Using Supportive Coevolution to Evolve Self-Configuring Crossover

Nathaniel R. Kamrath
Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri, U.S.A.
nrkxwb@mst.edu

Brian W. Goldman
BEACON Center for the Study of Evolution in Action
Department Of Computer Science and Engineering
Michigan State University
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

Creating an Evolutionary Algorithm (EA) which is capable of automatically configuring itself and dynamically controlling its parameters is a challenging problem. However, solving this problem can reduce the amount of manual configuration required to implement an EA, allow the EA to be more adaptable, and produce better results on a range of problems without requiring problem specific tuning. Using Supportive Coevolution (SuCo) to evolve Self-Configuring Crossover (SCX) combines the automatic configuration technique of multiple populations from SuCo with the dynamic crossover operator creation and evolution of SCX.

This paper reports an empirical comparison and analysis of several different combinations of mutation and crossover techniques including SuCo and SCX. The Rosenbrock, Rastrigin, and Offset Rastrigin benchmark problems were selected for testing purposes. The benefits and drawbacks of self-adaptation and evolution of SCX are also discussed. SuCo of mutation step sizes and SCX operators produced results that were at least as good as previous work, and some experiments produced results that were significantly better.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.6 [**Artificial Intelligence**]: Learning—*parameter learning*; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program modification, program synthesis*

General Terms

Algorithms

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.

Keywords

Coevolution, Dynamic Crossover, Linear Genetic Programming, Parameter Control, Self-Adaptation, Self-Configuration, Supportive Coevolution

1. INTRODUCTION

The performance of most Evolutionary Algorithms (EAs) is strongly correlated with the configuration of many parameters and operators [11]. The optimal configuration of an EA is typically problem specific and usually requires the knowledge and skills of an expert to obtain [9]. Furthermore, it is likely that the optimal parameter values and operators change throughout the run of an EA [2]. One solution to these problems is an EA which can configure itself and optimally adjust its parameters and operators throughout its run. This would help reduce the need for an expert while reducing the need for parameter tuning, giving the EA the potential to adapt to different problems and perform well without problem specific tuning. To address the previously stated issues, this paper introduces a novel EA which employs Supportive Coevolution (SuCo) to evolve Self-Configuring Crossover (SCX).

The evolution of SCX by means of SuCo creates an EA which can benefit from the strengths of both techniques. SuCo allows for the evolution of parameters and operators in support populations along with the primary solution population. The support populations supply the primary population with the best parameters and operators found at that point. When SCX is added to SuCo in a support population of crossover operators, more flexibility is added and the need for configuration is further ameliorated.

2. BACKGROUND

Previous work has been performed on automatic parameter and operator creation and evolution. In [12], the automatic creation of Genetic Algorithm (GA) mutation operators was discussed. [8] implemented control methods for all the parameters in an EA. This method was more generally applicable than less adaptive EAs; however, it could not achieve the same solution quality as a correctly tuned EA on specific problems.

Self-adaptation of EA parameters has also previously been investigated; in [5, 10] the offspring operators were con-

trolled. However, self-adaptation has limitations. When a suboptimal mutation method randomly creates a single high quality individual, there is no guarantee that the self-adapted operator will create another high quality individual. This individual may have difficulty creating high-quality offspring of its own due to its self-adapted genes. Thus, a method is needed to separate the operator's fitness from the individual's fitness.

2.1 Supportive Coevolution

SuCo is a method of automatic, self configuration which consists of a primary population, and any number of support populations [4]. The primary population is the population of solution candidates that are evaluated by the target fitness function identically to a traditional EA. The support populations are composed of individuals which represent parameters and operators. Support populations give SuCo the ability to separate operator fitness from individual fitness. All primary and support populations are evolved simultaneously, and, when the EA needs operators and parameters to create new primary individual offspring, a random individual from each support population is chosen to create the offspring.

The evolution of the support individuals throughout a run gives the EA the ability to configure itself. Based on the fitnesses of the support individuals, the EA can determine which parameters and operators are performing best at the current state of the EA. The support individuals who survive are then allowed to create offspring and continue the evolution of their population in an attempt to continue to find the optimal parameters and operators. As in [4], support individual fitness assignment is based on two factors: the fitness difference between parents and child and the genetic difference between parents and child. Equation 1 defines the fitness difference between parents and their offspring.

