On the Hybridization of Genetic, Population Based Incremental Learning, and Particle Swarm Optimization Algorithms

Abstract
This work investigates the tri-hybridization of techniques taken from Genetic Algorithms, Population Based Incremental Learning, and Particle Swarm Optimization. In particular, this work analyzes various combinations of such techniques with respect to their performance in optimizing multi-dimensional functions. The hybrid algorithm is found to improve performance functions with significant local minima and lessen performance on flatter functions.

I Introduction

Genetic Algorithms generate a population of individuals. After an evaluation of the fitness of the individuals, the most fit individuals are selected and undergo a process of mutual recombination and individual mutation to produce theoretically more fit individuals. The population is updated with these new individuals, and the cycle repeats as desired.

Population-Based Incremental Learning (PBIL) is an evolutionary algorithm that maintains a probability vector instead of a population. The probability vector generates individuals, whose relative fitness determine the subsequent modification of the probability vector. Theoretically, as the probability vector is modified toward better individuals and away from worse individuals, the overall quality of the generated individuals increases.

Particle Swarm Optimization (PSO) is an algorithmic method that imitates the process of a large collection of individuals searching for the best location in an area, through constant recalculation of direction. Each individual communicates with a neighborhood of other individuals, and this constant communication influences the direction in which each individual decides to investigate. Individuals also explore the search space based on the direction of the best solution.

A central theme in the design of nature-inspired algorithms is the delicate interplay between exploration and exploitation: the continuous balance between embracing randomness and pursuing greediness. The techniques implemented in Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) demonstrate this quality. This work investigates whether the exploration and exploitation techniques of GA can enhance the performance of PSO algorithms.

With regard to exploitation, GAs explicitly prefer the best fit individuals to produce the next generation; similarly, individuals in PSO update their direction of exploration towards their neighborhood's best solution. As for exploration, GAs generate individuals through mutation and recombination operators applied to the best fit individuals, while individuals in PSO consider the direction of their personal best solution as well.

The optimization algorithm was used to find the global minimum of a given function. Thus, in implementing the hybrid GA-PSO, the standard of comparison was the minimum value returned by each particular GA-PSO configuration compared to the standard PSO configuration under identical parameters. Configurations varied across choice of neighborhood and swarm size. Within GA-PSO specifically, our work further investigates the choice of crossover method, as well as mutation and crossover probabilities.

We found that given standardized parameters, the hybrid outperforms PSO in terms of final solution quality on functions with significant local minima and performs worse otherwise. It also takes longer to converge and will always have a longer runtime. The GA in the hybrid always performed better without selection, and the PBIL was a poor choice in algorithm in trying to determine the best cut ratio for the hybrid.

Section II describes the functions optimized, Section III describes Genetic Algorithms, Section IV describes PBIL, Section V describes PSO, Section VI describes topological neighborhood choices for PSO, Section VII describes the method of hybridization, Section VIII describes our experimental design, Section IX describes results, Section X describes further work, and Section XI describes conclusions.

II Functions[1]

The goal of this implementation of the algorithm is to calculate the global minima of various mathematical functions. Thus, our algorithm is essentially tasked with solving an optimization problem, and differences between algorithm conditions are judged based on their relative performance in optimization. These functions can have any number of dimensions greater than or equal to two, which allows for investigations on the performance of the PSO algorithm with respect to the number of variables and the problem's dimension size.

The initial particles are initialized to random values in the function. However, the range of random values does not include the minima; thus, particles are not instantiated with positions biased towards the center of the search space. Each function also requires the initial velocity of the particle to be initialized with values in an acceptable range; such an acceptable range allows particles to immediately begin exploring the solution space to find more optimal results. The five functions tested in this paper are listed below.

2.1 Rosenbrock

The equation for the Rosenbrock function:

[1] Bratton, D., & Kennedy, J. (2007). Defining a Standard for Particle Swarm Optimization. 2007 IEEE Swarm Intelligence Symposium.

$$f = \sum_{i=1}^{D-1} \{100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\}$$

where D is the dimension.

Feasible bounds for this function are: $(-30,\ 30)^D$. Initialization ranges used for the position vector and velocity vector were [15.0, 30.0] and [-2.0, 2.0], respectively.

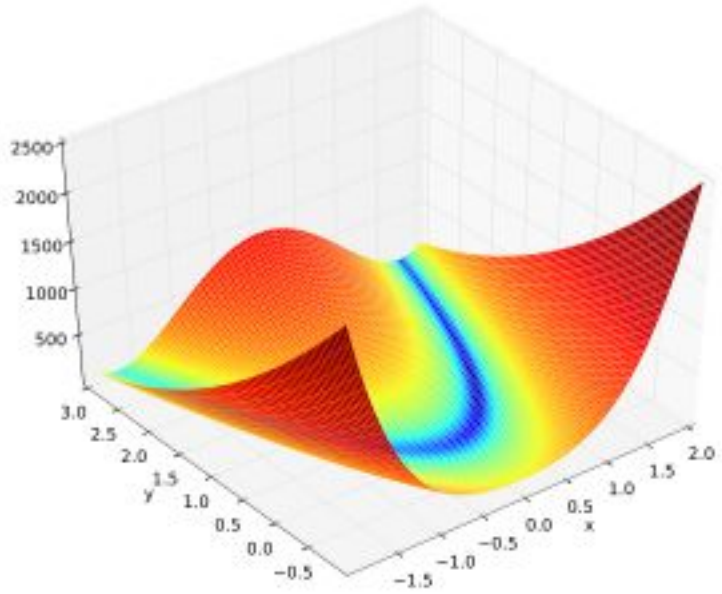


Figure 1: Rosenbrock function with two variables (in three dimensions).

2.2 Ackley

The equation for the Ackley function:

$$f = -20exp\{-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\} - exp\{\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i)\} + 20 + e$$

where D is the dimension.

Feasible bounds this function are: $(-32,\ 32)^D$. Initialization ranges used for the position vector and velocity vector were [16.0, 32.0] and [-2.0, 4.0], respectively.

The difficulty in optimizing the Ackley function is that, as evidenced by Figure 2, the function offers numerous local minimums. Compared to the global minimum, the local minimum are clearly inferior.

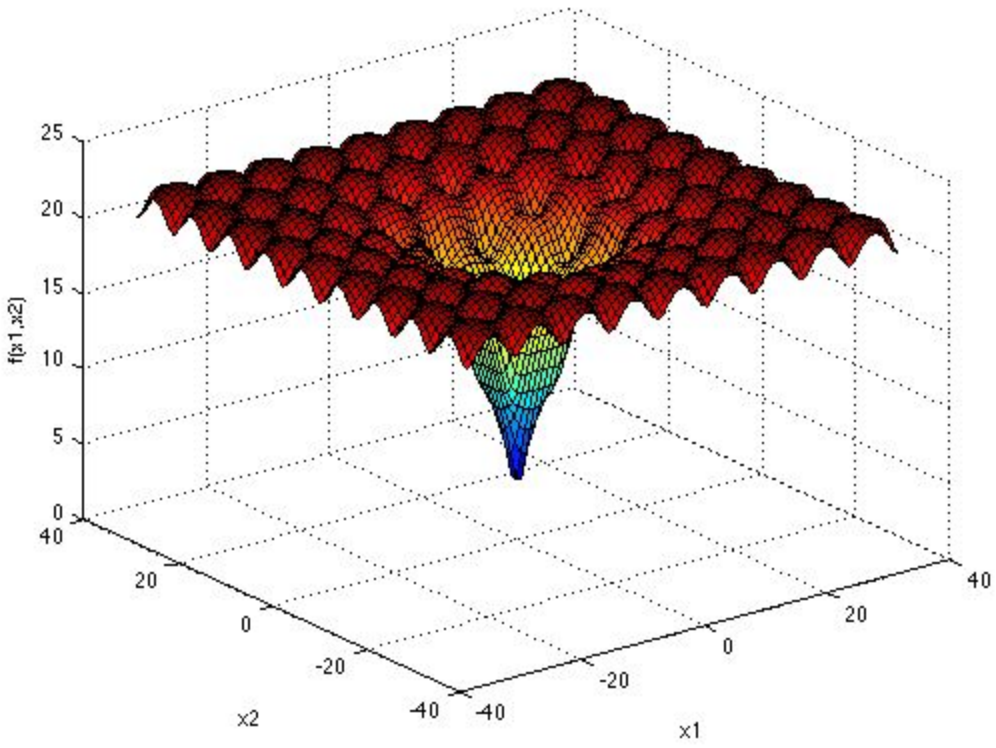


Figure 2: Ackley function with two variables (in three dimensions).

2.3 Rastrigin

The equation for the Rastrigin function:

$$f = \sum_{i=1}^{D} \{x_i^2 - 10cos(2\pi x_i) + 10\}$$

where D is the dimension.

Feasible bounds for this function are: $(-5.12,\ 5.12)^D$. Initialization ranges used for the position vector and velocity vector were [2.56, 5.12] and [-2.0, 4.0], respectively.

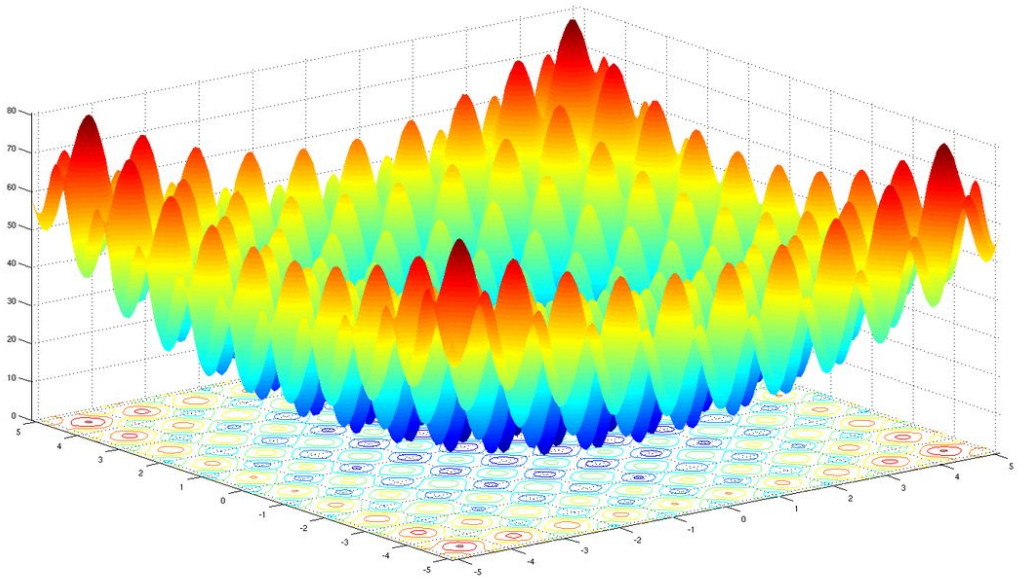


Figure 3: Rastrigin function with two variable (in three dimensions).

2.4 The Zakharov function has no local minima except the global one. Figure 4 represents it in its 2-dimensional form, and this is the equation for its d-dimensional form.

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^4$$

Global Minimum:

$$f(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (0, \ldots, 0)$$

The function is usually evaluated with $x_i \in [-5, 10]$, for all $i = 1, \ldots, d$.

