

Genetic Algorithms

and their Application to Continuum Generation

Tracy Moore, Douglass Schumacher
The Ohio State University, REU, 2001

Abstract

A genetic algorithm was written to aid in shaping pulses used in continuum generation. Mating techniques, fitness evaluation methods, mutation rate, population size, and generation size were studied and tested. Of the parameters investigated, the best combination of parameters is an initial combination of large population, high mutation rate and “Fitness Is Evaluation” fitness evaluation method with a final parameter combination of low mutation rate and “Linear Normalization” fitness evaluation method.

Continuum Generation

Continuum generation is the generation of a broad spectrum ($\Delta\lambda$ exceeding 500 nm) from a small bandwidth ($\Delta\lambda \sim 10$ nm) laser pulse. This phenomenon occurs when extremely intense laser light is passed through a medium because a medium’s index of refraction depends on the intensity of the light as well as the light’s wavelength according to:

$$n = n_0(\lambda) + n_2 * I(t),$$

Here n is the index of refraction, n_0 is the intensity independent part of the index, n_2 is a parameter characterizing the nonlinear response of the medium, and $I(t)$ is the time-dependent intensity. For optically transparent solid media, n_2 is typically of order 10^{-15} cm^2/W . This equation is actually an approximation of a more complicated equation that can only be solved numerically¹. In general, the dependence between the input laser light

intensity and output wavelength cannot be easily predicted. A genetic algorithm may be a useful means to design a laser pulse shape that will generate a desired spectrum by continuum generation.

Genetic Algorithms

A genetic algorithm is a method of problem solving based on Darwin's theory of natural selection. Some terms needed to discuss genetic algorithms include; an individual, or a possible solution to the problem; a chromosome, or the parameters that define a solution; a gene, or a parameter; fitness, or the how similar a solution is to the desired solution. A genetic algorithm operates by:

- 1) randomly generating an initial population of solutions, for example numbers, that can be represented in bit strings,
- 2) evaluating each individual's fitness via an appropriate fitness function,
- 3) creating new individuals by mating the current ones by mixing their bit strings together,
- 4) deleting some or all of the population to make room for new members,
- 5) repeating all but step (1) for the desired number of generations.

This algorithm is straightforward but involves many variable parameters including: fitness evaluation method, how the mating parents are chosen; mating techniques, how each parent's representative bit string is mixed with others to create new individuals; mutation rate, the probability any bit may change value; population size; and the number of generations.

Experiment

In this report, I describe a study of which combinations of parameters resulted in the best solutions for a given fitness evaluation function. Initially I planned on modifying John Greffentette's genetic algorithm, Genesis. However, it was concluded that writing my own genetic algorithm program would be more efficient. The majority of my summer was spent learning to program in C, studying genetic algorithms², and actually writing a genetic algorithm program.

The program I wrote allows many of the previously mentioned parameters to be varied according to a variety of methods². The first variable parameter investigated was population size. It was varied between sizes 20, 50, 100, 250, and 600.

The fitness evaluation method parameter was also varied. The fitness evaluation method options considered were "Fitness is Evaluation," "Windowing," and "Linear Normalization²." The following table describes the different methods.

Parent Selection Method:	Fitness Is Evaluation	Windowing	Linear Normalization
Description:	- Most basic fitness evaluation method.	- Fitness evaluation method intended to increase competition between similar individuals.	- Fitness evaluation method intended to increase competition between similar individuals.
How It Selects:	<ul style="list-style-type: none"> - Each individual's fitness is calculated with the fitness function. - Every individual's fitness is summed. - A random number is chosen between zero and this sum. - If this number is lower than the first individual's fitness then that individual is chosen to reproduce, if not the next individual's fitness is added to the first individual's fitness and if the random number is lower than this sum then the second number is chosen, if not the process continues in the same manner (this process is called roulette wheel selection). The process is repeated for as many parents as desired. 	<ul style="list-style-type: none"> - Each individual's fitness is calculated with the fitness function. - The fitness of the least fit individual is subtracted from every individual's fitness. - Parents are selected by roulette wheel selection described in Fitness Is Evaluation section. 	<ul style="list-style-type: none"> - Each individual's fitness is calculated with the fitness function. - The individuals are ordered from least to most fit. - Each individual's fitness becomes its order number. - Parents are selected by roulette wheel selection described in Fitness Is Evaluation section.