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

Using the relative fitness improvement between child and parent fitness as a way to rate the effectiveness of a support individual should be combined with a way of rewarding genetic diversity between parents and offspring to help prevent convergence. Doing so reduces the chance of evolving low risk operators which minimize genetic changes to reduce the chance of making detrimental changes. Scaling the relative improvement by the relative distance between parents and their child encourages genetic exploration. Equation 2 defines the relative distance between parents and their offspring.

$$RelativeDistance = \frac{distance(o, p_1) \cdot distance(o, p_2)}{distance(p_1, p_2)} \quad (2)$$

This equation is a simplification for two parent systems of the equation used in [4]. The distance referred to by Equation 2 is defined as:

$$\sum_{i=1}^{n} \left(firstGenes_i - secondGenes_i\right)^2 \quad (3)$$

where $firstGenes$ are the genes of the first individual and $secondGenes$ are the genes of the second individual. Assigning a support individual's fitness based on the product of relative fitness and relative distance rewards support individuals for creating primary offspring which are more fit and genetically dissimilar to their parents.

SuCo, thus far, has only been used to evolve mutation step sizes. Mutation step size individuals were represented as lists of floating point numbers which contained the same number of genes as a primary individual. Then, when mutation was performed, each mutation step size gene was used to control a Gaussian mutation which was added to the corresponding locus in the primary individual's genes. SuCo has the potential to evolve more than a single parameter. Since it has been shown that evolving a parameter with SuCo produces promising results [4], this paper describes research aimed at discovering if evolving multiple populations of support individuals by means of SuCo is a feasible approach to further automating EA configuration. Additionally, this research has another novel aspect. As the evolution of an entire operator through SuCo has yet to be explored, evolving crossover operators is also a novel extension of previous work.

2.2 Self-Configuring Crossover

With the ability of SuCo to evolve support populations of parameters and operators, it would be optimal to create support populations of individuals which are designed to be easily evolvable and highly expressive. The operator support individuals should be evolvable by standard EA techniques. With operators, this task is slightly more complicated. [1] employed a Meta-Genetic Program (Meta-GP) to evolve crossover operators. However, this method is unable to adapt within an EA run and is extremely computationally expensive. SCX is a method of automatic crossover creation and evolution which is computationally feasible and can be easily evolved by an EA [3].

SCX is encoded employing a linear structure similar to Linear Genetic Programming (LGP) and composed of a list of primitive functions. The first primitive function, Swap, was designed to represent crossovers that move genetic information between parents and between positions in a single parent such as n-point crossover, uniform crossover, and most permutation based crossovers. The second primitive function, Merge, was designed to represent crossovers that create genetic material by combining genes in a manner similar to arithmetic crossover. Both primitives use three parameters. The first two parameters correspond to the gene locations (in either parent) where the operation will occur. The final parameter depends on the primitive function. Swap's final parameter is a width which determines how many genes after each gene location should be swapped. This allows for blocks of consecutive genetic information to remain together. Merge's final parameter is a weight. This weight is used to combine the value at each selected location which is represented by the first two parameters in the Merge primitive. Equation 4 shows how Merge combines two values using a weighted average where $g(i)$ is the gene at the fist position, $g(j)$ is the gene at the second position, and α is the weight.

$$g(i) = \alpha \cdot g(i) + (1 - \alpha) \cdot g(j) \quad (4)$$

When using SCX to perform a crossover, the genes of both parents are first concatenated. Then, the copy of the genes is modified by applying each primitive in the SCX individual in sequence. After each primitive has been applied to the list of genes, half the combined genome is used to create an offspring. To allow for further dynamic behavior, each parameter in a primitive can be set in multiple ways. The first is the Number construct, which uses a static value set at primitive creation every time the primitive is used. The

second is the Random construct which creates a new random number every time the primitive is used. The final construct is the Inline construct. Inline forces the genetic operations to be performed between the same gene locations in each parent. See Figure 2 for an example of applying SCX. In previous work, SCX was evolved using techniques similar to self-adaptation. This was achieved by giving every primary solution individual its own crossover operator. Thus, there was no separate population for the SCX operators.

SCX has several qualities which make it attractive as a candidate for an operator in SuCo. First, SCX reduces the search space of crossover operators by using a structure similar to LGP and by basing its functionality on primitives. Since the search space of crossover operators is very large and the operators need to be evolved within a single run of the EA, the reduction of search space is vital to the search for optimal crossover operators. Second, even with the reduced search space, SCX still has the potential to be extremely dynamic. Its size, primitive combination, parameters, and even the way the parameters are defined, are all dynamic properties of SCX. Lastly, SCX's structure makes it easy to evolve using standard EA operations. With these reasons in mind, and the fact that SCX has shown promising results [3], SCX has been chosen as the dynamic crossover operator to be evolved by SuCo in the research presented in this paper.