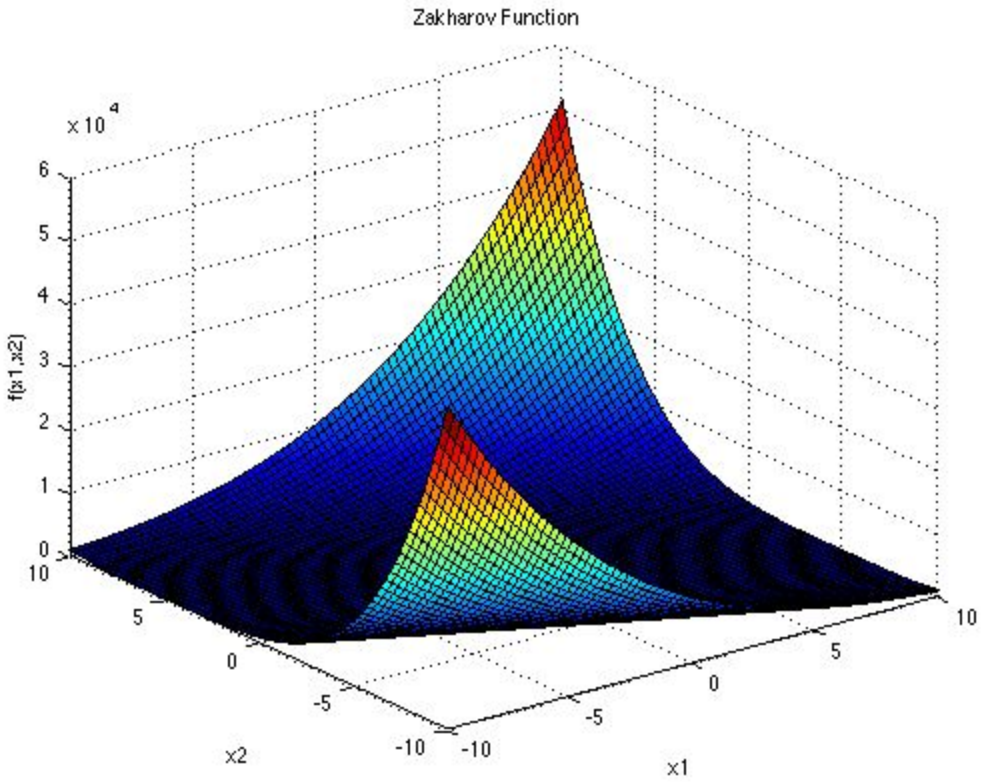


Figure 4: Zakharov function in three dimensions.[2]

2.5 The Griewank function is shown in its 2-dimensional form in figure 5. This is it's d-dimensional equation.

$$f(\mathbf{x}) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Global Minimum:

$$f(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (0, \ldots, 0)$$

The function is usually evaluated for $x_i \in [-600, 600]$, for all i = 1, …, d.

The Griewank function has many, regularly distributed, local minima. The complexity is shown in the zoomed-in plots.

[2] https://www.sfu.ca/~ssurjano/zakharov.html

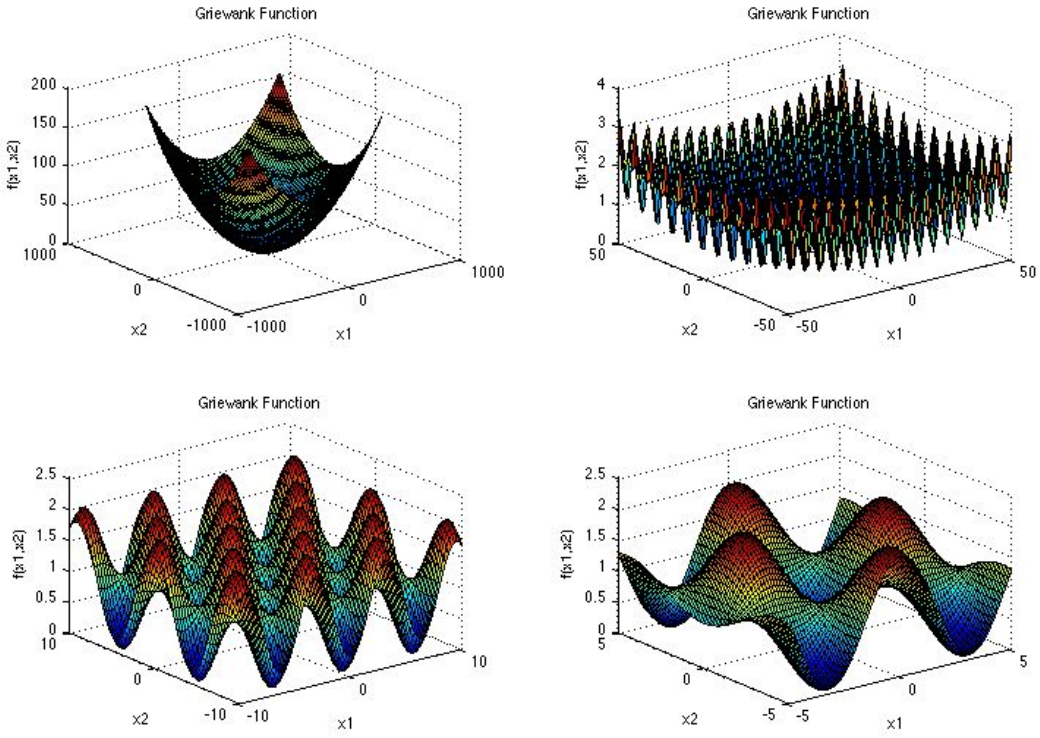


Figure 5: Griewank function as inspected at various intervals.[3]

III Genetic Algorithms

A Genetic Algorithm first generates a random population of candidate solutions, each of which is a string of symbols. After the application of the fitness function, individuals are selected for the breeding pool and crossover—through the recombination of pieces of strings—is applied to produce children. Finally, mutation is achieved by changing each symbol in each candidate solution's string with some small probability.

Genetic Algorithms typically include selection techniques such as tournament, rank, and Boltzmann selection to generate the breeding pool. However, in our work, we omitted selection operators; through initial experiments, we quickly found that the above selection operators, which select individuals with replacement, performed noticeably worse compared to a simple *q*-selection method, which simply takes the best *q* number of solutions, without replacement.

Regardless, after the application of the selection operator, the recombination phase modifies selected candidate solutions through crossover. Candidates in the breeding pool are paired, and

[3] https://www.sfu.ca/~ssurjano/griewank.html

for each pair, crossover is applied with some probability p. In our work, the two crossover methods implemented were 1-point crossover and uniform crossover.

In 1-point crossover, a crossover point for a pair of candidate solutions is randomly selected between the inclusive range of integers $[0,\ N-1]$, where N is the size of a candidate solution string. The crossover point corresponds to the index past which the candidate solutions swap substrings. Following the exchange of substrings, a child is randomly chosen among the two new candidate solutions.

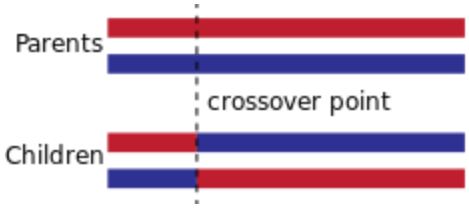


Figure 1. 1-point crossover (http://www.cs.vu.nl/~gusz/ecbook/ecbook-illustrations.html).

In uniform crossover, each symbol in the child's solution string is randomly selected between the corresponding pairs of parent symbols. In other words, the first symbol is chosen from the first pair, the second symbol from the second pair, etc.

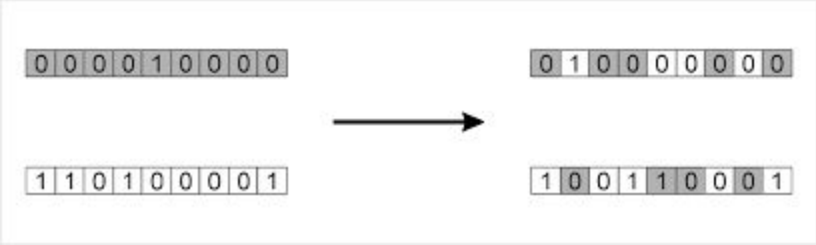


Figure 2. Uniform crossover (https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)).

Following the recombination phase, each child is subject to mutation. Each symbol in each child's solution string sic changed with some small probability. Following the mutation phase, the population of candidate solutions is now updated with these new, possibly mutated children.

Through this cycle of fitness evaluation, selection, recombination, and mutation, a Genetic Algorithm over time theoretically improves the quality of candidate solutions within its population.

3.1 GA on Real-values: Design choices

The question of mutation with regard to bitstrings is straightforward, as bits can only exhibit one of two values: a zero or a one. However, note that in our work, solutions are vectors of real numbers: specifically an array of *n* real numbers that are fed to an *n*-dimensional function.

Thus, each symbol in the solution String has an infinite selection of possible values for mutation. Clearly, a good mutation method does not uniformly select among the entire reals. Ideally, the effect of mutation should be greater when individuals are more concerned about exploration, and the effect should be lower when individuals are more inclined to exploit best solutions.

Michalewicz' Non-Uniform Mutation algorithm embodies these two characteristics; in our work, the mutation for GA is implemented using this algorithm, which proceeds as follows.[4]

Given a solution string *i* of length *N* at iteration *t*, i.e. $i^t = i_1{}^t,\ i_2{}^t, i_3{}^t, ..., i_N{}^t$, the individual i^{t+1} (i.e. the solution string *i* at iteration $t + 1$) is generated probabilistically as follows.

First, consider the following notation $\Delta(t,\ x)$, which is defined as $\Delta(t,\ x) = x(1\ -\ u^{(1\ -\ \frac{t}{T})})^b$. Note that *u* is drawn from the uniform distribution of the interval [0, 1]; *T* is the maximum number of iterations that the algorithm runs, *t* is the current iteration, and *b* is a constant. Note that *b* directly controls the desired impact of the mutation operator; higher values result in less mutation, as the quantity for which *b* is the exponent is a decimal less than 1.

Note that the inclusion of $\Delta(t,\ x)$ in the mutation operator, notably the parameters *T* and *t*, achieves the flexibility of stronger exploration early in the algorithm, followed by stronger exploitation in later stages.[5]

Specifically, $\Delta(t,\ x)$ can achieve maximal values of *x* in early iterations, but these maximal values decrease logarithmically as iterations increase. However, the limitation of the operator is that *T* must be specified at runtime; in other words, generalizations about *T* must be made beforehand if other runtime termination conditions are desired, such as the condition where the algorithm terminates if solutions do not improve over a certain amount of iterations.

Now, considering the actual mutation operator itself: for each j^{th} symbol $i_j{}^t$ of *i*, the first decision to be made is whether to mutate that symbol positively, by adding a positive number, or mutate it negatively, by subtracting a positive number. Note that the probability of either event occurring is equal. If positive mutation is chosen, $i_j{}^{t+1} = i_j{}^t + \Delta(t,\ i^u - i_j{}^t)$, where i^u is an upper bound on the value that a solution symbol can take. If negative mutation is chosen,

[4] Janikow, Cezary Z. and Michalewicz, Zbigniew. "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms.." ICGA, 1991. (page 4)

[5] Deep, K., Thakur, M. "A new crossover operator for real coded genetic algorithms." Applied Mathematics and Computation 188 (2007) 900.

$i_j^{t+1} = i_j^t + \Delta(t, i_j^t - i^l)$, where i^l is a lower bound on the value that a solution symbol can take.