Table 1: Basic Parent Selection Techniques

My program allows the user to choose between these basic parent selection technique options and three variations on them. Windowing with a minimum value is a variation on windowing that reassigns the fitness of all individuals less fit than a certain minimum value to that minimum value². Linear normalization with a factor is a variation of linear normalization that adds a constant value to the fitness of every individual². I also

introduced a new variation called nonlinear scaling. This is a variation of linear normalization that scales the fitnesses of the ordered individuals according to any equation the user chooses. The equation I considered for nonlinear scaling was:

$$scaling = e^{\sqrt{fitness}},$$

which *reduces*, in a controlled fashion, the ability of weaker individuals to compete.

The fitness function used in all trials for this work was:

$$fitness = 1/(x-1),$$

where x is the individual's numerical value. Due to time constraints, I was only able to evaluate the performance of the three basic fitness evaluation methods this summer, although data was collected for all cases.

The program also allows the user to switch fitness evaluation methods during a run. The user may dictate if the switch(es) occur after a certain number of generations; after a certain percentage of the generations has been completed; or after the average fitness, RMS spread, the ratio between RMS spread and average fitness, best fit individual, or worst fit individual has reached a certain value.

The program also allows the user to choose one of the three mating techniques : one point, two point, or uniform crossover.²

Crossover type:	One Point	Two Point	Uniform
Explanation:	A bit is chosen randomly and the first child receives the first parent's bit string up to that point and the second parent's bit string after it. The second child receives the bit strings not yet given to the first child.	Two bits are chosen randomly and the first child receives the first parent's string up to the first bit and after the second bit and the second parent's string between the two bits. The second child receives the remaining strings from each parent.	Each bit position is given a random binary value. The first child receives the first parent's bits if the randomly generated value corresponding to that bit's position is one. It receives the bits in the zero value positions from the second parent. The second child receives the bits from each parent not given to the first.
Examples:	Parent1: 100100101 Parent2: 011000110 Randomly chosen crossover bit: 5 Child1: 10010 0110 Child2: 01100 0101	Parent1: 100100101 Parent2: 011000110 Randomly chosen crossover bits: 2, 8 Child1: 10 100011 1 Child2: 01 010010 0	Parent1: 100100101 Parent2: 011000110 Randomly generated Map: 011100001 Child1: 000100111 Child2: 111000100

Table 2: Mating techniques (Davis)

The data taken to investigate the performance of each mating method was not analyzed due to time constraints.

Mutation is also important to the mating process. The mutation rate is the probability that a given bit will have a *chance* to change value. For example, if the mutation rate is 1%, then every bit would be assigned a random number between 1 and 1000 and all the bits with a value between 0 and 10 would have a new (binary) value randomly generated. This value has a 50% chance of changing so if the mutation rate is 1% approximately 0.5% of the bits in the population change value. The mutation rates investigated by my program were: 0.2%, 0.5%, 1.0%, 2.5%, 5.0%, and 10.0%.

Lastly, the program was automated to vary fitness evaluation method, population size, and mutation rate so it performed every combination possible for 15 different trials. The data collected in output files during each run included the generation number, the best fit individual's fitness, the worst fit individual's fitness, the average fitness, the fitness RMS spread, and the ratio between the fitness RMS spread and the average fitness. The data was analyzed with GENPLOT, a plotting and analysis program. A portion of the data has been graphed and analyzed with this program.

