A Study of Fixed-Length Subset Recombination

Kelly D. Crawford
Amoco Corporation
kcrawford@amoco.com

Cory J. Hoelting
The University of Tulsa
hoeltc@euler.mcs.utulsa.edu

Roger L. Wainwright
The University of Tulsa
rogerw@penguin.mcs.utulsa.edu

Dale A. Schoenefeld
The University of Tulsa
schoend@utulsa.edu

Abstract

While bit-based, order-based and real-valued genetic algorithms have been well-studied in the literature, the fixed-length subset representation has received relatively little attention. We discuss various crossover operators for this representation and the pitfalls associated with each. In particular, we explore the ratio of the subset size to the set size. This important ratio is a major contributor to the rate of convergence.

1 INTRODUCTION

Given a set, S, of the integers from 1 to N inclusive, a fixed-length subset is defined to be any subset of S of cardinality n. For example, if $N = 4$ and $n = 2$, then $S = \{1, 2, 3, 4\}$, and the fixed-length subsets of size n are {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, and {3,4}.

A fixed-length subset problem is one where candidate solutions are represented by fixed-length subsets. There are numerous examples of fixed-length subset problems in the literature. Radcliffe (1991, 1993) described improved crossover operators for fixed-length subsets. Radcliffe and George (1993) examined fixed-length subset recombination on a highly epistatic function they developed. Lucasius and Kateman (1992) developed crossover heuristics for a fixed-length subset problem in chemometrics. Crawford, Wainwright and Vasicek (1992, 1995), constructed a genetic algorithm to search for n outliers in a set of regression data of size N (Crawford and Wainwright, 1995). Hoelting, *et al.* (1995), used fixed-length subsets to find solutions to the p-median problem.

The effect of fixed-length subset crossover on convergence in genetic algorithms has not been studied in detail. This paper examines the problem. We begin with an overview of some existing representations and operators for fixed-length subset crossover. The problems with these representations and operators are then examined. Detailed analysis of the better operators leads to insight into the pitfalls of dealing with fixed-length subsets. Of particular interest is the ratio between the set size and the fixed-length subset size. Finally, we present a method that takes this ratio into account and show successful test results.

2 REPRESENTATION AND RECOMBINATION

There are numerous ways to encode a fixed-length subset. One could use a list of N bits, B, where $B_i = 1$ indicates that data point i is part of the subset. This representation requires exactly n 1-bits, and $N - n$ 0-bits. A permutation of N integers could also be used, where the first n elements (or some set of n elements) represent the subset. A simpler, more intuitive representation, referred to as the subset encoding, is to keep a list of n distinct integers from the set $\{1, ..., N\}$.

There are problems with recombination for any of these encodings. The 2-change operator is the simplest. Randomly select an element in the subset, randomly select an element outside the subset, and exchange them. This is a one-parent operator, however, and is more properly termed a mutation than a crossover. We will use the 2-change operator for comparison.

The permutation encoding can make use of many existing operators, making it an initially attractive representation. However, without modifications, there is no guarantee that the n elements in question will be changed after crossover. There are $N!$ possible permutations, but only $\binom{N}{n}$ possible subsets. Each subset is represented $n!(N - n)!$ different ways with the permutation encoding.

Bit encoding is 1-1 and onto. Subset encoding represents each subset only $n!$ times, rather than $n!(N - n)!$ times. However, in either case, standard crossover on these chromosomes usually produces invalid offspring. With the bit encoding, 1-point, 2-point, or uniform crossover will often produce offspring where the number of 1-bits is not equal to n. Similar crossover for the subset encoding often results in offspring that contain duplicate alleles, leaving less than n distinct alleles. Penalty functions or post-crossover fixups must be applied to address these problems. For example, using subset encoding, uniform crossover with fixup (UXF) lines up the parents and crosses them over in the typical uniform fashion. After this, duplicate alleles are removed from the offspring and replaced by alleles not already found in the offspring.

UXF is a *locus-based* crossover operator, only exchanging alleles between parents at fixed loci. To illustrate this, notice that an allele at locus i will never be moved to any locus j, $j \neq i$. The only exception to this is when duplicate alleles are removed from the offspring during the fixup stage, potentially resulting in offspring alleles not found in either parent. This is an example of an *implicit mutation*. An *explicit mutation* is the use of a mutation operator in the normal course of the genetic algorithm.

