Linkage Tree Genetic Algorithms: Variants and Analysis

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

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

ABSTRACT

Discovering and exploiting the linkage between genes during evolutionary search allows the Linkage Tree Genetic Algorithm (LTGA) to maximize crossover effectiveness, greatly reducing both population size and total number of evaluations required to reach success on decomposable problems. This paper presents a comparative analysis of the most prominent LTGA variants and a newly introduced variant. While the deceptive trap problem (Trap-k) is one of the canonical benchmarks for testing LTGA, when LTGA is combined with applying steepest ascent hill climbing to the initial population, as is done in all significant LTGA variations, Trap-k is trivially solved. This paper introduces the deceptive step trap problem (StepTrap-k,s), which shows the novel combination of smallest first subtree ordering with global mixing (LTS-GOMEA) is effective for black box optimization, while least linked first subtree ordering (LT-GOMEA) is effective on problems where partial reevaluation is possible. Finally, nearest neighbor NK landscapes show that global mixing is not effective on problems with complex overlapping linkage structure that cannot be modeled correctly by a linkage tree, emphasizing the need to extend how LTGA stores linkage to allow the power of global mixing to be applied to these types of problems.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

LTGA, Linkage Learning, Deceptive Trap, Deceptive Step Trap, Nearest Neighbor NK Landscapes, Optimal Mixing

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

1. INTRODUCTION

Linkage learning has been shown to benefit the performance of evolutionary algorithms and a variety of linkage algorithms have been developed which use non-tree structures [3]. The canonical Linkage Tree Genetic Algorithm (LTGA) variant was introduced in 2010 [7, 8] with two promising additional variants being introduced in 2011 [5, 9]. The basis for all LTGA variants is the identification of linked substructures in the genome by examining the entropy in the current population and representing those linkages as a hierarchical tree. Each node in the tree represents a cluster of linked genes and the children of each node are non-overlapping subsets of their parent. To exploit the linkage information, LTGA performs a multistage crossover designed to allow for significant exploration without disrupting interdependent genes. The LTGA variants differ in the exact method for how they construct their linkage tree, apply it during crossover, and select parents on which to perform crossover; there has been little analysis of the advantages and disadvantages of these options. Furthermore, most of the work done with LTGA employs Steepest Ascent Hill Climbing (SAHC) on the initial population to help the initial tree construction, the effects of which warrant further investigation. Table 1 shows the principal differences between the LTGA variants in terms of option combinations. To aid in explaining how LTGA works and how all of these option combinations affect evolution, a running example is presented which uses the population in Table 2. These example individuals are evaluated using a trap-like fitness function in which the gene loci *abc*, *def*, and *ghi* are evaluated as groups, such that the gene configuration of 111 results in the best fitness and 000 the second best.

The next few sections provide an introduction to how LTGA works and explain the rationale driving each of the different option combinations. The effects of each LTGA option are discussed and some experimental evidence is provided to support this discussion. To isolate the effects of each, a new LTGA variation consisting of a unique option combination is introduced. Also, a new benchmark is introduced in Section 7 to better understand how LTGA exploits the local search on the population. Finally, Section 9 summarizes the findings and indicates which variant is best under what circumstances.

2. INITIAL POPULATION

In LTGA's canonical form, the initial population undergoes SAHC before starting evolution. In each SAHC iteration, all of an individual's neighbors are evaluated. If any

Dimension	Options
Initial Population	Random / SAHC
Clustering	Full / Pairwise
Subtree Ordering	Least Linked First / Smallest First
Crossover	Two-Parent / Global Mixing / Global Best Next
Variations	**Options Used**
Original [8, 7]	SAHC, Full, Least Linked First, Two-Parent
Original+	SAHC, Pairwise, Least Linked First, Two-Parent
Pairwise [5]	SAHC, Pairwise, Smallest First, Two-Parent
LT-GOMEA [9, 1]	SAHC, Pairwise, Least Linked First, Global Mixing
LTS-GOMEA	SAHC, Pairwise, Smallest First, Global Mixing

Table 1: Existing variations on LTGA

of those neighbors are better than the current individual, then the best neighbor replaces it. SAHC continues until no improvement can be found. On binary problems, the SAHC neighborhood consists of all individuals within one bit flip of the original individual. One reason why LTGA uses SAHC on the initial population is to discover linkage information [5]. The linkage tree is constructed using all of the individuals in the current population and is built before any type of selection occurs. As a result, if SAHC is not performed on the initial population and no other steps are taken, LTGA attempts to determine the linkage of randomly generated genes. Not only does this create a meaningless tree, [5] argues that applying those trees can be very harmful to the search's success. If SAHC is not used, they suggest some form of selection be performed prior to the initial tree creation, such as tournament selection. Table 2 provides an example of how SAHC application can discover linkage information. After SAHC is applied to the randomly initialized individuals, the linked groups are clearly visible.

