Apr 20th, 11:00 AM - 12:00 PM

Elitist Schema Overlays: A Multi-Parent Genetic Operator

Nick Nichols
Illinois Wesleyan University

Mark Liffiton, Faculty Advisor
Illinois Wesleyan University

Elitist Schema Overlays: A Multi-Parent Genetic Operator

Nick A. Nichols
Illinois Wesleyan University
Bloomington, IL 61701 USA
nnichols@iwu.edu

April 26, 2013

Abstract

Genetic Algorithms are programs inspired by natural evolution used to solve difficult problems in Mathematics and Computer Science. The theoretical foundations of Genetic Algorithms, the schema theorem and the building-block hypothesis, state that the success of Genetic Algorithms stems from the propagation of fit genetic subsequences. Multi-parent operators were shown to increase the performance of Genetic Algorithms by increasing the disruptivity of genetic operations. Disruptive genetic operators help prevent suboptimal genetic sequences from propagating into future generations, which leads to an improved fitness for the population over time. In this paper we explore the use of a novel multi-parent genetic operator, the *elitist schema overlay*, which propagates the matching segments in the genetic sequences of the elite subpopulation to bias the global search towards the best known solutions. We investigate the parameters that drive the behavior of elitist schema overlays to determine the most successful model, and we compare this to successful multi-parent and traditional genetic operators from the literature. Both elitist schema overlays and multi-parent Genetic Algorithms were found to perform better than traditional Genetic Algorithms in our experiments, but elitist schema overlays degraded performance when used in conjunction with other multi-parent Genetic Algorithms.

1 Introduction

Optimization problems are some of the most difficult problems to solve exactly; however, checking how good a particular solution is usually involves a fairly inexpensive computation. Thus, a natural strategy for problems like this would be to generate a large, random pool of solutions and use information from a quality check to find better solutions. This is central to how Genetic Algorithms operate. We can encode solutions to these problems as genetic sequences and

then replicate the process of natural evolution on them. The quality checks, performed by the *fitness function*, let us score a solution's fitness, or quality. The *selection* operator uses these scores to determine which solutions performed well and should be kept for additional processing and those that did poorly and need to be discarded. We then derive additional potential solutions from those that were kept, which is done by a *recombination* operator. Finally, it is helpful to change small components in these new solutions to expand our search with a *mutation* operator [4].

This methodology lets us quickly generate approximate solutions to these problems. Genetic Algorithms are not guaranteed to find the optimum solution, but they are able to find good approximations very efficiently [22]. Much research has gone into improving these approximations by modifying the strategies and heuristics used, and this paper aims to do the same.

Historically, Genetic Algorithms have been modeled closely after observations from Biology. For example, both biological reproduction and recombination in traditional Genetic Algorithms always occur with either 1 or 2 parent(s) involved; however, this is only a restriction in Biology [10]. For Genetic Algorithms, the number of parents used during recombination can easily be expanded beyond 2. Papers describing techniques for multi-parent recombination started appearing as early as 1966, but little was reported about their behavior early on [6]. The strategies investigated showed promise, and they have since drawn more attention and research[8].

Curiously, many of these new operators were extensions of traditional recombination operators modified only to accommodate for more parents. In this paper, we follow the trend of diverging from the restrictions of Biology by introducing a new genetic operator, the *elitist schema overlay*, and the more general concept of a *genetic overlay*. This operator attempts to amplify the benefits of both crossover and fitness-based scanning by attempting to identify and propagate the genes correlated with the most successful solutions that have been discovered [22].

In the next section, we will define the core terminology of Genetic Algorithms. Section 3 covers traditional genetic operators and two of the most successful genetic operators from the literature, diagonal crossover, or diagonalization, and fitness-based scanning. In Section 4, definitions for both genetic overlays and elitist schema overlays are given. Additionally, we have also presented the case for why this methodology is successful and how it follows from Genetic Algorithms theory. In Section 5, we define the numerical optimization functions and NP-Hard problems that we will test against. The encoding formats and support functions used to compute these are also defined in that section. Section 6 details how our experiments were set up, and an analysis of the data from those experiments.

2 Terminology

Genetic Algorithm research borrows terms from both Biology and Computer Science, so we have set several conventions on terminology for this paper. A *Genetic Algorithm* is a local search technique that simulates biological evolution on solutions to a given problem [22]. A *genetic sequence* g_k, often also called an *individual*, is a sequence of values $a_1 a_2 \dots a_n$ that represents a potential solution to a particular problem. Each value a_i within a genetic sequence is referred to as an *allele* and can take on values from a specified domain D_i, commonly the set $\{0, 1\}$. A *population* refers to the set G whose elements are the genetic sequences for a given instance. A *schema*, pluralized as *schemata*, is a partial genetic sequence where each a_i can be left unspecified and is not required to take any value [22]. In this paper, the character '-' will represent a value that has been left unspecified, which is commonly called a *don't care* value [14].

Genetic Algorithms have two distinct phases, *recombination* and *selection*, which occur every iteration, also called a *generation*. During the selection phase, a *fitness function* takes a $g_k \in G$ as input and returns an $x \in \mathbb{R}$. This x, or *fitness value*, measures how well g_k solves a given problem instance. *Elitism* is a strategy that takes the individual with the highest fitness value in a population and copies it into the next generation.

Recombination utilizes *genetic operators*, which are functions from genetic sequences to genetic sequences. The probability of a particular genetic sequence being chosen for selection is typically determined by its fitness value. Genetic operators of *arity* n take n *parent* genetic sequences to produce m new, *child*, genetic sequences. It should be noted that m and n are two not necessarily distinct natural numbers. Historically, the phrase *mutation operator* has referred to the case where $n = 1$, *recombination operator* has specified $n = 2$, and *multi-parent recombination operator* was reserved for $n > 2$ [8]. These operators are defined in greater detail in the following section.

3 Current Genetic Operators

Traditional Genetic Operators

Genetic Algorithms were designed to emulate evolution by natural selection, and traditional genetic operators closely resemble their biological counterparts. Crossover, mutation, and selection build the core of many Genetic Algorithms, and are direct translations of biological processes into the computational world. [22]. For the rest of this paper, the term "traditional Genetic Algorithm" will be used to refer to 1-point crossover and mutation together.

1-Point Crossover

1-point crossover is the computational analog to sexual reproduction. This binary genetic operator takes two genetic sequences, and then chooses a splitting

point in their genetic sequences. In the example below, the splitting point is between the fourth and fifth alleles:

$$\text{Parent 1} : 0\ 1\ 1\ 0 \mid 1\ 0\ 1\ 1\ 0$$
$$\text{Parent 2} : 0\ 1\ 0\ 0 \mid 0\ 0\ 1\ 0\ 1$$

The alleles that occur after the splitting point are then swapped between the two parents to create two new child genetic sequences. The parents from the example above would produce the children below:

$$\text{Child 1} : 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1$$
$$\text{Child 2} : 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0$$

Uniform Mutation

Mutation is a unary genetic operator designed to help Genetic Algorithms break out of local optima [22]. This operator scans through a genetic sequence of length n, and at each point will change that allele to another value in that position's domain with a given probability p. This probability is typically set to $\frac{1}{n}$, but further research has demonstrated that a dynamically set p produces better behavior [2]. An example has been provided below:

$$\text{Before Mutation} : 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1$$
$$\text{After Mutation} : 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1$$

Swapping Mutation

Problems that require their genetic sequences to be permutations, like the Traveling Salesman Problem, utilize swapping, instead of uniform, mutation. Both operators scan through their respective genetic sequences, but the difference occurs when a change is made in the case of permutations. These mutation operators choose a random point in the rest of the genetic sequence and swap the values at these two locations. An example can be seen below:

$$\text{Before Mutation} : 8\ 4\ 2\ 1\ 9\ 7\ 3\ 5\ 6$$
$$\text{After Mutation} : 8\ 7\ 2\ 1\ 9\ 4\ 3\ 5\ 6$$

Steady-State Selection

Steady-state selection is used to ensure that underperforming genetic sequences are eliminated from the population, thus preventing their genes from propagating further. Once recombination and mutation have finished adding individuals

to the population for a particular generation, this operator assigns each individual a fitness value with the fitness function for the current problem. The individuals are then sorted in descending order with respect to their fitness values. The first k individuals in this list become the population operated upon in the next generation.

Multi-Parent Genetic Operators

The literature concerning multi-parent genetic operators highlighted two central operators: fitness-based scanning and diagonal crossover. These two strategies have shown success as multi-parent genetic operators, and are the direct conceptual descendants of traditional genetic operators. Both of these operators will be tested independently, as well as in conjunction with each other. This *mixed-operator* methodology has shown performance gains in previous research [25].

Fitness-Based Scanning

The first operator from the literature that has been implemented and tested is fitness-based scanning as proposed by Eiben, et al [8]. The generalized form of scanning iterates through the empty genetic sequence of a child and determines its value based upon the values present in n selected parents [5]. Fitness-based scanning makes a roulette wheel selection to choose each allele. At each allele, the probability that a parent, from the sub-population, consisting of the parent genetic sequences selected for this recombination, S, named i with a fitness value of $f(i)$, will donate its allele to the child's genetic sequence has probability $P(i)$ as described below [8]:

$$P(i) = \frac{f(i)}{\sum_{i \in S} f(i)}$$

Thus, the expected number of alleles inherited from a parent i is $E(i)$ [8]:

$$E(i) = P(i) * (\text{ Chromosome length })$$

In previous studies, fitness-based scanning had mixed results across several numeric optimization functions, and performed well on the Traveling Salesman Problem [8, 10]. Overall, fitness-based scanning managed to surpass the results of traditional crossover and other multi-parent genetic operators [8].

Diagonal Crossover

Diagonal crossover was introduced by Eiben, et al. to extend the concept of crossover into the realm of multi-parent genetic operators [6]. A diagonal crossover of arity n can be described easily. Take n individuals from a population, select $n-1$ crossover points, and create n children by selecting one subsequence from each piece of the genetic sequences given [10]. Figure 1 depicts

how this would work [10]. Note that a_i, b_i, c_i, and d_i are genetic subsequences of arbitrary length.

a_1	a_2	a_3	a_4	parent a
b_1	b_2	b_3	b_4	parent b
c_1	c_2	c_3	c_4	parent c
d_1	d_2	d_3	d_4	parent d

$\rightarrow$

a_1	d_2	c_3	b_4	child a
b_1	a_2	d_3	c_4	child b
c_1	b_2	a_3	d_4	child c
d_1	c_2	b_3	a_4	child d

Figure 1: Diagonal crossover applied to four parents

The rationale behind expanding this operator into the realm of multi-parenthood was to increase the disruptiveness, and by extension the exploratorivity, of traditional crossover [7]. This meant that the population would need a large degree of similar genetic sequences before the search would narrow and converge [10]. It should also be noted that in the special case of $n = 2$ is identical to traditional 1-point crossover [10]. High arity versions of this operator increased the performance of Genetic Algorithms in research, though they were ultimately less successful than scanning operators; however, it was also noted that it is much less expensive to compute, meaning that larger populations could be processed in the same time [10]. In our research, the population sizes are set to be equal across all operators.