The first true subset crossover, RAR_w, was proposed by Radcliffe (1993). Using subset encoding, parents are compared for similarity. While only the members of the subset are stored, the members outside the subset, referred to as *barred* alleles, are also considered.

Four categories of alleles are identified. The subset of alleles common to both parents is denoted by w_2, those that show up in neither parent by $\bar{w}_2$ (i.e., those that show up as barred alleles in both parents), those found in only one parent by w_1, and those that show up as barred alleles in only one parent by $\bar{w}_1$. Note that $\bar{w}_1$ always contains exactly the barred versions of w_1.

RAR_w works by filling a "draw bag" with both regular and barred alleles. These alleles are then randomly drawn (without replacement) from the bag (or, *multiset*) and used to create new offspring. The w in RAR_w determines the ratio of the number of copies of w_2 and $\bar{w}_2$ alleles versus the number of copies of w_1 and $\bar{w}_1$ alleles to place into the draw bag. Note that in RAR_w, the order of alleles in the chromosome is ignored, making it a *set-based* operator (as opposed to a *locus-based* operator).

When a barred allele is drawn from the bag, it means that this allele cannot show up in the offspring. When a regular allele is drawn, it is placed into the offspring unless the matching barred allele has already been drawn. An interesting note is that after $N - n$ barred alleles have been drawn, the offspring is fully specified as the n remaining alleles. This special case can result in the offspring containing alleles not found in either parent (although the barred alleles are found in both parents). Complete implementation details can be found in Radcliffe (1993).

If the offspring is not fully specified after the bag has been emptied, alleles are randomly generated and placed into the offspring. Note that this can only happen for the special cases of RAR_0 and RAR_∞. In RAR_0, no w_2 or $\bar{w}_2$ alleles will ever be placed into the draw bag. In RAR_∞, no w_1 or $\bar{w}_1$ alleles will ever be placed into the draw bag. The only way a bag can be emptied without fully specifying an offspring is if it does not contain at least $N - n$ barred alleles. Thus, to empty the bag, we must have $|\bar{w}_1| + |\bar{w}_2| < N - n$. As long as the w in RAR_w is not 0 or ∞, this inequality never holds.

To illustrate a typical application of RAR_w, suppose $N = 6$ and $n = 3$. We have $\{1, 2, 3, 4, 5, 6\}$ as possible alleles, and $\{\bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}, \bar{6}\}$ as possible barred alleles. Given the parents $p_1 = \{1, 2, 3\}$ and $p_2 = \{2, 3, 4\}$ (which implies $p_1' = \{4, 5, 6\}$ and $p_2' = \{1, 5, 6\}$) we identify the following sets: $w_1 = \{1, 4\}$, $\bar{w}_1 = \{\bar{1}, \bar{4}\}$, $w_2 = \{2, 3\}$, $\bar{w}_2 = \{\bar{5}, \bar{6}\}$. Assume we are using $w = 2$ (i.e., RAR_2). This means we will place 1 copy each of all w_1 and $\bar{w}_1$ alleles and 2 copies each of all w_2 and $\bar{w}_2$ alleles into the bag. Thus, we place 1 copy each of 1, 4, $\bar{1}$ and $\bar{4}$ into the bag, and 2 copies each of 2, 3, $\bar{5}$ and $\bar{6}$ into the bag. In fact, this is exactly one copy of each of the alleles (normal and barred) found in both parents, making RAR_2 an intuitive starting choice. The bag now contains $\{1, 4, \bar{1}, \bar{4}, 2, 2, 3, 3, \bar{5}, \bar{5}, \bar{6}, \bar{6}\}$. Finally, we randomly draw from the bag (without replacement) until we have constructed an offspring as illustrated in Table 1. The resulting offspring is $\{2, 3, 4\}$. As another example, suppose the first three draws were $\bar{1}$, $\bar{4}$ and $\bar{5}$. At this point we can quit drawing alleles from the bag. We have now fully specified the offspring as $\{2, 3, 6\}$. Note that the allele 6 is not found in either parent, making this an example of an implicit mutation.