IV PBIL

Population-Based Incremental Learning (PBIL) is an evolutionary algorithm that maintains a probability vector instead of a population. Each entry within the vector indicates the probability that a generated individual will have that particular trait. Thus, PBIL is generally applied to problems where candidate solutions can be represented as bit strings; the probabilities in the vector influence the binary value of the corresponding bits.

After individuals are generated, a fitness function applies fitness values to each individual. The probabilities within the probability vector are then modified for the next generation based upon the fitness of these individuals, in a process called competitive learning. Specifically, the vector is adjusted towards the traits of the most fit individual, with the following equation:

$$p_i = (p_i (1 - a)) + S_i a$$

where p_i is the i^{th} entry in the probability vector, a is the learning rate, and S_i is the bit value of the individual's i^{th} trait. Similarly, the vector is adjusted away from the traits of the least fit individual using the following equation:

$$p_i = (p_i (1 - a)) + \neg S_i a$$

where p_i and a are the same as above, but $\neg Si$ denotes the opposite bit value of the least fit individual's i^{th} trait. For example, if the i^{th} bit of the most fit individual is a 1, the i^{th} entry of the probability vector is appropriately modified to indicate a higher probability that later generations of individuals display an i^{th} bit of 1. A similar process occurs if the i^{th} bit of the least fit individual is a 0, although the magnitude by which the vector is updated can differ based on the specified parameters.

The probabilities in the vector also undergo mutation; based on a probability p, each entry in the vector is shifted towards 0 or 1 (if possible). Note that if a probability is already the exact value of 0 or 1, it can only be mutated towards the other binary value.

In principle, this two-fold approach to the modification of the probability vector increases the probabilities that generated individuals display "better" traits. The cycle of generating individuals and updating the probability vector continues for a given number of iterations specified at run time.

The solution returned by a PBIL algorithm is directly derived from the probability vector: the solution simply rounds the entries of the vector to the nearest integer. Because the entries are probabilities, the possible integers are clearly limited to 0 or 1. In other words, the solution is equivalent to the individual most likely to be generated by the probability vector.

V Particle Swarm Optimization

In our work, the individuals were particles, each of which contains vectors describing their current location, their current velocity, and the location of its personal best solution. Thus, given that the solution space is the Real numbers of dimension d (R^d), each vector consists of d real numbers. Particles are guided by both their personal and neighborhood bests.

Particle Swarm Optimization is a method used to find the maximum or minimum values of a problem. In our work, we attempted to locate the minimum values of three functions.

A PSO algorithm first initializes all particles at random points within the solution space; in our work, we specifically set the initialization ranges away from the global minimum, thus avoiding any possible algorithmic initialization bias towards the desired values. Without such restrictions on initialization ranges, designing a "best" algorithm simply devolves into an exercise of figuring out how to initialize particle coordinates as close to the global minimum as possible.

In addition to the initial position, each particle is also initialized with an initial velocity. During each iteration, particles move in the direction of their velocity, as well as in the direction of the neighborhood best position. Following this movement to a new position, to decide whether the new location is a new personal best, each particle evaluates its new position coordinates with respect to the function.

More formally, the velocity is updated as follows:

$$V_i = \chi [V_i + U(0, \phi_1) \otimes (p_i - x_i) + U(0, \phi_2) \otimes (g_i - x_i)]$$

Regarding the above equation, the following notation key should be followed:

V_i is the velocity vector.

$U(a, b)$ is a vector of random entries, where each entry takes on a value between a and b.

ϕ_1, ϕ_2 are the random update weight of the personal best and the random update weight of the global best, respectively.

g_i is the global best vector.

p_i is the personal best vector.

$\otimes$ denotes component-wise multiplication.

χ is the constriction factor, usually defined as:

$$\chi = \frac{2k}{\phi - 2 + \sqrt{\phi^2 - 4\phi}}$$

where $\phi = \phi_1 + \phi_2$, and usually $k = 1$.

In our work, $\phi_1 = \phi_2 = 2.05$.

Note that the personal best position vector is the location of the best value that the particle has found. The neighborhood best for a particle *P* is the location of the best value found by any of the particles within the neighborhood of particle *P*. A neighborhood of particle *P* is simply a group of particles in the particle swarm assigned to particle *P*; various methods of selecting such groupings are described in the next section.

In summary, the particles were each initialized with a random position and velocity within the solution space. The treatment of randomness is particularly significant: it is particularly important to ensure that particles are not initialized in locations that already return the best solution within the search space. Otherwise, the performance of a specific construction of a PSO algorithm simply depends on how well it can initialize particles in location that most easily lead to discovering the location of the best solution. Then, on each iteration, individuals velocities were updated based on personal and neighborhood bests, positions were updated based on these new velocities, and the value of the current solution (essentially the value of the function f that is being optimized) was calculated at the new position. Following the update of personal and neighborhood bests (if necessary), the next iteration begins.

IV Neighborhood Topologies

Neighborhood topologies differ from geographic ones in two key aspects: first, unlike geographic neighborhoods, a particle's location has no effect on the particle's topological neighborhood. Thus, it follows that topological neighborhoods are necessarily fixed, as in the set of particles within a topological neighborhood never changes (except for Random topologies; however, Random topologies do not change as a results geographic reasons). Such topological neighborhoods are "totally independent of geometry," and the generally share the property that "neighborhood composition remains constant."

We implemented Particle Swarm Optimization while comparing the performances of various neighborhood topologies. The four topologies investigated were Global, Ring, von Neumann, and Random.

In a global neighborhood, every particle belongs in every particle's neighborhood. Thus, the personal best of any neighborhood is always equivalent to the global best of the swarm.

In Ring neighborhoods, particles are arbitrarily organized “single-file” in a circular ring. Then, a particle *P*’s neighborhood consists of the particles adjacent to *P*. One can also visualize such a configuration as a line with connected ends.

In von Neumann neighborhoods, a two-dimensional grid is arbitrarily filled with the particles. A particle *P*’s neighbors are the particles directly up, down, left, and right of *P*. In other words, these are the particles of “taxicab distance” 1 away from the *P*. If *P* is on an edge of the grid, it’s neighborhood extends to the opposite edge to select its neighbors.

In random neighborhoods, a particle has random neighbors chosen without repetition. In subsequent iterations, the particle has a twenty percent chance to have its random neighborhood re-created (randomly).

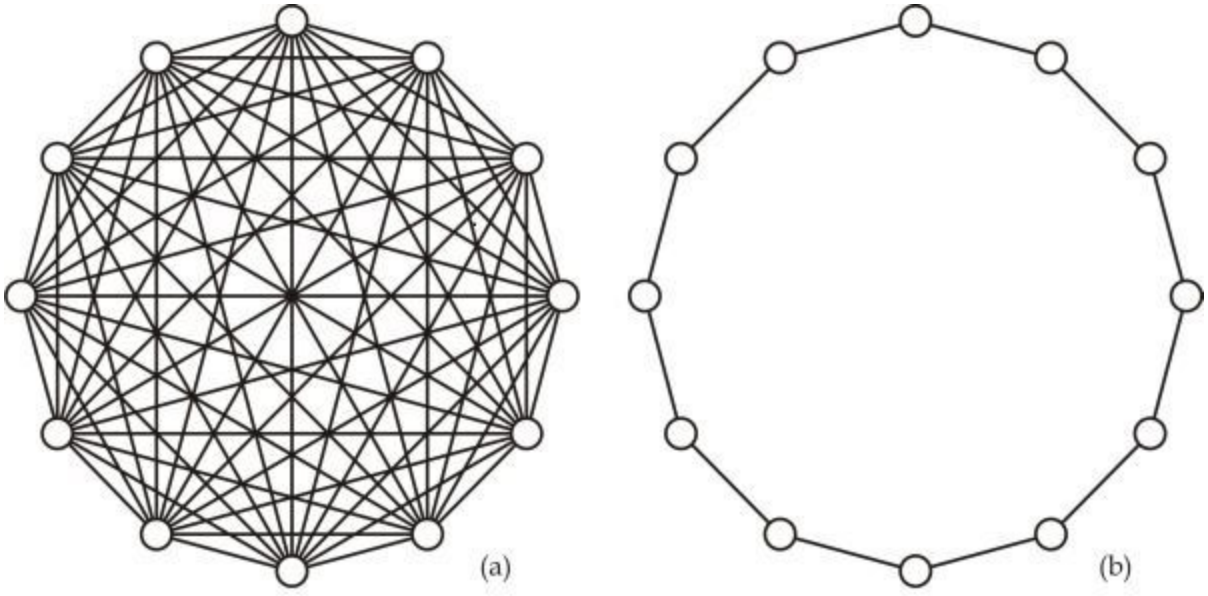


Figure 4: Global topology (a) and Ring topology (b)

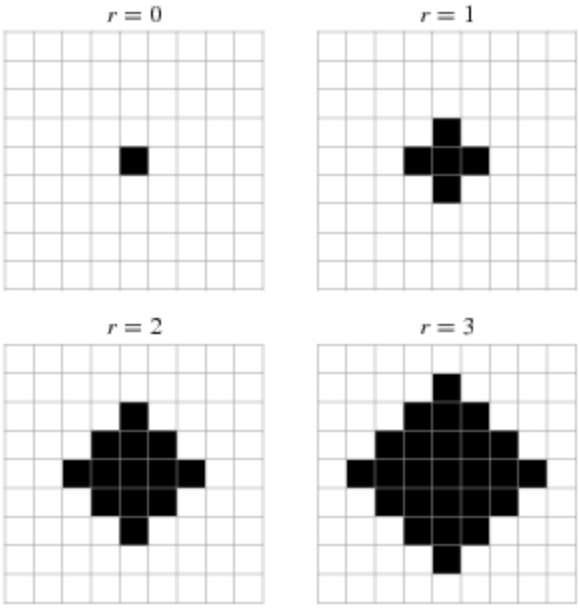


Figure 5: Von Neumann neighborhoods of various distances.

VII Hybridization

Our work implements a tri-hybridization of techniques derived from three types of nature-inspired algorithms: Genetic Algorithms, Population Based Incremental learning, and Particle Swarm Optimization. Specifically, our work is divided into two parts. The first is an investigation of the specific parameters of a GA-PSO hybrid that allow it to return the best results in function optimization. The motivation behind such a hybrid lies in the possibility of merging the merits of both techniques: the communication aspect of PSO combined with the fine-tuning process of evolution through selective randomness in GA.[6]

In our work, the method of GA-PSO hybridization builds on the work of Kao and Zahara. Specifically, at each iteration, n solutions are ranked by their fitness; then, given some q in $(0,1)$, the nq best solutions are modified by GA techniques. GA techniques are applied to the nq best solutions, and PSO techniques are subsequently applied to the entire population. As such, this method is essentially a PSO variant with a GA enhancement at each iteration, rather than a true GA-PSO split.

In the literature, specifically of Kao et al, $q=0.5$. The second part of our work examines this value q, which our work defines as the cut threshold: the proportion of the population on which GA techniques are applied. This investigation is carried out using PBIL techniques.

[6] Kao, Y., & Zahara, E. (2008). A hybrid genetic algorithm and particle swarm optimization for multimodal functions. Applied Soft Computing.