Now that we have defined each of the genetic operators from the literature, we will define our contribution: the genetic overlay and its concrete implementation, the elitist schema overlay.

4 Genetic Overlays

Fitness values are the determining force behind whether or not the genetic sequence of an individual will survive and propagate. The selection phase of Genetic Algorithms determines the probability of a given individual being chosen for recombination. The technique to ensure the survival of good genes was conceptually expanded upon with elitism, which allows the best performing individual to survive into the next generation. Elitist schema overlays ingrain this concept within a genetic operator, and are a special instance of genetic overlays:

Definition 1. *Genetic Overlay - A genetic operator which modifies an individual's genetic sequence to match the specified alleles of a given schema, while leaving the unspecified alleles at their original values.*

The following is an example of a genetic overlay that operates upon binary genetic sequences. The *defining length* of a schema is defined to be the number of specified values in an overlay, in the case of the provided example, we have the value 4.

$$\begin{aligned}
\text{Initial Individual} &: 0\ 1\ 0\ 0\ 1\ 0 \\
\text{Genetic Overlay} &: 1\ -\ 0\ 1\ -\ 0 \\
\text{Resultant Individual} &: 1\ 1\ 0\ 1\ 1\ 0
\end{aligned}$$

This operator allows us to quickly give a set of individuals very similar genetic sequences, thus allowing us to search the neighborhood of the provided schema. The choice and density of the schema that we use is very important. If our schema is very dense, i.e. it consists of mostly specified alleles, then the population will converge very quickly. Early convergence is problematic because it lessens the probability that the optimal solution has been found [4]. Likewise, a very sparse schema will spend excess computation cycles applying very few changes. Additionally, we want to ensure that our schema consists of alleles that yield high fitness values. Each of these factors must be carefully considered when developing a strategy for creating genetic overlays.

Elitist Schema Overlays

There are many reasonable methodologies that we could utilize to build effective genetic overlays. The central focus of this paper is the use of *elitist schema overlays*, and they are defined below.

Definition 2. *Elitist Schema Overlay - A genetic overlay that is built from the matching alleles of the elite sub-population.*

To define this formally, let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n individuals selected for recombination. Consider $T = \{t_1, t_2, \ldots, t_k\}$ as the set of the k individuals with the highest fitness rankings in P. We will now construct the genetic overlay, named s, from the genetic sequences in T. We want our genetic overlay to have specified values only where every individual in T has the same value at that same position. The notations $s[i], t_j[i]$, and $p[i]$ represent the value, or lack-thereof in the case of the schema, at position i.

$$s[i] = \begin{cases} t_1[i] & \text{if } t_1[i] = t_2[i] = \ldots = t_k[i] \\ - & \text{if otherwise} \end{cases}$$

Below is an example for $k = 3$.

$$\begin{aligned}
\text{Elite Individual 1} &: 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0 \\
\text{Elite Individual 2} &: 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0 \\
\text{Elite Individual 3} &: 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \\
\text{Resultant Genetic Overlay} &: -1\ 0\ -\ -0\ 1\ -\ 0
\end{aligned}$$

From here, we will apply s as a genetic overlay to each individual in P. An application of the example elitist schema overlay created before can be seen below:

Unmodified Individual : 1 0 0 1 1 0 1 1 1

Elitist Schema Overlay : $-1\ 0\ -\ -0\ 1\ -\ 0$

Resultant Individual : 1 1 0 1 1 0 1 1 0

At this point, we may continue to apply additional genetic operators to P as need be, or continue to the next generation. If our best performing individuals all have the same values at various alleles, this may hint that these assignments are correlated with success. Thus, it would be reasonable to search the neighborhood of this partial solution more thoroughly.

It should be noted that the members of the set P can be selected from the unmodified population or from the population resulting from the application of any other genetic operator. This was done intentionally to explore the behavior of elitist schema overlays as both a mutative and reproductive genetic operator.

Admittedly, the effects of this method will depend greatly upon our choice of k. For instance, the choice $k = 1$ will replace every individual that has the genetic overlay applied with the highest ranked member of the population. This is problematic, since it will instantly lead us to convergence. Likewise, a large value of k will greatly reduce the probability that $t_1[i] = t_2[i] = \ldots = t_k[i]$ is true. This would lead us to checking a large number of alleles and, in the likeliest case, doing little to nothing with that information.

To determine reasonable values for k, it is helpful to know what the probability of an allele being specified in an elitist schema overlay is. We will denote α_i to be this probability, and x_i to represent the size of the domain of the ith allele for a genetic sequence of length n. Note that this assumes that each value for an allele has an equal probability of showing up in a selected genetic sequence at allele i. Given that this assumption is rarely true, as more successful alleles should be present with a higher frequency, the following is a lower bound of the probability of a matching allele over k parents.

$$\alpha_i = P(k \text{ parents matching } i\text{th allele}) = \left(\frac{1}{x_i}\right)^{k-1}$$

With this, we can easily derive the probabilities of having no matches and a total match for a chromosome of length n in terms of α_i.

$$P(\text{No matches}) = \prod_{i=1}^{n}(1 - \alpha_i)$$

$$P(\text{All matches}) = \prod_{i=1}^{n}(\alpha_i)$$

Thus, our choice of k should be made carefully, but there are a large number of possibilities that we could easily consider. Large values for k will make α_i small, and thus the probability that no matches have been found will be high.

Likewise, low values for k increase the value of α_i, making the probability of every allele matching high. We need to find a methodology for selecting k that balances these two extremes. Else, we run the risk of wasting computations looking for very unlikely matches or causing our population to converge prematurely. A fixed and predetermined k could be chosen, k could be randomized for each generation, or k could be set to the number of individuals above a certain fitness threshold; however, the probabilities mentioned above lead to an alternative approach. We can create an inverse linear relationship between k and the number of generations remaining. By changing the value of k over time, we hope to achieve the following properties:

1. Narrowing Search - Since the initial k values will be very large, it is unlikely that $t_1(i) = t_2(i) = \ldots = t_k(i)$, and so we will not disturb the initial natural diversity with the operator. Likewise, as our k drops, our genetic overlay has a higher chance to be mostly specified, meaning that we will probably be searching a progressively narrowing neighborhood based upon the best performing solutions discovered so far [20].

2. Bounded Convergence - We can control how quickly k decreases, and by this, how quickly our population will converge towards the best known solutions. When we set $k = 1$ we will instantly converge the entire target population of this operator, and so we have a bound for the minimum convergence rate.

Our tests compare how well elitist schema overlays perform with a random k being selected each generation, k being set as a fixed percent of the population, and k being set by a linear inverse relationship with the number of generations remaining.

Rationale

Elitist schema overlays with an effective choice of k are intentionally related to the theoretical bases of Genetic Algorithms [12, 14]. The schema theorem, in Holland's own words, states, "The adaptive system must, as an integral part of its search of a, persistently test and incorporate structural properties associated with better performance [14]." Likewise, the building block hypothesis states that the propagation of building blocks, high fitness schemata with low defining lengths, are integral to the successes of Genetic Algorithms [12]. Thus, if we can identify the alleles associated with high performance, then we can use a genetic overlay to incorporate them into our entire population. This will lead to the identified schemata being tested by the fitness function more often since it is present in more individuals. Since premature convergence will prevent solutions from improving much more, we should be careful to only build genetic overlays that are correlated with high fitness values while leaving ample room to search [1].

Strangely, when Forrest and Mitchell tested the performance of Genetic Algorithms against Hill-Climbing Algorithms on the Royal Road function, whose

definition is tightly coupled with the building-block hypothesis, Genetic Algorithms were out-performed by Random Hill-Climbing Algorithms [11]. Results like this have lead to criticism of the strength of the underlying assumptions and the narrowness over which the schema theorem and building-block hypothesis could be applied [3, 23]. This has lead to the use of effective fitness measures and coarser graining on the size of building-block schemata to describe the evolutions of schemata over time [26].

The underlying mechanics of successful Genetic Algorithms are still being debated, but the exploitation of current, successful genetic sequences is still fundamental to the field as a whole [22, 23]. Genetic overlays were designed to speed up the propagation of schemata in a population, and elitist schema overlays search the fittest individuals for useful schemata to propagate. This operator will be explored as an addition to current models, modifying the population between the recombinant operators, fitness-based scanning and diagonal crossover, and the mutation operator.

This decision was made because elitist schema overlays act similar to both recombinant and mutative operators. Since they are produced from k genetic sequences, we can consider them as multi-parent operators; however, the application of a genetic overlay is a unary procedure. Thus, we utilize elitist schema overlays as an addition to, not a replacement for, current genetic operators. To determine the effects of including elitist schema overlays in Genetic Algorithms, we will now define the test problems used for our experimentation.

5 Testing Benchmarks

Improvements to Genetic Algorithms are frequently compared based upon their abilities to solve various benchmark numerical functions and real world problems like the Traveling Salesman and Job Scheduling problems [19]. In order to get a good grasp on the general behavior of elitist schemata overlays in comparison to the other tested operators, we have chosen a variety of problems from both of these fields.

Our research found a set of common minimization functions used as benchmarks of the performance of new Genetic Algorithms [4, 10, 28]. These functions should give us a picture of how well our algorithm compares to more traditional ones. In order to make some reasoning about the behavior of elitist schemata overlays on real world problems, we will also test performance on several instances of the Knapsack and Traveling Salesman problems. Finally, each of the genetic operators will be run on several NK-Landscapes, a common testing tool for Genetic Algorithms. Definitions for each of these can be found below.

Minimization Functions

Definitions

The functions below display a wide array of properties as a group that will test each of the behavior of the genetic operators in different ways. By varying

levels of ruggedness, how much values differ from those in their neighborhood, and deceptiveness, the number of local optima, we get a broad picture how our operators will behave under certain conditions. The definitions for each of these numerical optimization problems can be found in the literature [19, 32].