Using SAHC on the initial population can result in on the order of μN^2 evaluations, where μ is population size and N is genome size. On some problems it is possible to do partial reevaluation such that neighbors of an individual can be evaluated at a much lower cost than a brand new individual [1]. In such cases, the total number of SAHC iterations is counted, which requires on the order of μN steps. As a promising offset to SAHC's cost based on population size, [1] shows how performing local search can reduce the population size required to achieve success.

	Initial	After SAHC
Individual 1	100010001	000000000
Individual 2	101100100	111000000
Individual 3	010011010	000111000
Individual 4	001001110	000000111

Table 2: Example population before and after SAHC

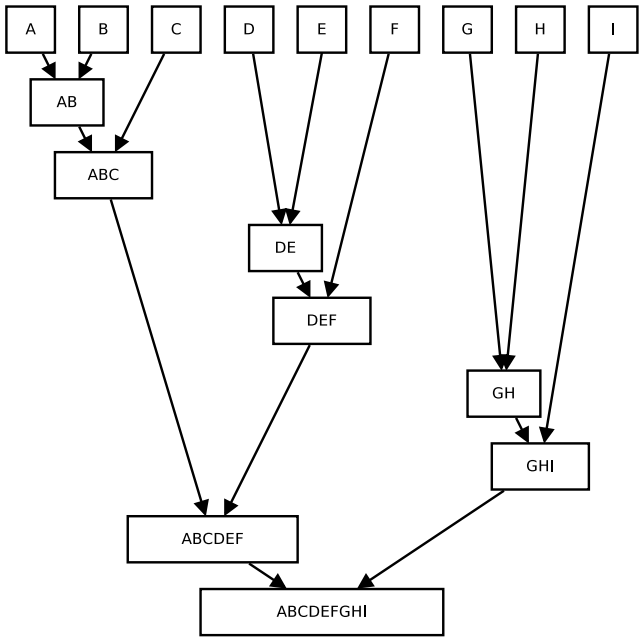

Figure 1: Example Linkage Tree

3. CONSTRUCTING TREES

LTGA stores linkage information as a tree. Each node in the tree represents a cluster of genes that LTGA believes to be linked. A tree is built by first creating a subtree with one node for each locus in the genome. LTGA then iteratively joins the two subtrees that it considers the most linked (with ties broken stochastically) into a binary tree whose root contains all of the loci of both subtrees. As a result, the number of subtrees is reduced by one each iteration, meaning the total number of nodes in the final tree is one less than twice the number of dimensions.

Figure 1 provides an example linkage tree construction for the running example. When constructing this tree, the first merge combines a and b. In the next iteration, LTGA can combine any of the remaining individual loci with each other, as well as with the newly created subtree ab, but it can no longer combine loci with just a or b. This represents one of the limitations of LTGA, since it is unable to fully capture a relationship such as a linked to b and b linked to c without a linked to c. The closest approximation LTGA can make to this type of linkage is if a is more tightly linked to b than b is to c, then the linkage tree can contain both ab and abc, which is one of the primary advantages of a tree representation over simple clusters. Also, since LTGA rebuilds the linkage tree each generation, changes in the population can allow it to shift priorities, for instance from ab to bc. It is important to note that the tree provided in Figure 1 is not the only one that can be constructed for this population and that LTGA contains no bias based on where loci are positioned in the genome. LTGA also has no bias on merging larger or smaller subtrees first.

3.1 Cluster Distance

The linkage between genes is determined using the normalized variation of information, defined by Eq. 1, where D is the distance between clusters C_i and C_j, and H is the entropy of a cluster, as given by Eq. 2. S represents all possible strings that cluster C can be, and $p_c(s)$ is the fraction of the current population that has cluster C set to s.

$$D(C_i, C_j) = 2 - \frac{H(C_i) + H(C_j)}{H(C_i \cup C_j)} \quad (1)$$

$$H(C) = -\sum_{s \in S} p_c(s) \log(p_c(s)) \quad (2)$$

As a result, $D(C_i, C_j)$ provides the ratio of the entropy of the clusters separately to the clusters joined. The minimum distance is achieved when for each s in C_i there is exactly one s' in C_j such that all individuals who have s also have s'. This method of calculating distance, measures how well the contents of one cluster can predict another, which implies they are linked.

Using the example population in Table 2 after SAHC and starting after the first merge, the distance between the cluster ab and the clusters c and d is calculated as follows. For cluster ab, $p_{ab}(00) = \frac{3}{4}$, $p_{ab}(11) = \frac{1}{4}$, and all other possible strings for ab never occur in the population. Both c and d have $p(0) = \frac{3}{4}$ and $p(1) = \frac{1}{4}$. This means that the numerator in Eq. 1 will be the same for both $D(ab, c)$ and $D(ab, d)$ and also that all three clusters have the same value for H. Joined, $p_{abc}(000) = \frac{3}{4}$ and $p_{abc}(111) = \frac{1}{4}$ for abc and $p_{abd}(000) = \frac{2}{4}$, $p_{abd}(110) = \frac{1}{4}$ and $p_{abd}(001) = \frac{1}{4}$ for abd. Because the entropy for abd is higher than abc, the distance value for ab to c will be lower, and LTGA will choose to join abc first.

