Supportive Coevolution

Brian W. Goldman
Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri, U.S.A.
brianwgoldman@acm.org

Daniel R. Tauritz
Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri, U.S.A.
dtauritz@acm.org

ABSTRACT

Automatically configuring and dynamically controlling an Evolutionary Algorithm's (EA's) parameters is a complex task, yet doing so allows EAs to become more powerful and require less problem specific tuning to become effective. Supportive Coevolution is a new form of Evolutionary Algorithm (EA) that uses multiple populations to overcome the limitations of other automatic configuration techniques like self-adaptation, giving it the potential to concurrently evolve all of the parameters and operators in an EA.

As a proof of concept experimentation comparing self-adaptation of n uncorrelated mutation step sizes with Supportive Coevolution for mutation step sizes was performed on the Rastrigin and Shifted Rastrigin benchmark functions. Statistical analysis showed Supportive Coevolution outperforming self-adaptation on all but one of the problem instances tested. Furthermore, analysis of instantaneous mutation success rate showed that this new technique is better able to adapt to the changes in the population fitness. Further study using multiple evolving parameters is needed to fully test Supportive Coevolution, but the results presented here show a promising outlook.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.6 [**Artificial Intelligence**]: Learning—*parameter learning*

General Terms

Algorithms

Keywords

Coevolution, Parameter Control, Self-Adaptation, Supportive Coevolution

GECCO'12 Companion, July 7–11, 2012, Philadelphia, PA, USA.

1. INTRODUCTION

The behavior of an Evolutionary Algorithm (EA) is controlled by a number of parameters and operators, each of which can affect the solution quality of the EA. Unfortunately, operator and parameter selection are not separable search spaces, and optimal settings are problem dependent. As a result, the difficulty of solving the target problem is often transformed into the difficulty of properly designing and tuning the EA. In order to allow EAs to be easier to use as well as more generally effective, a method is needed to reduce the impact of this problem specific tuning or remove the need for it entirely.

A further difficulty in EA configuration is the need for parameter values to change dynamically during the course of a single run [2]. This compounds the problem of EA tuning combinatorially: instead of just determining a single best value for each parameter, each parameter's value needs to change during different stages of a run in an optimal manner.

In order to automatically determine quality parameter and operator settings for each stage of an EA, Supportive Coevolution evolves these settings concurrently with candidate solutions. Supportive Coevolution differs from previous techniques by decoupling the evolution of parameters and operators from the main population allowing any number of settings to be evolved in this way.

2. BACKGROUND

2.1 Parameter Setting

The most commonly automated parameter in an EA is the mutation operator. One of the earliest mutation control methods used offspring quality to evaluate mutation quality [13]. While this method was able to make the EA less sensitive to a priori tuning as well as achieve better solution quality, it relied on a manually created feedback structure that has been greatly outperformed by self-adaptation.

Self-adaptation increases an individual's genome to include genes used for parameter control. For example, CMA-ES is a very successful method in which each individual encodes its own control values for how to perform mutation [6]. The quality of each individual's encoded mutation impacts the quality of its offspring. Therefore, if an individual has a good mutation operator, it is more likely to create good offspring, allowing that gene to propagate. If its mutation gene is bad, it will likely be unable to produce quality offspring, preventing propagation. Many other EA parameters have also been encoded using self-adaptation. For example, the operators controlling how offspring are created have

been controlled [4, 15, 3], and mate selection has been controlled allowing each individual to evolve their own mate selection [14].

Self-adaptation has limitations. First, the fitnesses of the self-adapted genes are indirectly linked to their survival chance. Just because an individual is highly fit does not mean its self-adapted genes are fit. Consider the case where a suboptimal mutation method – which contains a normally distributed stochastic element – randomly creates a single high quality individual. Even though the operator worked once, that does not mean it will work again. Jointly, this individual may have difficulties creating offspring of its own, hampered by its self-adapted genes. In order to overcome this issue, a method is needed to separate the operator's quality from the individual's quality.

This problem of indirect fitness becomes even more apparent when adapting multiple parameters. In [11], control methods were devised for all of the parameters in an EA. While this method was more generally applicable than less adaptive EAs, it was unable to achieve the same levels of solution quality as correctly tuned EAs on specific problems. If a self-adaptive approach attempted to control all of the parameters in an EA, all of an individual's encoded parameters would share the same fitness, independent of their actual fitness.

2.2 Coevolution

Coevolutionary Algorithms (CoEAs) determine the fitness of an individual in relation to other individuals. This field originally used a competitive model, in that individuals competed against each other to achieve their own fitness in a host parasite relationship [7]. This technique was later expanded to include a cooperative design, in which individuals worked together toward a common goal by splitting up the problem search space [8]. In both methods, all individuals are directly evaluated by the target problem's fitness function, either as antagonist or protagonist.