The De Jong hypersphere function is both convex and unimodal with a global minimum of 0 located at $(0, 0, \ldots, 0)$. The n-dimensional version of the hypersphere function is as defined:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i^2 \text{ for } -5.12 \leq x_i \leq 5.12$$

The De Jong hyper-ellipsoid function is defined similarly, and maintains the global minimum of 0 located at $(0, 0, \ldots, 0)$.:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} i x_i^2 \text{ for } -5.12 \leq x_i \leq 5.12$$

We also selected the closely related sum of different powers function, whose minimum is also of 0 located at $(0, 0, \ldots, 0)$:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} |x_i|^{i+1} \text{ for } -1 \leq x_i \leq 1$$

The Griewank function is known for its multi-modality and also has its global minimum of 0 located at $(0, 0, \ldots, 0)$:

$$f(x_1, x_2, \ldots, x_n) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \text{ for } -600 \leq x_i \leq 600$$

Rastrigin's Function is another highly multi-modal function with a global minimum of 0 at $(0, 0, \ldots, 0)$:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right) \text{ for } -600 \leq x_i \leq 600$$

Rosenbrock's function has a global minimum of 0 at $(1, 1, \ldots, 1)$:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n-1} ((x_i - 1)^2 + 100(x_{i+1} - x_i^2)^2) \text{ for } -5 \leq x_i \leq 5$$

The two dimensional Michaelwicz function has a global minimum approximately equal to -1.8013 at the approximate points $(2.2032, 1.5705)$ and is defined below:

$$f(x, y) = -\sin(x)\sin^{20}\left(\frac{x^2}{\pi}\right) - \sin(y)\sin^{20}\left(\frac{2y^2}{\pi}\right) \text{ for } 0 \leq x, y \leq 5$$

The two-dimensional six-hump camel back function has a global minimum of approximately -1.0316 at the coordinates $(0.0898, -0.7126)$ and $(-0.0898, 0.7126)$:

$$f(x,y) = \frac{x^6}{3} - 2.1x^4 + 4x^2 + xy + 4y^4 - 4y^2$$

Schubert's function is multi-modal and has 18 global minima of approximate value -186.7309 in the given search domain :

$$f(x,y) = \left(\sum_{i=1}^{5} i\cos(i+ix+x)\right) * \left(\sum_{i=1}^{5} i\cos(i+iy+y)\right) \text{ for } -10 \leq x,y \leq 10$$

As a side note, the JGAP version used for our testing disallowed for negative fitness values [17]. To account for this, a constant offset was added to each of the fitness scores returned for the Michaelwicz, six-hump camel back, and Schubert functions to ensure that only non-negative results were returned.

Numerical Encoding

Genetic sequences have traditionally been encoded as both traditional binary strings and *Binary Reflective Gray Codes* [31]. Gray Codes are modified interpretations of bit strings that minimize the variation in representation of adjacent values [2]. In the traditional binary system, the unsigned values of 2^n and $2^n - 1$ have no bits in common over the first n places, even though their values only differ by 1. In a Gray Coding, all adjacent values differ by exactly one bit [18]. Given a bit string $a = a_1 a_2 \ldots a_n$, we can described the translation between traditional binary strings and Binary Reflective Gray Codes mathematically. The traditional binary bit string a can be rewritten as a Binary Reflective Gray Code bit string $b = b_1 b_2 \ldots b_n$ with the following function [2]:

$$b_i = \begin{cases} a_1 & \text{if } i = 1 \\ (a_{i-1} + a_i) \bmod 2 & \text{if } i > 1 \end{cases}$$

Likewise, we can translate between a Binary Reflective Gray Code bit string b to the traditional binary bit string a with this procedure [2]:

$$a_i = \begin{cases} b_1 & \text{if } i = 1 \\ (a_{i-1} + b_i) \bmod 2 & \text{if } i > 1 \end{cases}$$

Empirically, Genetic Algorithms that employ Gray Codes have outperformed those using traditional binary genetic sequences [2]. For that reason, we have implemented Gray Codes in our research. This gives us a good means of representing integers, but our numerical optimizations are defined over the real numbers.

To describe continuous domains, like those in the test functions presented, we must map our genetic sequences of discrete values to the real numbers. Suppose we wish to operate on a variable x whose domain $D = [a,b] \subseteq \mathbb{R}$, and that we require k decimal places of precision for x. To accomplish this, we

need to partition D into $10^k(b-a)$ ranges of equivalent length. So we find the smallest $n \in \mathbb{N}$ such that $10^k(b-a) \leq 2^n - 1$. To assign values to x given a genetic sequence g and a decoding function d, which takes a genetic sequence and evaluates it to its decimal equivalent, we use the following formula [18]:

$$x = a + d(g)\left(\frac{b-a}{2^n - 1}\right)$$

This methodology is also used in our research.

0-1 Knapsack Problem

Definition

The 0-1 Knapsack Problem is an NP-Hard optimization problem. Let X be the set of items $\{x_1, x_2, \ldots, x_n\}$, C be the set of cost values $\{c_1, c_2, \ldots, c_n\}$, and B be the set of benefit values $\{b_1, b_2, \ldots, b_n\}$. Each item x_i, with the domain $\{0, 1\}$, has an associated cost c_i and benefit b_i, where each $b_i, c_i \in \mathbb{R}^+$. Given a cost limit L we want to maximize the sum of the benefit values while not allowing the sum of our cost values to exceed L:

$$max\left(\sum_{i=1}^{n} b_i x_i\right) \text{ subject to } \sum_{i=1}^{n} c_i x_i \leq L$$

Encoding

The 0-1 Knapsack Problem is typically also represented with bit strings, but these are not evaluated in the same manner as those for continuous domains. For a genetic sequence $g = g_1 g_2 \ldots g_n$, each g_i, where $1 \leq i \leq n$, represents the value of the corresponding x_i in the item set X for a given instance of the problem.

Traveling Salesman Problem

Definition

The Traveling Salesman Problem is another NP-Hard problem. Let V be a set of vertices $v_1, v_2, \ldots v_r$ of a graph G. The set of edges E is defined such that each $e_i \in E$ is an ordered tuple of the form $(v_j, v_k), j \neq k$ where $v_j, v_k \in V$ are the vertices that e_i connects. For each $e_n \in E$, there is an associated $w_n \in W$ where w_n is a positive real number that is the weight of the edge e_n. The goal of a given Traveling Salesman Problem instance is to find the Hamiltonian Circuit that has the minimum total weight. In other words, we must find a series of connected edges $T = e_1, e_2, \ldots, e_r$, where r is the number of vertices, that minimizes the following [13]:

$$\sum_{e_i \in T} w_i$$

Where w_i is the weight of the edge e_i. Additionally, to ensure that each of our generated graphs has a Hamiltonian circuit, we force them to satisfy Dirac's Theorem, which states:

Theorem 1. *Every simple, connected graph with $n \geq 3$ vertices that satisfies $degree(n) \geq \frac{n}{2}$ has a Hamilton circuit [30].*

Encoding

Permutation based problems, like the Traveling Salesman Problem, break away from the tradition of utilizing binary values to build genetic sequences. Naïve bit string representations create long genetic sequences that commonly encode infeasible solutions, those that do not meet the constraints of the problem instance, or incorrect genetic sequences, those that are not permutations [21]. To help correct this, Traveling Salesman Problems of size n can be encoded as a sequence of integers from the set $\{1, 2, \ldots, n\}$ to minimize the length of the genetic sequences. Another common optimization is to fix a single point in all genetic sequences to reduce the number of representations of an identical solution, and we have followed this trend in our work.

Permutation Repair Algorithm

For most applications, the described genetic operators will function correctly unmodified; however, for permutation based problems that allow for only one instance of a given allele, like the Traveling Salesman Problem, a slight change is necessary. For instance, the application of an overlay might leave identical values in multiple positions in the genetic sequence. Many genetic operators have this effect, and two main solutions are utilized in practice: New genetic operators tailored to these problems, and repair algorithms [16]. To keep our choice of operators consistent, we have devised a repair algorithm to use in every necessary case.

$$\begin{aligned} \text{Genetic Overlay} &: -1\ 3\ -\ -6\ 2\ -\ 9 \\ \text{New Individual} &: 2\ 1\ 3\ 4\ 5\ 7\ 6\ 8\ 9 \\ \text{Resultant Individual} &: 2\ 1\ 3\ 4\ 5\ 6\ 2\ 1\ 9 \end{aligned}$$

As we can see above, the application of the genetic overlay results in an individual that contains two instances of both 2 and 1 in its genetic sequence. Since this is no longer a permutation, it cannot be a potential solution to the Traveling Salesman Problem. Thus, we need a repair algorithm can be used to correct this example, while maintaining the given genetic overlay.

Our repair algorithm, whose pseudocode can be found in Figure 2, iterates through a genetic sequence and counts how often each value is used. On the second pass, we check each value to see if it has been used exactly once in the permutation. If this is true, then we advance to the next allele. We will also skip alleles that are defined in the given genetic overlay. If neither of these conditions

are true, we replace the current allele with a value that is not already in the permutation. With this structure, we maintain the invariant that the genetic subsequence from the beginning up to the point we are operating on is a valid permutation.

In order to guarantee that we both do not disturb a given genetic overlay and that our end result is a permutation, we must require that every genetic overlay is a permutation. This constraint is feasible in practice, and we have proven below that no elitist schema overlay will ever contain a duplicated value if the population consists of permutations only.

Theorem 2. *An elitist schema overlay will contain no duplicated elements, given that all members of the population are k-permutations of n elements where* $k \leq n$.

To begin, let e be an elitist schema overlay created from a population P. Let us assume towards a contradiction that e contains at least one value that has been duplicated. Thus, the must exist distinct alleles e_p, e_q, such that $e_p = e_q$. So, for each individual I used to construct e, we know that $I_p = e_p = e_q = I_q$ by the definition of elitist schema overlays. However, $I_p = I_q$ implies that I is not a k-permutations of n elements. This contradicts our assumption that every individual in the population must be a k-permutations of n elements, and so each e must only contain distinct elements. ■

Given the code in Figure 2, we will now demonstrate that this approach will always return permutations given that our genetic overlays contain no duplicated elements.

