ solutions heavier, leading them to be more likely to be chosen to produce offspring. Tournament selection works by using the fitness-proportionate values and systematically weighting solutions against each other in a tournament format, until the two remaining ‘best’ parents are left. These are left to crossover and produce offspring. In the elitism format, the top N solutions are propagated to the next generation of the population. The remaining solutions are generated using the tournament selection method. In [LU10], the author makes a claim that the crossover component to memetic algorithms is typically the most important, as it produces the new and improved individuals to add to the population. The authors also differentiate between the two classifications of crossover operators; the assignment operator, and the partition operator. They propose that the partition operator acts superiorly to the assignment operator, because the partitions allow for the differentiation of ‘good’ and ‘bad’ properties to pass to offspring. Their paper also explains that an offspring

should be inserted into the population as a replacement for the ‘worst’ parent, and should not be identical, or too closely similar, to an existing individual in the population.

#### **IV. MUSIC COMPOSITION**

In accordance with Western music theory, a musical scale is a series of notes that start with a root, or tonic, note, and ascend in order by following a specific pattern in relation to the appropriate scale. The twelve notes in Western music are; A, A#, B, C, C#, D, D#, E, F, F#, G, and G#. These notes circle back onto themselves. G# would then move back to A. A half step is simply the distance, or interval, between a note and its immediate successor. A whole step is two half steps. Therefore, to build, for example, a C major scale, you begin with the tonic note, C. The notes then follow a pattern of whole and half steps as such; whole, whole, half, whole, whole, whole, half. We then end up with the series of notes C, D, E, F, G, A, B. These are the notes of a C major scale. Major scales always follow the same pattern of steps. There are, however, a number of other scales one can build using different patterns. For the sake of the first generation of this memetic algorithm, only major scales will be used.

The duration of notes are typically referred to as note values, but for the sake of clarity in this paper, they will be referred to as durations to prevent confusion. The various durations in music theory are as follows; a longa (four notes long), a breve (two notes long), a semibreve (one note long), a minim (half a note), a crotchet (quarter of a note), quaver (eighth), semiquaver (sixteenth), demisemiquaver (thirty-second), hemidemisemiquaver (sixty-fourth), and quasihemidemisemiquaver (hundred twenty-eighth). These durations refer to how long the note is played for. This paper will deal with only whole notes (semibreve) to thirty-second notes (demisemi-).

Time signatures, or meter, are a representation of how many beats are in each measure, and what note duration constitutes one beat. The most commonly used time signature in Western music is that of 4/4. This means there are 4 beats in a measure (top numeral), and each beat is made of a quarter note, or a crotchet. As stated, 4/4 is the most commonly used time signature, and is referred to as ‘common time,’ and therefore, our memetic algorithm will work within this meter.

## V. A MEMETIC ALGORITHM FOR COMPOSITION

```
population P = initialization();
while(!stopping_conditions)
{
    chromosome C1,C2 = local_search(P);
    offspring O = crossover(C1,C2);
    mutate(O);
    P = regeneratePopulation(O);
    evaluate(O);
}
```

Figure 1: Memetic Algorithm

### V.I Initialization

This procedure is to generate an initial population from which to proceed with. It will generate  $N$  members of a population, all with random genetic information. In this algorithm, a member of the population, hereafter chromosome, will consist of a series of values that indicate the notes and duration of those notes for one musical bar. Middle C, considered the ‘middle’ note in music composition and is found in the middle of a piano or keyboard, will be designated as the value of 60, in keeping with the MIDI protocol standards, and the values will range within 2 octaves; the octave leading to middle C, the octave beginning at middle C, and the C above this

octave. The integer values from 48 to 72 will represent all of the natural, sharp and flat notes within these octave ranges (see Table 1). Also, a 0 will indicate a rest, or no note being played. For randomization, each note will be equally likely to be chosen. The values for the duration of the note will be represented as follows; thirty-second notes (1), semiquavers (2), quavers (4), crotchets (8), minims (16), and semibreves (32). For randomization, durations will be assigned a weighted probability, with quavers, crotchets, and minims being more common than the other durations. In the event of a dotted note, which indicates the note be held again half the length of the original duration, the value will reflect 1.5x the original value assigned (e.g. a dotted quaver will be represented as 6). A chromosome will consist of two sets of values; one to hold the values of the notes, and one to hold the values of the durations (see Table 2). This algorithm, as stated earlier, will assume 4/4 time. This means that there will be the equivalent of a semibreve, or whole note, within each measure. The set of values for duration, therefore, must summate to be less than or equal to 32 in order to be fit. The number of values in the set for duration will equal the number of values in the set for pitch, assuring that each pitch will have a duration assigned to it (E.g. A set of 4 note values will indicate a set of 4 note durations).

**Table 1: Note value representations**

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71
72											

Table 2: A chromosome's note values and duration values

52	55	56	52	50	58
8	8	4	2	2	8

Table 3: A chromosome's converted note values and duration values.

E	G	G#	E	D	A#
1/4	1/4	1/8	1/16	1/16	1/4

## V.II Stopping Conditions

The stopping conditions chosen for the initial implementation will be whenever every chromosome conforms to a specified key (Cmaj, Amin, etc) and the melody within any given chromosome holds within one octave.

## V.III Local Search

The local search feature is that which defines a memetic algorithm apart from a genetic or evolutionary one. Its purpose is to enhance the quality of the solution by causing improvements in the individual chromosomes. The localized search for this algorithm will traverse through each

chromosome in the population, and will improve the chromosome by taking the first note value in the series of note values that does not belong to the specified key, and shifting it up or down to the nearest value that does. It will also check to ensure that the summation of values for duration is equal to 32. In the event that it is greater than 32, a check for dotted values will take place. If one is found, it will be reassigned as an un-dotted duration. If not, the largest value for duration will be found, and the next largest will be substituted in its place. If the summation is less than 32, the difference will be found and assigned as the duration of a rest at the end of the measure. In the event that each note does belong to the specified key, it will swap two of the note values and two of the duration values within itself at random. It will also assign a fitness value to each chromosome; one point for each note out of the key, one point if the duration does not equal 32, and one point if the notes within the chromosome are not within one octave. Therefore, higher values will equate with lower fitness. The two chromosomes with the lowest values are assigned as parents. In the event of a tie, they are chosen at random from the eligible chromosomes. The pseudocode can be found in Figure 2.

```
for each chromosome
{
    if summation of duration values != 32
    {
        if summation > 32
        {
            if dotted value exists
                undot value;
            else reduce largest value;
        }
        else
        {
            add new duration = to difference;
            add rest note value for this duration;
```

```

    }
  }
  for each note value
  {
    if note value does not belong in key
      shift value into key;
    else if all values in key
    {
      randomly swap two note values;
      randomly swap two duration values;
    }
  }
}

```

Figure 2: Local search function

### V.III Crossover and Mutation

The purpose of the crossover and mutation procedures is to generate new and improved individuals to repopulate the solution with. To do this, the procedure uses the results of the local search in finding the parents by comparing their fitness values.

The process for crossover in this memetic algorithm will be as follows. The child will inherit at random the duration values of either parent 1 or parent 2. Then the child will fill the number of available note values as determined by the number of duration values by randomly inheriting notes from both its parents. This way, the genetic information of each parent can be found in the child.

The process for mutation will follow as such after the crossover has finished. The duration values of the resulting child will be rearranged at random, so as to create a wholly new set of durations.

Once both procedures have completed, a unique individual will be created from the genetic information of both parents.

#### **V.IV Population Regeneration**

Repopulation occurs after the creation of a new offspring. The process goes as follows. The new offspring has its fitness value calculated, just as every other member of the population during the localized search. Then this fitness value is compared to all the members of the existing population. If the fitness value of the new individual is lower than the worst fitness value of an existing member (highest value equates with worst fitness), then the least fit member is removed from the population, and is replaced with the new individual.

#### **V.V Evaluation**

The evaluation procedure is used to determine whether the stopping conditions have been achieved. Each chromosome is evaluated individually to determine if it satisfies the stopping condition. If all chromosomes are found to satisfy the conditions, the algorithm completes, and the chromosomes are returned to the user. If any chromosome fails to satisfy the conditions, the algorithm continues to perform.

## VI. FITNESS FUNCTION

In order to evaluate the individuals of a population, the algorithm must have a fitness function to ‘score’ each member. The algorithm will stop when all of the members of the population have an acceptable fitness value.

The fitness function has four separate components; one for tonality, one for melody, one for harmony, and one for rhythm. For the rest of this section, consider  $P_n$  to represent the  $n$ th member of a population  $P$ .

The fitness function for tonality,  $F_t(P_n)$ , is based on the notes within a given individual. At the beginning of the algorithm, a note is specified as the desired key for the optimal solution. Also defined is the type of scale; in the current version, simply major or minor. The fitness function measures how well the notes of an individual conform to the desired key.  $F_t(P_n)$  is defined as the number of notes out of scale within that  $n$ th member of the population.