3. METHODOLOGY

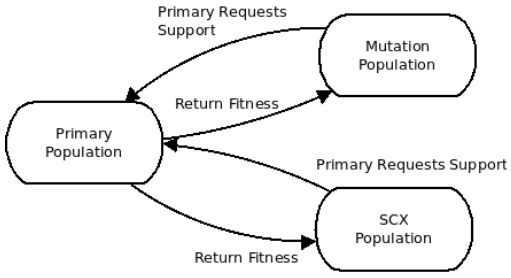

Figure 1: Suco Primary and Support Population Interaction

SuCo was implemented with two support populations: one for mutation step sizes and one for crossover operators as shown in Figure 1. The mutation step size individuals were represented using lists of floating point numbers. The crossover individuals were implemented using SCX. When the EA starts, each support population is randomly initialized along with the primary population. After initialization, the creation of each new individual requests a crossover operator and a mutation operator from the support populations. The SCX operator creates an offspring from two parents. Then, the offspring is mutated by applying the mutation step sizes from each of the mutation support individual genes to the offspring genes at the same locus. Support individuals are chosen randomly and the same support individual combinations are prevented to ensure different combinations of support individuals are evaluated. Figure 2 demonstrates the production of offspring using the described method.

Once the new primary individual is created and evaluated, the support individuals that were requested for its creation are assigned the product of Equation 1 and Equation 2 as fitness. This encourages genetic exploration while awarding fitness improvements. Each support individual can be evaluated multiple times to more accurately assess its true potential to generate quality offspring.

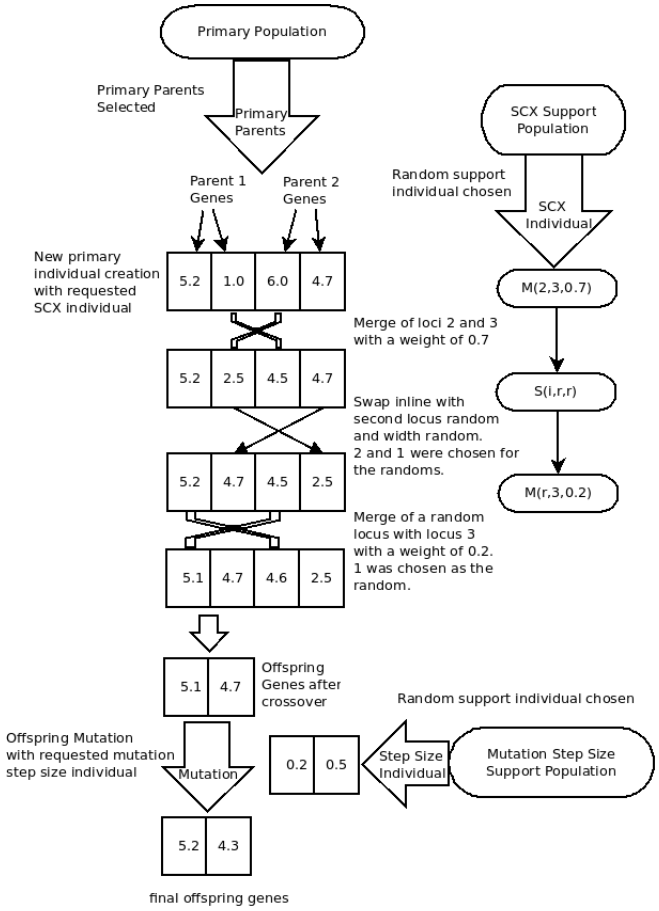

Figure 2: SCX and Mutation Support Individuals in Offspring Creation

Two adaptations were made in this research; the first to SuCo and the second to SCX. SuCo had to be adapted to allow one of its support populations to be configured for SCX individuals. Since SCX was originally self-adapted as part of other individuals, it was slightly modified to allow for evolutionary techniques to be executed on each individual.

The SuCo adaptations were required because a support population for a crossover operator had not been previously explored. A similar approach to the mutation step size population was adopted. Individuals were randomly initialized at the start and given to the primary population when requested by the EA throughout the run. k-tournament selection with replacement was used for parent selection and without replacement for survivor selection. The mutation and crossover methods for this support population were handled by standard SCX methodologies as specified in [3].