In our work, the PBIL algorithm represents q as a binary decimal expansion. Specifically, optimizing q to three significant digits requires a binary symbol string of length 10: $a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}$, where a_i is in $\{0,1\}$. Then $q = ½ * a_1 + ¼ * a_2 + ⅛ * a_3 + \ldots$ Note that the factor associated with a_{10} will be $(½)^{10} = 1/1024 \approx 0.001$, i.e. an approximate precision to three significant digits.

The individuals in the PBIL algorithm are GA-PSO hybrids, which inherit cut thresholds generated from the q probability vector.

VIII Experimental Design

In addition to the cut threshold q, our work examines other parameter values as well. For the GA-PSO hybrid, our work experiments with various neighborhoods topologies, swarm sizes, and crossover methods. Though optimal settings for these parameters have been discovered through thorough testing in previous works, new optimal parameter values may arise because of the interplay between GA and PSO.

First, we compared various cut thresholds of 0.0, 0.05, 0.1, 0.2, 0.5, and 1.0, to test the effectiveness of GA techniques. Each set of cut thresholds was tested on each set of parameters 5 times for 10000 iterations; at every 1000 iterations, our calculates the mean best value found. The choice of mean over median is that while both metrics give a sense of the generally quality of good solutions, the mean also offers some insight into the consistency of such good solutions. Through early testing, we found that under certain sets of parameters, the hybrid algorithm would occasionally return awful solutions, which would be captured by the mean metric.

Regarding the sets of parameters for which we tested the hybrid, our work inspects cut thresholds on every possible combination of the following parameters: number of particles, neighborhood topology type, crossover method, and function choice. For number of particles, our work tests 30 and 49, as those are the numbers tested in a previous work on PSO; thus, we are able to offer comparisons of our hybrid to PSO, although our hybrid with cut threshold of 0.0 demonstrates this comparison as well. In addition, we added a 144 particle swarm size on the recommendation of Kao et al that the number of particles in the hybrid be roughly four times the number of dimensions, which for us was 30.[7]

For neighborhood topology type, our work tests random (ra), ring (ri), and von Neumann (vn) topologies; for crossover methods, our work tests 1-point crossover (1c) and uniform crossover (uc).

In our work, the set of investigated parameters did not include mutation or crossover probabilities. Specifically, our work implements a .2 probability for mutation probability, and 1.0 for crossover probability, as those are values recommended by Kahara et al.[8] A paper by

[7] Kao, Y., & Zahara, E. (2008). A hybrid genetic algorithm and particle swarm optimization for multimodal functions. Applied Soft Computing, 8(2), 849-857.
[8] Kao, Y., & Zahara, E. (2008). A hybrid genetic algorithm.

Michalewicz et al also implies that such values are standard.[9] While the typical standard mutation value found in the literature is a mutation probability of 0.01, such recommendations assume a symbol solution string that takes on a finite set of discrete values. Thus, in these contexts, mutation has a quite a large effect; for example, in a bitstring, mutation results in a change to the only other symbol option. This phenomena is known in the literature as the Hamming Cliff effect, and properties of this are well recorded in the literature.[10] However, for real numbers, the effects of mutation can be incredibly minute, based on the generated random number and iteration heuristics.

This work also ignores the implementation of selection operators for many of its trials, a centerpiece of standard GA algorithms. Note that our motivation for the GA-PSO hybrid lies in the exploitation of GA's randomness properties, which would theoretically be beneficial when attempting to escape local minima. Testing on the hybrid using Tournament, Rank, and Boltzmann selection also confirmed that the addition of these types of selection on top of the cut ratio split created algorithms that were relatively inefficient. Thus, the standard selection operators, namely rank, tournament, and Boltzmann, seemed inappropriate for the aims of our hybrid, especially given their quality of sampling with replacement. Such sampling with replacement results in an exploitation of the best solutions, rather than an exploration of all good solutions.

However, despite the aforementioned omission of a direct exploration of mutation probability and crossover probability, our work indirectly investigates these two parameters through the thorough testing of the cut threshold. Note that this cut threshold decides the determines of individuals selected for the breeding pool. Thus, the cut threshold establishes the proportion of individuals that undergo mutation and crossover operators, which can arguably be considered a property in line with the purpose of the mutation and crossover probabilities. In essence, these three factors control the contribution of randomness to the algorithm. Unfortunately, the sole exploration of the cut threshold does not allow for a comparison between the relative effectiveness of mutation versus that of crossover, with regard to function optimization.

In our work, performance will be judged primarily in terms of solution accuracy, given that speed of convergence and algorithm run time are fairly quick. The performance of our hybrid will be compared to the optimal values of parameters that we discovered in our previous work regarding the standard PSO algorithm.

We plan to test our algorithm on its ability to discover the global minimum of a function; thus, the problems on which we will be testing our hybrid will be drawn from standard optimization functions used in the literature.[11]

---

[9] Janikow, C. Z. and Michalewicz, Z. "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms.."
[10] Herrera, Lozano, & Verdegay. Tackling Real-Coded Genetic Algorithms (1998). p282

[11] Bratton, D., & Kennedy, J. (2007). Defining a Standard for Particle Swarm Optimization. 2007 IEEE Swarm Intelligence Symposium.

IX Results

All permutations of the four neighborhood types, two crossover methods, and three swarm sizes were tested on all five functions. For the sake of clarity, only the graphs with parameters that were later determined to be ideal will be included. Other graphs can be made available upon request. Assume results are of running the algorithms without selection unless otherwise specified.

The two crossover methods were first to be examined. The following figures display the closest a particle has gotten to the global minimum of the function over the iterations of the algorithm.

Figure 1: UC vs 1c on Rastrigin

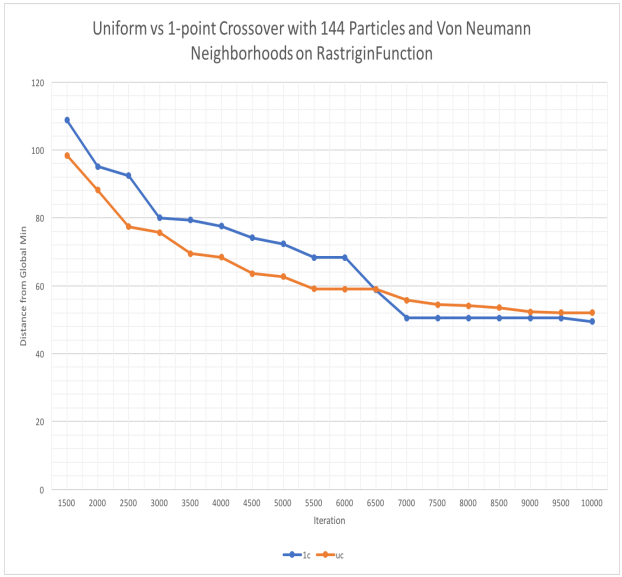


Figure 2: UC vs 1c on Rosenbrock

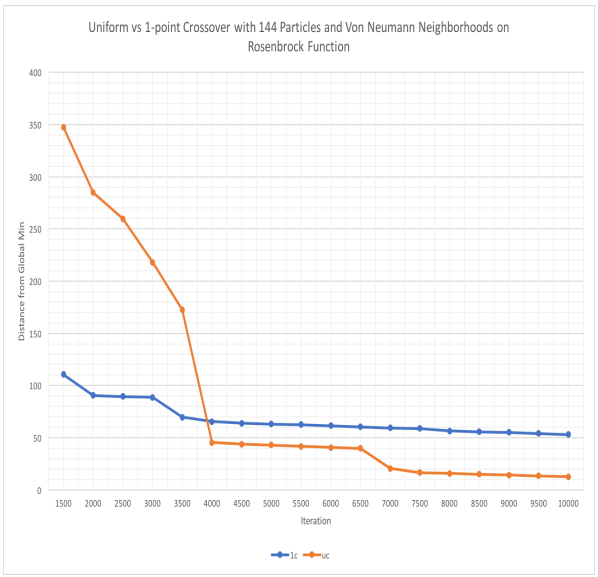


Figure 3: UC vs. 1c on Ackley

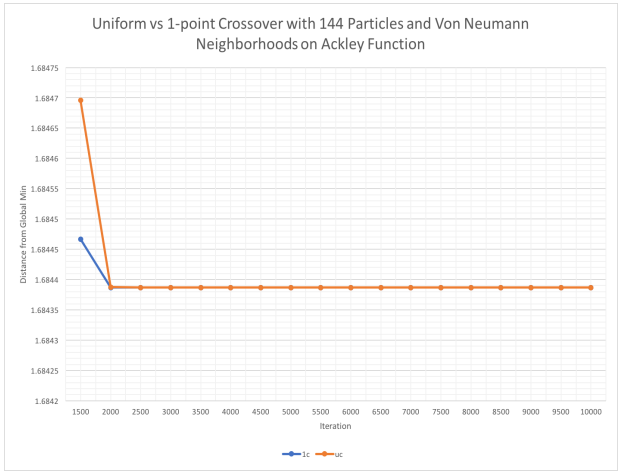
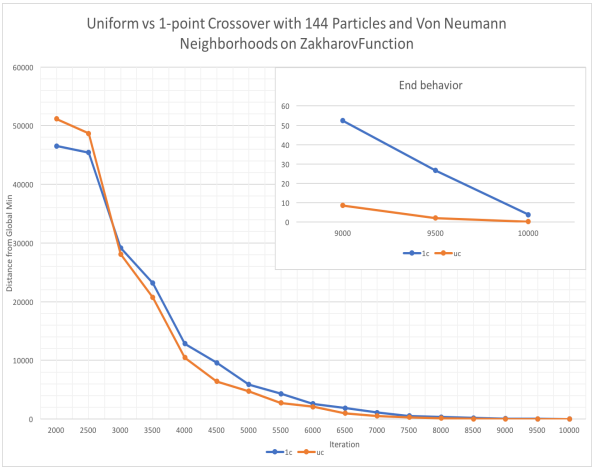


Figure 4: UC vs 1c on Zakharov



The performances of the two crossover methods were largely comparable across all functions. Only in Figure 2 (Rosenbrock) did we see a significant difference in quality of solution. For that reason, uniform crossover was determined to be slightly better and was used as an ideal parameter throughout the rest of testing. Note that the Griewank function is missing and will be discussed later.

Neighborhood type was next to be tested. The following graphs examine Ring, Global, Von Neumann, and Random topologies across different functions.