A weakness of other operators is that they typically ignore anything outside the subset. This leads to loss of diversity in the population and premature convergence. RAR_w still suffers from this problem, but is a much improved operator due to the introduction of barred alleles.

draw	offspring				
3			3		
$\bar{5}$			3		$\bar{5}$
$\bar{1}$	$\bar{1}$		3		$\bar{5}$
$\bar{5}$	$\bar{1}$		3		$\bar{5}$
1	$\bar{1}$		3		$\bar{5}$
4	$\bar{1}$		3	4	$\bar{5}$
$\bar{4}$	$\bar{1}$		3	4	$\bar{5}$
2	$\bar{1}$	2	3	4	$\bar{5}$

Table 1: RAR_2 example

3 RECOMBINATION PITFALLS

In actual tests, we found the fixed-length subset operators lacking, with premature convergence as the most common problem. However, since the available literature on the subject reported a fair measure of success, further analysis was required. We surmised that crossover was the problem. To test this conjecture, we devised a method to study the crossover operators themselves. We termed this tool *static analysis* (Crawford, 1996).

Static analysis is the use of a genetic algorithm to study crossover in the absence of selection pressure, fitness and mutation. The objective is not fitness related, but instead is simply to see how long it takes for a population to unify when only crossover is a factor. Such an analysis will provide a measure of the inherent bias of a crossover operator. By using a fitness function that always returns 1, setting the crossover rate to 1 and setting the mutation rate to 0, almost any genetic algorithm package can be turned into a static analyzer. For our tests we used the LibGA package (Corcoran and Wainwright, 1993) under the generational model (subsequent testing of the steady-state model produced similar results).

To understand why a population would unify under static analysis, consider the nature of locus-based crossover operators. Viewing a population of bit strings as a set of columns, notice that an allele in column i (i.e., locus i), will only be exchanged with other alleles in column i. Now consider what happens when you randomly select alleles from one of these columns in order to create a new column. Over time, sampling error (because we have a finite population size) will unify the column. A single allele will begin to dominate the column, and eventually, the entire column will contain a single allele. Note that as the population size increases, the number of generations required for unification increases.

All bit-based crossover operators we tested eventually unified their populations under static analysis (the final chromosome was different for each initial random seed). Tests showed that in general, if it takes G generations to unify the population, between 80% and 90% of the alleles are lost after only $\frac{G}{2}$ generations. Explicit mutation operators never unify under static analysis. Without selection pressure, they simply produce random movement within the search space.

For bit-based and order-based crossover operators, testing showed that the rate of convergence under static analysis is a function of the operator, the population size and the chromosome length. For fixed-length subset crossover, however, it is a function of the operator, the population size, the set size N and the subset size n. This was our first indication

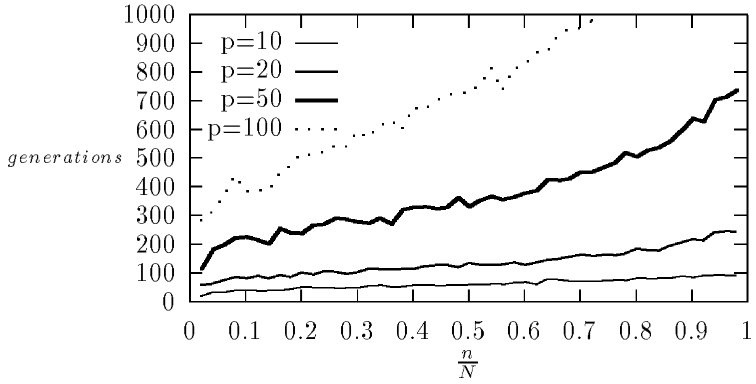

Figure 1: Static analysis convergence for UXF, $N = 100$

of the importance of the ratio between n and N.

The performance of each operator varied with respect to $\frac{n}{N}$. This variation is difficult to quantify since the behavior of the objective function also changes as n changes. Rather than behaving as a function parameter, changes in n tend to create new fitness landscapes. Nonetheless, this observation is critical to understanding and analyzing the behavior of fixed-length subset operators. We will limit this discussion to subset encodings using the crossover operators UXF and RAR_w.

3.1 UXF

Static analysis provided us with much insight into the behavior of UXF. As the ratio between $\frac{n}{N}$ varies, the number of generations required to converge the population also varies. Figure 1 shows results of static analysis tests for $N = 100$. The x-axis shows the different values of n expressed as a ratio with N, the y-axis shows the number of generations required for the entire population to unify, and the population size is represented by p. The same information is presented in Figure 2, but with the number of generations divided by the population size. Note the correlation between the different lines, especially for lower values of n.

Overall, empirical testing (see Section 4.3) showed very poor performance for small values of $\frac{n}{N}$, with somewhat better performance for larger values. Static analysis supports these results by showing that even in the absence of selection pressure, UXF converges quickly for small ratios. When selection pressure is added, premature convergence is the result.