Coevolution of mutation step size was proposed in [12]. This algorithm used a two-population approach, with one population encoding solutions to the target problem and the other encoding offset vectors. Each generation, a single individual was created using a form of randomly weighted arithmetic crossover. The difference in genetic position between the offspring of this crossover and its nearest parent was added to the population of offset vectors. The rest of the offset vectors were initialized to zero, with random perturbations added upon each application. The rest of the offspring in the main population for each generation were created by adding and subtracting a random offset vector to a random individual, choosing the best of the two for each. This algorithm reported limited success.

There are a number of weaknesses in evolving mutation step size in this way. First, by using the offset vectors as discrete steps instead of step size probabilities, each offset vector has a much lower chance of being widely useful as it can only move an individual to two points in the search space. Second, the use of truncation selection based on relative fitness, with no reward for improving diversity, will likely cause offset vectors to trend to zero as smaller vectors have a lower risk of making changes drastic enough to an individual to significantly reduce fitness quality. As the offset vectors trend to zero, the only method to prevent premature convergence is crossover's creation of individuals and offset vectors. Unfortunately, the maximum magnitude of the offset vector created by crossover is equal to half the span of the individuals in the main population, meaning that the more individuals converge, the less mutation can change individuals. Furthermore, the offset produced by crossover is not rated on the quality of the offspring created.

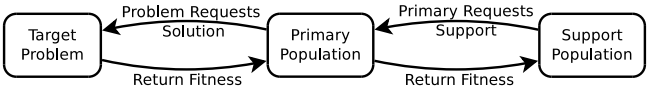

Figure 1: Supportive Coevolution information flow

Endosymbiotic Evolutionary Algorithms are a variant on cooperative coevolution which also splits the solution to a target problem into multiple species [9]. To evaluate a solution, individuals must be chosen from all of the populations and evaluated together. This method differs from standard cooperative coevolution in that individuals who work particularly well together are occasionally combined and evolved together in a new population. While this style of coevolution borrows its inspiration and name from cooperative natural systems composed of primary and support organisms, Endosymbiotic EAs do not use this type of relationship.

3. METHODOLOGY

Supportive Coevolution (SuCo) uses a multi-species, multi-population CoEA design. There are two main types of species: *Primary* and *Support*. The former is used to evolve solutions and the latter is used to evolve configuration information.

3.1 The Primary Population

The *Primary* population interacts with the target fitness function identically to a traditional EA. This means that each individual in the Primary population encodes all of the genetic information required to represent a problem solution. To evaluate an individual in the *Primary* population, the fitness function can be directly applied to its genes, as shown in Figure 1. All of the other operations in SuCo happen transparently around individuals of the *Primary* population.

3.2 The Support Populations

Instead of using tunable parameters and operators, the *Primary* population relies on the *Support* populations to evolve parameters and operators for each stage of evolution. This relationship is represented by the right half of Figure 1. Each *Support* population is used for a different parameter or operator. The research presented in this paper was restricted to using a single *Support* population to encode mutation step size, but any combination of parameters could be used, for example population size and the recombination operator.

To determine the fitness of an individual in the *Support* population, it is used by a *Primary* population to perform a task. The *Primary* population then rates how useful the *Support* individual was. As such, the supportive individual is never directly evaluated by the target fitness function, but instead receives an indirect fitness from the primary population. In the case of mutation step size, this is achieved by using an individual to create offspring in the *Primary* population. The fitness of the offspring created in this way is used to determine how fit the *Support* individual is. Because each *Support* individual is applied to different parents, the

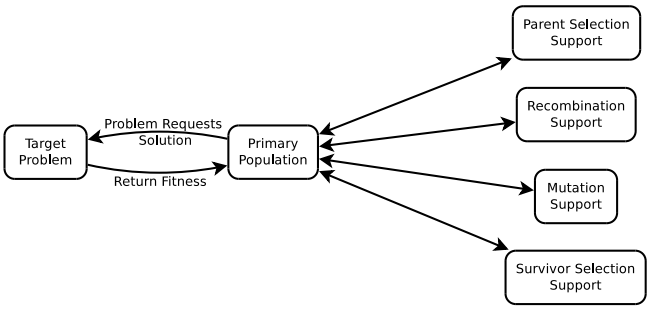

Figure 2: Supportive Coevolution multiple *Support*

offspring's fitness relative to its parents needs to be considered. Equation 1 is a natural counter measure to this problem, but other effective methods may exist. This equation subtracts the average parent fitness from the offspring's fitness to determine if the offspring is a relative improvement.

$$RelativeFitness = offspring - average(parents) \quad (1)$$

If only Equation 1 was used, the *Support* population would likely converge to very small mutation step sizes to reduce the risk of exploration [13]. To reward exploration, the fitness from Equation 1 can be scaled by Equation 2. In this equation, all of the distances between the offspring and its parents are multiplied. This encourages *Support* individuals to create *Primary* individuals that are more genetically dissimilar from all of their parents, while still emphasizing the need to improve the fitness of the offspring.

$$RelativeDistance = \frac{\prod distance(off, parents)}{\prod allPairsDistance(parents)} \quad (2)$$

By using each *Support* individual multiple times and accounting for parent fitness, the true fitness of that individual can be approximated. To use SuCo for other parameters and operators, similar techniques can be devised.