3.2 Pairwise Approximation

In the original LTGA variant [8, 7], hereafter referred to simply as Original, the method given in Section 3.1 was used to merge all clusters. However, a follow up paper showed that this method does not scale well, as calculating the entropy between all possible clusters becomes very time consuming when solving high-dimensional problems [5]. To overcome that problem, that paper proposed an estimation method, provided in Eq. 3.

$$D'(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{c_i \in C_i} \sum_{c_j \in C_j} D(c_i, c_j) \quad (3)$$

Instead of finding the entropy of an entire cluster, this measure only finds the entropy between all pairs of loci in the population. For clusters of sizes greater than one, this finds the average of the pairwise entropy for all loci in C_i with all loci in C_j. While the intent behind this metric was to reduce the time complexity used to construct trees, in which it succeeded, using the pairwise distance also led to a drop in the population size and number of evaluations required to achieve success on a number of problems [5]. The first algorithm to use this method is hereafter referred to as Pairwise, and Original using this method is called Original+.

4. CROSSOVER

The linkage tree is used to perform crossovers that do not disrupt gene linkage, while still exploring the search space. Each subtree in the linkage tree is used as a crossover mask, such that all of the genes in that subtree's cluster are crossed between the parents. When creating a single offspring, all of the crossover masks are used sequentially. Crossovers that create an improvement are kept, while detrimental changes are discarded. As was discussed in the context of LT-GOMEA [9, 1], this incremental change can be seen as a form of local search. Since the leaves of the linkage tree each contain a single locus, LTGA's crossover will mimic a simple hill climber as well as bit flip mutation.

Parents	111000000	000111000	Changes Kept?
abcdef	000111000	111000000	No
ghi	111000000	000111000	No
gh	111000000	000111000	No
def	111111000	000000000	Yes
de	111001000	000111000	No
abc	000111000	111000000	No
ab	001111000	110000000	No
Single Bit Crossover Here			

Table 3: Two parent crossover with least linked first ordering as used by Original. Shading indicates which genes were crossed.

Tables 3–6 provide examples of how LTGA performs crossover for different variants. The left column in each shows which subtree cluster is in use, the middle column(s) show how the offspring are being modified, and the right column indicates if the changes are kept or reverted before applying the next crossover. Each offspring is created through a series of $2N - 2$ crossovers. This includes all crossovers of length one and all other subtrees in the linkage tree. The root of the tree, which contains all of the loci, is not crossed because it will not create any new individuals.

Since the most linked genes are clustered first and are treated as a unit in all further clustering, few crossovers will occur that disrupt their linkage. For instance, the two genes that have the minimum distance (tightest linkage) will be merged first. All other clusters will either contain both genes or neither, so only two of the $2N - 2$ crossovers act on one gene but not the other. Conversely, pairs of clusters with the highest distance (weakest linkage) will be merged last, allowing many crossovers to mix their gene values. In the example, a and b were the first merged cluster, meaning that only their leaf nodes can perform a crossover that splits their values, while a and g are not linked until the root of the linkage tree, and as a result the subtrees a, g, ab, abc, gh, ghi, and $abcdef$ can all freely mix their gene values.

In LTGA, crossover occurs inside all clusters, with the potential of increasing their entropy, making it less likely to form them in the future without impacting diversity. This allows LTGA to recover from some amount of confusion of gene linkages, unlike nonhierarchical crossover.

4.1 Repeated Evaluations

If no preventative measures are taken, LTGA will create an immense number of duplicate individuals. For example, 14 of the 32 individuals created in Table 4 are duplicates of either individuals created during this crossover or of the original parents. As a slight prevention, all of the published LTGA variants avoid evaluating individuals that are identical to their parents, or at least make no mention of deviating from the original LTGA formulation that did this. While

Parents	111000000	000111000	Changes Kept?
Single Bit Crossover Here			
ab	001000000	110111000	No
de	111110000	000001000	No
gh	111000000	000111000	No
abc	000111000	111000000	No
def	111111000	000000000	Yes
ghi	111111000	000000000	No
abcdef	000000000	111111000	No

Table 4: Two parent crossover with smallest first ordering as used by Pairwise. Shading indicates which genes were crossed.

Clone	111000000	Changes Kept?
abcdef	000000000	No
ghi	111000111	Yes
gh	111000001	No
def	111111111	Yes
de	111001111	No
abc	000111111	No
ab	001111111	No
Single Bit Crossover Here		

Table 5: Global mixing crossover with least linked first ordering as used by LT-GOMEA. Shading indicates which genes were crossed.