Results

The goal of this research project was to determine which parameter combinations yielded the "best" results. However, there are multiple possible definitions of "best" results, including: populations that converge quickly, populations whose average fitnesses approach the optimal fitness, and populations who converge to the same final value regardless of initial value (yielded by "robust" algorithms). All three definitions of best were considered when analyzing the data collected.

Due to time constraints, not all the data collected was analyzed. The mating technique used for all trials discussed in this report was one-point crossover, and 600 generations were evaluated. Generally, the population converged after approximately 100 generations, and so results are not presented beyond this point. Also, data and results found are dependent on the linear fitness function used and other fitness functions should be considered. The results described here consider the variation of only three parameters: fitness evaluation method, mutation rate, and population size. However, the program

offers many other options. Both mutation rate and population size are discussed here as they vary with fitness evaluation method.

Mutation rate seemed to be one of the most influential variables investigated. The far left column shown in Fig. 1 gives the results when the “Fitness is Evaluation” fitness evaluation method was used for various mutation rates. It is clear that a low mutation rate is necessary for convergence near the optimal fitness value, 1.0. Runs taken with a mutation rate 0.2%, or 1.0% converged very near this optimal fitness. However, a 10.0%, or 5.0% mutation converged to a lower fitness in fewer generations.

Mutation rate effects populations differently depending on fitness evaluation method. “Fitness Is Evaluation” and “Windowing”, columns 1 and 2 in Fig 1, both converge more quickly at higher mutation rates. “Linear Normalization”, column 3 of Fig. 1, performs better at lower mutations; it converges quicker and to a more optimal value. However, “Fitness Is Evaluation” and “Windowing” both converge more quickly than “Linear Normalization” regardless of mutation rate. The general trend indicated in fig 1 is that high mutation rates will approach an optimal value quickly and lower rate will more nearly approach the optimal value. Therefore, the mutation rate should change from an initial high mutation rate for quick convergence to a final low mutation rate for optimal convergence value, at some point during the run, when the fitness reaches a certain value. The program currently does not offer this option.

Population size was also investigated. The “Fitness is Evaluation” fitness evaluation method, shown in column 1 of Fig. 2, indicates that higher population sizes converge more quickly. Figure 2 also demonstrates that the higher the population size, the more quickly the average fitness converges for all fitness evaluation methods. Fitness

evaluation method “Windowing”, shown in column 2, appears not to converge at low population sizes. This poor result is most likely due to a bug in the “Windowing” code. “Fitness Is Evaluation” and “Windowing”, columns 1 and 2, both converge more quickly than “Linear Normalization”, column 3. Therefore, fitness evaluation methods should be switched during the run. The initial fitness evaluation method should be “Fitness is Evaluation” for quick convergence and the final fitness evaluation method should be “Linear Normalization” for optimal convergence value.

Conclusions

Based on the data investigated, for practical applications, the initial parameters should include high mutation combined with “Fitness Is Evaluation” fitness method. After the average population fitness grows to a desired value the algorithm should switch to an extremely low mutation rate and the “Linear Normalization” parent selection method. The population should be as large as possible.