Since all SCX operators are being evolved in their own population, a way to perform crossover on two separate primary individuals with one crossover operator was devised. The functionality is very similar to the original SCX method. Both parents' genes are first copied and concatenated. Then, all the primitive functions of the SCX support individual are applied to the copied genes. Once this is complete, half the

Parameter	Rastrigin	Offset Rastrigin	Rosenbrock
A	10	10	100
Genome Size	100	100	50
Evaluations	10,000	100,000	100,000
Population	172	418 (37)	355 (200)
Offspring	5	6 (50)	2
Mutation Rate	0.009838	0.009738	0.213557
Mutation Step	1.47144	0.587874	0.033593
Parent Tournament	114	172 (35)	339 (139)
Survival Tournament	63	421 (27)	344 (190)
Evals Per Generation	9	1	10

Table 1: Experiment parameters used for testing with novel EA parameters in parentheses

genetic material in the resulting list of genes is used to create an offspring. This is different from the standard SCX implementation in that it employs a single SCX individual from a separate population to perform the crossover operation.

Since both crossover and mutation are being evolved, it can become difficult to determine what to attribute the fitness of an individual to: crossover, mutation, its parents, or random chance. For many techniques, this is difficult to determine. However, SuCo explicitly attempts to separate these factors by normalizing for parent fitness, repeatedly using the same operator to adjust for noise, and using operators in different combinations to assign operator fitness independent of the other operators it is paired with. Also, since SCX is evolving in a support population as opposed to being self-adapted, there are several potential benefits as well as several potential disadvantages.

One possible disadvantage is that a crossover operator may work well with only one specific primary individual and perform poorly with all other primary individuals. This could potentially lead to the production of unfit individuals. However, evolving SCX in a support population allows the primary population to have access to fit crossover operators instead of just one crossover operator which it is attached to as in self-adapted SCX. Another benefit from SuCo of SCX is it allows the fitness of the operator to be decoupled from the primary individual's fitness. Thus, the operator can receive its own fitness which is indicative of that operator's ability to produce quality, genetically diverse offspring. This allows for better evolution of crossover individuals and a better SCX operator population from which the primary population can draw.

4. EXPERIMENTAL SETUP

The main goal of this research is to determine if SuCo is a feasible way for EAs to perform self-configuration and evolve their own parameters and operators. Thus, results and analysis strictly based on fitness are not the only product of these tests. They will serve as a measure of the potential of SuCo when employed to evolve multiple support populations as an indication for future work.

The base settings for the parameters are shown in Table 1. These parameters were taken from [3] which implemented a meta-EA to search for optimal parameters. However, some basic hand tuning for Offset Rastrigin and Rosenbrock was performed on the EA which evolves both mutation step sizes and SCX operators since this combination has never been tested before. This EA's parameters are shown in parentheses where different. All EAs used the same parameters for the Rastrigin problem. Since the goal of this research is to reduce the amount of parameter setting required by EAs, all support population parameters were identical to the primary population's parameters. Further analysis of the parameters used is available in Section 6.2.

Several benchmark problems were employed to test performance and produce results for analysis and comparison. The following test problems were selected to match previous work and allow for backward comparisons. For a real-valued, highly multimodal problem with no gene interdependence, the Rastrigin Function was chosen [6]. The Rastrigin Function is given in Equation 5:

$$An + \sum_{i=1}^{n} [x_i^2 - Acos(2\pi x_i)] \; \forall x \; \in \; [-5.12, 5.12] \qquad (5)$$

The Rosenbrock problem [7], as defined in Equation 6, was used to represent real-valued problems with gene interdependence.

$$\sum_{i=1}^{n} [(1 - x_i)^2 + A(x_{i+1} - x_i^2)^2] \; \forall x \; \in \; [-5, 10] \qquad (6)$$

In both of these problems, the optimum value for each gene is the same. Thus, the Offset Rastrigin problem, introduced in [3], was implemented as the final real-valued benchmark problem to test the EA's ability to solve problems where not every gene has the same optimum value. The Offset Rastrigin problem is a specially designed modification of the Rastrigin problem which, at the start of every run, generates an offset vector O. Every element in O is randomly assigned a value from [-2.5, 2.5] at intervals of 0.5. In the Rastrigin function, if x_i is a local best, $x_i + .5$ is a local worst, and if x_i is near the global optimum, then $x_i + 1$ will be near a local optimum. This makes the problem much harder for SCX since simply swapping a gene which is optimal in one position with a gene in another position has a high probability of resulting in a poor fitness value.