Clone	111000000	Changes Kept?
Single Bit Crossover Here		
ab	001000000	No
de	111110000	No
gh	111000110	No
abc	000000000	No
def	111111000	Yes
ghi	111111111	Yes
abcdef	000000111	No

Table 6: Global mixing crossover with smallest first ordering as used by LTS-GOMEA. Shading indicates which genes were crossed.

helpful, this does not catch all of the duplicates. For this study, no individual is evaluated twice. This provides an accurate gauge of how many unique evaluations each LTGA variant uses and removes any bias against LTGA variants that are more likely to create duplicate individuals.

4.2 Subtree Ordering

The order in which subtrees are used to perform crossover is one of the primary differences between LTGA variants. In Original and LT-GOMEA, subtrees are applied in least linked first order. In this ordering, the crossover masks are stored in a stack. All of the individual loci are pushed randomly onto the stack. During each iteration of tree construction, the newly formed cluster is also pushed onto the stack. Table 3 shows the ordering for the running example. Since *abcdef* was the last cluster to be merged, it is the first to be applied. The focus of this ordering is for crossover to perform solution refining. The initial crossovers are likely to be large and will be based on the least amount of information in the linkage tree. These steps provide search space exploration. The more linked LTGA believes a cluster of genes to be, the more likely it believes that keeping those genes together will allow for an individual to improve by crossing them. As such, this method starts with the least predictable crossovers, followed by something very similar to local search, bringing the solution to a local optima.

Pairwise employs an alternative subtree ordering which sorts the clusters based on size instead of linkage, with equal sized clusters applied in random order. This method is more akin to optimizing the parts of the problem you know the most about before using the optimized pieces to make larger changes. Table 4 provides an example for this ordering. Smallest first is not the direct opposite of least linked first. One major difference is that least linked first has the potential to deal with the same set of genes in a number of sequential crossovers, where as in smallest first that is very unlikely. In the example, *de* and *gh* are crossed after *ab* and before *abc*, which means if there is a linkage between *ab* and either *de* or *gh* that was previously undiscovered, this ordering has the potential to try it before modifying *c*.

4.3 Parent Selection

In Original and Pairwise, all crossovers for a single offspring are based on two parents, similar to canonical Genetic Algorithms (GAs). To create each child, two parents are selected at random. For each cluster in the linkage tree, two children are created by swapping the genetic material of the parents for that cluster. If either of the children created in one step of the crossover are better than the best of the parents, both of the parents are replaced by the children before continuing on to the next subtree. This method of replacement ensures that the genetic diversity inside a single crossover is unchanged, as whatever patterns existed in the parents before crossover will still exist after. Often this has a polarizing effect, as the best parent is likely to improve by taking the best parts of the weaker. After all of the linkage clusters have been applied, the best child created during the entire process is copied into the next generation. This means that each parent pairing creates a single child. As LTGA uses generational survivor selection, to achieve a steady population size parents must participate in creating more than one offspring each generation. Since parents are modified during the creation of their offspring, subsequent offspring created by the same parent will benefit from the learning performed during the creation of their first. Tables 3 and 4 provide examples for this type of crossover.

The LTGA variant Linkage Tree Global Optimal Mixing Evolutionary Algorithm (LT-GOMEA) differs from Original and Pairwise by performing crossover in a method more similar to Estimation of Distribution Algorithms than GAs. Instead of selecting two parents to pull genetic material from, LT-GOMEA starts by cloning a single parent to initialize the child. For each subtree crossover mask, LT-GOMEA chooses a random parent from the population to donate genetic material to the child for that mask. If the child is improved, the changes will be kept, otherwise they are reverted. While it is not explicitly stated as part of LT-GOMEA, the donated genetic material is assumed here to not be identical to the child's gene values, as doing so would waste a chance at im-

proving the individual. Tables 5 and 6 provide examples of performing global mixing crossover using the population provided in Table 2.

A variant to LT-GOMEA was proposed in [1] which tried all possible donations for each crossover before choosing to keep any. This change allows LTGA to perform a steepest ascent local search on each subtree cluster for each individual, finding the global next best. Unfortunately, doing so incurs significant overhead and causes premature convergence due to large subtrees copying so much of an individual as to allow the current best individual to spread too rapidly.

5. EXPERIMENTS

While most of the permutations listed in Table 1 are valid ways to configure an LTGA, this paper focuses on the variants most likely to achieve interesting results. From the work with LT-GOMEA and Pairwise, it appears that subtree ordering and crossover represent the ways in which the state-of-the-art LTGA variations differ. To examine the effects of these variations, empirical test results are presented for all four combinations of least linked first linkage, smallest first linkage, two parent crossover, and global mixing, which are represented by Original+, Pairwise, LT-GOMEA, and the novel combination Linkage Tree Smallest First Global Mixing Evolutionary Algorithm (LTS-GOMEA). While global next best is an option for crossover, [1] tested LT-GOMEA against global next best and found it to be sufficiently inferior to not warrant further investigation.