It is important to note that although order is not important in a fixed-length subset problem, UXF still respects order, being a locus-based crossover operator. Thus, rather than having $\binom{N}{n}$ possible subsets, UXF considers a search space of size $n!\binom{N}{n} = \frac{N!}{(N-n)!}$. In other words, there is a considerable amount of duplication in the search space imposed by UXF. There are two major implications of this analysis. First, since for larger values of n, $n! >> \binom{N}{n}$, the search space is much larger than required (although proportionately, there is a like increase

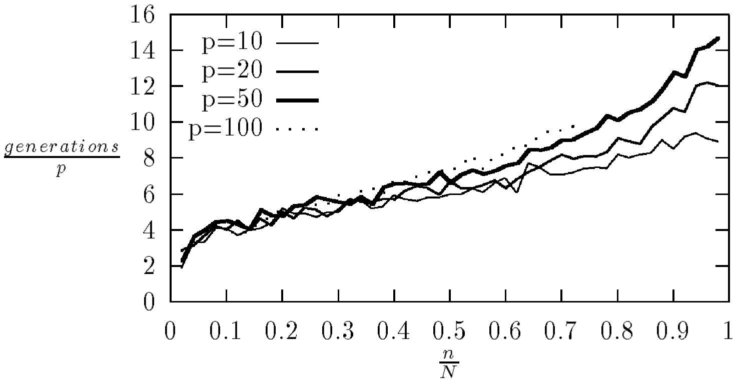

Figure 2: Static analysis ratio of convergence for UXF, $N = 100$

in the number of solutions). Second, while $n!$ continues to grow as n gets closer to N, $\binom{N}{n}$ reaches its peak at $n = \frac{N}{2}$. In fact, for all n, $\binom{N}{n} = \binom{N}{N-n}$. The result is more and more duplication as n increases.

Given the static analysis results, our empirical test results and the success reported in the literature for other fixed-length subset problems, it became clear that the culprit was the ratio between n and N. The tests shown in the literature always used $\frac{n}{N} = .5$, or very nearly so. Our tests were usually concerned with $\frac{n}{N} < .2$. This observation was the initial basis for our research into fixed-length subset crossover.

One might conjecture that since UXF performs better for $\frac{n}{N}$ close to 1, a complement encoding could be used for small n. In a subset encoding, the list of n integers corresponding to the subset is stored in the chromosome. In a complement encoding, the list of $N - n$ integers corresponding to the members outside the subset is kept in the chromosome. However, empirical testing showed the complement encoding to perform more or less the same as subset encoding.

UXF was a consistently poor performer in all of our tests, often being beaten by a simple 2-change operator. The locus-based nature of UXF together with the problem of the ratio $\frac{n}{N}$ combine to make it a poor choice for fixed-length subset crossover.

3.2 RAR_w

Given the effect of the ratio $\frac{n}{N}$ on UXF, we suspected the same for RAR_w. Static analysis, however, was initially inconclusive. After every crossover operator that was tested converged a population to a single chromosome under static analysis, it was initially surprising to find that RAR_w did not. However, the set-based nature of RAR_w prevented unification. Alleles would disappear and reappear in the population over time, with the total number of alleles lost at any one time hovering around a fixed number. This fixed number is a function of population size, w, N and n. Table 2 shows the results of static analysis tests for various n

Operator	n								
	10	20	30	40	50	60	70	80	90
$RAR_{0.5}$	2.78	0.11	0.01	0.00	0.00	0.00	0.00	0.00	0.00
RAR_1	3.21	0.17	0.01	0.00	0.00	0.00	0.00	0.00	0.00
RAR_2	4.55	0.61	0.11	0.02	0.00	0.00	0.00	0.00	0.00
RAR_4	13.45	6.79	3.87	2.14	1.01	0.46	0.10	0.01	0.00
RAR_8	43.37	35.77	28.65	21.96	13.39	8.86	4.68	1.62	0.17

Table 2: RAR_w maximum allele loss under static analysis for $N = 100$

sizes with $N = 100$ and a population size of 100.

Clearly, the ratio $\frac{n}{N}$ still plays a role in RAR_w. We define a difference measure (D) between parents that will assist us in our analysis (Crawford, 1996). Given two parents, each of length n, we count the number of alleles in common (this is $|w_2|$). The difference measure, D, is defined to be $n - |w_2|$. Assuming $\frac{n}{N} \leq 0.5$, identical parents will have $D = 0$, while parents with no alleles in common will have $D = n$.