Theorem 3. *The result of applying the permutation repair algorithm on a genetic sequence, s, and an overlay, o, of length k is always a genetic sequence, with o intact, that is a k-permutation of the n possible values for each allele, given that* $k \leq n$ *and that the genetic overlay contains no duplicated elements.*

To prove this theorem via weak induction, we shall begin with the base case of genetic sequences of length 1. Thus, neither s nor a o can have any duplicated values. Thus, applying o to s will create no duplicate elements and will leave o intact.

Now we assume that the permutation repair algorithm will work on any genetic sequence and genetic overlay of length $k-1$, and will return a $(k-1)$-permutation of $n-1$ elements with the genetic overlay intact. From here, we show that the case of $k \leq n$ follows. For this proof, assume that the n permutable values are contained in a set N. We must consider the following cases:

1. The first value of the genetic overlay is defined. To begin, we have already applied o to s. Now we apply the inductive hypothesis to the subsequence $s_2 \dots s_k$ and set of values $N' = N \setminus \{o_1\}$. Thus, $s_2 \dots s_k$ is a $(k-1)$ permutation of the $n-1$ elements of N' with $o_2 \dots o_k$ intact. Since, $o_1 = s_1$ we know that o is intact with respect to s. Further, since $s_1 \neq n_i, \forall n_i \in N'$, we know that s can contain no duplicate values.

input : A genetic sequence, *seq*, of permutable values and a genetic overlay, g

output: *seq*, with values that form a permutation.

```
// Find out how many times each value is used in seq
for value ∈ seq do
    use[value] ← use[value] + 1
end
// Correct the sequence
for i ← 0 to seq.length do
    value ← seq[i]
    // Only check undefined locations in the overlay
    if g[i] = defined then
        continue
    end
    // Check if the value has been reused
    if use[value] > 1 then
        // Find an unused value
        for j ← 0 to use.length do
            // Update the usage array and genetic sequence
            if use[j] = 0 then
                seq[i] ← j
                use[j] ← 1
                use[value] ← use[value] − 1
                break
            end
        end
    end
end
```

Figure 2: Permutation Repair Algorithm Pseudocode

2. The first value of the genetic overlay is undefined. Consider the set $N' = N \setminus \{o_t | \forall o_t \in o \text{ where } o_t \text{ is defined}\}$. Since o_1 is undefined and the length of o is less than or equal to n, we know that at least one element exists in N'. We now set s_1 to be an arbitrary member of N'. Now we apply the inductive hypothesis to the subsequence $s_2 \dots s_k$ and set of values $N'' = N \setminus \{s_1\}$. By the inductive hypothesis and the assumption that o_1 is undefined, o must be intact with respect to s. Additionally, because $s_2 \dots s_k$ cannot contain duplicate values and cannot contain s_1, since $s_2 \dots s_k$ is a $(k-1)$ permutation of N'', we know that s contains no duplicate values.

In either case, we are left with a genetic sequence that is a k permutation of

n elements with our genetic overlay, o, intact. ■

NK-Landscapes

NK-Landscapes are mathematical models that have appeared in many disciplines, and finding the global minimum or maximum of one of these models has been demonstrated to be in NP-Hard [29].

NK-Landscapes assign values to bit strings of length N by applying a function f to each allele and its K neighbors [15]. A string S, consisting of the bits $b_1, b_2, \ldots, b_N$ has a fitness value V defined to be:

$$V = \sum_{i=0}^{N} f(S, i)$$

The function f takes the string and evaluates the necessary number of bits using the function g.

$$f(S, i) = g(b_i, b_{i+1 \bmod N}, \ldots, b_{i+K \bmod N})$$

Larger values of K increase the ruggedness of a model by increasing the interplay between the bits, and lower values of K make for smoother landscapes [15]. Rugged models have large numbers of local optima, and so tuning K gives us varying degrees of difficulty for local search methods like Genetic Algorithms [22, 24]. Likewise, larger values for N lead to search spaces of exponentially increasing size. Since multi-parent genetic operators have had success on NK-Landscapes in previous studies, we will test each genetic operator on several NK-Landscapes of varying size and ruggedness [9, 24].

6 Experimentation

Now that the operators and benchmark problems have been identified and defined, we will empirically test the performance of elitist schema overlays, multi-parent genetic operators, multi-operator strategies, and traditional Genetic Algorithms. We seek to compare each of these configurations in terms of their solutions found and runtime; additionally, we seek to verify previous results in multi-parent operator research. These results and experiments will help us answer the following questions:

- Do fitness-based scanning and diagonal crossover perform as previously reported, with more parents tending to higher success rates?
- Do fitness-based scanning and diagonal crossover perform better when used separately or in conjunction with each other?
- Which of the three methods of selecting k, assigning random values, selecting a fixed percentage, or by computing a linear relationship with generations left, performs best?

- How do each of the above methods of k selection affect the convergence rates of their respective populations?
- With which genetic operator(s) does the best performing elitist schema overlay configuration perform best?
- Do elitist schema overlays improve solution quality?
- How efficiently can elitist schema overlays be computed and applied in relation to other genetic operators?

Each of the operators and benchmarks were implemented with the JGAP framework. JGAP is an open source Genetic Algorithms package for the Java programming language developed by Meffert et al [17]. This work is built upon JGAP version 3.6.2, which was the latest stable release at the time this paper was written.

To determine how well any Genetic Algorithm performs, it is important to know what parameters were used for testing. To reduce the number of variables in our experiments, the same setup was used for each experiment measuring solution quality, as outlined in Figure 3. While measuring runtime, the number of generations was decreased from 500 to 150. These parameters are similar to those found in the literature to ensure that our results are comparable to previous research [27, 10, 9].

problem type	minimization
parents used for recombination	2-10, by steps of 2
selection type	steady-state
selection mechanic	best first
diagonal crossover rate	70 %
fitness-based scanning rate	70 %
elitist schema overlay rate	100 %
mutation rate	dynamic [2]
population size	fixed at 200
termination condition	500 generations elapsed
trials per configuration	100

Figure 3: Parameters used for testing

All of our experiments measuring runtime were executed on an Intel Core i5-2500 processor clocked at 3.30 gigahertz to ensure that the data gathered was comparable. The computer also had 16 gigabytes of RAM and was running version 5.5 of the CentOS operating system. The full data set generated from these experiments can be provided by the author at request.

Benchmark Parameters

To ensure that our tests are reproducible, the parameters used to generate each of our test cases have also been included. The n-dimensional numerical optimization problems were tested on genetic sequences of length 100, split into 4 dimensions of 25 boolean alleles: For reference, they have been listed below:

- De Jong's Hypersphere Function
- De Jong's Hyper-ellipsoid Function
- The Sum of Powers Function
- Griewank's Function
- Rastrigin's Function
- Rosenbrock's Function

The 2-dimensional functions were tested on genetic sequences of length 62, with the boolean alleles split evenly between the two dimensions. These functions have also been listed below for reference:

- Michaelwicz's Function
- The Six-Hump Camel Back Function
- Schubert's Function

Both the Knapsack and the Traveling Salesman problems were tested on several randomly generated instances of size 50 to 500 by increments of 50. 10 instances of each size were generated to help prevent against a particular operator having an advantage over another due to its ability to exploit a feature of a given instance. NK-Landscapes were similarly generated for $N = 50$ to $N = 250$ with increments of 50. 20 instances were generated for each size, 10 with $K = 5$ and the remainder with $K = 10$. Both NK-Landscapes and Knapsack Problem instances were tested using boolean alleles, and the Traveling Salesman Problem utilized chromosomes built of integer alleles.

Results

To determine the usefulness of elitist schema overlays as a genetic operator, we compared our work to current successful operators. To do so, we have analyzed both how efficient our operator is and how well it contributes to solution quality.

To measure solution quality for each operator, we measured the average fitness of the most fit solution found for each of the numerical optimization problems. To measure the solution quality of problems in NP-Hard, we measured the average fitness found by an operator at a given arity and compared it to best known solution across the problems all operators with the following:

$$\frac{\text{averageFitness}}{\text{bestFoundFitness}} * 100\%$$

It should be noted that the best found solutions during experimentation for k-value selection were only compared against other methodologies in that section; the best found solutions during experiments comparing elitist schema overlays to multi-parent operators were drawn from across both of these datasets. We tested several instances of each size of each problem to reduce error, and collected frequency data to allow us to compare solutions of different instances that were the same size. Multiple instances were used because internal features of a given instance may play a role in how well a particular operator performs.

Since each of the problems was tested as a minimization task (as shown in Figure 3), we wish for our reported fitness values to be as low as possible. For the charts measuring solution quality by problem, our y-axes on the provided charts always measures the average fitness value while the x-axes provide information on operator arity. For numerical optimization problems, this is the direct result of the mathematical formulas provided. In the cases of problems from NP-Hard, the fitness value as a percent of best found solution is provided.

To answer our research questions, we will first explore the genetic operators from the literature. Secondly, we compare the various methodologies used to select a k-value for elitist schema overlays to determine which we will test in conjunction with existing genetic operators. Finally, we will analyze how elitist schema overlays perform when used in conjunction with existing genetic operators.

Existing Genetic Operators

To determine if elitist schema overlays improve the overall performance of Genetic Algorithms, we must first establish a performance baseline from the existing research. To do so, we have tested both existing operators, diagonal crossover and fitness-based scanning, individually and in conjunction with each other against our benchmarks. Diagonal crossover and fitness-based scanning were both tested with arities of 2, 4, 6, 8, and 10. When combined, both operators were set to the same arity from the previous list. These experiments will help to verify the previous experimental results from multi-parent and multi-operator research [25, 10].

Each of the operator configurations consistently found the same minima for 6 of the 9 optimization problems. Since these data do not differentiate the operators, we will focus upon the remaining three functions: Griewank's (shown in Figure 4), Rastrigin's (Figure 5), and Rosenbrock's (Figure 6).