In order to ensure a minimal population size, each LTGA variant was tuned to each problem using a modified version of bisection [6]. In order to remove operator bias and further generalize the method, the bisection used here starts by setting the initial population size to the minimum possible for the algorithm, which in this case is two. It then proceeds by testing to see if its guess meets the success criteria. If not, then it doubles its guess and tries again. Once the criteria are met, the current guess is used as the maximum population size and the previous guess is the minimum population size for bisection. This ensures the starting minimum is below the population size required and the starting maximum is at least as high as required. In this instance, the required success rate was 24 of 25 runs finding the global optimum. When actually gathering experimental results, all experiments used 100 runs.

To reduce the noise generated by the initial population, a set of individuals was created for each run of each population size on each problem. These sets of individuals were then all optimized using SAHC. Having all versions of LTGA share initial populations for the same population size facilitates examining how each algorithm utilizes the information in those populations. Furthermore, any difference in population size is then more clearly indicative of a need for different initial information, and not just caused by random chance.

6. DECEPTIVE TRAP

The canonical benchmark for LTGA is the deceptive trap (Trap-k) problem [2], which divides the genome into separate traps of length k bits. Each trap is scored using Eq. 4, where t is equal to the sum of the bit values in the trap.

$$trap(t) = \begin{cases} k-1-t, & t < k \\ k, & t = k \end{cases} \quad (4)$$

N=50	Population Size	Rarest Trap Count	Local Search Steps	Evals To Success (Standard Dev.)
Original+	26	1.94	513.93	1946.53 (481.90)
LT-GOMEA	26	1.94	513.93	21.31 (5.55)
Pairwise	26	1.94	513.93	2200.38 (468.01)
LTS-GOMEA	26	1.94	513.93	89.81 (0.48)
N=100				
Original+	24	1.52	922.33	6134.35 (1244.64)
LT-GOMEA	24	1.52	922.33	47.91 (10.19)
Pairwise	24	1.52	922.33	6794.50 (1307.67)
LTS-GOMEA	24	1.52	922.33	179.82 (0.48)
N=150				
Original+	32	2.26	1827.23	15430.62 (2625.32)
LT-GOMEA	32	2.26	1827.23	79.65 (14.06)
Pairwise	32	2.26	1827.23	15507.38 (2577.15)
LTS-GOMEA	32	2.26	1827.23	269.78 (0.48)

Table 7: Deceptive Trap Results, k=5

Each trap is fully deceptive, in that all non-optimal values lead away from the global optimum of all ones toward the local optimum containing all zeros. As a result, any crossover that affects less than a complete trap will likely cause both individuals to move away from the global optima. This need to preserve linkage, coupled with trap independence, makes Trap-k an ideal proof of concept benchmark for LTGA.

LTGA needs an initial population that contains the optimized value for each trap in at least one individual to find the global optima, as the deceptive nature of each trap makes creating the optimized value during evolution unlikely. Local search on the deceptive trap function ensures that all traps in all individuals are initialized to either the optimum (all ones) or the local optimum (all zeros). LTGA's first benefit is the complete removal of entropy for each trap. Independent of trap size, after local search all traps will contain less than one bit of information. As a result, when LTGA builds the gene clusters, all genes in a trap will have the minimum possible distance from each other. Furthermore, when performing a crossover using a subset of the cluster (i.e., a subtree of size less than k), new trap values cannot be created as there is no way to change less than k bits in any individual to improve its fitness. Thus, the only way for LTGA to lose track of a trap's grouping is if the entire population agrees on the value between two traps. In such an event it is still unlikely that LTGA will create a trap with a value other than all ones or all zeros. When performing a crossover that contains one or more complete traps, LTGA's behavior changes depending on the type of crossover in use. The final best offspring of Original and Pairwise, using two-parent crossover, will contain all of the optimized traps that started in either individual. LT-GOMEA and LTS-GOMEA will solve even faster, as after all complete traps have been crossed over, the individual produced will contain all optimized traps that exist anywhere in the population. The second benefit of local search for LTGA is the increased number

N=49	Population Size	Rarest Trap Count	Local Search Steps	Evals To Success (Standard Dev.)
Original+	64	1.83	1486.62	2148.50 (571.09)
LT-GOMEA	64	1.83	1486.62	13.41 (3.43)
Pairwise	64	1.83	1486.62	2454.71 (579.23)
LTS-GOMEA	64	1.83	1486.62	90.91 (0.27)
N=98				
Original+	78	1.76	3541.38	12983.56 (2990.20)
LT-GOMEA	78	1.76	3541.38	31.89 (7.08)
Pairwise	78	1.76	3541.38	14082.40 (2884.49)
LTS-GOMEA	78	1.76	3541.38	181.92 (0.26)
N=147				
Original+	80	1.57	5410.32	27150.74 (5581.73)
LT-GOMEA	80	1.57	5410.32	53.68 (11.26)
Pairwise	80	1.57	5410.32	30189.41 (4738.40)
LTS-GOMEA	80	1.57	5410.32	272.96 (0.17)