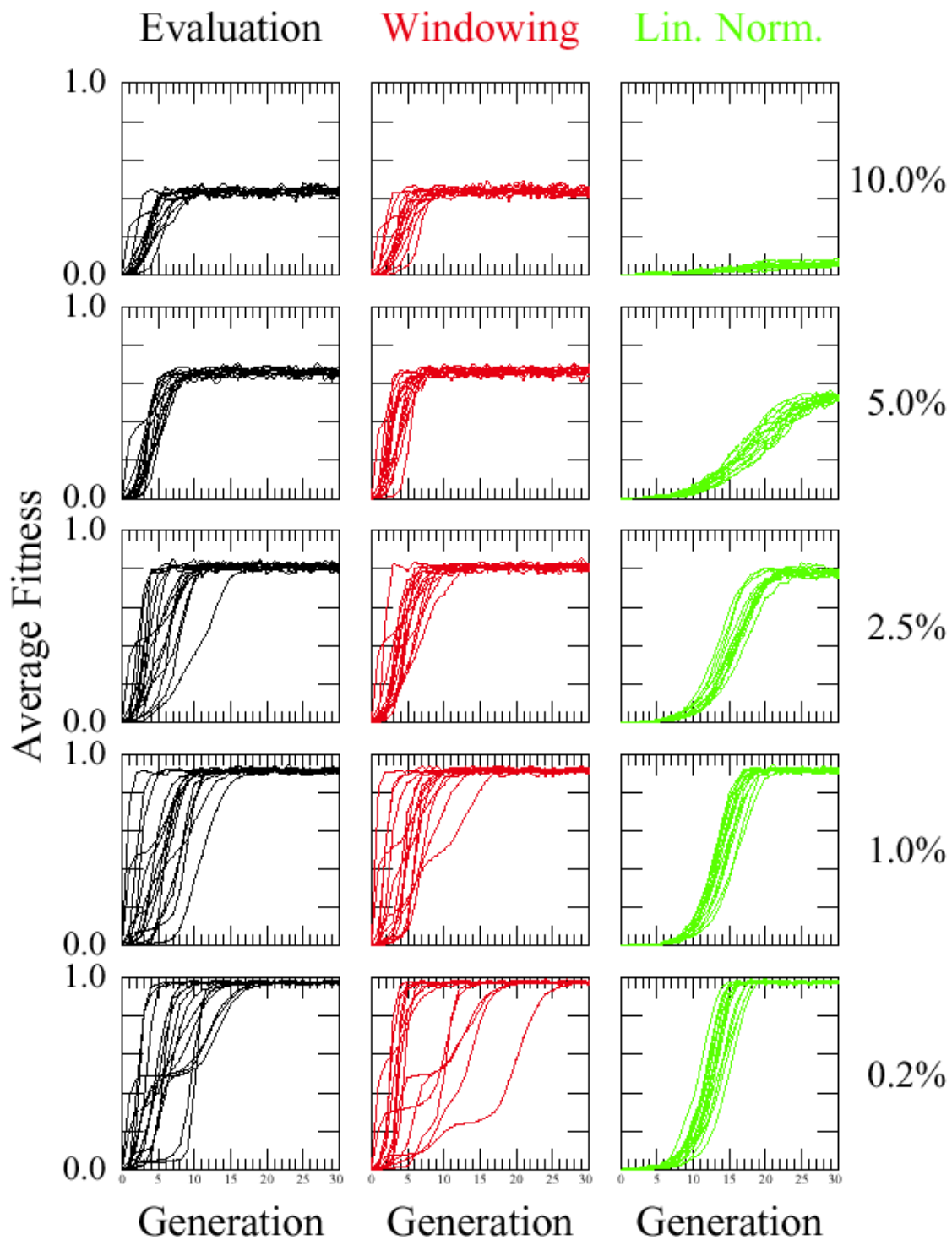


Fig. 1: Effect of varying mutation rate for different fitness evaluation methods. The fitness method is listed at the top of each column and the mutation rate is given to the right. In all cases, one point crossover was used with a population size of 600.