Several combinations of mutation and crossover were implemented for testing. First, a static mutation step size combined with arithmetic crossover (static + arithmetic) was implemented as a base line. Next, the following combinations were implemented for empirical comparisons: static mutation step size with self-adapted SCX (static + SA-SCX), SuCo of mutation step sizes with arithmetic crossover (SuCo mut + arithmetic), static mutation step size with SuCo of SCX (static + SuCo SCX), SuCo of mutation step sizes with self-adapted SCX (SuCo mut + SA-SCX), and SuCo of both mutation step sizes and SCX (SuCo mut +

	Static Mutation	SuCo Mutation
Arithmetic	-55.925 (6.4379)	-61.249 (1.5353)
SA-SCX	-0.0072 (0.00465)	-0.01711 (0.00690)
SuCo SCX	-0.00051 (.00233)	-0.00025 (0.00078)

Table 2: Rastrigin Results: mean final best fitness with standard deviation in parentheses

	Static Mutation	SuCo Mutation
Arithmetic	-0.12568 (0.18911)	-0.15197 (0.19091)
SA-SCX	-0.02718 (0.01818)	-0.02366 (0.01194)
SuCo SCX	-0.30794 (0.46271)	-0.02579 (0.01022)

Table 3: Offset Rastrigin Results: mean final best fitness with standard deviation in parentheses

SuCo SCX). These combinations were picked to explore all possible combinations for static or evolved mutation step sizes with static, self-adapted, or evolved SCX operators.

Since none of the new techniques have significant asymptotic complexities, all experiments rely on counting evaluations as the measure of search efficiency. Experiments with SuCo count only evaluations of the primary population as the primary individuals are the only individuals who make calls to the fitness function. Support individuals are assigned a fitness value based on the results of the primary individual's fitness evaluation which means support individuals do not need to be directly evaluated.

A complete package containing our results as well as the source code and configuration files used in testing is available from our website[1].

5. RESULTS

The results comparing the different combinations of mutation step size parameter and crossover operator implementation are presented in Table 2, Table 3, and Table 4. In these tables, each column represents the mutation method used (labelled by the first entry in the column) and each row represents the crossover method used (labelled by the first entry in each row). SA-SCX represents self-adapted SCX, SuCo SCX represents SuCo evolved SCX, and SuCo Mutation represents SuCo evolved mutation step sizes. Each experiment shows the mean final best fitness and standard deviation found over 30 runs. A series of Mann-Whitney U tests were performed comparing the results from SuCo mut + SuCo SCX to the results from all other combinations. SuCo mut + SuCo SCX was found to be the best combination on the Rastrigin and Rosenbrock problems. On the Offset Rastrigin problem, there was no statistical difference between SuCo mut + SuCo SCX and SuCo mut + SA-SCX. A confidence level of $\alpha = 0.01$ was used in the statistical analysis.

6. DISCUSSION

The results from the tests in Section 5 show that the SuCo mutation + SuCo SCX EA performed well on the Rastrigin and Rosenbrock problems. This was expected since allowing SCX to evolve in a support population allows for a better evaluation of SCX operators and gives the primary population access to higher quality crossover operators. However,

[1]`https://github.com/natekamrath/SuCo-SCX`

	Static Mutation	SuCo Mutation
Arithmetic	-71.873 (39.489)	-85.468 (41.840)
SA-SCX	-28.999 (22.847)	-24.444 (23.446)
SuCo SCX	-22.638 (23.201)	-17.941 (22.824)

Table 4: Rosenbrock Results: mean final best fitness with standard deviation in parentheses

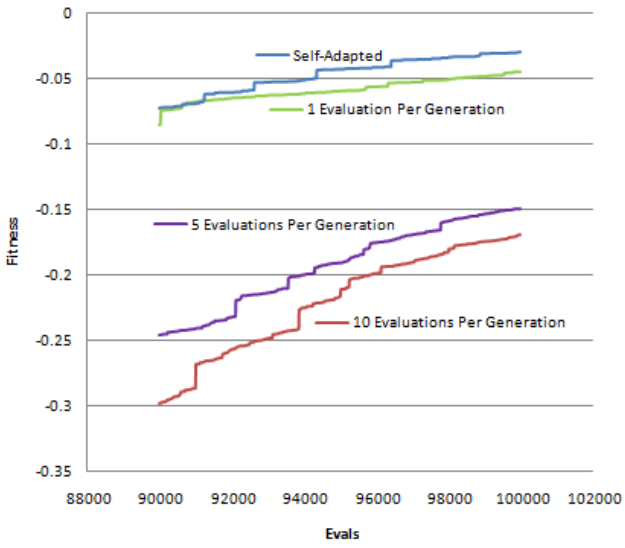

Figure 3: Fitness vs Evals for Different Numbers of Evaluations per Generation - 100,000 Evals

it is interesting that the SuCo mutation + SuCo SCX EA did not perform as well on the Offset Rastrigin problem. This could be because of its increased complexity which is difficult for SCX to overcome since different genes have different optimum values and there is no gene interdependence. Further explanation is provided in the following sections.