Table 8: Deceptive Trap Results, k=7

N=50	Population Size	Rarest Trap Count	Local Search Steps	Evals To Success (Standard Dev.)
Original+	73	8.66	414.4	13744 (2925.2)
LT-GOMEA	67	8.09	383.1	9643 (2606.3)
Pairwise	74	9.43	422.3	13475 (2901.1)
LTS-GOMEA	49	5.34	278.2	11192 (2163.7)
N=100				
Original+	86	9.78	893.7	46579 (7085.9)
LT-GOMEA	84	9.41	874.1	32533 (4820.6)
Pairwise	97	11.42	1011.6	50669 (7921.5)
LTS-GOMEA	62	6.32	642.7	36272 (4755.3)
N=150				
Original+	101	11.35	1520.8	91652 (13411.8)
LT-GOMEA	96	10.64	1449.6	58591 (7269.7)
Pairwise	107	12.54	1610.5	100951 (14306.7)
LTS-GOMEA	73	7.31	1103.0	68129 (10462.3)

Table 9: Deceptive Step Trap Results, k=5, s=2

of optimized traps in the starting population. The probability of a trap containing an optimal value in a single individual increases from $\frac{1}{2^k}$ to $\frac{k+1}{2^k}$, since SAHC will not change randomly generated optimized traps, and will improve all traps with a single incorrect bit to the optimum.

Tables 7 and 8 provide the experimental results for the LTGA variations on Trap-5 and Trap-7, respectively. As expected, the algorithms required the same initial population size, due to their identical requirements on the existence of optimized traps. Population size predictably increases with problem size, with the exception of $N = 50$ to $N = 100$, where the population size decreases. This minor drop is likely a bisection tuning artifact, which relies on stochastic processes to set the population size. The rarest trap count value given for each problem is the average number of individuals in the initial population containing the optimized version of the trap with the lowest frequency of individuals containing its optimized version over all successful runs.

The LTGA variants vary in the number of evaluations after local search that each requires to achieve success. Most notably, both versions using global mixing (LT-GOMEA and LTS-GOMEA) require far less than either form of two-parent crossover. When performing global mixing, each crossover mask that contains exactly one trap will result in the child having the optimized version of that trap after crossover. This is because the child either has the optimized version, and therefore no chance to improve for that crossover mask, or the suboptimal version and will therefore randomly select a parent with the only other genetic material for that crossover mask: the optimized value. Since two-parent crossover can only create offspring containing the optimized traps from its two parents, multiple crossovers are required to build individuals such that when they are paired, at least one has the optimized value for all traps. The difference between LT-GOMEA and LTS-GOMEA is trivial in this case, as the difference is how many crossover masks are tried before all single trap masks are applied. In LTS-GOMEA there will be $2N - \frac{N}{k}$ subtrees with a size less than or equal to k, as opposed to LT-GOMEA where, depending on tree construction, there can be as few as $2\frac{N}{k} - 1$ subtrees with less than or equal linkage to the single trap subtrees. These formulas are near perfect predictors of the experimental number of evaluations found in Tables 7 and 8.

7. DECEPTIVE STEP TRAP

The artificial nature of the Trap-k problem, as exploited by SAHC, makes LTGA's performance on this benchmark an unlikely predictor of its ability on more realistic problems. In order to create a very similar problem to Trap-k that cannot be exploited as much by SAHC, the creation of StepTrap-k,s as defined in Eq. 5 is proposed. Similar to Trap-k, StepTrap-k,s uses a series of non-overlapping trap functions, where each trap is scored using the sum of the bits set to one. The optimal value for each trap is all ones, with a local optimum of all zeros. All other configurations improve monotonically as they approach all zeros. The only difference between normal Trap-k and StepTrap-k,s is that StepTrap-k,s includes plateaus of length s. If s is set to one, then StepTrap-k,s is identical to Trap-k. For $s > 1$, up to $s - 1$ bits can change without a change in the fitness of non-optimal traps. By modifying Trap-k to StepTrap-k,s and setting s=2, SAHC will still create optimized traps at the same probability as in Trap-k, but it will leave the majority of traps in some partial state between all zeros and all ones. As such the loss of entropy for each trap will not be as extreme as in normal Trap-k, making LTGA more prone to clustering errors.

$$step\_trap(t) = \left\lfloor \frac{(k-s) \pmod s + trap(t)}{s} \right\rfloor \quad (5)$$

Tables 9 and 10 provide the experimental results for each

N=49	Population Size	Rarest Trap Count	Local Search Steps	Evals To Success (Standard Dev.)
Original+	120	4.30	535.6	36033 (4756.3)
LT-GOMEA	135	4.66	599.0	34349 (5035.3)
Pairwise	147	5.76	654.4	42689 (5746.5)
LTS-GOMEA	117	4.18	521.4	39870 (6170.5)
N=98				
Original+	233	8.71	1841.0	180075 (17685.8)
LT-GOMEA	197	7.20	1549.5	130998 (16004.0)
Pairwise	265	10.26	2090.6	222781 (25374.9)
LTS-GOMEA	143	4.37	1127.0	147621 (15173.7)
N=147				
Original+	256	9.58	2899.9	371343 (35206.8)
LT-GOMEA	317	12.17	3596.4	292347 (35440.8)
Pairwise	309	11.97	3509.9	485905 (50916.7)
LTS-GOMEA	200	6.61	2268.3	319875 (27276.5)