In these cases, traditional Genetic Algorithms consistently performed worse than multi-parent strategies. As a general trend, both utilizing more operators and increasing the arity of those operators increased performance, but there were exceptions. Fitness-based scanning saw performance degradation with arity 10 on the Rastrigin function, as did mixed operator usage on the Griewank function. Utilizing several operators performed the best on the Griewank and Ras-

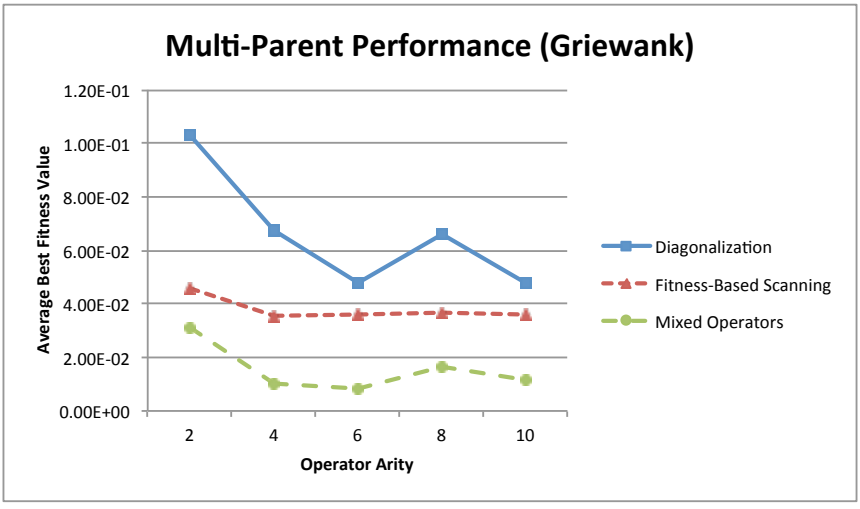

Figure 4: Multi-Parent Performance on Griewank's Function

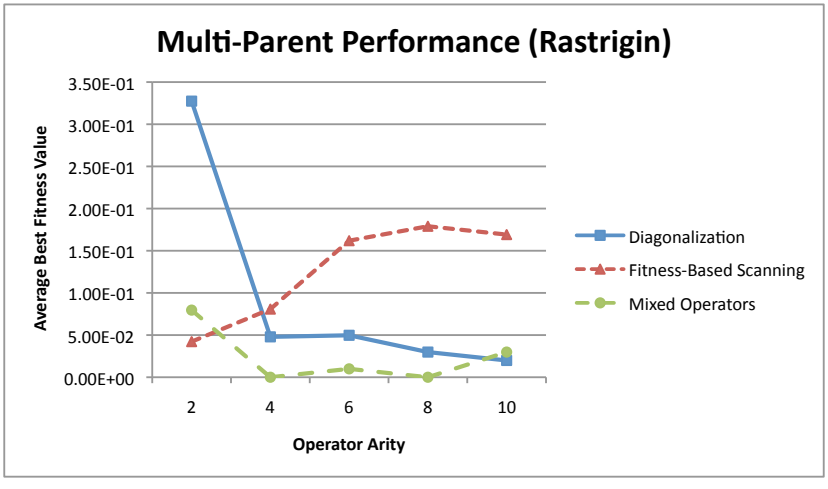

Figure 5: Mulit-Parent Performance on Rastrigin's Function

trigin functions, and was outperformed on the Rosenbrock function by fitness-based scanning.

We also observed that diagonalization was consistently outperformed on both the Rosenbrock and Griewank functions; however, diagonalization did outperform fitness-based scanning on the Rastrigin function. Each of these findings were in line with the literature on fitness-based scanning and diagonalization [8, 10]. We will now investigate how these operators performed on the NP-Hard test problems.

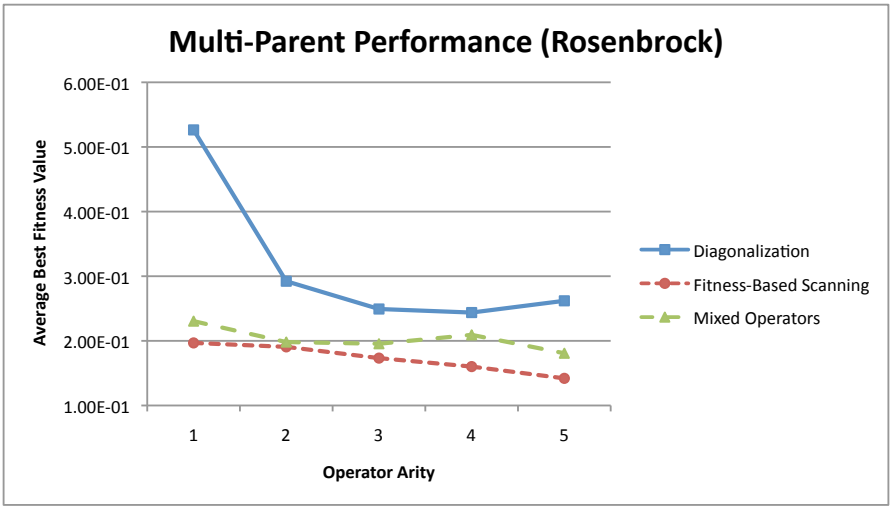

Figure 6: Multi-Parent Performance on Rosenbrock's Function

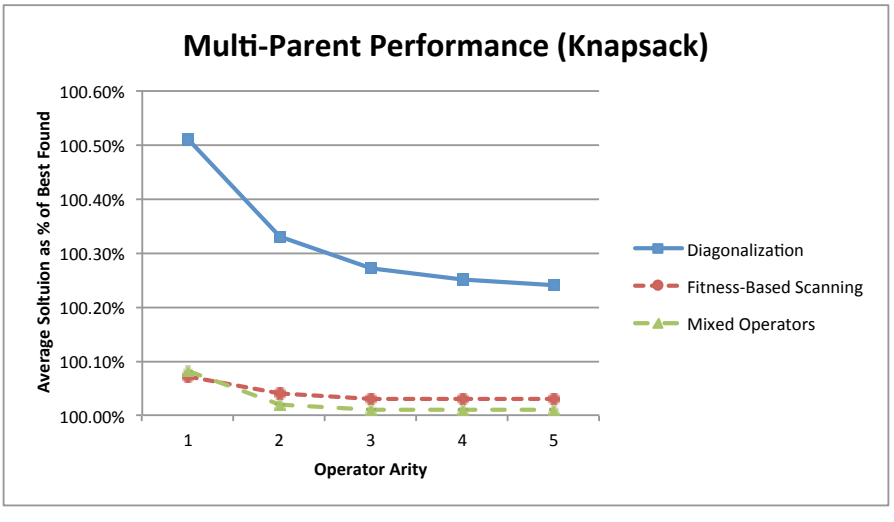

Figure 7: Multi-Parent Performance on the Knapsack Problem

For the Knapsack Problem (shown in Figure 7), diagonalization consistently performs worse than all other configurations; however, in the worst case, the instances with 500 objects, its average best solution was only 1.39% higher than the best found solution across all configurations, where the best strategy found solutions 0.01% greater than the optimal when averaging over all instances of all sizes. Within that range of performance, our previous observation that increasing the arity of the operators improved performance held true.

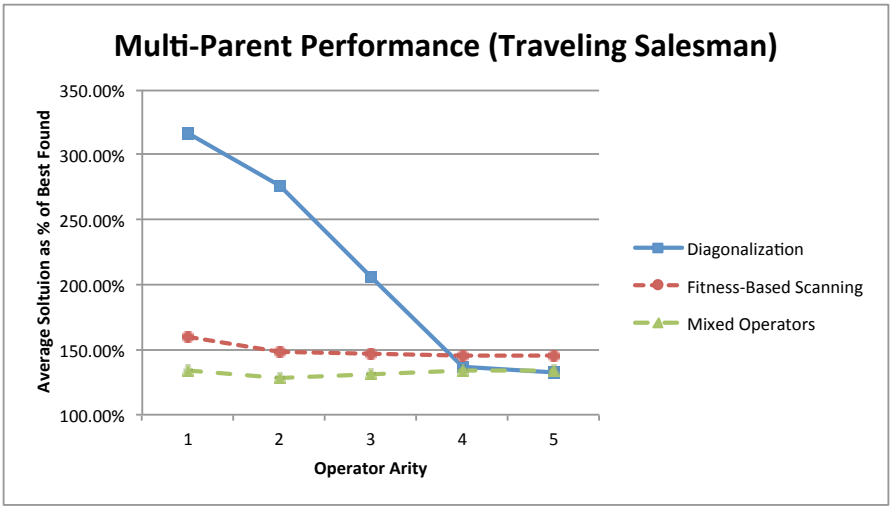

Figure 8: Multi-Parent Performance on the Traveling Salesman Problem

Within the tested instances of the Traveling Salesman Problem (shown in Figure 8), high-arity diagonalization outperformed fitness-based scanning, but was outperformed by mixed operator strategies. Mixed operator strategies performed the best with arity 4, but across the other two strategies, more parents typically increased performance. Once again, traditional Genetic Algorithms performed the worst, and found solutions 216.52% greater than the best found when averaged across all instances for all problem sizes, where the best averaged performance was 31.47% greater than the best found on average.

As previously found, multi-parent strategies performed the best when tested on NK-Landscapes [9]. For each problem instance, the best found solution was always found by each operator configuration.

Overall, this data supports previous conclusions about the success of genetic operators with arities greater than 2. Increasing the number of parents involved in recombination improved the success of both diagonalization and fitness-based scanning. Additionally, utilizing both of these operators in conjunction with each-other further increased performance on both the numerical optimization and NP-Hard test problems. Now we will compare the various methodologies of k selection used to build elitist schema overlays, and compare the best found strategy to the multi-parent operators tested here.

k-Value Selection Methodologies

Each of our k selection methodologies was used in addition to a traditional Genetic Algorithm to determine which methodology performed the best. When using k as a fixed percentage of the population, we ran tests with k set to $10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%,$ and 90%. During our tests with k

chosen as a linear relationship with the number of generations remaining, we determined k with the following formula:

$$k = \left\lfloor \frac{generationsLeft}{totalGenerations} * populationSize \right\rfloor$$

As before, 6 of the 9 numerical optimization problems lead to each of the operators converging to the same solution. The solution quality information from these cases does not help us differentiate these operators, but the convergence information from these tests will be explored later. The remaining functions to be analyzed for solution quality information are the following: Griewank's (shown in Figure 9), Rastrigin's (Figure 10), and Rosenbrock's (Figure 11).