Figure 5: Neighborhood Comparison on Rosenbrock

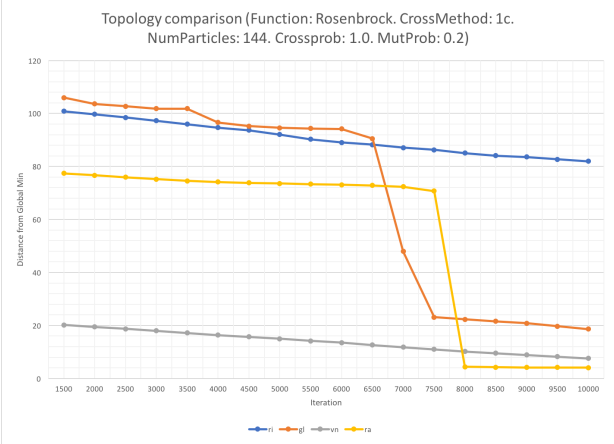


Figure 6: Neighborhood Comparison on Rastrigin

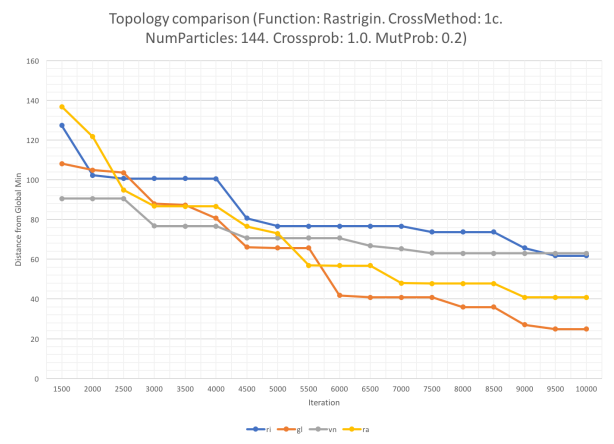


Figure 7: Neighborhood Comparison on Zakharov

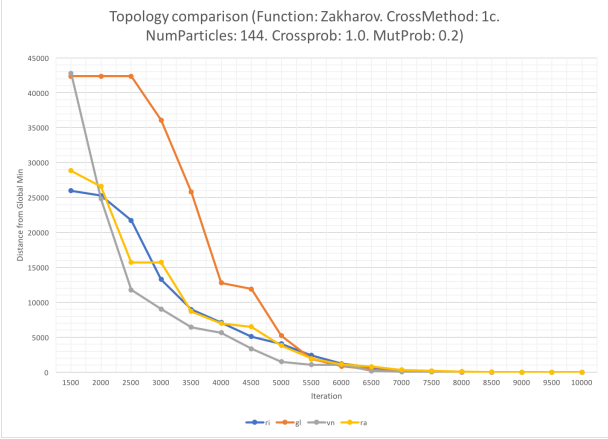
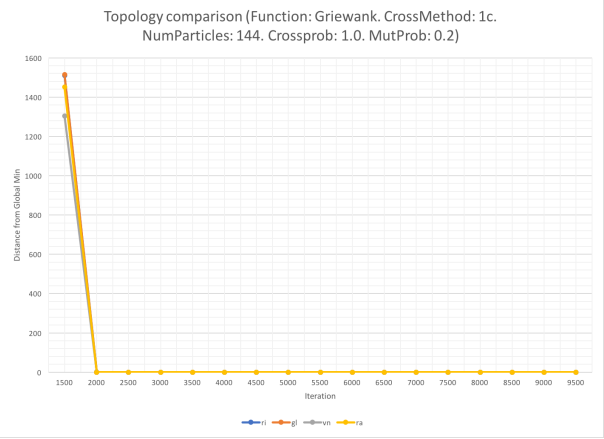


Figure 8: Neighborhood Comparison on Griewank

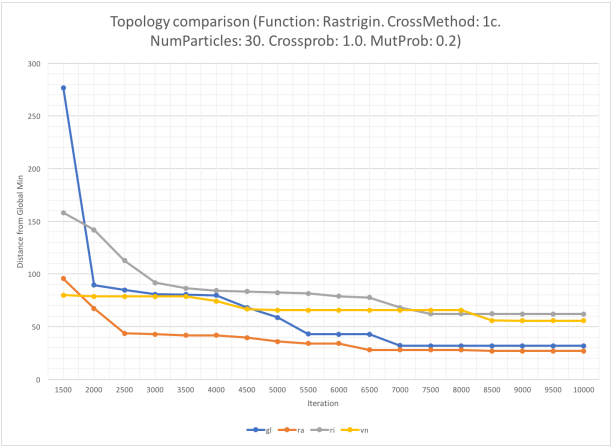


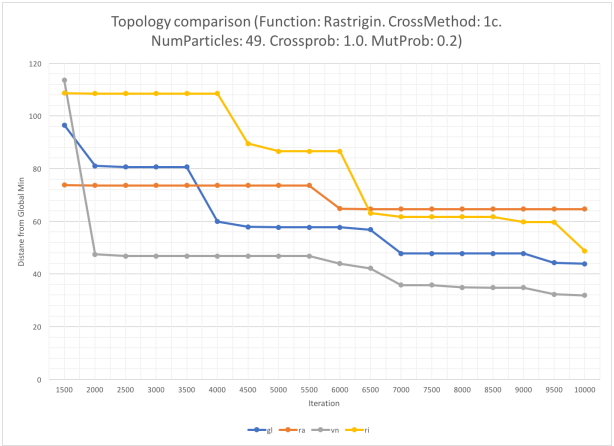
Again, there is no clear dominant topology, just as is the case with PSO. Like with a normal PSO, different topologies are best for different functions. Because all neighborhoods were fairly similar, a topology that performed strongly in all functions was selected for future tests: Von Neumann. With the performances so similar, topology was held constant for the rest of the trials in an attempt to create as few uncontrolled variables as possible. Though Figure 6 may show that global neighborhoods work slightly better on Rastrigin, for example, the goal of this research is

to compare the hybrid algorithm to the PSO. Varying neighborhoods across each function would make this much more difficult, so a good solution was selected for all.

Number of particles was tested next. 30, 49, and 144 particles were chosen to be tested due to overlap with prior experiments and their ease of integration with Von Neumann neighborhoods. Following is the performance of all neighborhoods on Rastrigin for these swarm sizes. Again, more graphs can be made available for other functions upon request.

Figure 9: 30 particles with all neighborhoods on Rastrigin *Figure 10: 49 particles with all neighborhoods on Rastrigin*





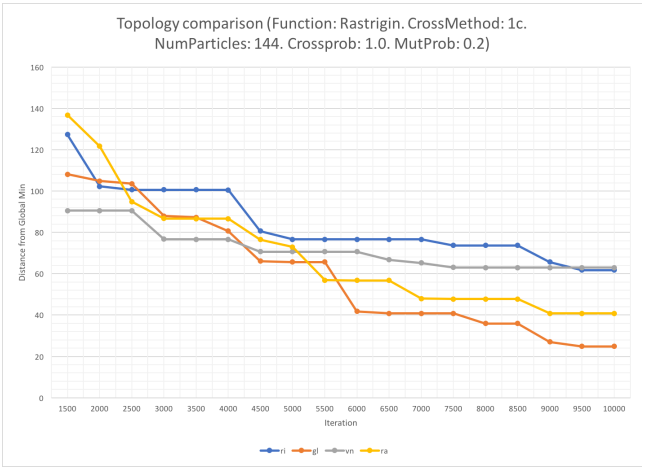


Figure 11: 144 particles with all neighborhoods on Rastrigin

As seen in Figure 9, 30 particles performs worse than 49 or 144 particles. However, there is little difference in performance between 49 and 144, so the improvement is not strictly linear. Supported by both the entirety of the data plus the data of Kao et al[12] who suggest a swarm size

[12] Kao, Y., & Zahara, E. (2008). A hybrid genetic algorithm.

of four times the number of dimensions, 144 particles was selected to be standard through the final tests.

Thus, the ideal parameters for the final comparison of the hybrid and PSO algorithms are uniform crossover, a Von Neumann neighborhood, and a swarm size of 144. Finally, we examine the performance of our hybrid against the classic PSO.

Figure 12: Hybrid vs. PSO on Zakharov

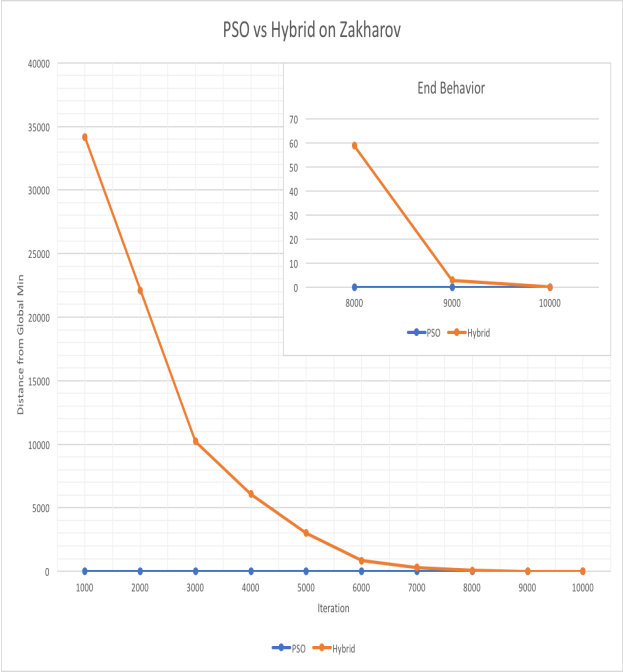


Figure 13: Hybrid vs. PSO on Rosenbrock

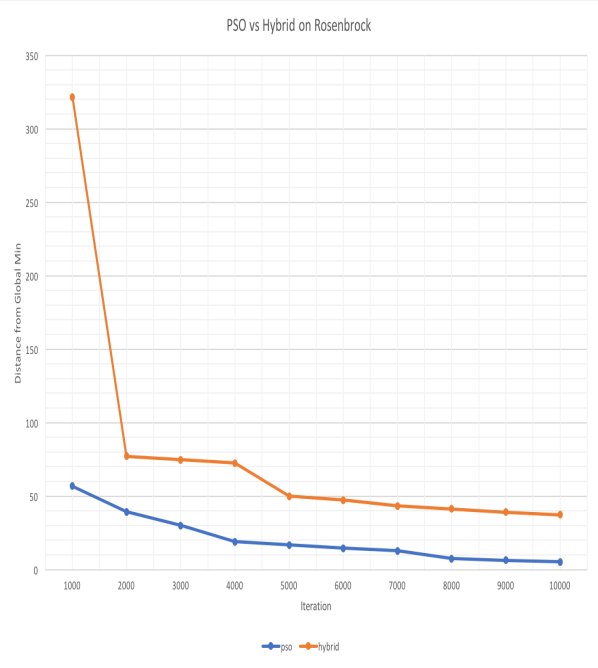


Figure 14: Hybrid vs. PSO on Griewank

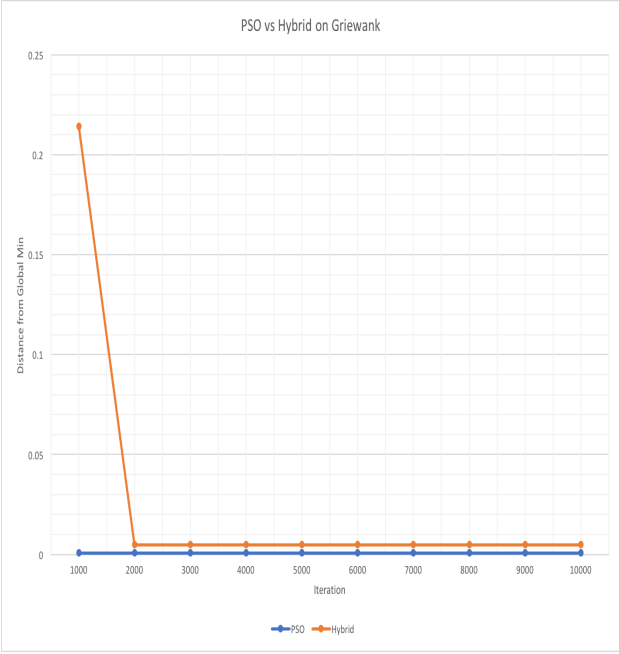
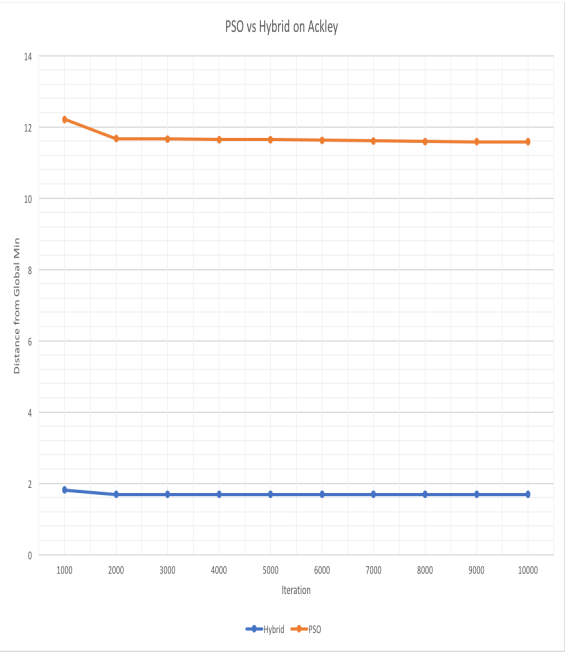