Table 10: Deceptive Step Trap Results, k=7, s=2

variation of LTGA on the StepTrap-*k,s* problem using s=2 with k set to 5 and 7 respectively. Unlike the uniform population sizes found on Trap-*k*, all LTGA variants needed a different sizes on StepTrap-*k,s*. For all N and k tested, LTS-GOMEA was the least destructive to existing optimized traps, and was therefore able to achieve the global optima with the smallest population sizes.

To understand why global mixing with smallest first ordering is the least destructive to existing optimized traps, we first need to examine how optimal mixing is able to lose an optimized trap. Incorrectly linked crossovers can include parts of multiple traps. If, when performing these crossovers, the amount that some of the traps improve is greater than the fitness lost by breaking one of the other traps, the change is still kept. Consider the example where a, b, c, and z are incorrectly linked. The first three are part of one trap that is not at either optima in the parents, and the fourth is part of another trap which one of the parents has optimized. The crossover abc improves its trap by increasing its fitness two points, which offsets the crossover of z, which decreases its trap value one point. As a result, the optimized trap is lost. Smallest first ordering is sometimes able to avoid this problem by making minimal changes first. In the example, if any of a, b or c are linked before all three are combined with z and crossing that subtree can achieve even a single point of fitness increase, then when crossing $abcz$ there will no longer be enough improvement to offset the damage.

Conversely, because smallest first is more likely to push individual traps to their local optima before performing larger crossovers, it is less likely that larger crossovers that include partial traps will make enough of an improvement to be considered successful. This means that until better linkage information is found, most of the larger crossovers in smallest first ordering will likely be wasted. Least linked does not have this problem, and in fact the most tightly linked crossovers at the end of search are more likely to make improvements, and help prevent wasted evaluations. As an indirect added bonus, every successful crossover reduces the entropy of most if not all of the traps it effects, making the next generation's linkage tree more accurate. This is because as traps are pushed to the extremes, StepTrap-*k,s* approaches the Trap-*k* problem, which is very easy for LTGA to solve, as shown in Section 6.

Two-parent crossover is even more likely to destroy optimized traps than global mixing, as shown in Tables 9 and 10. While two-parent crossover can have all of the same flaws as global mixing for optimized trap destruction, it can also lose a trap due to parent pairing. If two parents are paired such that the worse parent has an optimized trap in a location the better parent does not, and no crossover exists that can move that trap without significantly damaging the other parent, that genetic material can be lost. It is possible that this explains why smallest first does not help Pairwise as much as LTS-GOMEA, as even though crossover is not directly destroying traps, the inability for Pairwise to move traps between parents is resulting in their loss. One final consideration about the trade off between LT-GOMEA and LTS-GOMEA is the inclusion of the initial search steps as evaluations. If either is applied to a problem where partial reevaluation is not an option, the number of evaluations required for local search will be about N times larger than the number of local search steps. LTS-GOMEA's smaller population size allows it to solve all of the StepTrap-*k,s* problems except N=49 in less evaluations than LT-GOMEA if local search evaluations are included.

Similar to normal Trap-*k*, if LTGA is able to correctly link each trap independently before all optimized versions of any trap are lost, it is very unlikely that any LTGA variant will fail to reach the global optimum. This is because any crossover that moves a single trap from one parent to another can only improve that trap's fitness, which will shift the population toward both extremes, again making it resemble Trap-*k*. The only way to lose an optimized trap now would be to cross multiple traps into a child, where more traps are made optimal than are made non-optimal. Smallest first ensures that any crossover that moves multiple traps will first optimize the individual traps, which again reduces the likelihood of this happening. Two-parent crossover is also unlikely to have this issue as, independent of what order traps are crossed, optimal traps that are lost in one step can be reclaimed in subsequent crossovers with the same parent or will be optimized before they can contribute to a lost optimized trap.

8. NEAREST NEIGHBOR NK

NK landscapes are a set of randomly generated benchmark problems [4]. An NK landscape is defined by the number of dimensions used (N), the number of other genes each gene relies on to calculate its own fitness (k), the epistasis table defining the linkages between genes, and N randomly generated fitness functions which determine the fitness of each collection of linked genes. Nearest neighbor NK landscapes are a subset of NK landscapes in which each gene is linked to the k genes that directly follow it, with the final genes wrapping around. This subset is of interest for testing LTGA as it contains overlapping blocks of genes with varying levels of linkage. Furthermore, by utilizing dynamic programming it is possible to determine the optimal value for nearest neighbor NK landscapes in polynomial time [10].