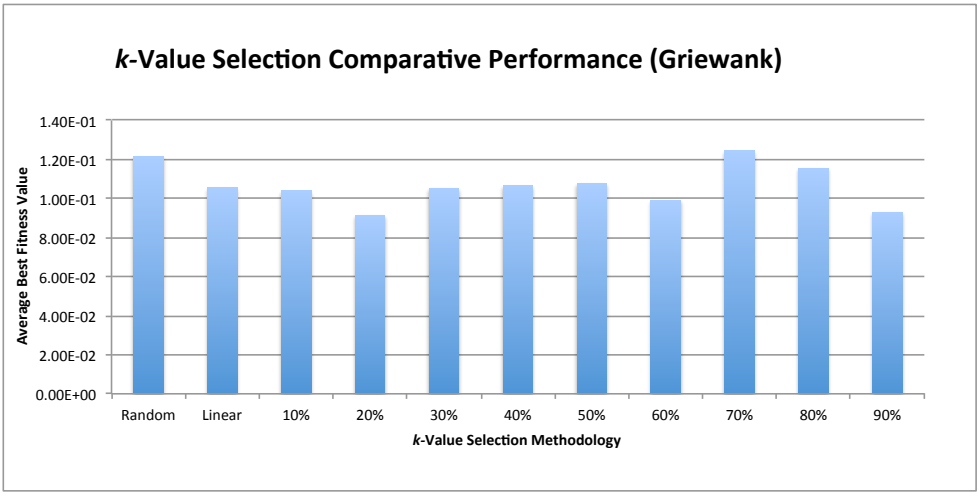

Figure 9: Elitist Schema Overlay Performance on Griewank's Function

Across each of these functions, no single methodology consistently dominated the others. Further, when analyzing the data across the percentage based selection methods, the percent used and the overall performance did not appear to be correlated. Selecting k randomly consistently converged the slowest, taking 153.28 generations on average, and selecting $k = 30\%$, the fastest converging methodology, lead to convergence after 130.7 generations on average. We will now investigate the behavior of traditional Genetic Algorithms with elitist schema overlays on problems in NP-Hard.

The solution quality data gathered from the Knapsack Problem (referenced in 12) is almost indistinguishable, much like the numerical optimization data was. Every methodology found on average a solution that was between 100.19% and 100.21% of the best found solution for that given instance. The convergence data was more varied. Selecting $k = 20\%$ converged on average after 161.78 generations, and $k = 40\%$ converged the fastest, taking 142.09 generations on average.

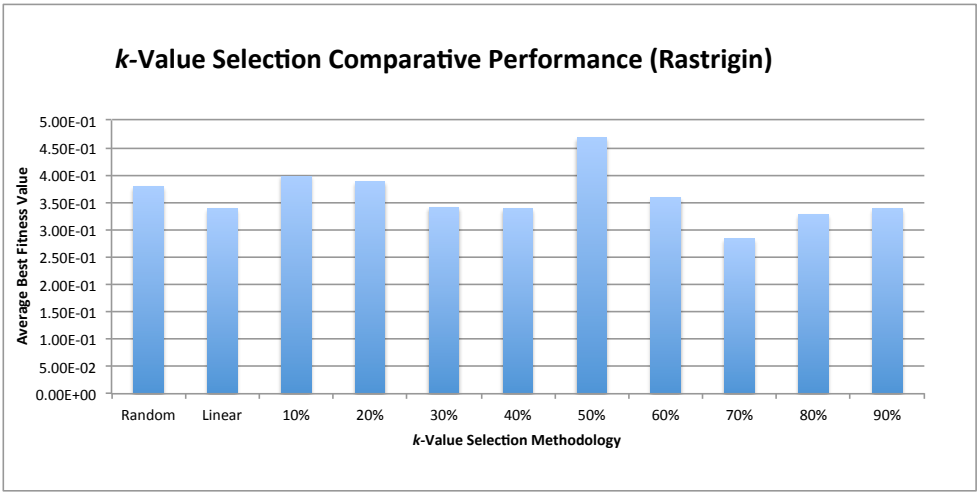

Figure 10: Elitist Schema Overlay Performance on Rastrigin's Function

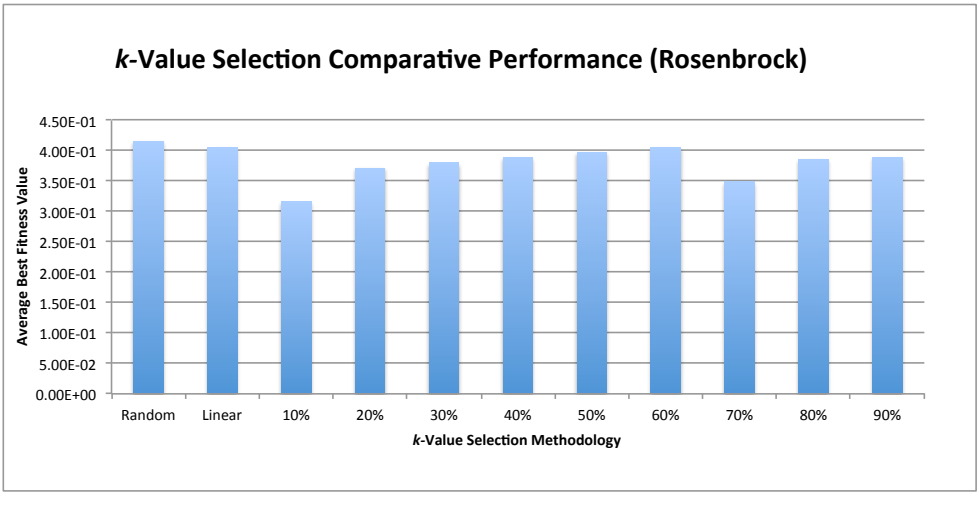

Figure 11: Elitist Schema Overlay Performance on Rosenbrock's Function

Data gathered for the Traveling Salesman Problem (referenced in 13) showed more variation in solution quality. The best average solutions were found by $k = 40\%$, and found solutions 125.77% of the best found solutions on average. $k = 40\%$ also converged the quickest on average, usually converging after 339.69 generations. The worst average solutions were found by $k = 30\%$, which were on average 133.08% of the best found solutions. The slowest converging methodology was $k = 90\%$, which took on average 364.85 generations.

As was the case with multi-parent Genetic Algorithms, our tests with elitist

Methodology	Solution Quality	Average Generations to Convergence
Random	100.20%	144.84
Linear	100.19%	157.24
10%	100.19%	146.54
20%	100.19%	161.78
30%	100.21%	147.94
40%	100.19%	142.09
50%	100.19%	149.27
60%	100.19%	143.39
70%	100.19%	150.19
80%	100.19%	143.47
90%	100.19%	151.64

Figure 12: Elitist Schema Overlay Performance on the Knapsack Problem

Methodology	Solution Quality	Average Generations to Convergence
Random	127.25%	349.77
Linear	126.30%	353.50
10%	126.34%	350.03
20%	126.29%	355.57
30%	133.08%	353.42
40%	125.77%	339.69
50%	129.55%	355.16
60%	125.33%	342.35
70%	128.00%	357.78
80%	126.29%	342.98
90%	127.40%	364.85

Figure 13: Elitist Schema Overlay Performance on the Traveling Salesman Problem

schema overlays on NK-Landscape instances gave us little data to differentiate methodologies (referenced in 14). Following the trend of the other problems in NP-Hard, $k = 20\%$ converged the slowest, taking 33.97 generations on average, and $k = 40\%$ converged the fastest, taking 22.48 generations on average. We now combine the convergence data from the problems in NP Hard and the numerical optimization problems to determine the overall convergence behavior of our k-value selection methodologies.

To gather generalized convergence rate data, we averaged the number of generations to convergence across all runs of all problem instances (Referenced in Figure 15). There appears to be no correlation between the number of parents in fixed percentage methodologies and the average rate of convergence. As was the case before, $k = 20\%$ converged the slowest on average while $k = 40\%$ converged

Methodology	Average Generations to Convergence
Random	26.29
Linear	29.18
10%	27.18
20%	33.97
30%	27.95
40%	22.48
50%	28.64
60%	24.30
70%	29.38
80%	25.34
90%	30.07

Figure 14: Elitist Schema Overlay Convergence on NK-Landscapes

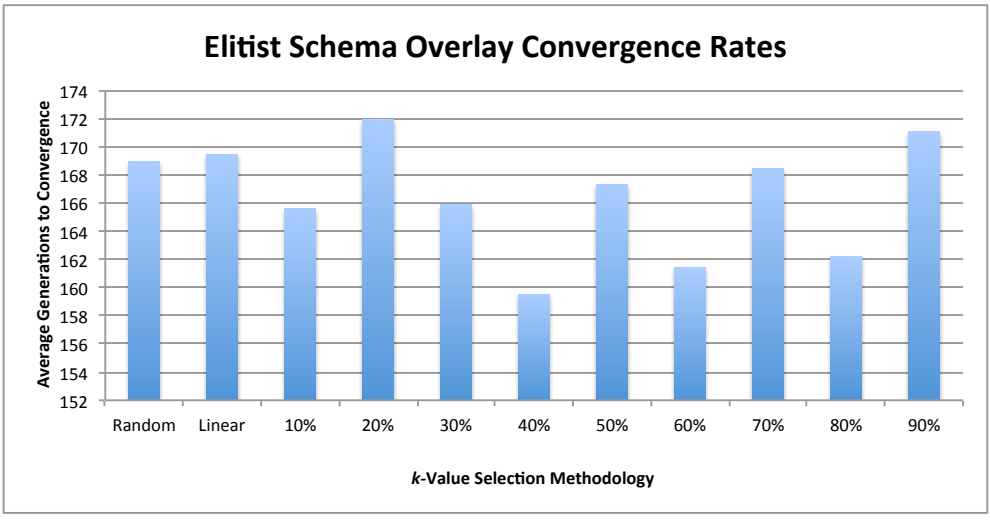

Figure 15: Elitist Schema Overlay Rates of Convergence

the quickest. This data helped us select a k-value selection methodology to utilize with both traditional Genetic Algorithms and multi-parent operators to determine empirically how they affect performance.

Elitist Schema Overlays with Existing Operators

We chose to run our further experiments with k set to be a fixed 20% of our population. The rationale behind this choices was based upon the observed convergence rates in our experiments. Since most of the solutions were comparable in quality, we focused upon the methodology that converged the slowest. This

decision was made in an attempt to help prevent premature convergence [1]. We also aimed to maintain the disruptiveness of diagonalization and fitness-based scanning, which is thought to be central to the successes of multi-parent operators [10]. This was used in conjunction with the 15 different genetic operator configurations used in the experiments with the existing genetic operators. We will begin by comparing performance on the numerical optimization functions.