We will refer to the four sets w_1, $\bar{w}_1$, w_2 and $\bar{w}_2$ collectively as w^*. The relationships between D and the cardinality of the w^* are shown in equations 1 through 4.

$$|w_1| = 2 * D \tag{1}$$
$$|\bar{w}_1| = 2 * D \tag{2}$$
$$|w_2| = n - D \tag{3}$$
$$|\bar{w}_2| = N - (D + n) \tag{4}$$

Consider the case where $N = 8$. We can quickly calculate the cardinality of the w^* for $n = 2..6$. To demonstrate, we constructed representative pairs of parents in Table 3 to cover all values of D for $n = 3$. We can do likewise for the other values of n. From this, we can construct Table 4, Table 5 and Table 6, which show the cardinality of the w^* for $n = 2$, $n = 3$ and $n = 4$, respectively. For example, Table 5 shows that for $n = 3$ and $D = 2$ we will have 4 alleles of type w_1, 4 of type $\bar{w}_1$, 1 of type w_2 and 3 of type $\bar{w}_2$. Thus, we can quickly see the distribution of alleles in the bag for RAR_w based on the difference measure D and the subset size n. Note that the table for $n = 5$ will be the same as Table 5 (with $n = 3$), except that w_2 and $\bar{w}_2$ are switched. The same is true for $n = 2$ and $n = 6$.

It is clear that for $\frac{n}{N} \leq 0.5$ the largest D is n and for $\frac{n}{N} > 0.5$ the largest D is $N - n$. Figure 3 illustrates. Note that the largest maximum D value occurs at $\frac{n}{N} = 0.5$.

Initially, when the population is most random, the average D will be at its highest. Over time, as the population begins to converge, the average D decreases. When the average D reaches 0, the population has unified. When D is 0, RAR_w will always produce offspring identical to the parents (except for RAR_0, where w_2 and $\bar{w}_2$ alleles are never placed into the bag).

Again, with static analysis suggesting that RAR_w suffers less from allele loss when $\frac{n}{N}$ is close to 1, why not use a complement encoding when $\frac{n}{N}$ is small? The reasons for this are that the

D	parents					
0	a	b	c			
	a	b	c			
1	a	b	c			
		b	c	d		
2	a	b	c			
			c	d	e	
3	a	b	c			
				d	e	f

Table 3: Example parents representing all possible D values for $N = 8$, $n = 3$

	D		
	0	1	2
$\|w_1\|$	0	2	4
$\|\bar{w}_1\|$	0	2	4
$\|w_2\|$	2	1	0
$\|\bar{w}_2\|$	6	5	4

Table 4: $N = 8$, $n = 2$

	D			
	0	1	2	3
$\|w_1\|$	0	2	4	6
$\|\bar{w}_1\|$	0	2	4	6
$\|w_2\|$	3	2	1	0
$\|\bar{w}_2\|$	5	4	3	2

Table 5: $N = 8$, $n = 3$

	D				
	0	1	2	3	4
$\|w_1\|$	0	2	4	6	8
$\|\bar{w}_1\|$	0	2	4	6	8
$\|w_2\|$	4	3	2	1	0
$\|\bar{w}_2\|$	4	3	2	1	0