Figure 15: Hybrid vs. PSO on Ackley



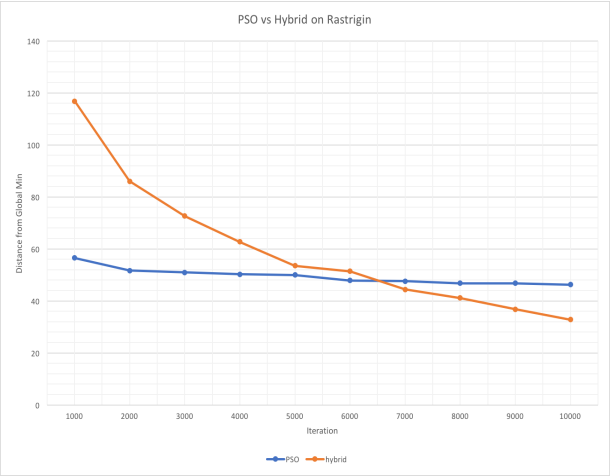


Figure 16: Hybrid vs. PSO on Rastrigin

The PSO used the same parameters as the hybrid.

Overall, the hybrid algorithm performed better on Ackley and Rastrigin and worse on Rosenbrock and Zakharov. In all functions but Ackley, it seems to converge more slowly. Figure 15 shows both algorithms quickly converging to their final solution, but because the initial data collection interval was too large, which one converged faster is inconclusive. This is likely due to the randomness of the genetic algorithm. The high mutation probability makes even particles that are approaching excellent solutions be sent away. With the high number of iterations, this randomness was eventually rewarded. Fewer iterations may have proved more challenging for the hybrid.

The two functions the hybrid did improve on share an important trait: they have many local minimum. The two functions the hybrid did worse on are flat. This follows the logic that the GA is providing additional randomness into our PSO algorithm. On flat surfaces, it is very easy to identify the correct direction for particles to move. The PSO's natural greediness allows them to move continuously toward the global minimum. The hybrid, however, sends these particles that are correctly identifying the best direction all throughout the search space. As the GA is performed on all the most fit particles, they are unlikely to reach the lowest values as quickly or effectively as a normal PSO.

For functions with significant local minima, the hybrid improved the quality of the final solution. Where the PSO may get caught, the randomness of the GA pops the particles out of the local minima where they are caught. Though this still seems to require more iterations, the PSO algorithm fails to find a solution as good as that of the hybrid.

The Griewank function is solved extremely quickly by both algorithms. This function deserves more testing, as the logic does not quite work out. The local minima in Griewank are plentiful, but they are smaller than that of Ackley and Rastrigin. It could be that the minima are so small that it is essentially flat to the two algorithms. Thus, PSO would perform very well on it, as it does. However, the hybrid performs equally as well (Figure 14). It could be because the function is fairly bowl shaped (in 3D), so there is a larger space for quality solutions. Perhaps even the randomness of the GA is not enough to create particles with bad solutions.

The PBIL implementation in order to determine the best q, or cut ratio, was not effective. When testing the PBIL on very small sizes of individuals and iterations the values did not converge, but that was not very surprising, and these tests were only done to ensure the code was functional.

This PBIL was then run on larger values, but through this testing it was discovered that running the larger problems was very computationally inefficient. The PBIL algorithm required running the Hybrid algorithm every time to test if a specific bit string of Q was a good answer. That means that the Hybrid algorithm had to be run for each individual, every iteration, so the combined PSO-GA algorithm ran the number of individuals times iterations every time the PBIL ran. Then that had to be tested on different parameter combinations. After the PBIL data collection loop had run for over 10 hours without returning its results, it was determined that a PBIL was computationally inefficient for solving this kind of problem. Even if the PBIL did converge with the larger parameter set, which there was no guarantee it would, the time taken to run the algorithm made it useless as a solution. The convergence may have been a problem because of the dynamic nature of the results of the hybrid algorithm. The same result is not guaranteed for a set of parameters, and the PBIL may have had a difficult time with a solution that was shifting around (if the solution even exists).

Another option to obtain information on the cut ratio was to gather information on it by running the Hybrid algorithm with different parameters and different cut ratio values, and take the mean of the best found values and of points along so many iterations to observe the speed of convergence. This data is recorded on 60 graphs appended to this report which have five main patterns depending on the function, which are displayed here/ These patterns held when the other parameter values were changed. These were all done with no selection.

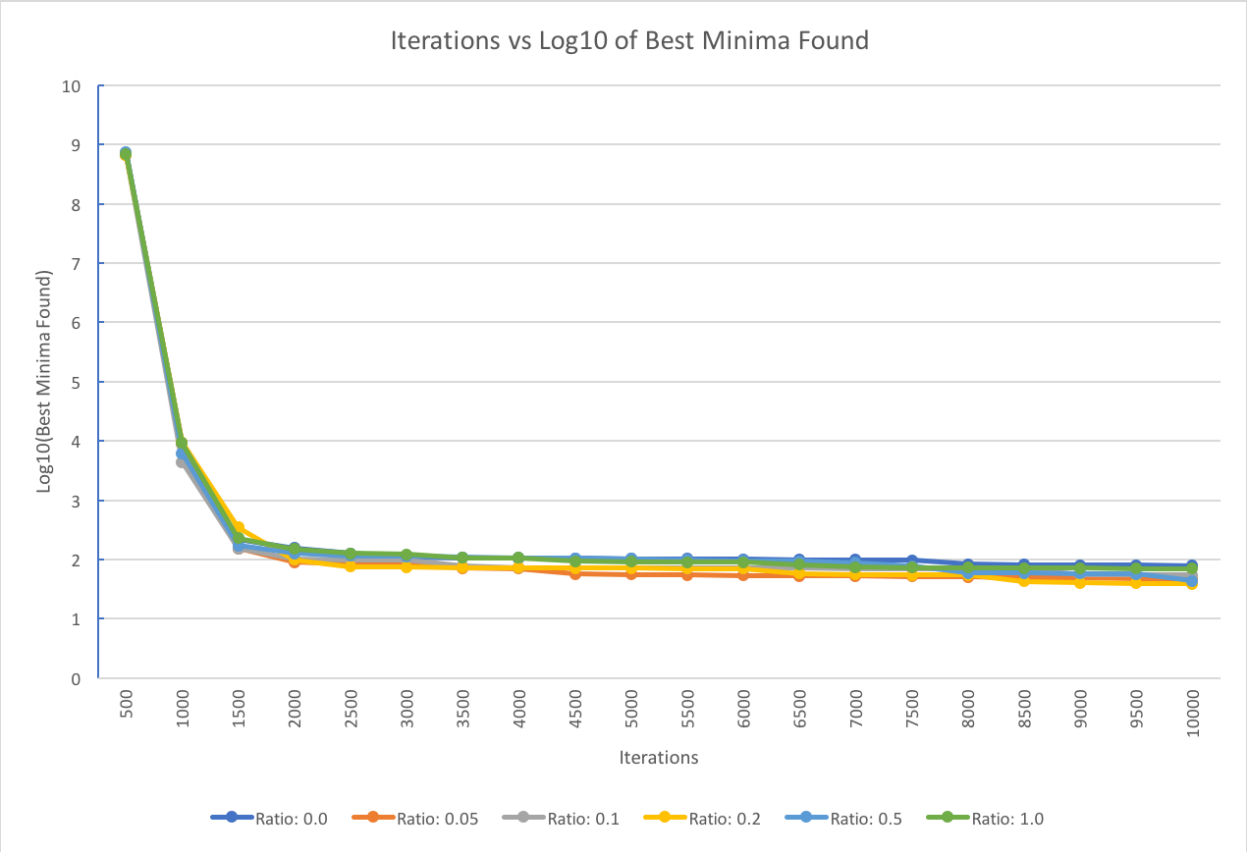


Figure 17: Comparison of Iterations to the scaled best minima for the cut ratio. Parameters = 16 particles, Random Nbhds., 1c; Function = Rosenbrock

Figure 17 shows that the Rosenbrock function has values that converge rather quickly for all of the cut ratios, and the values tend to remain the same around the. This is one of the more consistent functions with the values tending to converge at the same rate and have around the same best values no matter the cut ratio.

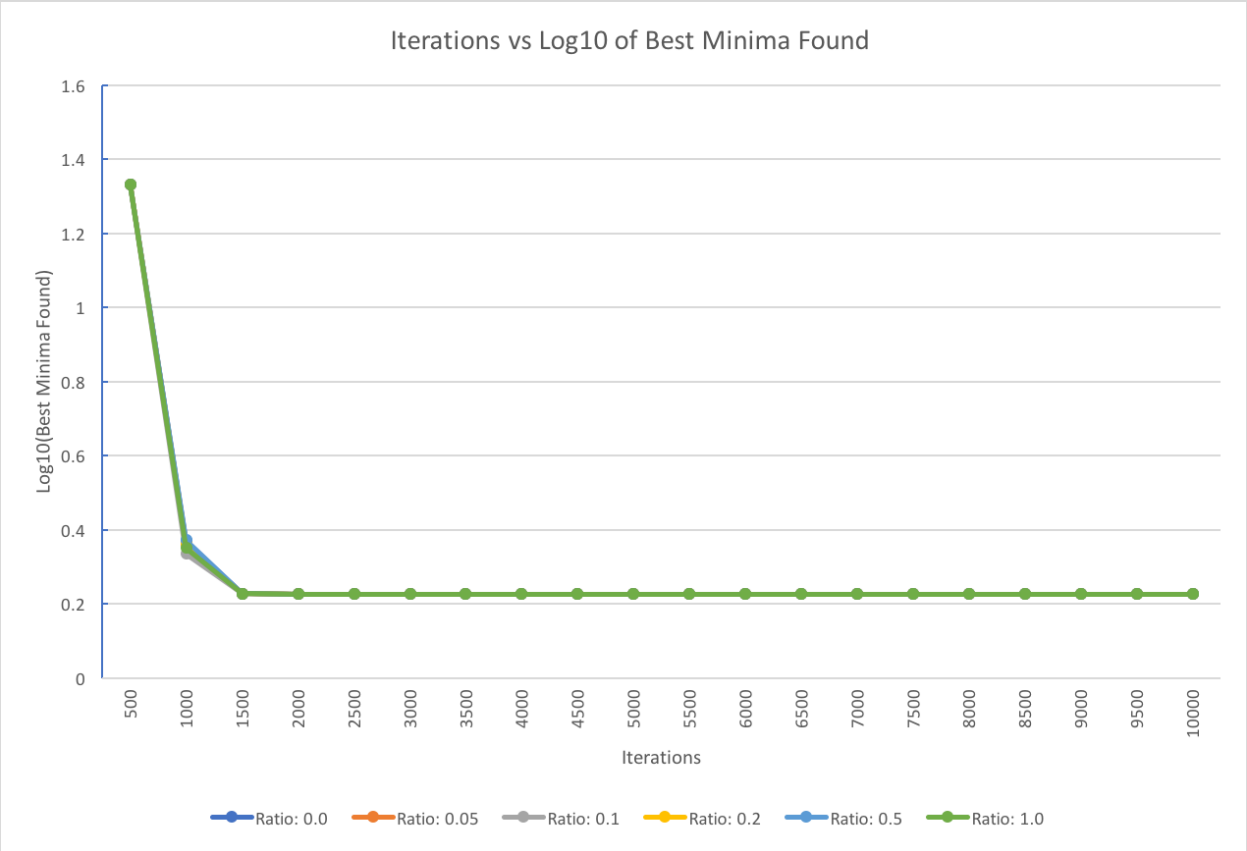


Figure 18: Comparison of Iterations to the scaled best minima for the cut ratio. Parameters = 16 particles, Ring Nbhds., 1c; Function = Ackley

Figure 18 shows that Ackley's function has almost exactly the same graph for any cut ratio. There is rapid convergence of all the ratios to one line that is not deviated from.