The melody fitness function,  $F_m(P_n)$ , is based on the intervals between the notes of an individual. The current model for this function is based on whether or not the interval between any consecutive notes in an individual is less than an octave.  $F_m(P_n)$  is defined as the number of intervals found in that  $n$ th member that exceed one octave.

The harmony function,  $F_h(P_n)$ , is used to determine the harmonic fitness of chord structures in an individual. The function evaluates each note in the chord structure and determines if the value is part of the appropriate scale.  $F_h(P_n)$  is defined as the number of notes in the chord of that  $n$ th member that are not part of the defined scale.

In order to account for the rhythm, or time signature, of an individual, we have incorporated a factor into the final fitness function to ensure that each individual conforms to the desired meter. Any individual who does not meet the desired time signature cannot reach 0, the ‘best’ an individual can be. In the full fitness function, 1 is added if the time signature is incorrect.

The full fitness function,  $F(P_n)$ , is the summation of the three aforementioned fitness functions and the rhythm factor, as shown in Equation (1).

*Fitness function*

$$F(P_n) = W_t * F_t(P_n) + W_m * F_m(P_n) + W_h * F_h(P_n) + R(P_n); \quad (1)$$

$F$  = full fitness function;

$P_n$  = nth member of population  $P$ ;

$F_t(P_n)$  = tonality fitness function;

$F_m(P_n)$  = melody fitness function;

$F_h(P_n)$  = harmony fitness function;

$R(P_n)$  = rhythm factor : = 1 if incorrect, = 0 if correct;

$W_t, W_m, W_h$  = integer values to represent weights

## VII. PARENT SELECTION METHODS

For the algorithm, we devised four different methods of parent selection. These methods include a Random selection technique, an Elitism technique, a Tournament selection, and a Roulette selection method.

### VII.I Random

The first selection method tested in the algorithm is the random selection method. As the name suggests, it simply selects chromosomes at random from the population pool to act as the parents for new individuals. The pseudocode for the random selection method can be found in Figure 3.

```
void randomParents(numParents, myPopulation, parents){  
    for(0 <= i < numParents)  
        generate random number t;  
        parents.add(myPopulation.elementAt(t));  
    }
```

**Figure 3: Pseudocode for Random parent selection**

## VII.II Elitism

The second selection method is elitism. In this method, each individual is assigned a fitness value via the fitness function. Using these scores, a percentage of the best, most fit individuals are used as parents. To start, the first numParents number of individuals from the population are chosen as the parents. Then, each member of the population after that is compared one by one to each of the parents. If a member of the population is found to be more fit than an existing parent, that parent is swapped out with that member. This continues until all the members have been compared against the existing parents. Then the original members of the population that were initially selected as parents are compared to the current set of parents to ensure that the fittest individuals are chosen as parents and a more fit individual was not replaced prematurely. The pseudocode for the Elitism method can be found in Figure 4.

```
void eliteParents(numParents, myPopulation, parents){
    for(0 <= i < numParents)
        parents.add(myPopulation.elementAt(i));
    for(numParents <= i < myPopulation.size)
        for each(chromosome : parents)
            if(chromosome.fitness > myPopulation.elementAt(i).fitness)
                replace chromosome with myPopulation.elementAt(i);
                break;
    for(0 <= i < numParents)
        for each(chromosome : parents)
            if(chromosome.fitness > myPopulation.elementAt(i).fitness)
```

```

        replace chromosome with myPopulation.elementAt(i);
    }
    break;
}

```

Figure 4: Pseudocode for Elitism parent selection

### VII.III Tournament

The third selection method is tournament selection. Every individual in the population is paired at random with another. The fitness values of each pair are compared. The fitter individual of the pair moves on to the next ‘round’, while the other is disqualified. This continues until there are a number of winners equal to the desired number of parents. Then this last group of winners are paired as the parents for new individuals.

```

void tournamentParents(numParents, myPopulation, parents){
    create temporary empty population;
    create vector of integers = size of myPopulation;
    add values 0 to index of last member of myPopulation;
    shuffle vector;
    if(myPopulation.size <= numParents)
        parents = myPopulation;
    else
        for(0 <= i < myPopulation.size/2){
            compare myPopulation element at i and myPopulation.size - i
            add most fit to temporary population;
        }
    tournamentParents(numParents, temporary population, parents)
}

```

```
}
```

Figure 5: Pseudocode for Tournament parent selection

## VII.IV Roulette

The final selection method that was created is the roulette selection method. This method works similarly to a roulette wheel, where the likelihood that an individual is chosen is proportional to its fitness value. Because a 0 is considered the ideal fitness for individuals in this algorithm, the size of each individuals 'slice' of the roulette wheel will be inversely proportional to their fitness value. Once the 'slices' have been determined, a number is generated at random. The individual with the range of numbers that contains this randomly generated number will be one parent. This continues until the desired number of parents is found.

```
void rouletteParents(numParents, myPopulation, parents){  
    int high = 0;  
    for each(chromosome : myPopulation)  
        if(chromosome.fitness > high)  
            high = chromosome.fitness;  
    create vector of integers = size of myPopulation;  
    for(0 <= i < myPopulation.size())  
        for(0 <= j < high - myPopulation.elementAt(i).fitness)  
            add i to vector of integers;  
    for(0 <= i < numParents)  
        shuffle vector of integers;  
    parents.add(myPopulation.elementAt(vector[i]));  
}
```

}

Figure 6: Pseudocode for Roulette parent selection

## VIII. CROSSOVER TECHNIQUES

Two crossover techniques were developed to determine the best way to create offspring. We felt it necessary to develop multiple ways of crossing over the genes from the parents in order to try and get different results and to evaluate the performance of each method.

### VIII.I Full Crossover

The first crossover technique tested is one in which the offspring gets both values of data, (note & duration), from a single parent. From either the mother or the father, an offspring will receive, one by one, entire encodings of note value and duration.

56	56	54	59
4	4	8	16

Figure 7: Parent one using Full Crossover

56	55	57	53	55
4	4	8	8	8

Figure 8: Parent two using Full Crossover

Using the parents shown in Figures 7 & 8, the offspring in Figure 9 will select the value pairs in the parents highlighted in bold, alternating between the two parents until it has enough notes and durations equal to the same number of notes and durations as the parent with the fewest number. Parent 1 has 4 notes, while parent 2 has 5. Therefore, the offspring will have 4 notes.

56	55	54	53
4	4	8	8

Figure 9: Offspring using Full Crossover

### VIII.II Selected Crossover

The second technique devised is different in that the offspring receives one half of a pairing from each of its parents, i.e. either all of its notes or all of its durations.

56	56	54	59
4	4	8	16

Figure 10: Parent one using Selected Crossover

56	55	57	53	55
4	4	8	8	8

Figure 11: Parent two using Selected Crossover

In the case of the above parents (Figures 10 & 11), the offspring in Figure 12 will have 4 notes, similar to technique one. The difference being that the offspring will receive all of its durations from the parent with the fewest notes, and the equivalent number of note values from the other parent. The bold numbers in the above example would be paired in the offspring.

56	55	57	53
4	4	8	16

Figure 12: Offspring using Selected Crossover

## IX. TESTING RESULTS

In order to find the most optimal version of the algorithm, three performance metrics were used to find an optimal solution; 1) the number of generations passed before finding an optimal solution, 2) the amount of time the algorithm takes to find an optimal solution, and 3) the ratio of ‘good’ to ‘bad’ offspring produced. A higher value of metric 3) indicates better performance. Metric 3) is the measure of how many offspring were accepted into the next generation’s population compared to those that were not.

The first test was to find the best of the parent selection methods described in Section VII. Next, we tested the weighting of the fitness function to determine how different weights would affect the performance of the algorithm. The next test was to change the percentage of the population to serve as parents and observe the performance of the algorithm. These percentages ranged from 10% to 50% in increments of 10 percentage points. Following that, we tested how the population size affected the algorithm. The sizes varied from the smallest being 100

individuals up to 1000 individuals. Lastly, we tested the two crossover techniques described in Section VIII to determine which was more effective.

### IX.I Selection Methods

In this section, we studied the performance of the algorithm utilizing the four different parent selection methods described in Section VII.