3.3 Multiple Support Populations

Figure 1 depicts SuCo using one *Primary* and one *Secondary* population. While in most previous work, only a single parameter or operator is evolved at once, SuCo can easily be expanded to evolve multiple aspects of an EA configuration concurrently, as shown in Figure 2. For each evolving parameter and operator, another *Support* population with its own fitness function can be created. When using multiple *Support* populations, the fitness of each individual can still be determined by using an individual multiple times in combination with different individuals from the other populations.

If a recombination *Support* population was used in conjunction with a mutation population, the fitness of a mutation individual would be determined by using it with multiple recombination individuals and determining its relative success. Increasing the number of supportive populations has no impact on how the primary population is assigned its fitness, and does not necessarily require any extra evaluations of the target fitness function.

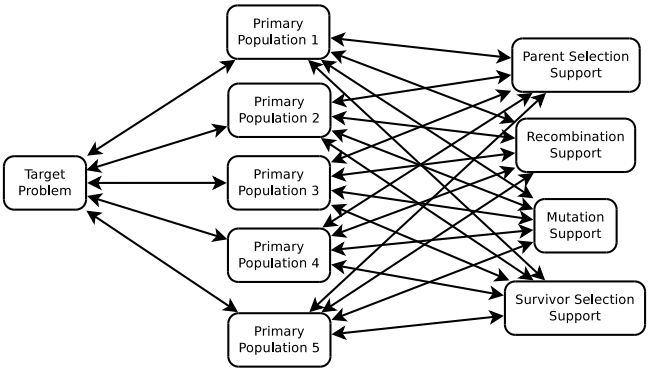

Figure 3: Supportive Coevolution multiple *Primary*

3.4 Multiple Parallel Primary Populations

The need to evaluate a *Support* individual multiple times to truly evaluate its fitness highlights another advantage of SuCo. Traditionally, EAs are run multiple times on the same problem to improve the odds that one run will stochastically obtain acceptable results. In these cases, the same EA is applied to the problem with no changes, or the only changes made are in parameter tuning. To make better use of these evaluations and allow the *Support* populations to react faster to the needs of the *Primary* population, SuCo can be extended to use multiple *Primary* populations at once, as shown in Figure 3.

When using a *Primary* population, Figure 1 shows the target problem requesting individuals from that population, evaluating them, and returning their fitness. To expand this to multiple *Primary* populations, the target problem simply requests for one individual from each population, one at a time. From the point of the target problem and the *Primary* populations, this is functionally identical to the multiple runs commonly done for stochastic reasons.

The advantage of parallel *Primary* populations is the ability for all of them to share the same set of *Support* populations. No matter how many *Primary* populations are in a SuCo, there is only one *Support* population for each parameter or operator being evolved. This allows the *Support* populations to receive fitness information from all *Primary* populations, and in effect allow them to evolve at an accelerated rate.

Another advantage to using multiple populations is the ability to evolve parameters that are typically more difficult for dynamic evolution, such as population size and survival rate. Because the *Support* population for each of these can receive feedback from multiple populations, it can evaluate the effects of using each value for these parameters at the same point in each *Primary* population's evolution.

4. EXPERIMENTAL SETUP

Since many of the previous methods for dynamic parameter control only allow for a single dynamic parameter, only a single *Support* population was used for this initial testing of SuCo. While this set does not test all of the proposed features of SuCo, such as multiple *Support* populations controlling more complex configuration options, starting simple

	Rastrigin		Shifted Rastrigin	
Parameter	***N*=10**	***N*=20**	***N*=10**	***N*=20**
Parallel Populations	11	5	7	9
Population Size	413	492	237	280
Offspring Size	75	238	9	24
Parent Tournament	8	5	29	27
Maximum Mutation Step Size	0.2698	0.2993	1.564	1.0445

Table 1: Experiment parameters by benchmark for SA

	Rastrigin		Shifted Rastrigin	
Parameter	***N*=10**	***N*=20**	***N*=10**	***N*=20**
Parallel Populations	6	13	8	5
Population Size	183	261	202	248
Offspring Size	204	161	89	343
Parent Tournament	2	6	12	15
Support Configuration				
Population Size	1	446	291	373
Offspring Size	312	85	187	210
Parent Tournament	16	1	25	12
Maximum Mutation Step Size	0.3756	0.4475	2.45	2.5072
Evaluations Per Generation	2	2	9	8

Table 2: Experiment parameters by benchmark for SuCo

Parameter	Value
Parent Selection	Tournament
Recombination	Arithmetic
Survivor Selection	Truncation
Evaluations	100,000
Runs	100