It is also interesting to note that one evaluation per generation was used for the Offset Rastrigin problem. Normally, allowing for multiple evaluations per generation helps to determine the true quality of support individuals. This allows them to be evaluated in different combinations and with different primary individual parents to ensure that one bad combination does not cause a potentially fit support individual to be removed from the support population. However, it was found that performing one evaluation per generation was more beneficial on the Offset Rastrigin problem. In order to determine why, data from the Offset Rastrigin problem was further analyzed.

6.1 Evolution Versus Self Adaptation of SCX

Figure 3 shows mean best fitness versus the number of evaluations for the last 10,000 evaluations of several 100,000 evaluation runs. These values were obtained by averaging values from the Offset Rastrigin problem over fifty runs for each different setting shown. The higher number of evaluations per generation produces large, sudden increases in fitness followed by periods of small improvement. The increases in fitness are more extreme than the graph may depict as the results are smoothed. The described changes in value indicate that either multiple runs all improve at identical evaluation numbers, or individual runs increased dra-

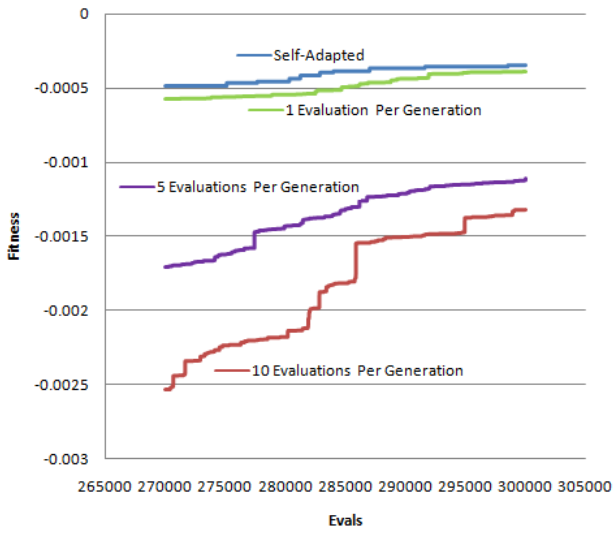

Figure 4: Fitness vs Evals for Different Numbers of Evaluations per Generation - 300,000 Evals

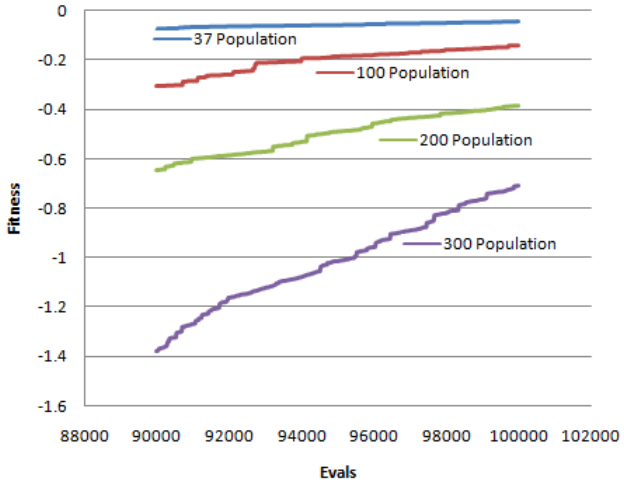

Figure 5: Fitness vs Evals for Different SuCo mut + SuCo SCX Population Sizes - 100,000 Evals

matically. As the number of evaluations per generation becomes smaller, the sudden increases in fitness happen more often, but have a smaller magnitude in change. As a result, support individuals are evolved more quickly and can adapt to the changing primary individuals faster. This creates the more constant, consistent fitness improvement curve observed with small numbers of evaluations per generation.

When observing the increase in fitness of the higher number of evaluations per generation, it would appear as though it would surpass the smaller number of evaluations per generation given more evaluations. However, Figure 4 shows the last 30,000 evaluations of a run which was allowed to continue to 300,000 evaluations. The data for Figure 4 was collected in the same way as the data for Figure 3, but with only 10 runs per experiment and the number of evaluations parameter varied. It is easy to see that the higher number of evaluations per generation still has not surpassed the lower evaluations per generation. This is because the large increases in fitness following an evolutionary cycle of the support populations become increasingly smaller and the period between evolutionary cycles of the support populations also yields significantly less improvements in fitness as the EA progresses.