Table 6: $N = 8$, $n = 4$

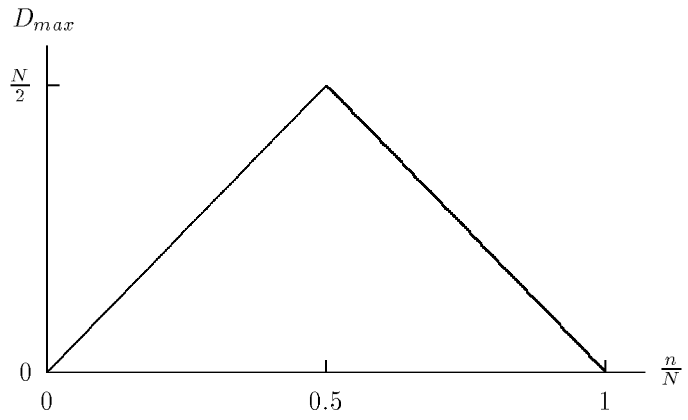

Figure 3: Maximum possible D values relative to n and N

search space size is the same, i.e. $\binom{N}{n} = \binom{N}{N-n}$, the fitness landscape is the same, and the maximum D value is the same. To illustrate, consider the following example. Let $N = 8$ and $n = 2$ for RAR_1. Assume we have $\{a, b\}$ and $\{b, c\}$ as our two subsets. These two subsets are represented by themselves in subset encoding. In complement encoding, these two subsets are represented by $\{c, d, e, f, g, h\}$ and $\{a, d, e, f, g, h\}$. When recombining the two subsets in subset encoding we have $w_1 = \{a, c\}$, $\bar{w}_1 = \{\bar{a}, \bar{c}\}$, $w_2 = \{b\}$, and $\bar{w}_2 = \{\bar{d}, \bar{e}, \bar{f}, \bar{g}, \bar{h}\}$. When recombining in complement encoding we have $w_1 = \{a, c\}$, $\bar{w}_1 = \{\bar{a}, \bar{c}\}$, $w_2 = \{d, e, f, g, h\}$, and $\bar{w}_2 = \{\bar{b}\}$. Hence, over all sequences of random draws from the bag, we are just as likely to draw $\bar{d}$ in the subset encoding before the individual is fully specified as we are to draw d in the complement encoding. This can be said for any allele or barred allele in the subset encoding and its corresponding barred allele or allele in the complement encoding. Furthermore, the meaning of an allele in subset encoding is the same as the meaning of its corresponding barred allele in complement encoding. Therefore, subset encoding will have, on the average, the same performance as complement encoding. Our empirical tests agreed with this analysis.

4 RECOMBINATION MODIFICATIONS

We attempted several modifications to RAR_w in order to alleviate the problem with small ratios of $\frac{n}{N}$ (Crawford, 1996). We first noted that RAR_w mandated that the number of copies of w_1 and $\bar{w}_1$ would be the same. Likewise for w_2 and $\bar{w}_2$. Separate controls for all four allele types might prove useful. Second, we note that there is a symmetry of the distribution of the w^* cardinalities in the draw bag when $\frac{n}{N} = .5$. Other ratios show the bag more skewed, so perhaps a forced symmetry in the draw bag would be helpful. Lastly, as $\frac{n}{N}$ ratios approach 0 or 1, the maximum range of D decreases (as shown in Figure 3). If we could set an average D threshhold for a population, we might be able to delay convergence long enough to find a solution.

	D				
	0	1	2	3	4
$\|w_1\|$	0	2	4	6	8
$\|\bar{w}_1\|$	0	2	4	6	8
$\|w_2\|$	4	3	2	1	0
$\|\bar{w}_2\|$	96	95	94	93	92

Table 7: $N = 100$, $n = 4$

4.1 Extended RAR (ERAR)

$ERAR$ takes 4 arguments: $|w_1|$, $|\bar{w}_1|$, $|w_2|$ and $|\bar{w}_2|$. Rather than a ratio, as in RAR_w, the arguments to $ERAR$ are the actual number of copies of each w^* allele type to put into the bag. For all cases where $|w_1| = |\bar{w}_1|$ and $|w_2| = |\bar{w}_2|$, $ERAR$ is equivalent to RAR_w, where $w = |w_2|/|w_1|$. This additional flexibility allows us to experiment with decreasing the number of barred alleles for smaller ratios of $\frac{n}{N}$. Tests showed a modest improvement in a few cases. However, it was not clear how to properly set the arguments to $ERAR$ or whether this would actually be useful in the general case.

4.2 Adjusting RAR (ARAR)

Note the symmetry for $n = 4$ as shown in Table 6. Also note that the w_1 and $\bar{w}_1$ values, when reversed, are the w_2 and $\bar{w}_2$ values times 2. This symmetry disappears as n moves away from $\frac{N}{2}$. This is particularly striking for larger values of N. For example, consider Table 7 where $N = 100$ and $n = 4$. Under RAR_2, $\bar{w}_2$ always dominates the draw bag. In Table 7, for $D = 2$, we see that $\bar{w}_2$ has a total of 188 entries in the bag (2 copies of each of the 94 alleles). w_1, $\bar{w}_1$ and w_2 have 4 entries each. Most of the time is spent selecting values outside the subset that will not be considered for insertion into the offspring. In other words, most of the alleles in the offspring are primarily chosen by drawing barred alleles, and this is true regardless of D.