Table 3: Common configuration across all algorithms on all problems

provides an initial proof of concept by comparing with the most similar, and most commonly used, effective method for dynamic configuration. As mutation step size is the most explored dynamic parameter, it was the logical choice for experimentation. For comparison, an EA using self-adaptation of n uncorrelated mutation step sizes (SA) was used, similar to a traditional Evolutionary Strategy. In this method, a mutation step size for each locus, denoted as σ_i, is added to each individual in the population. Whenever that individual creates an offspring, it first provides the offspring with a mutated version of its mutation step size information. The offspring then uses its own mutation step sizes to control a Gaussian mutation, as shown in Equation 3.

$$\begin{aligned} \sigma_i' &= \sigma_i \cdot e^{\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)} \\ x_i' &= x_i + \sigma_i \cdot N_i(0,1) \end{aligned} \tag{3}$$

A SuCo using a *Support* population for mutation step size was designed to act in a very similar method as the comparison algorithm. Each individual in the *Support* population encoded a mutation step size for each locus in the *Primary* population. When the *Primary* population needs to perform mutation, it uses a randomly chosen *Support* individual to provide the σ_i values for Equation 3. To determine the fitness of a *Support* individual, Equation 1 and Equation 2 were multiplied together and averaged for each application of the individual. This fitness metric may not be the best possible method, but was used as a natural example of SuCo's ability to directly manipulate the *Support* individual's fitness, a feature inherently unavailable to SA techniques.

To help ensure similarity of comparison, both SA and SuCo were allowed to use multiple concurrent populations. As such, a single run of either algorithm involves running multiple concurrent populations and splitting the maximum number of evaluations evenly among populations. As SA does not share information between populations, using multiple concurrent populations only allows this algorithm more chances to improve its best fitness found. One hundred runs of each multiple concurrent population EA was used to achieve statistical significance.

A set of benchmark problems was employed to perform comparative testing and analysis. As mutation step size is primarily designed for real-valued functions, the commonly used Rastrigin Function [10] was chosen. This function is a representative of highly multimodal functions that contain no gene interdependence, the hardest problem class that an EA using self-adapted uncorrelated mutation step sizes performs well on. The Rastrigin function is give in Equation 4.

$$An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i) \right], \forall x \in [-5.12, 5.12] \tag{4}$$

The global optima for the Rastrigin function occurs when all x_i are equal to zero. This artificial centering of the global optima in the search space has the potential to be exploited by algorithms in ways not applicable to real world problems. As such, the Shifted Rastrigin problem offsets the location of the global optima values. At initialization, the Shifted Rastrigin problem randomly generates an offset vector o, where $o_i \in [-5.12, 5.12]$. When evaluating a solution, Shifted Rastrigin temporarily adds the o vector to the solution vector before using Equation 4 to determine solution fitness.

N	SA	SuCo
Rastrigin		
10	-0.2390 (-0.4249)	-0.0199 (-0.1393)
20	-0.3494 (-0.4905)	-1.4132 (-0.7481)
Shifted Rastrigin		
10	-4.4215 (-1.8744)	-2.2518 (-0.9811)
20	-10.3967 (-4.3567)	-5.7718 (-2.2848)

Table 4: Mean (standard deviation) of final best fitnesses on each problem; statistical best highlighted

To maximize the effectiveness of both algorithms on each problem, REVAC [1] was used to tune all configurable parameters. Tuning was performing using 3000 REVAC evaluations, where each of those involved running the algorithm being tuned 10 times to completion. The fitness of a configuration was set as the median final best fitness found of those 10 runs. While SuCo has more parameters to tune (configuration of the *Support* population), the limited amount of tuning available and the increase in tuning search space likely prevents SuCo from achieving better specialization than SA. Problem specific tuned configuration parameters for SA and SuCo can be found in Table 1 and Table 2, respectively. A complete listing of all common configuration settings is provided in Table 3.

5. RESULTS

Table 4 shows the mean and standard deviation of the final best fitnesses achieved by SA and SuCo on each instance of the Rastrigin and Shifted Rastrigin benchmark functions. To determine the significance of the resulting values, a T-test was performed comparing SA with SuCo on each instance. All differences were found to be statistically significant using a confidence level of α=0.001.

In order to better understand the effectiveness of each mutation method, Figure 4 through Figure 11 are provided. During evolution, any time a mutation was about to be performed, the individual it was about to be applied to was evaluated. The mutation was then performed, and the resulting individual was reevaluated, and the change in fitness was recorded. If the mutation resulted in a positive change (made the individual more fit), it was considered a success. If the mutation resulted in a negative change (made the individual less fit), it was considered a failure. Mutations that did not change the individual's fitness, were not counted. At regular intervals the mutation success rate $(successes/(successes + failures))$ was recorded and then the number of successes and failures were reset to zero. Figure 4 through Figure 11 show how the instantaneous mutation success rate changed with respect to both the number of evaluations and the current best individual found by the population.