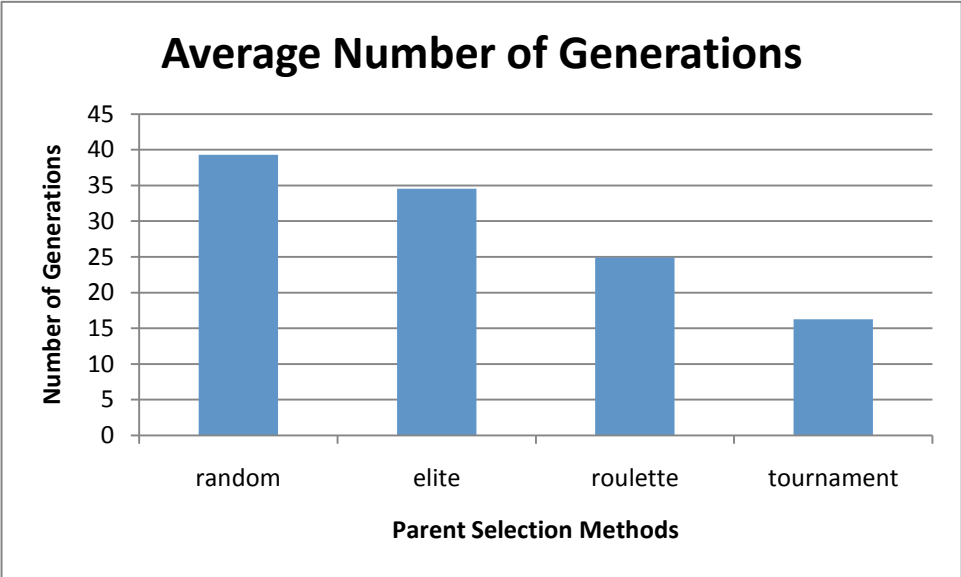


Figure 13: Average number of generations of parent selection methods

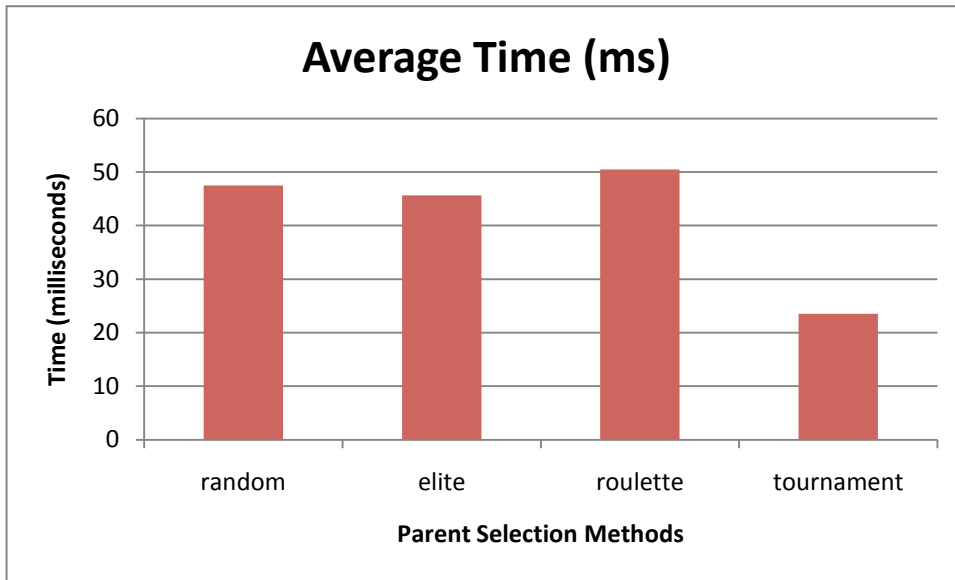


Figure 14: Average time to completion for parent selection methods

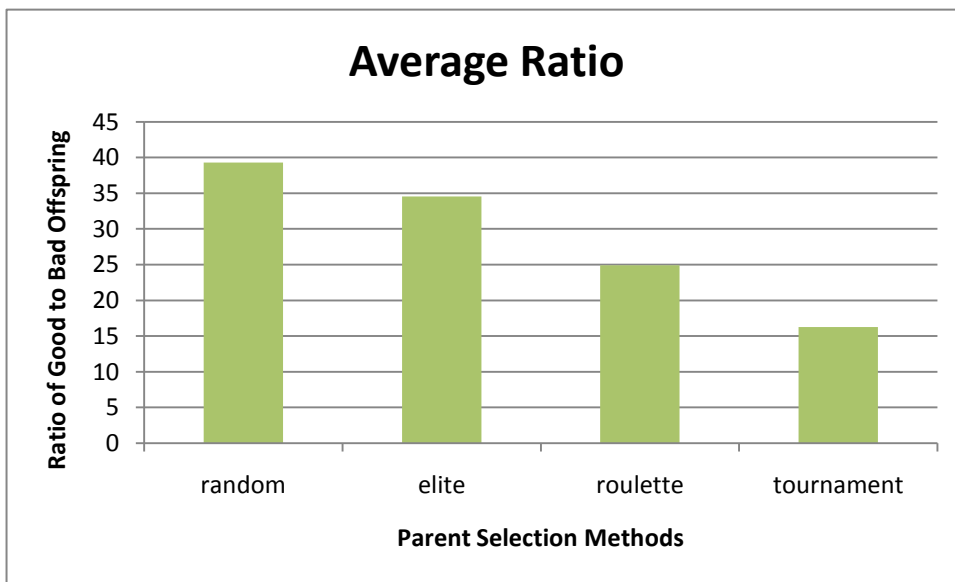


Figure 15: Average ratio for acceptable offspring for parent selection methods

These results were found using a population size of 100 with 20 parents after running the algorithm 50 times. The results show that the most effective method of parent selection is the

tournament method. Figures 13 and 14 show that it requires the least number of generations and the least amount of time to find the optimal solution, and Figure 15 shows it has the highest ratio of acceptable to unacceptable children produced during crossover. As expected, the least effective, as a whole, was the random selection method. Requiring the second longest amount of time, as shown in Figure 14, it requires the largest number of generations on average (Figure 13), and has the worst ratio of acceptable to unacceptable children (Figure 15). The results of an analysis of variance on each metric show that the differences in average time, ratio and generation are statistically significant, with a p-values of less than .0001 for each.

## **IX.II Fitness Function Weighting**

In this section, we studied the performance of the algorithm after using various weightings in the fitness function (see Section VI). For all experiments in the rest of this section, we used the tournament parent selection method as it was found in Section IX.I to be the most efficient. We used a population size of 100 individuals with 20% of the population acting as parents.