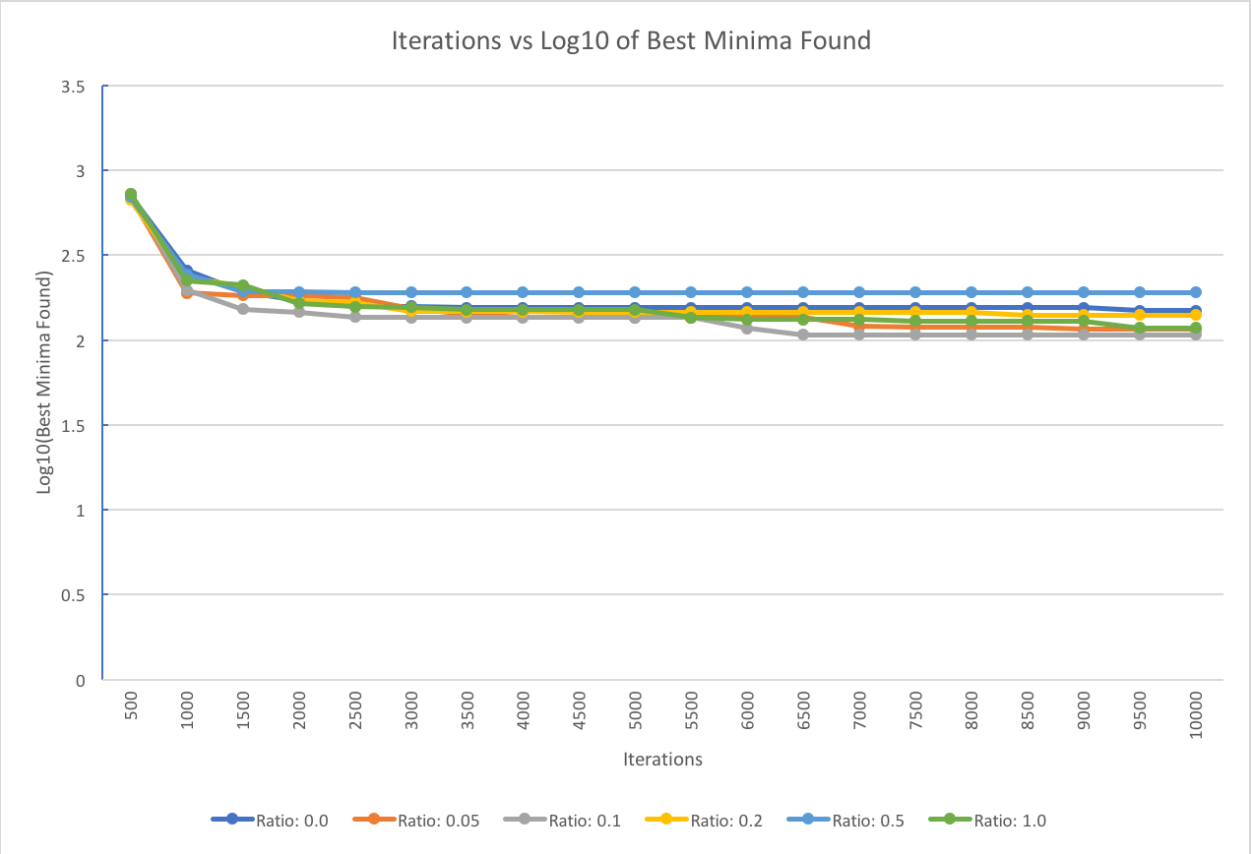


Figure 19: Comparison of Iterations to the scaled best minima for the cut ratio. Parameters = 16 particles, Ring Nbhds., 1c; Function = Rastrigin

Figure 19 shows that Rastrigin's function has a convergence within the first 1000 iterations, but the separate ratios spread out from each other slightly and some achieve slightly better results than the others. However, the best value is not always the same ratio, so no decisive conclusions can be drawn from the better values about a best cut ratio. However, a zero cut ratio is never the fastest converging or most accurate solution.

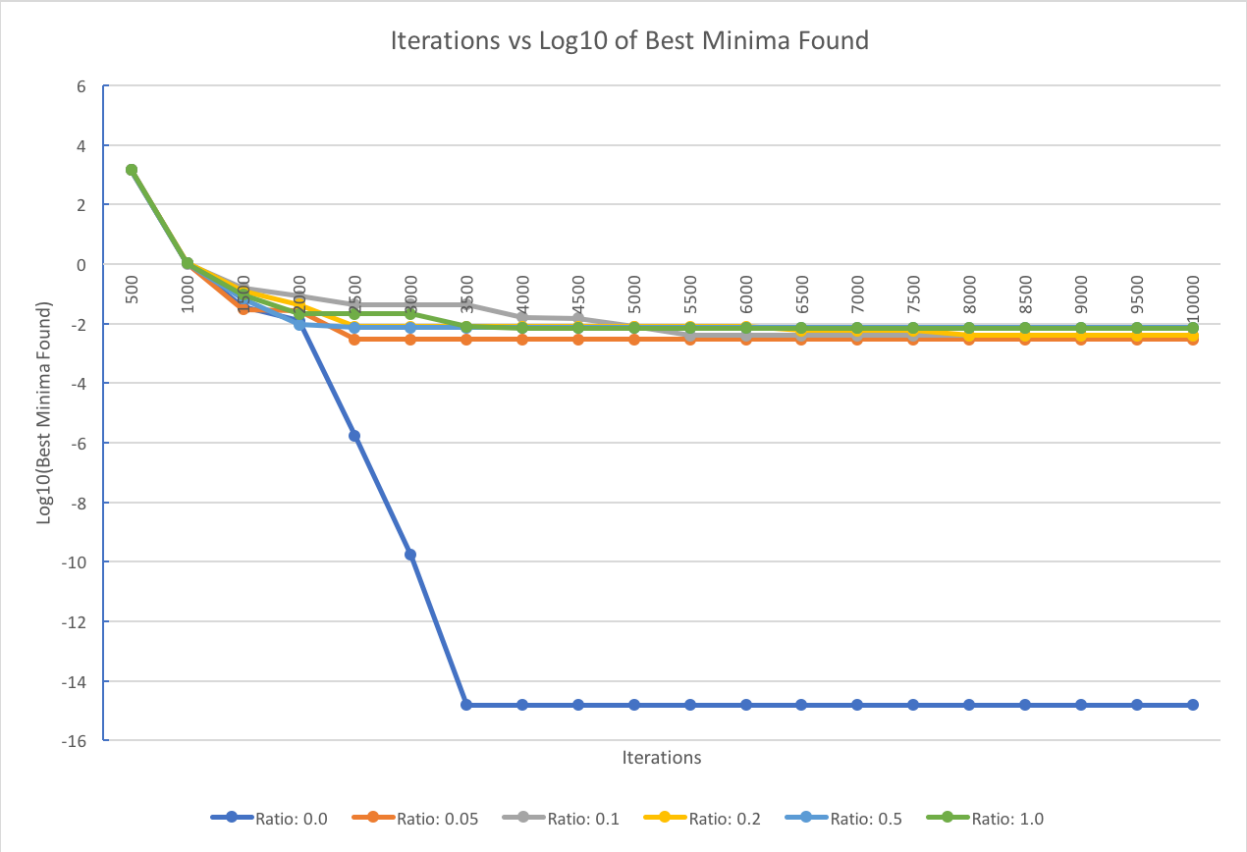


Figure 20: Comparison of Iterations to the scaled best minima for the cut ratio. Parameters = 16 particles, Ring Nbhds., 1c; Function = Griewank

The Griewank function tends to have a group of ratios converging together to a global minimum value and one or two outliers way below, as seen in figure 20. The outlier is not always the same cut ratio and appears to depend on which iteration of the algorithm happened to find the better values on that run.

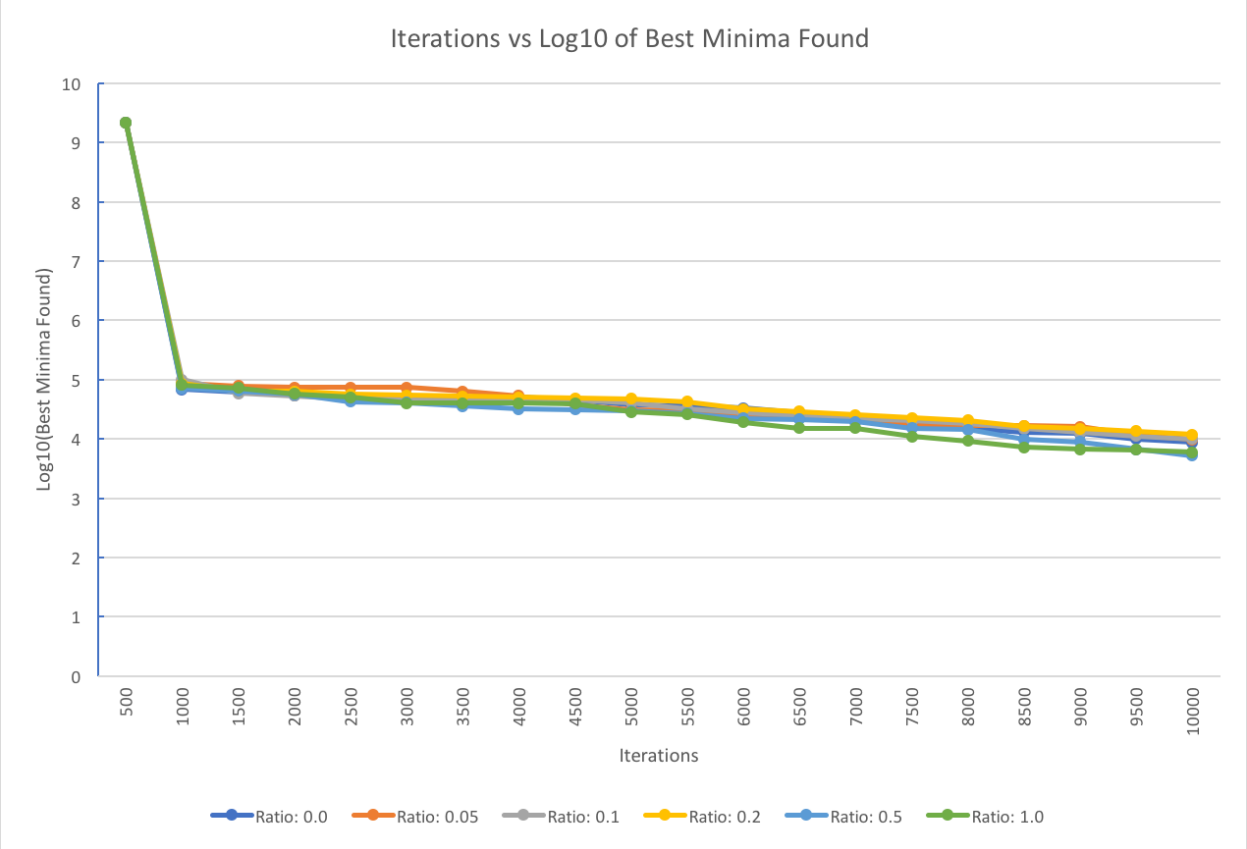


Figure 21: Comparison of Iterations to the scaled best minima for the cut ratio. Parameters = 16 particles, Ring Nbhds., 1c; Function = Zakharov