Using lower values of w (e.g., $RAR_{0.1}$) is one way to balance the bag distribution. Another is to construct an operator that will balance the bag as it is being constructed. We call this operator ARAR. We ran tests where the bag distribution was forced to be the same as when $\frac{n}{N} = 0.5$ (easily computed during an initialization step). As with ERAR, success was modest and varied with the fitness function. In most cases our testing did not support our initial assumption that a lack of symmetry in the bag distribution affects convergence.

4.3 Threshholding RAR (TRAR)

Our third modification was very successful. We observe that the genetic algorithm converges faster when w_2 and $\bar{w}_2$ alleles are selected from the bag. These alleles are selected more often as the population converges since most alleles will fall into these two categories as D decreases. Thus, convergence is accelerated.

We can detect when the population begins to converge by keeping track of the average value of D during crossover. When D_{avg} drops below a specified threshhold value, $TRAR$ temporarily sets the number of draw bag copies of the w_2 and $\bar{w}_2$ alleles to 0 until D_{avg}

Sequence	How Many Correct in the Sequence	Epistatic Credit
1:{0,1}	0	1
	1	0
	2	2
2:{2,3}	0	0
	1	1
	2	2
3:{4,5}	0	0
	1	1
	2	2
4:{6,7}	0	1
	1	0
	2	2
5:{8,9}	0	1
	1	0
	2	2

Table 8: Example credit for epistatic function with $N = 100$ and $n = 10$

climbs back above the threshhold.

To illustrate the success of threshholding we show empirical results on a slightly modified version of a function developed by Radcliffe and George (1993). Given a set of size N containing the integers modulo N ($\{0, 1, 2, ..., N-2, N-1\}$), the goal is to seek the size n subset containing the lowest n integers. Thus, the optimal solution will be $\{0, 1, 2, ..., n-2, n-1\}$. Each correct element of the subset is assigned 1 point, so the maximum fitness value is n.

As stated, this function is non-epistatic and quite easy for a genetic algorithm to solve. To add epistasis, the elements of the size n subset are grouped into pairs, namely $\{0, 1\}$, {2,3}, {4,5}, etc. If both elements of a pair show up somewhere in the subset, 2 points are added to the fitness. The fitnesses for finding 0 and 1 elements of a particular pair will be randomly determined at the start of the run to be 0 and 1, or 1 and 0, respectively. Thus, having 0 elements of some pairs will add 0 to the fitness, while 0 elements of other pairs will add 1 to the fitness. This makes the function very deceptive.

For $N = 100$ and $n = 10$, Table 8 shows an example of the randomly generated function. Note that sequence number 2 (which corresponds to the set {2,3}), adds credit of 0, 1 and 2 points to the fitness when 0, 1 and 2 members of the sequence are found in the chromosome, respectively. Sequence number 4, however, adds credit of 1, 0 and 2 points to the fitness when 0, 1 and 2 members are found.

Table 9 illustrates how the function described in Table 8 is used to evaluate 5 sample chromosomes. Consider child 3 in the top section of Table 9. Row 3 of the bottom section shows how the allele sequences match up with the expected optimum. Sequences 1 and 5 have 1 match and are both assigned a credit of 0 according to Table 8. All others have a perfect match and are assigned a credit of 2 each. The combined fitness is 6.

Note how child 2 has 0 matches for all sequences. Even so, Table 8 shows a credit assignment

Child	Example Chromosomes										Fitness
1	0	6	8	10	11	13	22	33	70	84	0
2	11	54	70	78	92	93	94	96	98	99	3
3	1	2	3	4	5	6	7	9	10	20	6
4	0	1	2	3	4	5	60	61	62	63	8
5	0	1	2	3	4	5	6	7	8	9	10
Child	Seq 1		Seq 2		Seq 3		Seq 4		Seq 5		Fitness
1	0						6		8		0
2											3
3		1	2	3	4	5	6	7		9	6
4	0	1	2	3	4	5					8
5	0	1	2	3	4	5	6	7	8	9	10

Table 9: Example chromosomes and fitnesses for epistatic function

of 1 for the sequences 1, 4 and 5, for a combined fitness of 3. Child 1 with 3 matches has a fitness of 0, thus illustrating the epistatic qualities of this function. Incomplete sequences will often lead selection away from the solution. Complete sequences must be found and recombined intact in order to locate the optimum.