6. DISCUSSION

6.1 Final Best Fitness

Table 4 shows SuCo achieving statistically better fitness on Rastrigin N=10, Shifted Rastrigin N=10, and Shifted Rastrigin N=20. Oddly, the most dramatic improvements are found on Rastrigin N=10, while on Rastrigin N=20

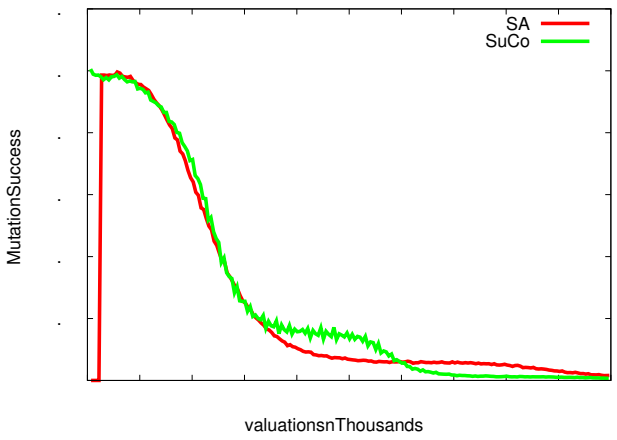

Figure 4: Mutation success rate with respect to evaluations, Rastrigin N=10

SuCo does significantly worse than SA. A likely cause of this deficiency is the parameter configuration that SuCo used for this instance.

To discern how SuCo's configuration for Rastrigin N=20 may have hindered its effectiveness, examine the differences between how SuCo's configuration changes with respect to dimensions on Shifted Rastrigin. By examining Table 2, it appears that on Shifted Rastrigin, SuCo's configuration favors increasing diversity and reducing parallel populations as the number of dimensions increases. This can be seen in the increase in population size and offspring size for both the *Primary* and *Support* populations. While the parent tournament size used by the *Primary* population increases, relative to the increases in population size, the selective pressure stays about the same. The *Support* parent tournament size drops dramatically, especially when considering the change in population size.

The configuration differences between Rastrigin N=10 and Rastrigin N=20 do not follow this pattern. While the population size does increase, the offspring size shrinks significantly, the parent selective pressure increases, and the number of parallel populations increases. Even though all of this suggests that SuCo's configuration for Rastrigin N=20 may not be the most effective possible, that configuration was found by REVAC, and further tuning would be unfair to the other algorithms.

6.2 Rastrigin Mutation Success

On the N=10 version of Rastrigin, SuCo's mutational success is very similar to that of SA. Figure 4 and Figure 5 show each algorithm's success in regard to evaluations performed and the fitness of the current best individual. Initially both algorithms start at a nearly 50% success rate, which makes sense as applying a random change to randomly generated individuals will likely give you an even chance of improvement. More precisely, when an individual exists near a peak or valley in the fitness landscape, large mutations have a high probability of pushing that individual off the peak or valley. When individuals have a fitness near the average of the landscape, mutations of any size will likely have an equal chance to improve or reduce fitness. To maintain a high mutation success rate as the population improves, the size of the

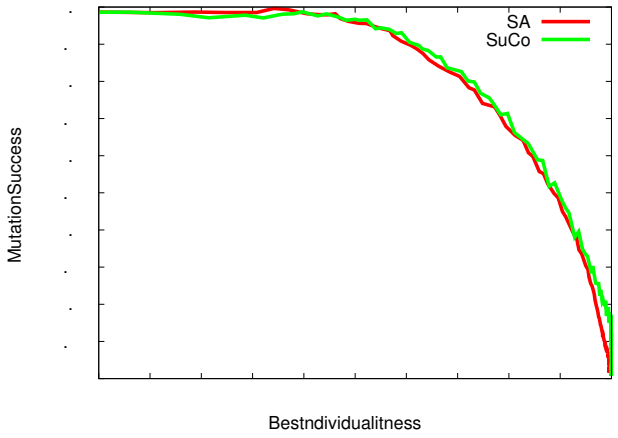

Figure 5: Mutation success rate with respect to best individual fitness, Rastrigin N=10