N=30	Population Size	Local Search Steps	Evals To Success (Standard Dev.)
Original+	132	1038.06	8253.86 (3473.91)
LT-GOMEA	131	1039.16	7743.29 (4804.82)
Pairwise	110	867.97	7144.20 (3426.76)
LTS-GOMEA	151	1219.60	10634.28 (5112.47)
N=40			
Original+	323	3286.34	33141.11 (12757.48)
LT-GOMEA	501	5102.29	36300.81 (21529.69)
Pairwise	250	2549.38	28875.14 (10652.87)
LTS-GOMEA	317	3233.64	40525.74 (16589.45)
N=50			
Original+	383	4786.59	67280.11 (22403.78)
LT-GOMEA	393	4905.07	60809.10 (24655.13)
Pairwise	371	4636.94	69266.90 (21912.95)
LTS-GOMEA	276	3451.42	62564.14 (19336.66)

Table 11: NK Results, k=5

Table 11 shows the results of each LTGA variant tested on nearest neighbor NK using k=5. Unlike the previous two benchmarks, Pairwise was able to outperform both global mixing strategies on N=30 and N=40 in both population size and total number of evaluations. Original+ was also able to achieve a lower number of evaluations on N=40. The most likely cause is LTGA's inability to capture overlapping dependencies inherent in nearest neighbor NK. When performing two-parent crossover, all donated information comes from the same individual. As such, any partial improvements made to a child are more likely to work well with subsequent crossovers with the same parent as both sets of genes were effective together in at least that parent.

When performing global mixing, no similar linkage exists between donated information. As a result, global mixing attempts to combine genetic material from many different parents, with only a limited ability to keep linked information together. Individuals created in this way are less likely to be an improvement over the existing individual, and will therefore be a wasted evaluation. The increased population size is likely a symptom of premature convergence caused by this inability to generate good solutions.

The reason why N=40 switches which type of mixing is most effective is probably because as the number of problem variables increases but the amount of overlap stays constant, the effect of incorrect linkage is lessened. This may explain why both global mixing variants were able to reduce their required population size from N=40 to N=50.

9. CONCLUSIONS

Using the Trap-k problem to test the different variations of LTGA provides little to no information, as SAHC converts a very difficult problem into a trivial one for any type of LTGA. In order to provide a better benchmark for non-overlapping linked genes, this paper introduces the StepTrap-k,s problem, which modifies the traditional Trap-k to contain fitness plateaus of width s. These plateaus prevent SAHC from pushing all traps to the extremes, providing a much more challenging problem for LTGA.

This new problem reveals global mixing's ability to converge faster and use smaller population sizes than two-parent crossover. By applying the smallest crossover masks first, LTGA better avoids destroying existing optimized structures in the population when incorrect linkage occurs at the expense of needing more search evaluations. As a result, on non-overlapping linkage problems where local search cannot exploit partial reevaluation, the novel LTGA variant LTS-GOMEA, which uses smallest first ordering, may outperform LT-GOMEA, which uses least linked first ordering.

Testing the LTGA variants on nearest neighbor NK landscapes provides insight to how each variant handles linkage styles less suited to tree structures. On problems with lots of overlap, the two-parent variants seem to perform better than global mixing due to the inherent linkage between subsequent crossovers between the same two parents. As global mixing is able to significantly outperform two-parent crossover in all other instances, improving how LTGA stores linkage information to allow for better representation of overlap is a very important next step for making LT-GOMEA and LTS-GOMEA more generally applicable.

10. REFERENCES

[1] P. Bosman and D. Thierens. The roles of local search, model building and optimal mixing in evolutionary algorithms from a BBO perspective. *GECCO*, pages 663–670, 2011.

[2] K. Deb and D. Goldberg. Analyzing deception in trap functions. *In Proceedings of FOGA II: the Second Workshop on Foundations of Genetic Algorithms*, pages 93–108, 1992.

[3] G. R. Harik, F. G. Lobo, and K. Sastry. *Linkage learning via probabilistic modeling in the extended compact genetic algorithm*. Springer–Verlag, Berlin, 2006.

[4] S. Kauffman. The origins of order: self organization and selection in evolution. *Oxford University Press*, 1993.

[5] M. Pelikan, M. Hauschild, and D. Thierens. Pairwise and problem-specific distance metrics in the linkage tree genetic algorithm. *GECCO*, pages 1005–1012, 2011.

[6] K. Sastry. Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master's thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, 2001.

[7] D. Thierens. Linkage tree genetic algorithm: first results. *GECCO*, pages 1953–1957, 2010.

[8] D. Thierens. The linkage tree genetic algorithm. *Parallel Problem Solving from Nature*, pages 264–273, 2010.

[9] D. Thierens and P. Bosman. Optimal mixing evolutionary algorithms. *GECCO*, pages 617–624, 2011.

[10] A. H. Wright, R. K. Thompson, and J. Zhang. The computational complexity of N-K fitness functions. *IEEE Trans. on Evolutionary Computation*, 4(4):373–379, 2000.