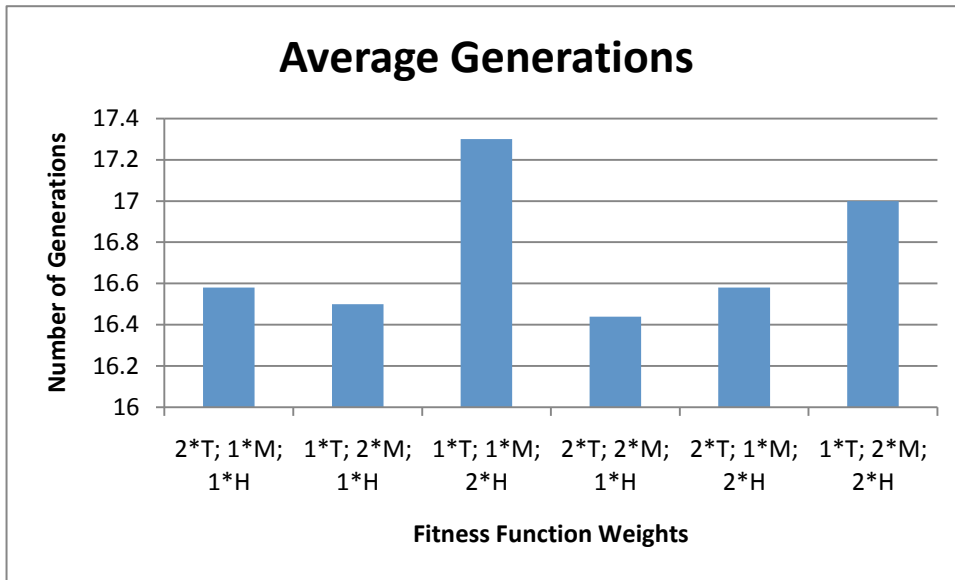


Figure 16: Average number of generations for fitness function weightings

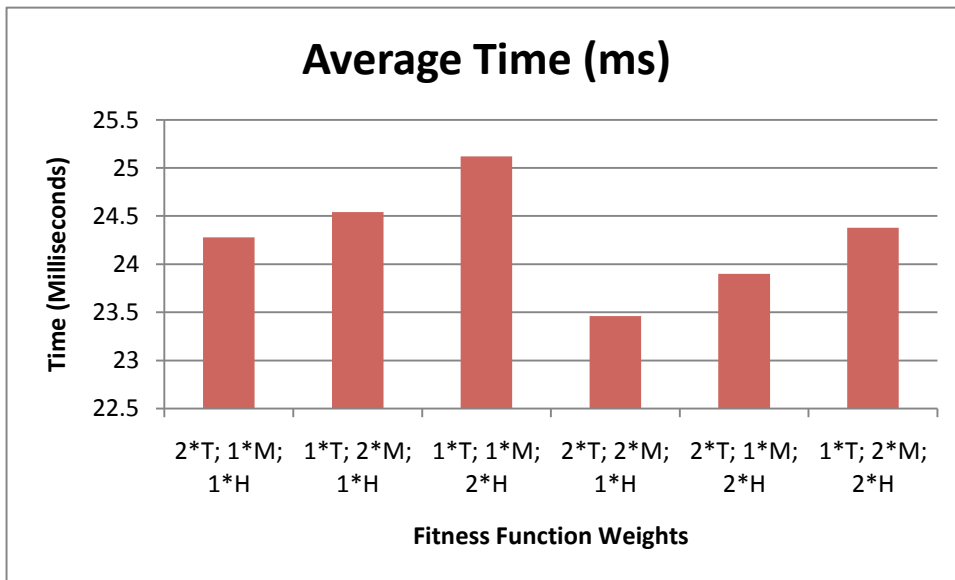


Figure 17: Average time for fitness function weightings

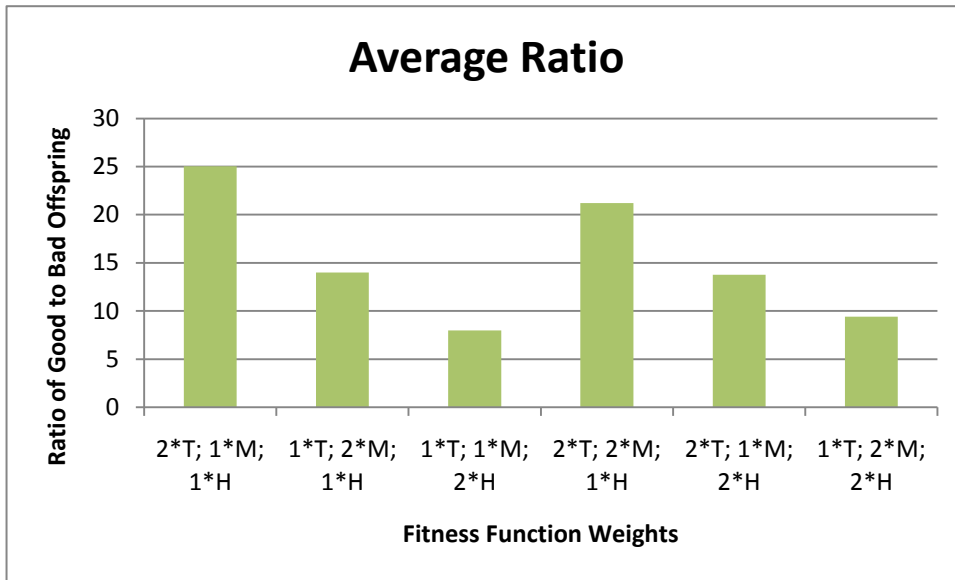


Figure 18: Average ratio for fitness function weightings

The test for various weightings of the fitness function reveal that the overall best weighting is  $2 * F_t(P_n) + 2 * F_m(P_n) + 1 * F_h(P_n)$ . By using this weighting, the algorithm requires the fewest generations on average and the least amount of time, on average, as shown in Figures 16 and 17, respectively. It also results in the second best ratio of acceptable to unacceptable children (Figure 18). The weighting of  $1 * F_t(P_n) + 1 * F_m(P_n) + 2 * F_h(P_n)$  provides the worst overall results.

### IX.III Percentage of Population as Parents

The tests in this section were to determine the effects of using different percentages of the population to act as parents. Using the tournament parent selection method that was determined to be the most effective in Section IX.I and the weighting found to be the most effective in

Section IX.II, with a population of 100 individuals, we used the performance metrics from Section IX to find the results found in Figures 19, 20 and 21.

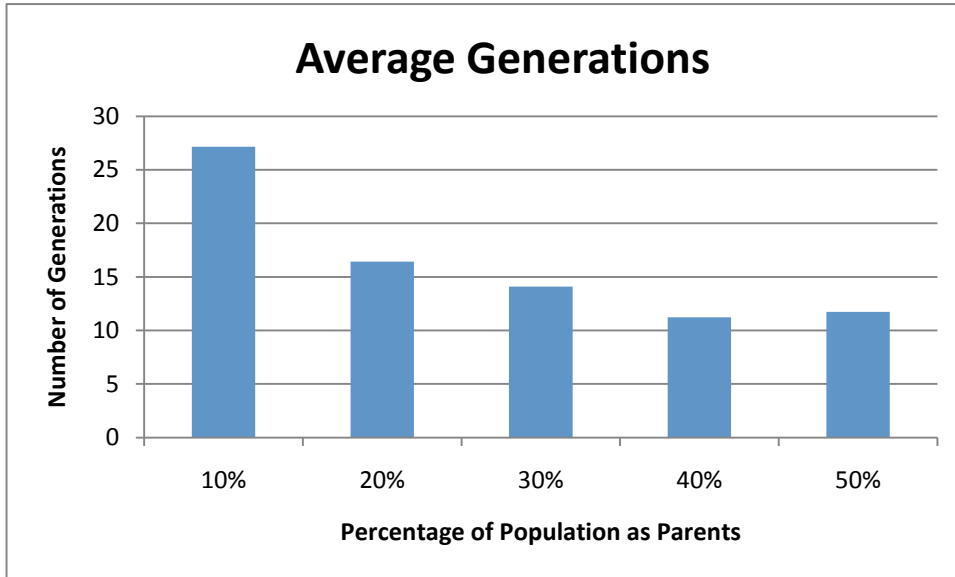


Figure 19: Average generations for various percentages

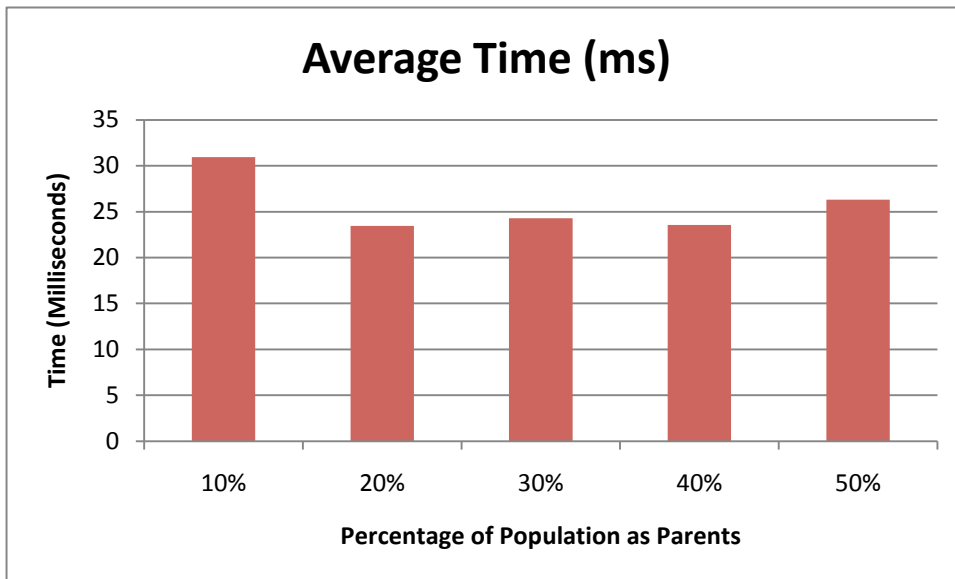


Figure 20: Average time for various percentages

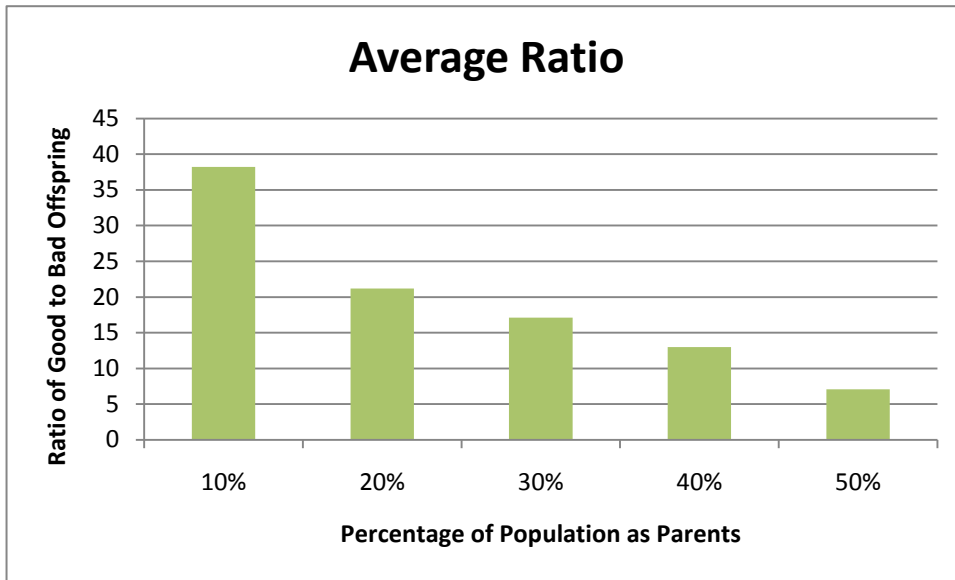


Figure 21: Average ratio for various percentages

The results show a trade off between the number of generations required for the algorithm to run to completion (Figure 19), and the average ratio of acceptable to unacceptable children produced during crossover (Figure 21). The higher the percentage of the population used, the fewer generations were needed, but the ratio became smaller. There seemed to be little correlation between the size of the percentage and the time required to complete the algorithm. The percentage chosen as the best is 20%. It yields the second highest ratio with the best time recorded, using the second largest number of generations.