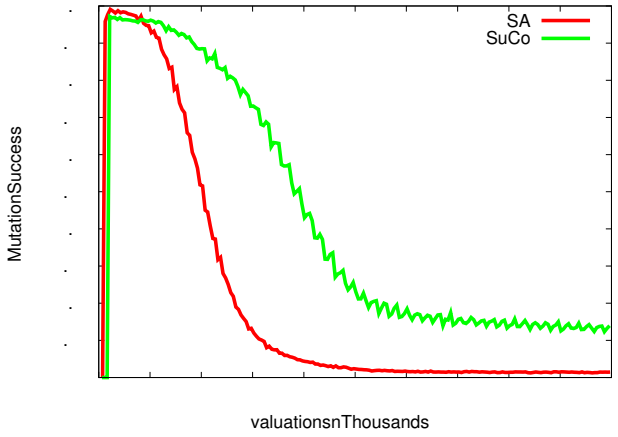

Figure 6: Mutation success rate with respect to evaluations, Rastrigin N=20

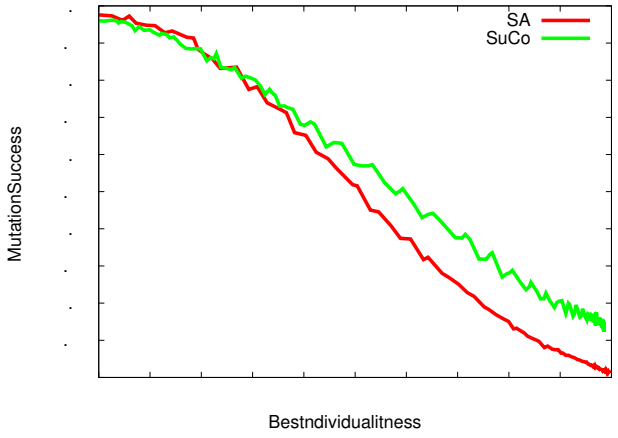

Figure 7: Mutation success rate with respect to best individual fitness, Rastrigin N=20

mutations made needs to correspond with this improvement. As such, the expectation for mutation success rate is to decrease with both the number of evaluations and the relative fitness of the current population. Furthermore, both SA and SuCo rely on evolution to perform mutation, meaning they both have the potential to suffer from diversity loss. Losing diversity can prevent the mutation step sizes from adapting to the shifting target needed by the population. Once the mutation step size gene pool has specialized, if it cannot adapt, you would expect to see a monotonic decrease in effectiveness as less and less individuals that benefit from the specialized mutation step sizes remain in the population.

When examining the mutation success rate relative to evaluations for Rastrigin N=10 shown in Figure 4, this pattern of monotonic decrease is apparent. Both methods quickly loose effectiveness, and begin approaching a zero success rate. The interesting divergent behavior occurs between 30 thousand and 70 thousand evaluations. Here SuCo is able to maintain a higher success rate longer, before returning to free fall again, dropping even lower than SA. To understand this behavior, its necessary to also examine Figure 5. This graph also shows a great deal of similarity between SA and SuCo, but the relevant section occurs at the far right of the graph. While both algorithms appear to lose effectiveness quickly at higher fitnesses, SuCo is able to sustain a higher quality near convergence. The likely explanation for this behavior is SuCo's increased ability to adapt and the ability for SuCo to decouple an individual from its mutation step size. Returning to the previous figure, the likely cause of SuCo's higher success rate is this effectiveness on highly fit individuals. When SuCo drops below SA in Figure 4 the likely cause is that the quality of individuals in the SuCo *Primary* population have improved beyond the quality of the SA individuals at the same number of evaluations, resulting in SuCo having individuals that are harder to improve.

Turning to the N=20 version of Rastrigin, Figure 6 and Figure 7 show that SuCo and SA have similar trends. SuCo is able to maintain a higher mutation success rate for a higher number of evaluations than SA, and is more successful at higher individual fitnesses. Table 4 shows that SA achieved better results than SuCo, which seems counter intuitive to the graphs on mutation success rate. Along with the explanation given in Section 6.1, possible explanations for this behavior are that the magnitude of the improvements made by SA are larger than SuCo, and that the types of mutations made by SA are more beneficial (or less detrimental) to the arithmetic crossover used by both algorithms. In either case, SA is able to achieve statistically better results while overall having a less successful mutation operator, suggesting the primary reason for the difference is not the effectiveness of the mutation method.

6.3 Shifted Rastrigin Mutation Success

The behavior of SuCo on Shifted Rastrigin is significantly different than both its behavior on normal Rastrigin as well as all of the observed behaviors of SA. When using N=10, SuCo breaks from the expected monotonic decrease of mutation success, as shown in Figure 8. From about 5 thousand to 30 thousand evaluations, SuCo's mutation success rate improves, before returning to the more expected shape of monotonic decrease which asymptotically approaches zero. Figure 9 shows a similar behavior with respect to fitness. SuCo appears to have a slower start than SA in both graphs,

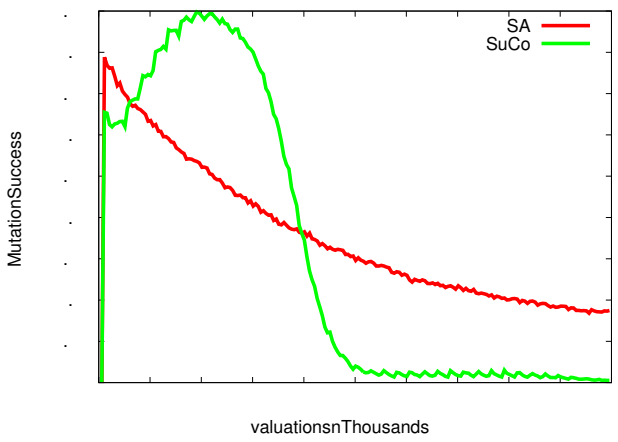

Figure 8: Mutation success rate with respect to evaluations, Shifted Rastrigin N=10