We ran a genetic algorithm on the epistatic function and gathered statistics. Each test was run 100 times with 100 different random seeds (the same seeds for each crossover operator, which guaranteed that the same 100 initial populations were used for each suite of tests). Other parameters included population size = 100, crossover rate = 1.0, mutation rate = 0.05 (per string), generational model and maximum number of generations = 200. Success was defined as finding the optimum. Over the 100 test runs, the percentage of time this optimum was found, as well as the average number of generations it took to find it, were used to benchmark the threshholding operator. Note that the average number of generations only includes those runs that found the optimal value. In cases where the optimal value was never found, the run either prematurely converged or was stopped after 200 generations.

Table 10 shows the results of the crossover tests on the epistatic function for $N = 100$ and $n = 10$. 2-change not only performs poorly, but the average number of generations was high relative to its poor performance. UXF was dismal, never locating the optimum value. The RAR_w variations showed a constant 9 to 10% success rate. When thresholding was added (in this case, the threshold = 6.5), the success rate climbed dramatically. Note that the iteration count increased, indicating that the non-threshold versions of RAR_w tended to prematurely converge. The addition of the threshold proved to be a significant aid in avoiding premature convergence.

5 CONCLUSIONS

We have examined a number of fixed-length subset recombination operators from the literature. A problem with the operators proposed to date is that none consider the important ratio $\frac{n}{N}$. We demonstrate analytically why this ratio is so important, and then support this with empirical tests. Analysis of the best fixed-length subset operator, RAR_w, provides us with insight into why it works well for $\frac{n}{N} = 0.5$, but struggles with smaller ratios. The

Crossover Operator	% of Time Optimum Found	Average Generations
2-change	7	42
UXF	0	-
$RAR_{0.1}$	10	18
$RAR_{0.25}$	9	16
$RAR_{0.5}$	10	16
$RAR_{0.75}$	9	17
$RAR_{1.0}$	10	18
$RAR_{2.0}$	9	17
$RAR_{3.0}$	9	15
$RAR_{4.0}$	9	16
$ERAR(20, 40, 20, 5)$	9	17
$ARAR(1, 2, 1, 2)$	24	17
T-$RAR_{0.1}$	63	61
T-$RAR_{0.25}$	65	55
T-$RAR_{0.5}$	68	57
T-$RAR_{0.75}$	76	56
T-$RAR_{1.0}$	63	50
T-$RAR_{2.0}$	76	55
T-$RAR_{3.0}$	68	52
T-$RAR_{4.0}$	75	52

Table 10: Epi10 test results

addition of threshholding to RAR_w showed significant performance improvements.

Acknowledgements

Special thanks to Michael Vose and Darrell Whitley for a helpful conversation at ICGA-6 that pointed us in the right direction on this work.

References

A. L. Corcoran and R. L. Wainwright. (1993) LibGA: A user-friendly workbench for order-based genetic algorithm research. In *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pages 111–117. ACM Press.

Kelly D. Crawford. (1996) *The Role of Recombination in Genetic Algorithms for Fixed-Length Subset Problems*. PhD thesis, The University of Tulsa.

Kelly D. Crawford and Roger L. Wainwright. (1995) Applying genetic algorithms to outlier detection. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann Publishers.

Kelly D. Crawford, Roger L. Wainwright, and Daniel J. Vasicek. (1992) Detecting multiple outliers in multiple dimensions using genetic algorithms. Technical Report UTULSA-MCS-92-4, The University of Tulsa, July.

Kelly D. Crawford, Roger L. Wainwright, and Daniel J. Vasicek. (1995) Detecting multiple outliers in regression data using genetic algorithms. In *Proceedings of the 1995 ACM/SIGAPP Symposium on Applied Computing*. ACM Press.

Cory J. Hoelting, Dale A. Schoenefeld, and Roger L. Wainwright. (1995) Approximation techniques for variations of the p-median problem. In *Proceedings of the 1995 ACM/SIGAPP Symposium on Applied Computing*. ACM Press.

C. B. Lucasius and G. Kateman. (1992) Towards solving subset selection problems with the aid of the genetic algorithm. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2*, Amsterdam. Elsevier Science Publishers, B. V.

Nicholas J. Radcliffe. (1991) Forma analysis and random respectful recombination. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann Publishers.

Nicholas J. Radcliffe. (1993) Genetic set recombination. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, San Mateo, CA. Morgan Kaufmann Publishers.

Nicholas J. Radcliffe and Felicity A. W. George. (1993) A study in set recombination. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann Publishers.