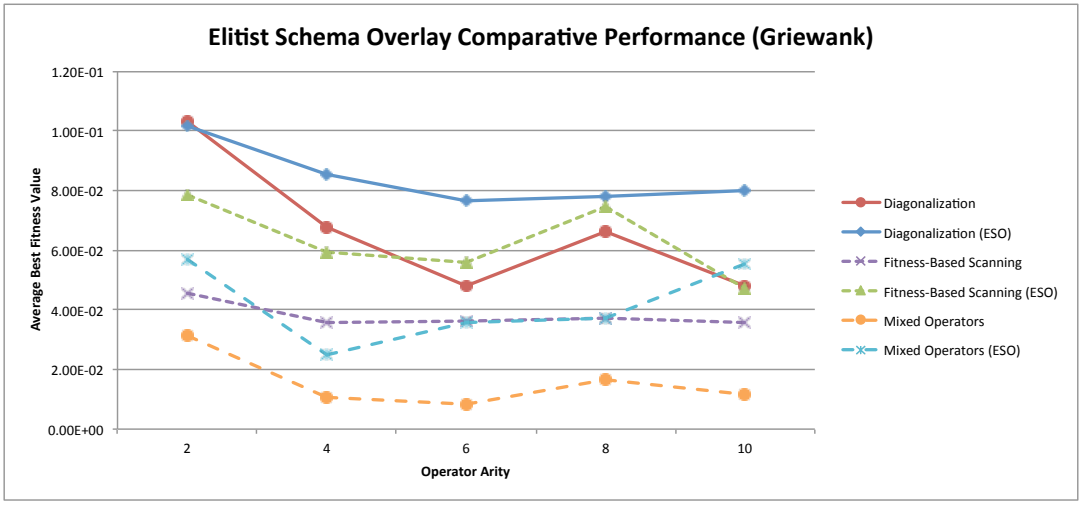

Figure 16: Comparative Performance on Griewank's Function

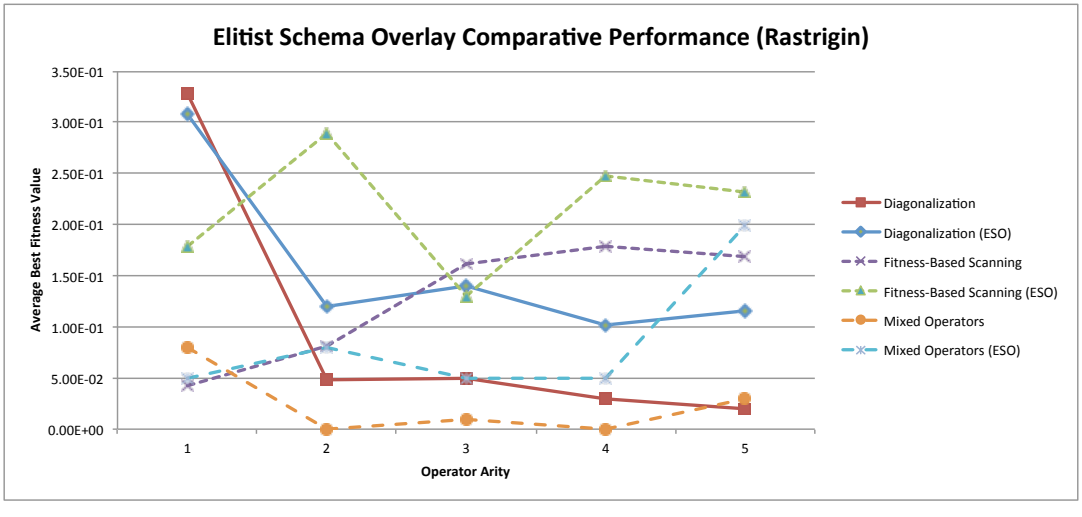

Figure 17: Comparative Performance on Rastrigin's Function

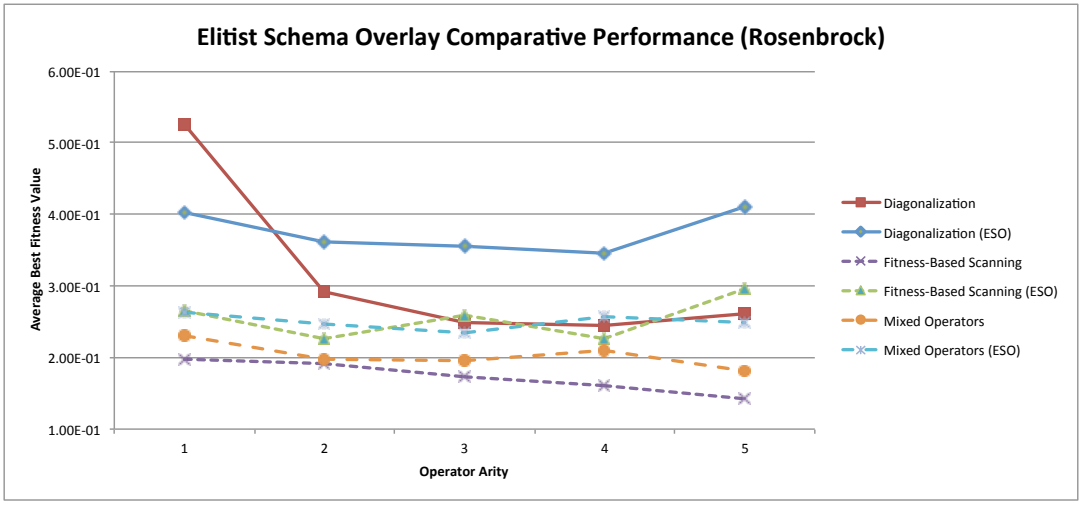

Figure 18: Comparative Performance on Rosenbrock's Function

As before, the observed performance on the numerical optimization problems only deviated during tests on Griewank's function (shown in Figure 16), Rastrigin's function (Figure 17), and Rosenbrock's function (Figure 18). Across these tests, multi-parent operators without elitist schema overlays tend to outperform those with elitist schema overlays. The tests run with elitist schema overlays show similar relative behavior when compared to each other as the multi-parent operators without elitist schema overlays; that is, diagonalization was usually outperformed by fitness-based scanning, which was in turn out performed by a multiple-operator strategy. In contrast to the previous experiment, increasing the number of parents no longer resulted in improved performance in the average case.

Within the data gathered for the Knapsack Problem (shown in Figure 19), the use of elitist schema overlays decreased the average performance of all operators by an average 0.05 percentage points; however, when utilized with traditional Genetic Algorithms, elitist schema overlays increased performance by 0.18%. Overall, the range of performance was small, with the worst average solution found evaluated at 0.51% above optimal. As before, the worst performance for any singular problem size were traditional Genetic Algorithms with instances of 500 items. Additionally, our previous observation that increasing the arity of the operators improved performance continued to hold true. The only exception to this trend occurred when higher arity operators were used in conjunction with each other and elitist schema overlays.

The Traveling Salesman Problem instances tested showed similar trends to the experiments with the Knapsack Problem (shown in Figure 20). When elitist schema overlays were used, the solutions found by traditional Genetic Algorithms were 78.03 percentage points closer to the best found solution when

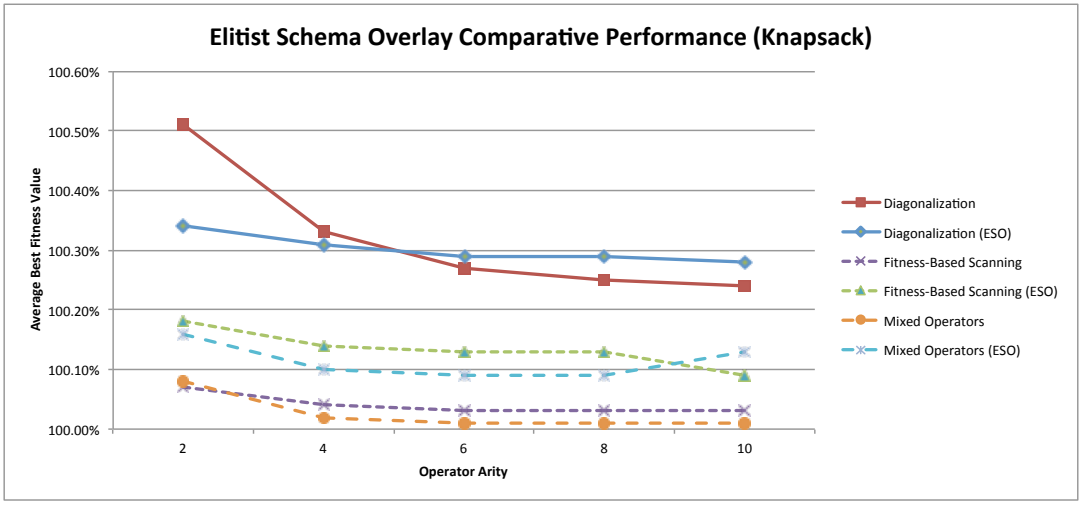

Figure 19: Comparative Performance on the Knapsack Problem

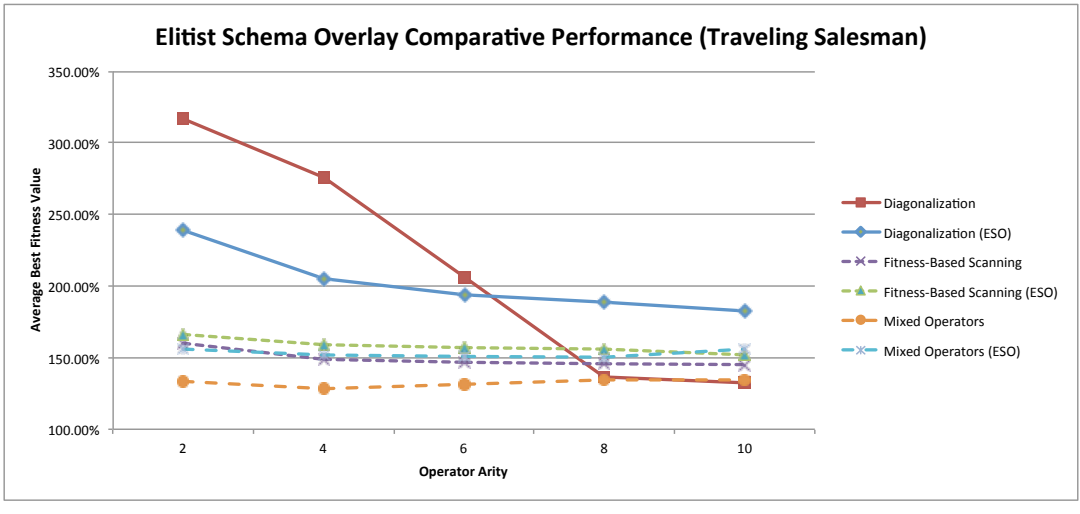

Figure 20: Comparative Performance on the Traveling Salesman Problem

averaged across all instances and problem sizes. Diagonalization with arity 4 also had significant gains when utilizing elitist schema overlays, with solutions improving on average by 70.75 percentage points; however, on average performance dropped by 5.92 percentage points.