## IX.IV Population Size

In this section, we study the effects of the population size on the effectiveness of the algorithm. Using the results from Sections IX.I, IX.II, and IX.III, we used varying population sizes to test the metrics.

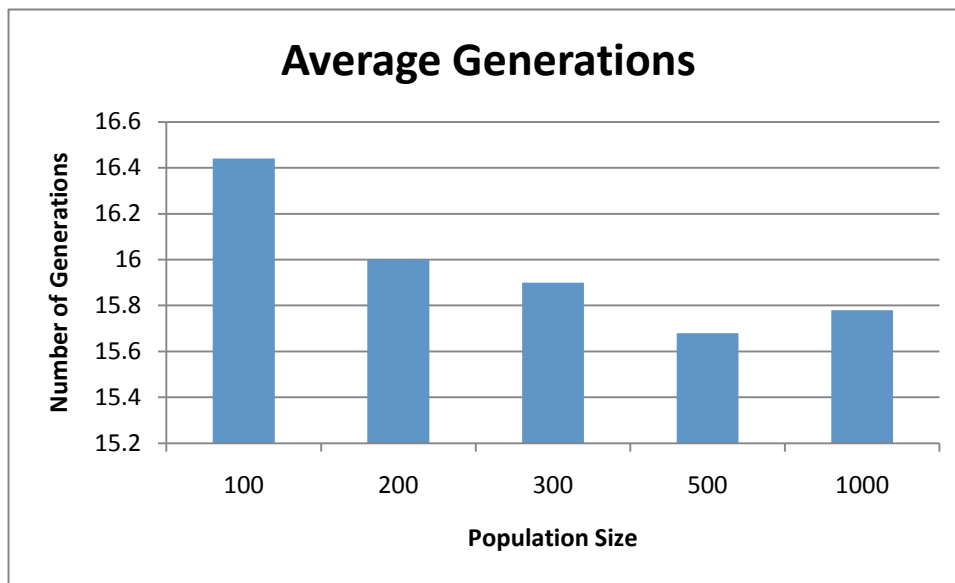


Figure 22: Average generations for various population sizes

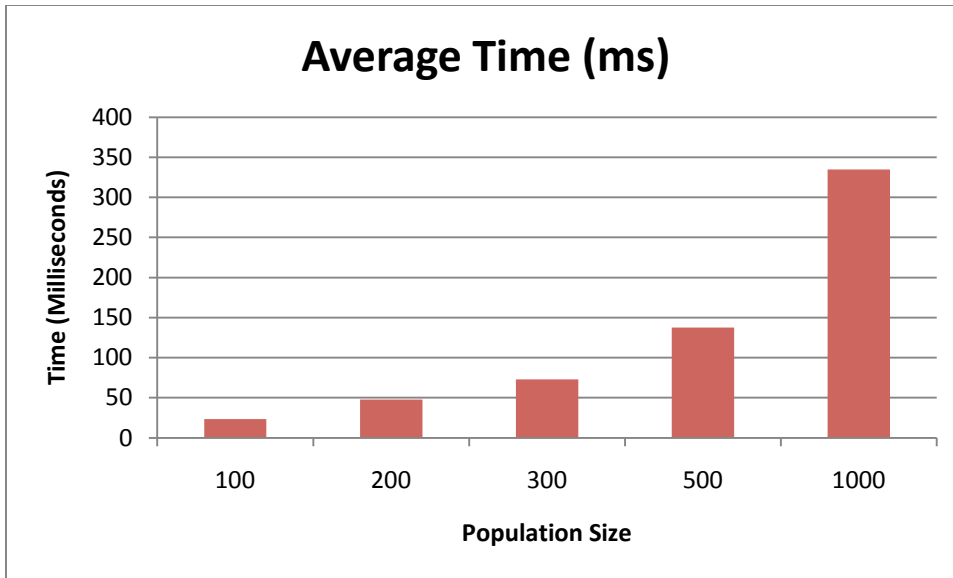


Figure 23: Average time for various population sizes

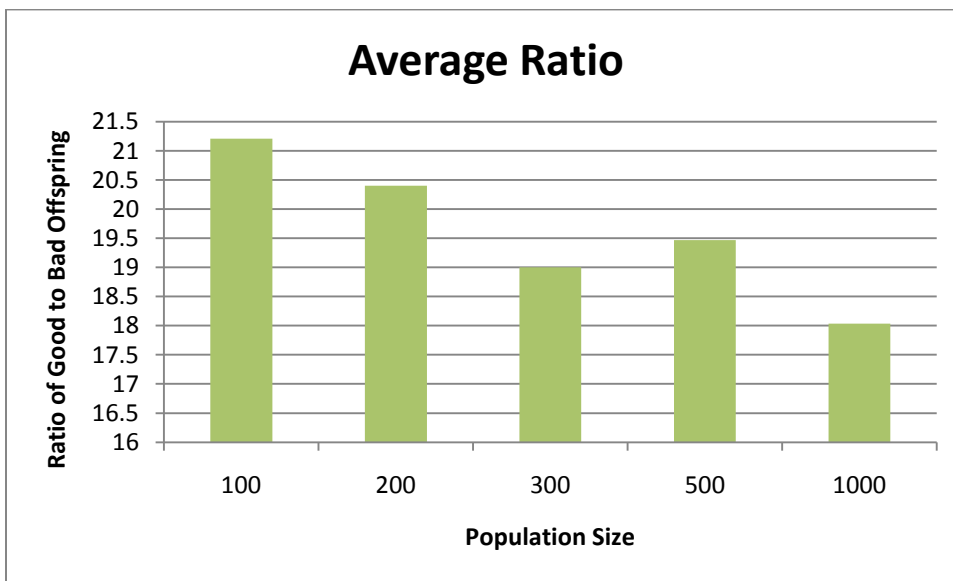


Figure 24: Average ratio for various population sizes

It appears that the larger the population size, the fewer generations are required for the optimal solution to be found (Figure 22). However, it also seems that a larger population size indicates a smaller ratio of acceptable to unacceptable children produced during crossover, as

seen in Figure 24. An anomaly appears with a population size of 500 that is, at this time, unaccounted for. Figure 23 shows that, as expected, it takes longer for the algorithm to run with a larger population than a small one.

### IX.V Crossover Techniques

In this section, we studied the performance of the two crossover techniques described in Section IV. We used the results from Sections IX.I, IX.II, IX.III and IX.IV to test.