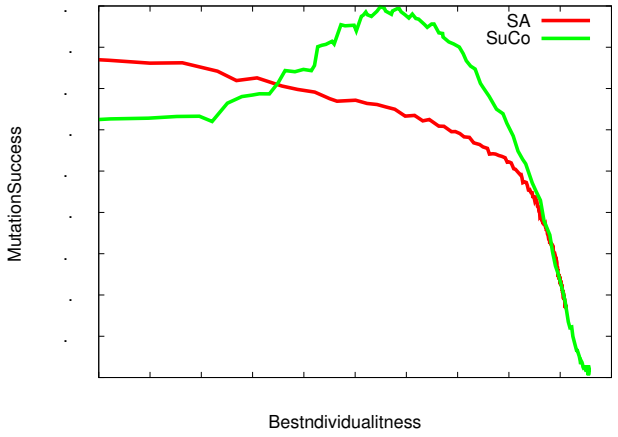

Figure 9: Mutation success rate with respect to best individual fitness, Shifted Rastrigin N=10

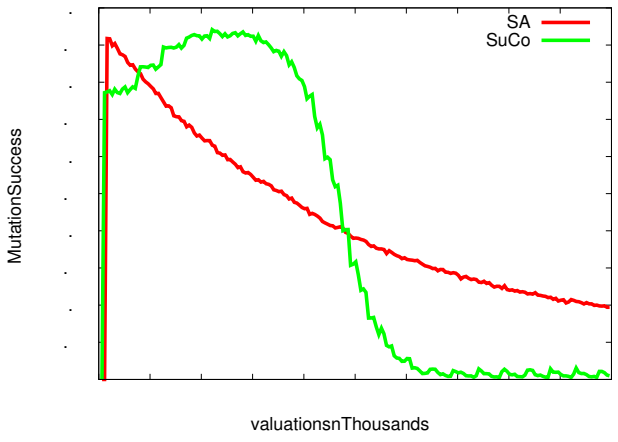

Figure 10: Mutation success rate with respect to evaluations, Shifted Rastrigin N=20

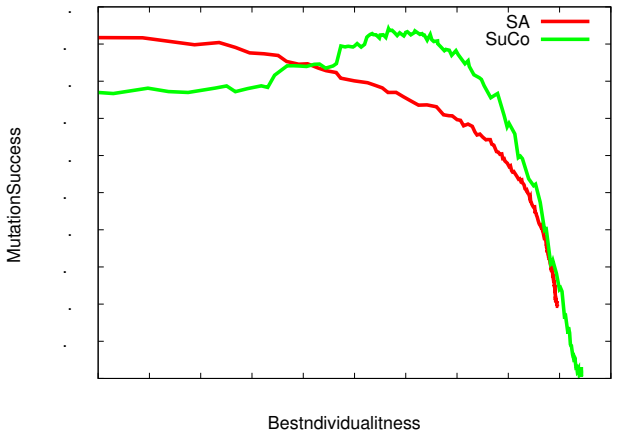

Figure 11: Mutation success rate with respect to best individual fitness, Shifted Rastrigin N=20

which is likely caused by it having a larger maximum mutation step size allowed than SA. Yet as evolution progresses, SuCo is able to achieve success rates greater than or equal to SA at similar levels of fitness. In respect to evaluations, SuCo's apparent collapse after 40 thousand evaluations is likely caused by a similar jump in population fitness quality, as discussed in Section 6.2.

The same behavior can also be observed on Shifted Rastrigin N=20 in Figure 10 and Figure 11. Again SuCo is able to improve its mutation success rate as evolution improves, with high performance maintained far into evolution. This is in comparison with SA, which does not appear to improve at all with evolution on this problem.

A likely cause for this change in behavior on Shifted Rastrigin is the inherent difference of the problem. Since the global optima for normal Rastrigin are artificially placed at the dead center of the search space, the arithmetic crossover used has a high probability of generating individuals near the global optima. This makes intuitive sense when you consider that if you generate two numbers randomly on some range and then average them, you would expect the result to be close to the center of the range. When the global optima is shifted randomly, this property breaks down, making evolution rely on mutation. As a result, Shifted Rastrigin requires a more powerful mutation operator than the normal Rastrigin problem, and is therefore able to better demonstrate the capabilities of SuCo.