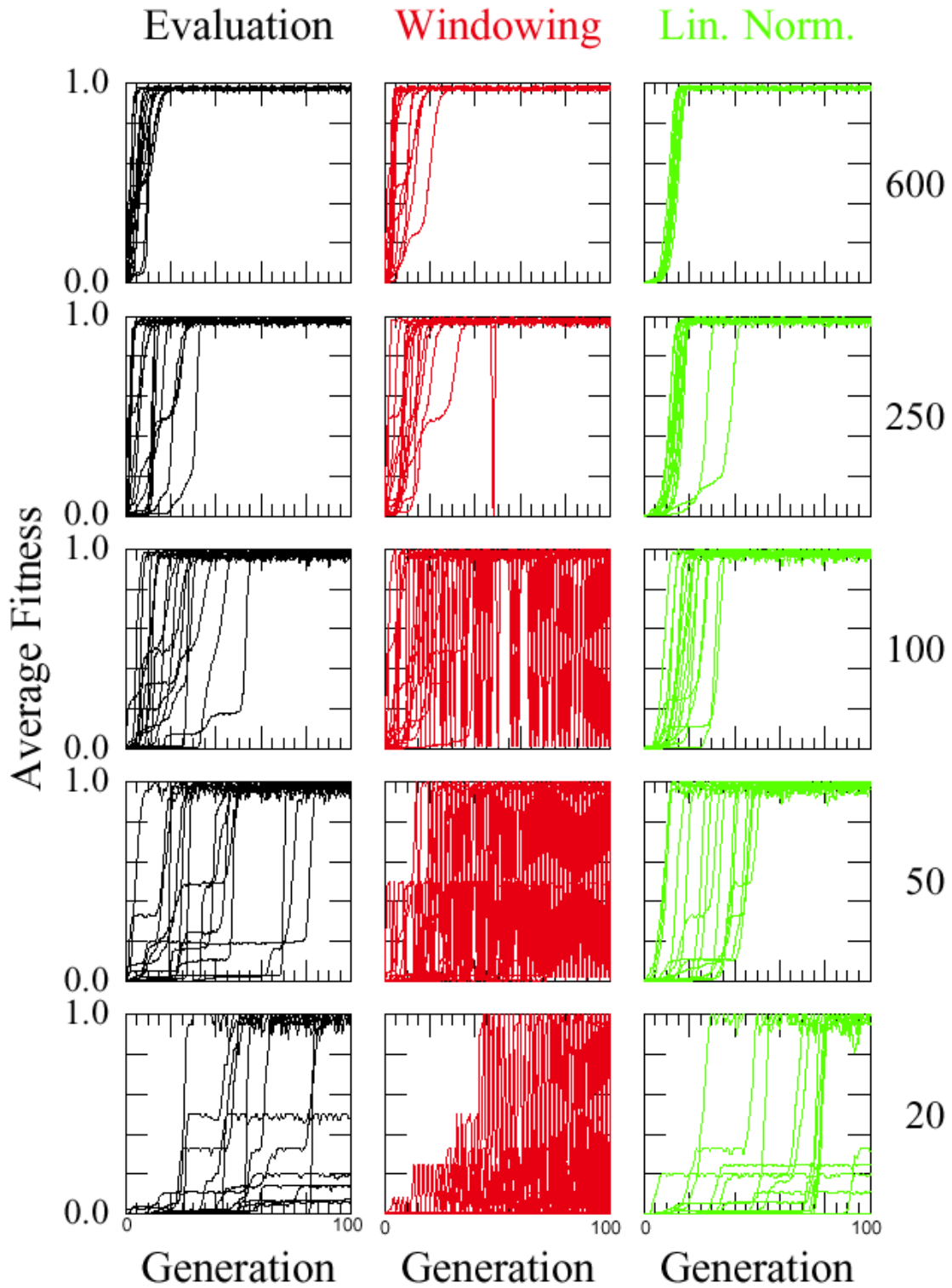


Fig. 2. Effect of varying population size for different fitness evaluation methods. The fitness method is listed at the top of each column and the population size is given to the right. In all cases, one point crossover and a 0.2% mutation rate was used.

Acknowledgements

Thanks to my advisor, Douglass Schumacher, for making this research opportunity possible and his continued advice and efforts and to Chris Schroeder for help with difficult programming tasks.

References

- 1) *Supercontinuum Laser Source*, R. R. Alfano, ed. (Springer-Verlag, New York, 1989).
- 2) Lawrence Davis, *A Handbook of Genetic Algorithms* (International Thompson Computer Press, Boston, 1996).