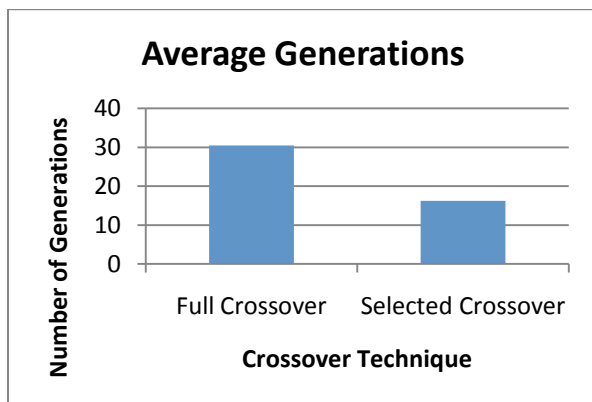


Figure 25: Average generations for crossover techniques

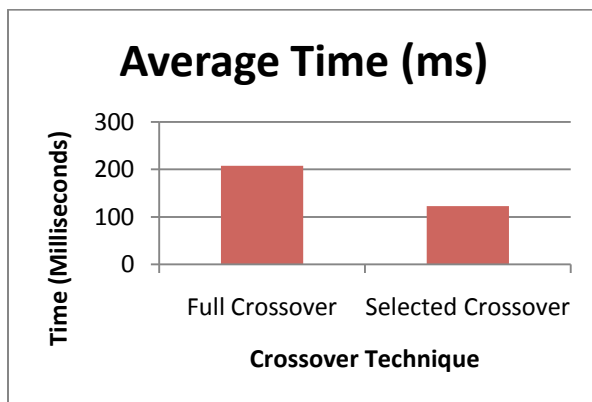


Figure 26: Average time for crossover techniques

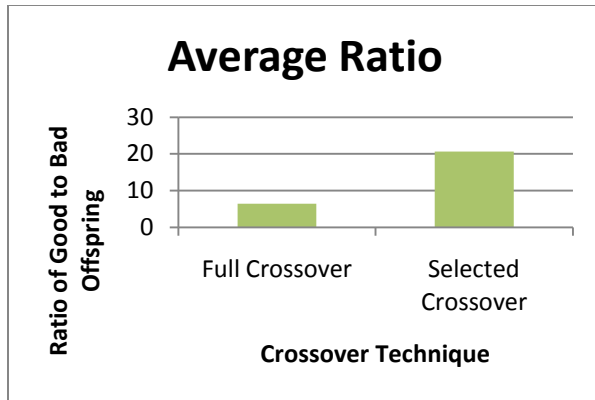


Figure 27: Average ratio for crossover techniques

The results for the crossover technique were conclusive. The Full Crossover technique, in which the note value and the duration were taken from the same parent, took far more generations (Figure 25), much longer time to reach the optimal solution (Figure 26), and the ratio was far lower than that of Selected Crossover (Figure 27). The ratio of acceptable to unacceptable children during crossover is the most important of the three measures, as it is a direct measure of how well the crossover technique creates offspring. The ratio for Selected Crossover is nearly four times that of Full Crossover.

## IX.VI Genetic vs. Memetic

The final test was to determine whether a memetic algorithm actually improves over its genetic predecessors. This final test was made using identical parameters for the memetic algorithm and for the memetic algorithm without the localized search feature, rendering it a

genetic algorithm. The parameters used were the results from Sections IX.I, IX.II, IX.III, and IX.V. We tested using population sizes of 100, 500 and 1000.

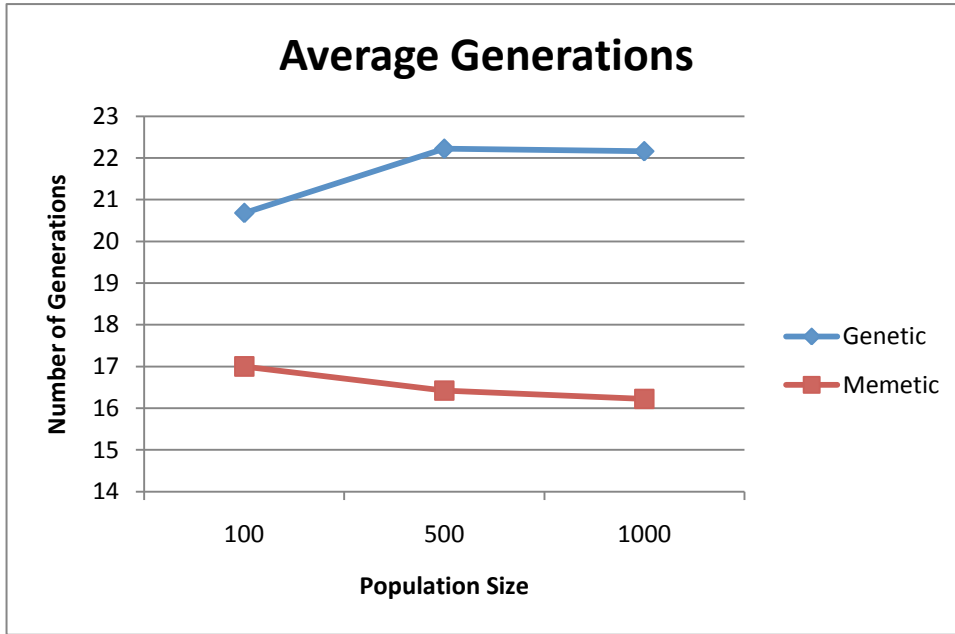


Figure 28: Average generations for genetic and memetic algorithms

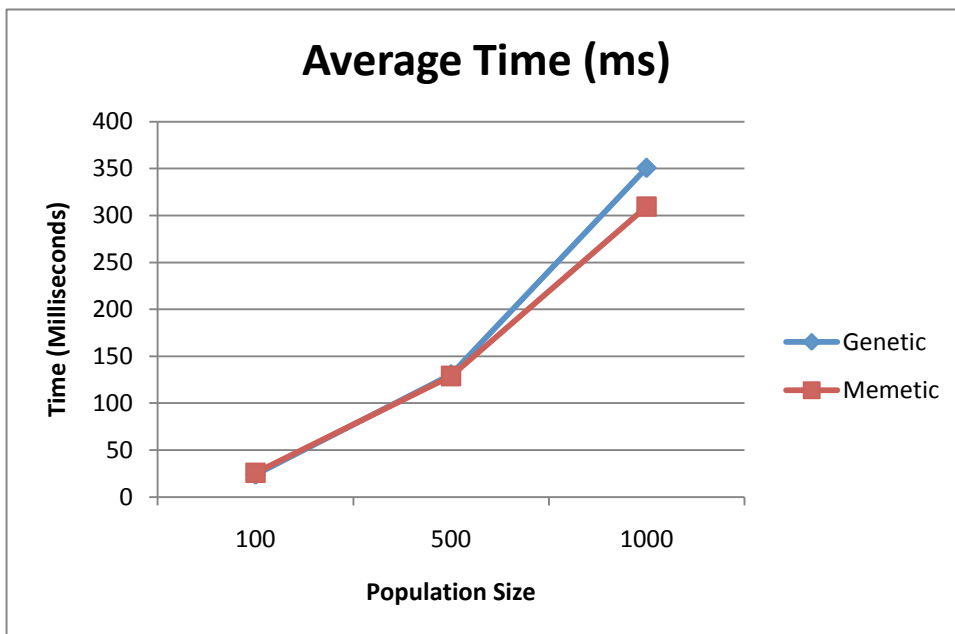


Figure 29: Average time for genetic and memetic algorithms

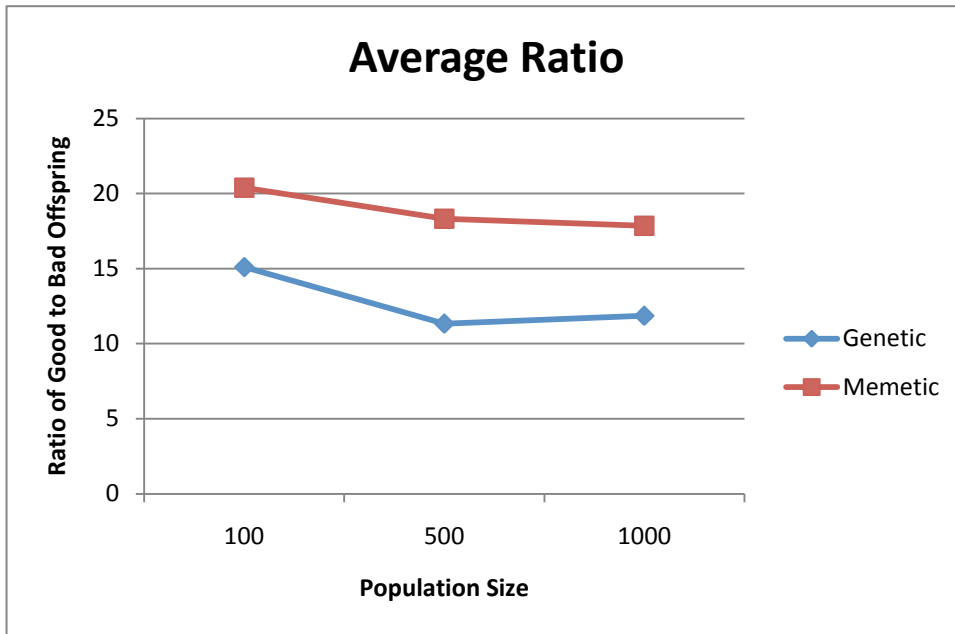


Figure 30: Average ratio for genetic and memetic algorithms

With each population size, the memetic algorithm outperformed its genetic counterpart in terms of average number of generations required to find an optimal solution (see Figure 28).

The difference in the average time required to complete the algorithm became more noticeable with increased population sizes, as seen in Figure 29. With the smallest population size (100), the genetic algorithm actually performed slightly better, on average. However, when the population began to grow larger, it became more and more apparent that the memetic algorithm was faster than the genetic.

Figure 30 shows the average ratio of acceptable to unacceptable children was much higher in all population sizes for the memetic algorithm than its genetic counterpart.