As shown in Figure 3 and Figure 4, it was found that allowing for multiple evaluations per generation through SuCo mut + SA-SCX yielded only marginally different results from static + SA-SCX. It was also observed that SuCo mut + SA-SCX performed better than SuCo mut + SuCo SCX when using the same parameters and multiple evaluations per generation. However, when only one evaluation per generation was used, SuCo mut + SuCo SCX performance improved. This can be attributed to the fact that self-adapting SCX always uses one evaluation per generation. SCX is never explicitly evaluated when being self-adapted because each operator belongs to a solution individual. Since each solution individual is only evaluated once per generation, each SCX operator is as well. When being evolved and evaluated multiple times per generation, it takes more evaluations to produce the same number of new support individuals as compared to self-adaptation. On a more complex problem, such as Offset Rastrigin, it may be easier to determine which SCX operators are more fit. Thus, one evaluation per generation of each support individual is sufficient to accurately determine the fitness of that support individual. It is also possible that one evaluation per generation allows the support individuals to evolve more quickly and continue to supply the primary population with new SCX operators. Both explanations could be reasons why SuCo performed better when performing only a single evaluation of each support individual per generation on Offset Rastrigin.

This demonstrates one of the benefits of using SuCo to evolve SCX in a support population as opposed to self-adapting SCX. On problems where complexity requires many evolutionary cycles to produce crossover operators which are highly developed, evolving SCX can mimic self-adapting SCX, in both behavior and performance, by adjusting the evaluations per generation. This can be seen in Figure 3 where the results from one evaluation per generation are very similar to the results from self-adaptation. However, on simple problems where the complexity is not necessary, evolved SCX operators can be evaluated multiple times per generation to truly assess the quality of more simplistic operators and produce more fit individuals. This is evident in the Rosenbrock results presented in Table 4, and even more evident in the Rastrigin results presented in Table 2. Clearly, evolving SCX operators in support populations increases the adaptability of an EA and allows it to perform more optimally over a wider range of problems.

6.2 Population Size when Evolving Multiple Support Populations

Another interesting result of evolving multiple support populations is observed when considering the population sizes used on the Offset Rastrigin and Rosenbrock problems. When SuCo was used to evolve both mutation step sizes and SCX operators, a smaller population size produced better results. This could partially be due to the ability of SuCo to produce high quality individuals from limited genetic material and the potential of SuCo to explore genetically. Since

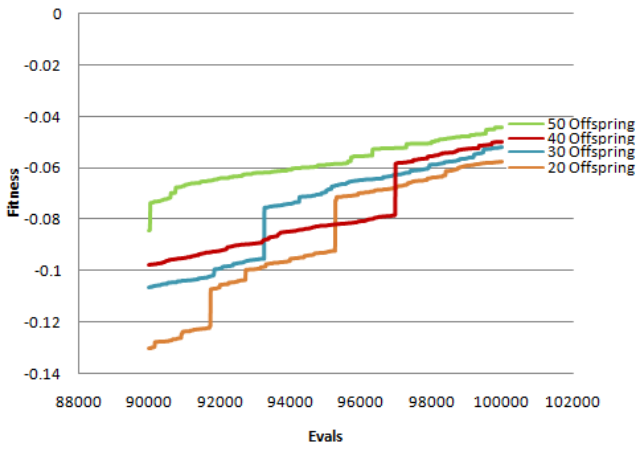

Figure 6: Fitness vs Evals for Different SuCo mut + SuCo SCX Offspring Sizes - 100,000 Evals

the parameters and operators are evolving to allow for a better search of the primary individual search space, less unique genetic material is needed in the primary population. The evolved parameters and operators can find new genetic material more easily which means unique genetic material does not have to be retained by individuals in the primary population. This was demonstrated by the configuration settings for the SuCo EA which evolved both mutation step sizes and SCX operators. Figure 5 shows the last 10,000 evaluations of several 100,000 evaluation runs where different population sizes were implemented. Each different population size experiment was performed fifty times and then averaged to produce the data shown in Figure 5.

Also, a higher number of offspring produced in each generation was preferred. This could be caused by the previously discussed effect which allows evolved parameters and operators to find new genetic material more easily. Another contributing factor could be that, since support individuals are encouraged to create genetic diversity, gene saturation induced by the large number of offspring produced is prevented. Figure 6 shows the last 10,000 evaluations of several 100,000 evaluation runs where different offspring sizes were implemented. The data for Figure 6 was collected in the same way as the data for Figure 5 with only the offspring size parameter being varied.