NK-Landscape problem instances generated the same behavior as before, and elitist schema overlays consistently converged to the best found solution across all trials, regardless of which operator it was used in conjunction with. Thus, little more can be gained from examining that data. Now that we have

finished analyzing the data concerning solution quality, we will examine how elitist schema overlays affect the runtime of Genetic Algorithms empirically.

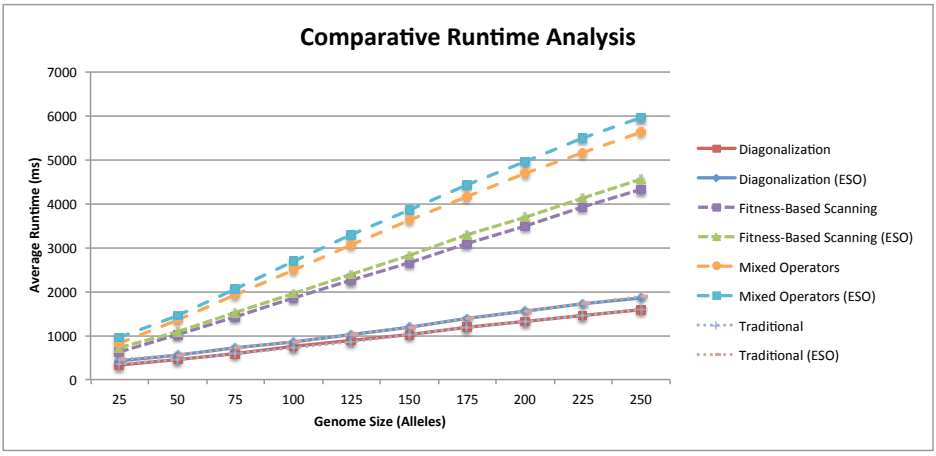

Figure 21: Comparative Runtime Analysis

The efficiency data in Figure 21 is drawn from runtimes while minimizing instances of increasing size of the first De Jong function over 150 generations. A numerical optimization function was chosen to limit error stemming from variance in the time required to read the input files that stored instances of NP-Hard problems. De Jong's Hypersphere function was specifically selected since the population converged to the optimal solution in each of our experiments. This allowed us to account for variance in the time necessary to construct elitist schema overlays with respect to the similarity of high performing genetic sequences. Along the x-axis, we have listed the sizes of the genetic sequences tested, and along the y-axis we measure the average runtime. This data gives us a look at how our choice of operators impact the overall runtime with respect to the size of the problem.

To determine if elitist schema overlays are prohibitively expensive in practice, we compared the runtimes of traditional Genetic Algorithms and multi-parent operators of arity 4 both with and without elitist schema overlays on instances of increasing size of De Jong's Hypersphere Function (shown in Figure 21). This data illuminates the cost of utilizing elitist schema overlays, and also shows how expensive traditional and multi-parent operators are in relation to each other. Traditional Genetic Algorithms ran the quickest in all instances, narrowly outperforming diagonalization. Fitness-based scanning took significantly longer than both of these, and multi-operators naturally took longer still. In each instance, the addition of elitist schema overlays slowed genetic algorithms down, but only slightly. Increasing the number of parents used in diagonalization from 2 to 4 had no major impact on the runtime; this indicates that the type of operators used has the strongest effect on runtime. Now that we have analyzed all of the data collected, we will draw our final conclusions.

7 Conclusions and Further Work

In this paper, we have introduced elitist schema overlays and their conceptual basis, genetic overlays. We tested this new genetic operator against two multi-parent genetic operators from the literature, diagonal crossover and fitness-based scanning [8]. We have also utilized the multi-parent operators in multi-operator configurations, a concept that has also been supported by the literature [25].

Our data supports the hypothesis that multi-parent strategies are an improvement upon traditional Genetic Algorithms. When paired with traditional Genetic Algorithms, elitist schema overlays improved performance in most cases, especially while testing instances of the Traveling Salesman Problem. Traditional Genetic Algorithms were also out performed by the operators from the literature: n-arity diagonalization, fitness-based scanning, and multi-operator strategies.

While elitist schema overlays increased the performance of traditional Genetic Algorithms, this effect did not carry over when elitist schema overlays were used with multi-parent operators. In most of the observed cases, adding elitist schema overlays to other n-arity genetic operators degraded performance; however, they still outperformed traditional Genetic Algorithms in most cases. While elitist schema overlays could not match the successes of other multi-parent operators, they provide another method to improve upon traditional paradigms.

Further Research

During our research, we came across and developed several unanswered questions that could be the focus of further research into elitist schema overlays and multi-parent Genetic Algorithms.

- What do these results tell us about the building block hypothesis and the schema theorem?
- What other ways of choosing and modifying the k value in an elitist schema overlay will produce better behavior, and why?
- What other strategies exist for creating effective genetic overlays?
- Why do the different percent values for k perform so differently from each other, especially in terms of convergence?
- Why do elitist schema overlays only produce gains when used in conjunction with traditional Genetic Algorithms, and degrade performance when used with other multi-parent genetic operators?
- How do high arity operators, with arities set close to the size of the genetic sequence, behave?
- What can be done to improve the computational performance of fitness-based scanning?

Acknowledgments

I would like to thank Professor Mark Liffiton for guiding me through this research project. Additionally, I would also like to Illinois Wesleyan University for providing both Professor Liffiton and Professor Andrew Shallue the resources to fund the cluster used for this research, and for allowing me to utilize it. I would also like to extend thanks to Professor Pablo Moscato and Professor Gusz Eiben for their time and for suggesting additional readings. Klaus Meffert and the team behind JGAP were also very helpful and generous for releasing their code as open source software.

References

[1] J. Andre, P. Siarry, and T. Dognon. An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. *Advances in Engineering Software*, 32(1):49 – 60, 2001.

[2] T. Bäck. Optimal mutation rates in genetic search. 1993.

[3] K. Burjorjee. The fundamental problem with the building block hypothesis. *arXiv preprint arXiv:0810.3356*, 2008.

[4] K. Deb and S. Agrawal. Understanding interactions among genetic algorithm parameters. *Foundations of Genetic Algorithms*, pages 265–286, 1999.

[5] A.E. Eiben. A method for designing decision support systems for operational planning. PhD Thesis, 1991.

[6] A.E. Eiben. Multiparent recombination in evolutionary computing. *Advances in evolutionary computing*, pages 175–192, 2003.

[7] A.E. Eiben and T. Bäck. Empirical investigation of multiparent recombination operators in evolution strategies. *Evolutionary Computation*, 5(3):347–365, 1997.

[8] A.E. Eiben, P. Raué, and Zs. Ruttkay. Genetic algorithms with multiparent recombination. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature — PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 78–87. Springer Berlin / Heidelberg, 1994.

[9] A.E. Eiben and C. Schippers. Multi-parent's niche: n-ary crossovers on nk-landscapes. *Parallel Problem Solving from Nature-PPSN IV*, pages 319–328, 1996.

[10] A.E. Eiben, C. van Kemenade, and J. Kok. Orgy in the computer: Multiparent reproduction in genetic algorithms. In Federico Morán, Alvaro

Moreno, Juan Merelo, and Pablo Chacón, editors, *Advances in Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 934–945. Springer Berlin / Heidelberg, 1995.

[11] S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. *Ann Arbor*, 1001:48109, 1993.

[12] D.E. Goldberg. Genetic algorithms in search, optimization, and machine learning. *Addison-Wesley Professional*, 1989.

[13] Y. Haxhimusa, E. Carpenter, J. Catrambone, D. Foldes, E. Stefanov, L. Arns, and Z. Pizlo. 2d and 3d traveling salesman problem. *Journal of Problem Solving*, 3(2):167–193, 2011.

[14] J. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. University of Michigan Press (Ann Arbor), 1975.

[15] S.A. Kauffman. *The origins of order: Self-organization and selection in evolution*. Oxford University Press, USA, 1993.

[16] P. Larranaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170, 1999.

[17] K. Meffert. Jgap - java genetic algorithms and genetic programming package. http://jgap.sf.net/, 2012.

[18] Z. Michalewicz. *Genetic algorithms+ data structures= evolution programs*. Springer, 1998.

[19] M. Molga and C. Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 2005.

[20] F. Neri, C. Cotta, and P. Moscato. *Handbook of memetic algorithms*. Springer, 2011.

[21] J Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63:337–370, 1996.

[22] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.

[23] N.I. Senaratna. Genetic algorithms: The crossover-mutation debate. *Degree of Bachelor of Computer Science of the University of Colombo*, 2005.

[24] B. Skellett, B. Cairns, N. Geard, B. Tonkes, and J. Wiles. Maximally rugged nk landscapes contain the highest peaks. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 579–584. Association for Computing Machinery, 2005.

[25] S. Smith. Using multiple genetic operators to reduce premature convergence in genetic assembly planning. *Computers in Industry*, 54(1):35–49, 2004.

[26] C. Stephens and H. Waelbroeck. Schemata evolution and building blocks. *Evolutionary computation*, 7(2):109–124, 1999.

[27] D. Sudholt. Crossover speeds up building-block assembly. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 689–702. ACM, 2012.

[28] S. Tsutsui, M. Yamamura, and T. Higuchi. Multi-parent recombination with simplex crossover in real coded genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 657–664, 1999.

[29] E.D. Weinberger. Np completeness of kauffman's nk model, a tuneably rugged fitness landscape. *Santa Fe Institute Technical Reports*, 1996.

[30] E. W. Weisstein. Dirac's theorem. http://mathworld.wolfram.com/DiracsTheorem.html. Visited on 19/3/2013.

[31] D. Whitley and S.B. Rana. Representation, search and genetic algorithms. In *Proceedings of the National Conference on Artificial Intelligence*, pages 497–502. John Wiley & Sons LTD, 1997.

[32] X.S. Yang. Test problems in optimization. *arXiv prepint arXiv:1008.0549*, 2010.