The Zakharov function's values also converge quickly relatively close together. The function also does not have a specific ratio that is better than the others, although the Zakharov function tends to continue to improve after the initial fast improvement, unlike many of the other functions.

No other patterns appeared in the cut ratio data based upon the other parameters tested (crossover type, particle numbers, and neighborhood topology). The functions probably caused different shapes in these graphs due to the shape of the function affecting the amount of convergence , not the change in cut ratio. The almost universal similarity in cut ratio across all the functions supports this conclusion, with no ratio ever performing significantly better than the others. The conclusions from these graphs do somewhat support earlier conclusions however, as a cut ratio of 0 on Rastrigin, which corresponds to only PSO, never performed best, whereas there seemed to be no patterns in Griewank, Zakharov and Rosenbrock. The values of Ackley were to similar in these graphs to draw any significant conclusions about cut ratio from.

The hybrid algorithm was also tested using (and not using) different types of selection on the particles it used, trying to run a complete Genetic algorithm within half of the PSO. Table 1 shows how in every case the Genetic algorithms with selection performed much worse than the hybrid without selection.

		None				
		Rosenbrock	Ackley	Rastrigin	Griewank	Zakharov
Particles	49	2.618054663	1.684386715	33.7686243	0.001479208	0.057826308
	144	0.769066109	1.684386715	41.34800437	0	0.000259963
	441	3.578886424	1.684386715	33.7686243	0	7.32901E-06

		Tournament				
		Rosenbrock	Ackley	Rastrigin	Griewank	Zakharov
	49	78818128.83	20.17459615	399.0301063	344.4492205	24837.18125
	144	71279783.83	19.92225331	374.006999	366.5283368	16782.7127
	441	49551014.63	19.61513868	334.5665603	313.5418001	15222.67063

		Rank				
		Rosenbrock	Ackley	Rastrigin	Griewank	Zakharov
	49	82234331.72	19.94415218	367.4403041	428.7402198	22588.55289
	144	67257126.81	19.9707453	357.0589799	389.2804516	17878.6142
	441	61378029.4	19.66612889	353.7000567	304.1885176	16329.25089

		Boltzman				
		Rosenbrock	Ackley	Rastrigin	Griewank	Zakharov
	49	85218349.25	20.09082215	397.490529	420.7102566	14978.69123
	144	91269798.77	19.94128388	381.9729743	400.7250509	12509.84772
	441	73537645.82	19.63787939	344.7716083	353.9770545	10655.63895

Table 1: Comparison of Selection types to no selection, mean global best value over 10,000 iterations with the given number of particles. Uniform Crossover, Von Neumann Neighborhoods, GA Crossover Probability 1.0, GA Mutation Probability 0.2

Table 1 also shows how the selection algorithms did improve as particle size increased. The conjecture from this data is that the selection most likely caused repeat particles to be created (because individuals can be chosen more than once and more "fit" individuals are more likely to be chosen) which then disrupted particle swarm optimization. The repeat particles would have the same position and velocity in the particle swarm, so they would essentially be the same particle, decreasing the particle size of the swarm by one, meaning the solution space would be explored less.

X Further Work

There are numerous directions in which one can pursue further work. One interesting direction is the idea of parameters that vary as iterations increase.

First, consider mutation. Recall that a desirable aspect of mutation is as follows: more extreme changes in earlier iterations, and more gradual changes in later iterations. Thus, one possible work could involve an investigation of the differences between various mutation equations, as well as varying rates of mutation over time. For example, a decrease in the maximal mutation value over time, coupled with increasing rates in mutation, would allow for greater, controlled mutation in late iterations, which certainly sounds desirable.

Further work could also examine mutation bounds, namely by inspecting more lenient lower and upper bounds, either by directly setting such bounds, or selecting equations that demonstrate an identical effect. Note that the motivation behind supplementing PSO algorithms with GA techniques is that the randomness inherent in GA techniques would theoretically allow individuals to overcome local minima. However, especially in later iterations, a mathematical comparison of the GA mutation operator and the PSO velocity update equation offers the argument that PSO actually contributes more randomness than GA techniques.

Specifically, the mutation operator essentially adds a factor that is multiplied by a power of a small decimal, which ultimately results in a tiny number. However, the PSO velocity update includes the addition a factor that is multiplied by a uniformly random number between 0 and 2.05, which will generally be larger in magnitude compared to the mutation operator's increment. Thus, increasing mutation bounds would allow an investigation of the greediness and randomness theory, with respect to the effective contribution of GA techniques.

Regarding variability over iterations, a further work could examine a variable cut threshold q. Note that in our work, while q is fixed for the duration of the algorithm, one could consider a variable q that evolves over time. Specifically, such a work would investigate whether a q that decreases over time performs better than a fixed q, and vice versa. In other words, what is the effectiveness of GA techniques as the solutions approach the optimal values. There are two ways to process this gradual change: either the optimal q value is the final value of q or the start value of q. It follows that there can also be an investigation regarding the choice of dq, namely the total change in q.

Further work could investigate how to properly implement selection methods, as well examine more complicated crossover methods. Ono et al have proposed the Normal Distribution Crossover (UNDX) method for real-valued GAs as a crossover operator that "preserves the statistical properties of the parental population well."[13] In essence, this crossover method is practically a probabilistically determined triangulation algorithm. A midpoint is first calculated between two parents, while a third parent provides adjustments only in the direction orthogonal to the line segment between the first two parents. Notably, these orthogonal adjustments are normally distributed.

At a high level, such a crossover method seems to be more applicable with regard to the realm of function optimization, versus the two crossover methods used in our work. Instead of scrambling between real values provided by parents, crossover operators such as UNDX allow for a more controlled and directed method of randomness.

Further work could examine a different method of GA-PSO hybridization, where PSO techniques are only applied on solutions below the cut threshold, in other words the worse candidate solutions that did not undergo GA. Applying PSO to worse solutions theoretically guarantees a steady improvement of the worst solutions, thus raising the floor of solution performance. Simultaneously, GA by either improving or worsening solutions, provides the randomness required to fine-tune good solutions. In this hybrid, PSO becomes a safety net against the potential bad solutions of GA.

Of note is that in this hybrid variant, neighborhoods must be reformed at each iteration. However, a previous work by J.S.B. et al[14] discovered that random neighborhoods perform very well in PSO algorithms; thus, the theory is that while neighborhoods must be reformed at each

[13] Kita, H., Ono, I., and Kobayashi, S. "Theoretical Analysis of the Unimodal Normal Distribution Crossover." Trans. of the Society of Instrument and Control Engineers, Vol.E-2, No.1, 193 (2002).
[14] John St. Belfield

iteration, the constant reorganization of neighborhoods ultimately does not negatively impact the performance of a PSO algorithm, or any variant thereof.

One possible issue with such an algorithm is that the separation between the initial best solutions and initial worst solutions essentially results in a lack of true hybridization, especially in early iterations. Solutions below the cut threshold would require many more iterations to breach the threshold, because of the absence of legitimate good solutions in their neighborhoods. However, given enough iterations, solutions should be able to breach the cut threshold with regularity.

XI Conclusion

The hybrid algorithm can be recommended under specific circumstances. Due to its longer iteration time (both a PSO and a GA iteration are being performed) and slower convergence, runtime must not be of the highest priority. Further, the hybrid should only be used on functions with many local minima. The PSO's greediness is more effective on flatter functions, but the randomness of the hybrid allows the particles to break out of their local minima in search of better positions. For ideal parameters of the hybrid, individual testing must be done on the target function. The neighborhood topology, swarm size, and crossover method all vary in effectiveness and speed on different functions, so we cannot make one generalization except to say the type of problem on which the hybrid can be effective.

The graphs detailing the performance of the algorithms for individual cut ratios appeared to help confirm the findings that the GA improved PSO due to the fact that the 0.0 cut ratio appeared to perform just as well as the other ratios in the functions without local minima and Griewank, and performed not as well in Rastrigin. The non-conclusive findings on a best cut ratio may be due to the fact that specific ratios may be better under different conditions, so while all the cut ratios seemed to be in the running with all of them, it possibly could be highly problem dependent and further testing is needed in this area to make a firm decision.

The attempt to use PBIL to predict the best cut ratio showed that some combinations of nature inspired algorithms are not necessarily a good idea, and the extravagantly long runtime is easily explainable by the excessive number of hybrid algorithm runs the PBIL would need to run.

The using no selection in the Genetic Algorithm inside the hybrid was clearly the most successful as the values were much closer to the actual minimum than the trials with selection. This result is hypothesized to be the case because the GA with selection can create repeat particles, which cas make the randomness of the PSO less effective.

XII Works Cited

Bratton, D., & Kennedy, J. (2007). Defining a Standard for Particle Swarm Optimization. 2007 IEEE Swarm Intelligence Symposium.

Deep, K., Thakur, M. “A new crossover operator for real coded genetic algorithms.” Applied Mathematics and Computation 188 (2007) 895–911.

Eshelman, L.J., Schaffer, J.D. Real-coded genetic algorithms and interval schemata, in: D.L. Whitley (Ed.), Foundation of Genetic Algorithms II, Morgan Kaufmann, San Mateo, CA, 1993, pp. 187–202.

Herrera, F., Lozano, M. & Verdegay, J. Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis. Artificial Intelligence Review (1998).

Janikow, C. Z. and Michalewicz, Z. "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms." Paper presented at the meeting of the ICGA, 1991.

Kao, Y., & Zahara, E. (2008). A hybrid genetic algorithm and particle swarm optimization for multimodal functions. Applied Soft Computing, 8(2), 849-857.

Kita, H., Ono, I., and Kobayashi, S. "Theoretical Analysis of the Unimodal Normal Distribution Crossover." Trans. of the Society of Instrument and Control Engineers, Vol.E-2, No.1, 187/194 (2002).