7. CONCLUSION

Supportive Coevolution (SuCo) utilizes a multiple-species, multiple-population coevolving system in which *Support* individuals evolve better methods for *Primary* individuals to evolve problem specific solutions. While the primary focus of this paper is to introduce the broad concept of SuCo, an example SuCo was compared against self-adaptation of n uncorrelated mutation step sizes (SA) on the Rastrigin and Shifted Rastrigin benchmark problems. In all but one of the tests performed, SuCo was able to statistically outperform SA, with the exception likely caused by poor tuning or the inherent artificial properties of the Rastrigin problem.

To better understand the advantages of SuCo, analysis was performed of the instantaneous effectiveness of the evolv-

ed mutation step sizes. In general, SuCo was able to create more successful mutation step sizes than SA, especially as the population begins to converge to higher fitness individuals.

8. FUTURE WORK

While these results are in no way comprehensive, they suggest that the general concept of SuCo is promising. Further work needs to be performed to test the claims made about SuCo's ability to evolve multiple *Support* populations at once, as well as its ability to handle parameters that are less commonly evolved such as population sizing and selective pressure parameters.

The problem of *Support* population configuration will need to be addressed. While it is likely that the newly created configuration parameters for the *Support* population will have less of an impact than the mutation size itself, that analysis needs to be performed. Another interesting question is if different *Support* populations can utilize the same configurations, or if each will require its own configuration. In either case, hopefully SuCo's ability to dynamically configure the *Primary* population will continue to outweigh the impact of the *Support* configuration.

On the topic of configuration, some of the experiments performed here raised the suspicion that improper configuration led to poor results. While REVAC was performed to try and mitigate this problem, more extensive and effective tuning may help determine the true potential of each algorithm.

While the intent of the SuCo testing was to provide a proof of concept, SuCo needs to be tested on a wider range of problems to fully understand its properties. There are an immense number of different problem classes to be tried and analyzed, and just as many existing techniques to be compared against. Applying SuCo to a real world problem and comparing against current best algorithms for dynamic configuration would help solidify SuCo's place from a practitioner's perspective.

Many of the mechanics created for the experiments performed (such as the method for calculating *Support* individual fitness for mutation step size) could use further investigation. These were created ad hoc and with little individual analysis. As one of the major advantages of SuCo is its ability to directly manipulate the fitness of the evolved genes, determining how much improvement you can receive by doing this is important. Similarly, methods of intelligently pairing *Support* individuals with *Primary* ones may help improve the success of SuCo. For example, a method of mate pairing similar to LIMP [5] or Endosymbiotic Evolution [9] may be applicable.

9. REFERENCES

[1] W. de Landgraaf, A. Eiben, and V. Nannen. Parameter Calibration Using Meta-Algorithms. *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 71–78, 2007.

[2] B. W. Goldman and D. R. Tauritz. Meta-Evolved Empirical Evidence of the Effectiveness of Dynamic Parameters. *Genetic and Evol. Comp. Conf.(GECCO-2011)*, pages 155–156, 2011.

[3] B. W. Goldman and D. R. Tauritz. Self-Configuring Crossover. *Genetic and Evol. Comp. Conf.(GECCO-2011)*, pages 575–582, 2011.

[4] J. Gomez. Self Adaptation of Operator Rates in Evolutionary Algorithms. In *Proceedings of GECCO 2010*, pages 162–173. Springer Berlin, Heidelberg, 2004.

[5] L. M. Guntly and D. R. Tauritz. Learning Individual Mating Preferences. *Genetic and Evol. Comp. Conf.(GECCO-2011)*, pages 1069–1076, 2011.

[6] N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. 9(2):159–195, 2001.

[7] W. D. Hillis. Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure. *Physica D 42*, pages 228–234, 1990.

[8] P. Husbands and F. Mill. Simulated Co-evolution as the Mechanism for Emergent Planning and Scheduling. *Fourth International Conference on Genetic Algorithms*, pages 264–270, 1991.

[9] J. Y. Kim, Y. Kim, and Y. K. Kim. An Endosymbiotic Evolutionary Algorithm for Optimization. *Applied Intelligence*, 15(2):117–130, 2001.

[10] H. Mühlenbein, D. Schomisch, and J. Born. The Parallel Genetic Algorithm as Function Optimizer. *Parallel Computing*, pages 619–632, 1991.

[11] G. Papa. Parameter-less Evolutionary Search. pages 1133–1134. Atlanta, GA, USA, 2008.

[12] P. Posik. Real parameter optimization using mutation step co-evolution. In *IEEE Congress on Evolutionary Computation*, pages 872–880, 2005.

[13] I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biolischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart, 1973.

[14] E. Smorodkina and D. Tauritz. Toward Automating EA configuration: the Parent Selection Stage. In *IEEE Congress on Evolutionary Computation*, pages 63–70, 2007.

[15] F. Vafaee, W. Xiao, P. Nelson, and C. Zhou. Adaptively Evolving Probabilities of Genetic Operators. *Machine Learning and Applications*, pages 292–299, 2008.