Overall, these test results show that a memetic algorithm should outperform its genetic predecessor nearly every time, with the exception of very small population sizes.

## IX.VII Subjective Test

Music, by its very nature, is subjective. Even after testing using the metrics described in the Section IX and finding the optimal parent selection method, fitness function weighting, parent percentage, and crossover technique, the ‘optimal’ solution may or may not be musically satisfying. This poses an opportunity to gather subjective testing results. In order to do this, 16 compositions (see Table 4) were generated using 2 examples of each combination of parent selection method and crossover technique. Once these compositions were created, a survey was open for people to listen to and rate the pieces of music. Each person responded by listing their three most favorite and three least favorite pieces of music. No person was aware of how the compositions were produced.

Table 4: Table of compositions used for subjective testing

<u>Name</u>	<u>Parent Selection Method</u>	<u>Crossover Technique</u>
Alpha Golf	Random	Full Crossover
Juliet Quebec	Elite	Full Crossover
Sierra Echo	Tournament	Full Crossover
Victor Delta	Roulette	Full Crossover
Foxtrot Kilo	Random	Selected Crossover
November Uniform	Elite	Selected Crossover
Zulu Lima	Tournament	Selected Crossover
Romeo Whiskey	Roulette	Selected Crossover

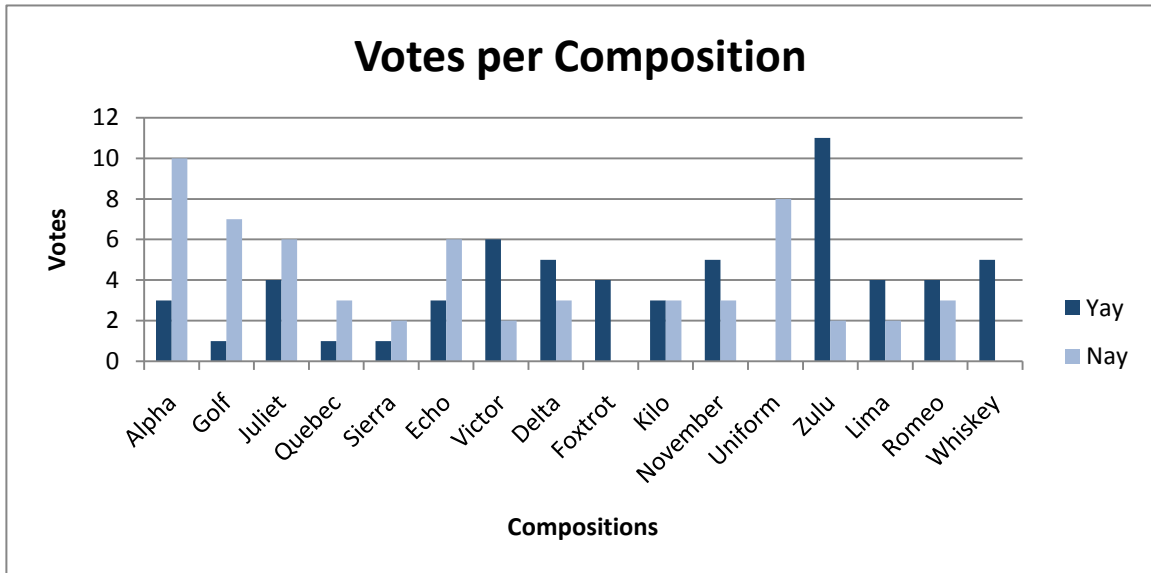


Figure 31: Votes yea and nay for each composition

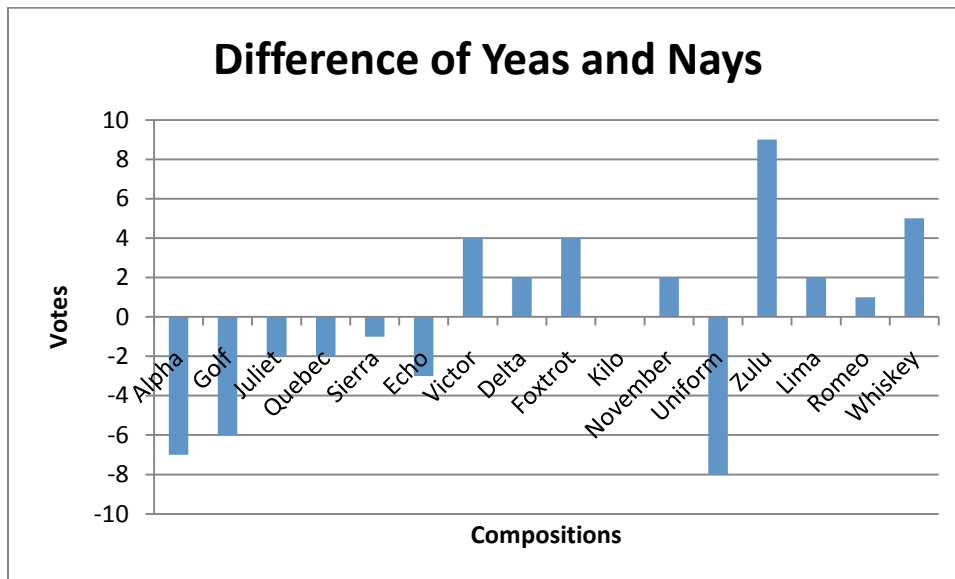


Figure 32: Difference of yeas and nays for each composition

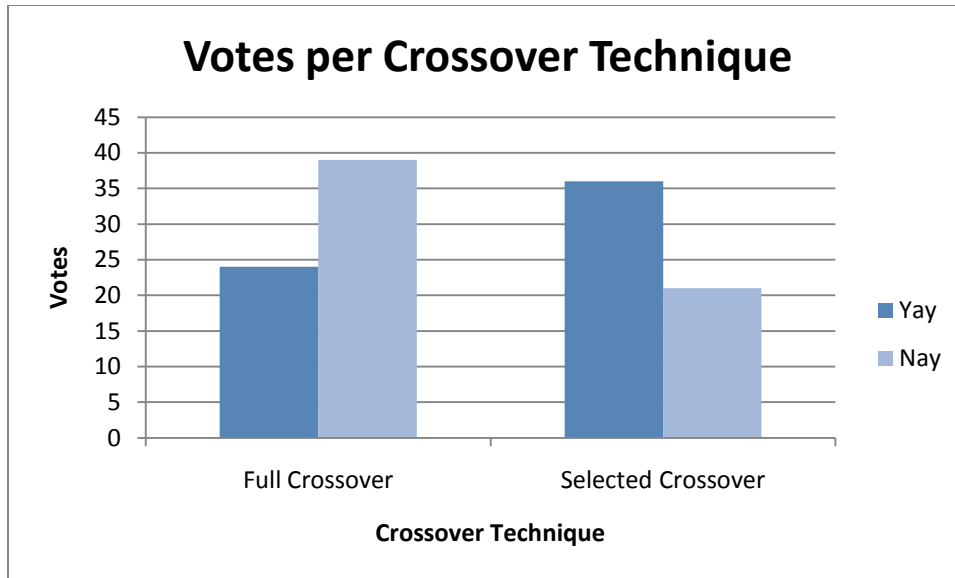


Figure 33: Votes yea and nay for each crossover technique

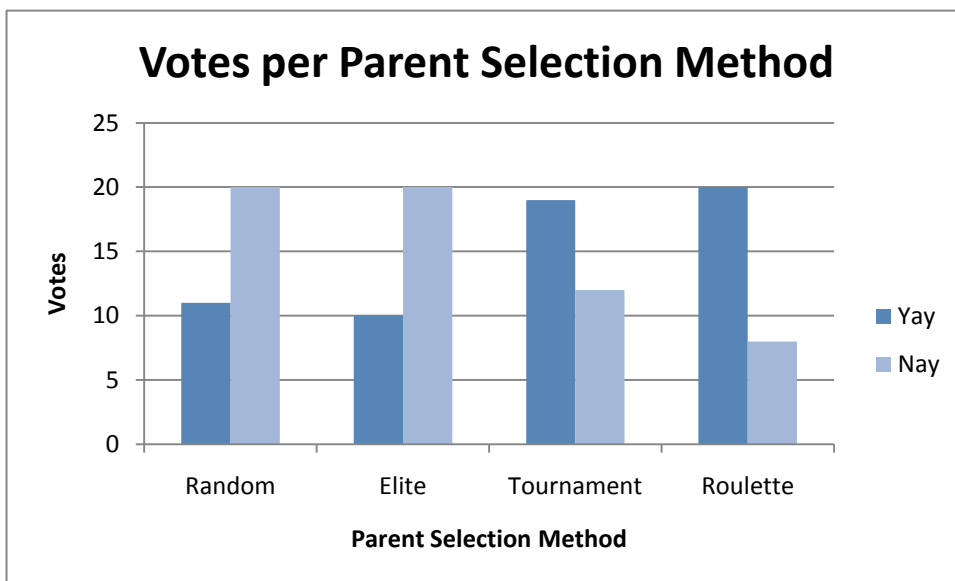


Figure 34: Votes yea and nay for each parent selection method

From Figure 31, we see that the composition with the most votes (11) was Zulu, created with the tournament selection method and Selected Crossover. In Sections IX.I and IX.V we showed that the tournament selection and Selected Crossover methods were the optimal, performing the

best according to our metrics. The composition from Figure 31 with the least votes was Alpha. The best performance based on the difference between the yea votes and the nay votes was, according to Figure 32, Zulu, with the worst performing being Uniform. Alpha was created with the random parent selection and Full Crossover, shown in Sections IX.I and IX.V to be the least optimal methods, respectively. Figure 33 confirms that Selected Crossover was the more liked than Full Crossover, which reflects the objective performance of the crossover techniques found in Section IX.V. Figure 34 shows that the most liked method was the roulette method, which was the second best parent selection method. The best parent selection method, the tournament method, received 1 fewer vote than roulette for most liked. The elite and random methods were not received well.