Another possible explanation for the small population size is the fact that the support populations all used parameters identical to the primary population. This method of setting parameters may not be optimal. The parameters observed could simply be the best parameters when considering all populations as a whole. If the support populations were each configured separately, different parameters may be found to be more optimal. This could also increase performance as the optimal configurations for each population could be found.

7. CONCLUSION

Proper EA configuration is a challenging task which often requires the experience of an expert. However, there are methods which allow an EA to configure itself optimally. An even more challenging problem is allowing an EA to reconfigure its parameters throughout its run. This ability allows the EA to adapt its parameters and operators in an attempt to find optimal settings. SuCo and SCX are both methods which allow for EA self-configuration and dynamic parameter control.

To build on previous work and determine if adding more parameters and operators to support populations in SuCo is beneficial, a new support population composed of SCX individuals was introduced to SuCo. It was determined that evolving SCX through SuCo as opposed to self-adaptation can further add flexibility to the EA. This is due to the fact that evolved SCX can operate in a mode functionally similar to self-adaptation or completely different depending on what is more optimal for the current problem. SuCo was shown to be not only a feasible way of evolving multiple parameters and operators, but also a promising way of doing so. Using SuCo to evolve multiple populations of parameters and operators has the potential to further reduce EA configuration needs, allow for dynamic parameter control, and produce high quality results.

8. FUTURE WORK

While the results shown here are promising, there is still a large amount of work which needs to be done. More research needs to be performed on the effects of adding additional parameters and operators to SuCo in support populations. Operators which are suitable for SuCo need to be empirically compared in order to determine which work well as support individuals. New methods of operator evolution may need to be investigated to create methods which lend themselves to being evolved in support populations. Methods to dynamically control population size also need to be researched. Automated methods for setting support population parameters and evaluations per generation need to be explored. The way fitness values are assigned to support individuals also needs more work. The empirical comparison of self-adaptation and evolution of SCX operators needs to be expanded on to further explain the benefits and drawbacks of each.

On a larger scale, SuCo needs to be compared to other methods of automatic configuration and dynamic parameter control. Performance on real-world problems needs to be studied and parameter sensitivity also needs further research.

9. REFERENCES

[1] L. Dioşan and M. Oltean. Evolving crossover operators for function optimization. In *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 97–108, Budapest, Hungary, 10 - 12 Apr. 2006. Springer.

[2] B. W. Goldman and D. R. Tauritz. Meta-Evolved Empirical Evidence of the Effectiveness of Dynamic Parameters. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '11)*, pages 155–156, Dublin, Ireland, July 2011.

[3] B. W. Goldman and D. R. Tauritz. Self-Configuring Crossover. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '11)*, pages 575–582, Dublin, Ireland, July 2011.

[4] B. W. Goldman and D. R. Tauritz. Supportive Coevolution. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '12)*, pages 59–66, Philadelphia, Pennsylvania, USA, July 2012.

[5] J. Gomez. Self Adaptation of Operator Rates in Evolutionary Algorithms. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC '04)*, pages 1720–1726, Portland, Oregon, USA, 20-23 June 2004. IEEE Press.

[6] D. S. H. Mühlenbein and J. Born. The Parallel Genetic Algorithm as Function Optimizer. *Parallel Computing*, 17(6–7):619–632, Sept. 1991.

[7] K. A. D. Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[8] G. Papa. Parameter-less Evolutionary Search. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1133–1134, Atlanta, Georgia, USA, 12-16 July 2008. ACM.

[9] E. A. Smorodkina and D. R. Tauritz. Toward Automating EA Configuration: the Parent Selection State. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC'07)*, pages 63–70, Singapore, Sept. 2007.

[10] F. Vafaee, W. Xiao, P. C. Nelson, and C. Zhou. Adaptively evolving probabilities of genetic operators. In *Proceedings of the 7th International Conference on Machine Learning and Applications (ICMLA '08)*, pages 292–299, La Jolla, San Diego, USA, 11-13 Dec. 2008. IEEE.

[11] A. E. W. de Landgraaf and V. Nannen. Parameter Calibration Using Meta-Algorithms. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC'07)*, pages 71–78, Singapore, Sept. 2007.

[12] J. R. Woodward and J. Swan. The Automatic Generation of Mutation Operators for Genetic Algorithms. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '12)*, pages 67–74, Philadelphia, Pennsylvania, USA, July 2012. ACM.