Chi-square tests were run on the results of the survey. We wanted to see if there was any statistical significance in the pairings of parent selection and crossover technique. The p-value of the chi-square test for yea votes was .0445, indicating that the results are statistically significant, as we believed they would be. The test for nay votes was even more significant, with a p-value of .0027. We also wanted to test the parent selection methods and the crossover techniques separately, to find whether one of them was more significant in the results than the other. A chi-square test for yea votes between the two crossover techniques gave a p-value of .1213, indicating that the results were not statistically significant. However, a chi-square test of the nay votes resulted in a p-value of .0201, which is statistically significant. A chi-square test for parent selection methods, yea votes, resulted in a p-value of .1406, not statistically significant. The chi-square for nay votes for parent selection methods resulted in a p-value of .0658, which is not quite statistically significant. Typically, .05 and smaller is the marker for statistically significant

results. The results of the chi-square tests can be found in Table 5. The significant results are highlighted in green, the insignificant in red, and the almost significant in black.

Table 5: Chi-square test results

Test	P-value
Selection method & crossover ; yea	.0445
Selection method & crossover ; nay	.0027
Selection methods ; yea	.1406
Selection methods ; nay	.0658
Crossover techniques ; yea	.1213
Crossover techniques ; nay	.0201

Overall, it appears that the subjective results from testing these compositions reflect similar results as the objective metrics we tested in Sections IX and revealed in Sections IX.I through IX.V. A chi-square test of the pairings of parent selection and crossover techniques proves that the results are statistically significant.

### IX.VIII Example

One can observe the results of using our algorithm by comparing Figures 35 and 36. Our parameters were that the resulting music must be in 4/4 time and be in the C-Major scale.



Figure 35: One measure of pre-algorithm music



Figure 36: One measure of post-algorithm music

In Figure 35, the pre-algorithm, the music does not conform to 4/4 time, and the presence of a sharp note indicates that it does not conform to the C-Major scale, either. After using the algorithm, Figure 36 shows that the resulting music does conform to both 4/4 time and falls in the C-Major scale.

## X. CONCLUSION

The research into the uses and designs of memetic algorithms is in its infancy. There are numerous real-world applications already being solved by the development of memetic computing. Algorithmic music composition is also a field that is growing. Many artificial intelligence techniques are already being employed in research to advance the field. This

proposal is an attempt to put forward a novel use for memetic algorithms for algorithmic music composition, a use that, at the time of this paper, has no published research. The belief is that this research, by improving upon the evolutionary and genetic algorithms already implemented in music composition, can help to find better solutions.

After testing the metrics described in Section IX, we have determined that the most effective version of our algorithm involved using a tournament style parent selection method, a fitness function in which the tonal and melodic components are weighted more than the harmonic component, using our Selected Crossover technique, with 20% of the population acting as parents. We also determined that a genetic version of our algorithm performs less optimally than our memetic version. Our subjective testing results showed that our objectively optimal parameters also provided aesthetically better results.

## XI. REFERENCES

[ARI05] Ariza, C. "Navigating the Landscape of Computer-Aided Algorithmic Composition Systems: A Definition, Seven Descriptors, and a Lexicon of Systems and Research." In Proceedings of the International Computer Music Conference. San Francisco: International Computer Music Association. (2005) 765-772.

[ASS99] Assayag, G., Rueda, C., Laurson, M., Agon, C., and Delerue, O. Computer-Assisted Composition at IRCAM: From PatchWork to OpenMusic. *Comput. Music J.* 23, 3 (Sep. 1999), 59-72.

[BON10] Bontoux, B., Artigues, C., and Feillet, D. 2010. A Memetic Algorithm with a large neighborhood crossover operator for the Generalized Traveling Salesman Problem. *Comput. Oper. Res.* 37, 11 (Nov. 2010), 1844-1852.

[CHI06] Chiu, Shih-Chuan, Shan, Man-Kwan, Computer Music Composition Based on Discovered Music Patterns. Proc. of 2006 IEEE International Conference on Systems, Man, and Cybernetics, Taipei (2006)

[DAW76] Dawkins, R. *The Selfish Gene*, Oxford University Press: 1976

[DIG04] J.G. Digalakis and K.G. Margaritis, Performance Comparison of Memetic Algorithms, *Journal of Applied Mathematics and Computation*, Elsevier Science, 158 (25): 237-252, 2004.

[DIO06] Dion, John A. Automated Music Composition: An Expert Systems Approach. Rivier College Online Academic Journal, Vol. 2, 1 (Spring 2006)

[FER10] Ferentinos, K. P. and Tsiligiridis, T. A. 2010. A memetic algorithm for optimal dynamic design of wireless sensor networks. *Comput. Commun.* 33, 2 (Feb. 2010), 250-258.

[KON07] Konstantinidis, A., Yang, K., Chen, H., and Zhang, Q. 2007. Energy-aware topology control for wireless sensor networks using memetic algorithms. *Comput. Commun.* 30, 14-15 (Oct. 2007), 2753-2764.

[LIU08] Lui, Y.-H. A Memetic Algorithm for the Probabilistic Traveling Salesman Problem. In: *Procs. 2008 IEEE Congress on Evolutionary Computation (CEC 2008)*, Hong Kong.(2008).

[LU10] Zhipeng Lu, Jin-Kao Hao, A memetic algorithm for graph coloring, *European Journal of Operational Research*, Volume 203, Issue 1, 16 May 2010, pgs. 241-250

[MAR00] M. Marques, V. Oliveira, S. Vieira, and A. C. Rosa, "Music composition using genetic evolutionary algorithms," in: *Proc. of 2000 Congress on Evolutionary Computation (CEC-2000)*, vol. 1, 2000, pp. 714–719.

[NAK09] Nakamura, Chisa, Onisawa, Takehisa. Music/Lyrics Composition System Considering User's Image and Music Genre. Proc. of 2009 IEEE International Conference on Systems, Man, and Cybernetics, San Antonio (2009)

[NGU09] Nguyen, Q. H., Ong, Y., and Lim, M. H. 2009. A probabilistic memetic framework. Trans. Evol. Comp 13, 3 (Jun. 2009), 604-623.

[NGU07] Q. H. Nguyen, Y. S. Ong, and N. Krasnogor, "A Study on the Design Issues of Memetic Algorithm", *IEEE Congress on Evolutionary Computation*, 2007.

[POP95] Pope, S. T. Fifteen years of computer-assisted composition. Computer Music Journal, 1995.

[REA01] Reader, S.M. The nature of the memetic beast. *Darwinizing Culture: The Status of Memetics as a Science* (book review), Trends in Cognitive Sciences, Vol. 5, p.365-366, Elsevier: 2001

[TIN10] Ting, C. and Liao, C. 2010. A memetic algorithm for extending wireless sensor network lifetime. Inf. Sci. 180, 24 (Dec. 2010), 4818-4833.

[SAM08] Samanlioglu, F., Ferrell, W. G., and Kurz, M. E. 2008. A memetic random-key genetic algorithm for a symmetric multi-objective traveling salesman problem. Comput. Ind. Eng. 55, 2 (Sep. 2008), 439-449.

[SHE08] Sheikholharam, P. and Teshnehlab, M. 2008. Music Composition Using Combination of Genetic Algorithms and Kohonen Grammar. In *Proceedings of the 2008 international Symposium on Computational intelligence and Design - Volume 01*(October 17 - 18, 2008). ISCID. IEEE Computer Society, Washington, DC, 255-260.

[UNE01] Unehara M, Onisawa T, Composition of Music Using Human Evaluation. Proc. of 2001 IEEE International Conference on Fuzzy Systems, Melbourne (2001)