Missouri Estimation of Distribution Algorithms Laboratory

Pairwise and Problem-Specific Distance Metrics in the Linkage Tree Genetic Algorithm

Martin Pelikan, Mark Hauschild, Dirk Thierens

MEDAL Report No. 2011001

February 2011

Abstract

The linkage tree genetic algorithm (LTGA) identifies linkages between problem variables using an agglomerative hierarchical clustering algorithm and linkage trees. This enables LTGA to solve many decomposable problems that are difficult with more conventional genetic algorithms. The goal of this paper is two-fold: (1) Present a thorough empirical evaluation of LTGA on a large set of problem instances of additively decomposable problems and (2) speed up the clustering algorithm used to build the linkage trees in LTGA by using a pairwise and a problem-specific metric.

Keywords

Linkage learning, genetic algorithms, linkage tree genetic algorithm, decomposable problems, prior problem-specific knowledge, problem decomposition.

Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Department of Mathematics and Computer Science
University of Missouri–St. Louis
One University Blvd., St. Louis, MO 63121
E-mail: medal@cs.umsl.edu
WWW: http://medal.cs.umsl.edu/

Pairwise and Problem-Specific Distance Metrics in the Linkage Tree Genetic Algorithm

Martin Pelikan
Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Dept. of Math and Computer Science, 320 CCB
University of Missouri in St. Louis
One University Blvd., St. Louis, MO 63121
pelikan@cs.umsl.edu

Mark Hauschild
Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Dept. of Math and Computer Science, 321 CCB
University of Missouri in St. Louis
One University Blvd., St. Louis, MO 63121
mwh308@umsl.edu

Dirk Thierens
Utrecht University
Dept. of Information and Computing Sciences
P.O. Box 80089, 3508 TB Utrecht, The Netherlands
Dirk.Thierens@cs.uu.nl

February 9, 2011

Abstract

The linkage tree genetic algorithm (LTGA) identifies linkages between problem variables using an agglomerative hierarchical clustering algorithm and linkage trees. This enables LTGA to solve many decomposable problems that are difficult with more conventional genetic algorithms. The goal of this paper is two-fold: (1) Present a thorough empirical evaluation of LTGA on a large set of problem instances of additively decomposable problems and (2) speed up the clustering algorithm used to build the linkage trees in LTGA by using a pairwise and a problem-specific metric.

Keywords: Linkage learning, genetic algorithms, linkage tree genetic algorithm, decomposable problems, prior problem-specific knowledge, problem decomposition.

1 Introduction

In genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989), crossover is typically the primary exploration operator. For reliable and efficient convergence to the global optimum, crossover must succeed in two important tasks (Goldberg, 2002; Thierens, 1999): (1) juxtapose partial solutions, and (2) prevent disruption of important partial solutions contained in high quality solutions. While it is relatively straightforward to design crossover operators that juxtapose partial solutions of two

or more solutions, preventing disruption of important partial solutions or building blocks is often more challenging. GAs capable of preserving important partial solutions are often referred to as *linkage learning* GAs due to their ability to learn and exploit important linkages between problem variables (Harik & Goldberg, 1996; Goldberg, 2002; Thierens, 1999). In the past two decades, a number of linkage learning GAs were proposed (ping Chen, Yu, Sastry, & Goldberg, 2007). In this paper we are going to discuss one of the most recent linkage learning GAs, the linkage tree GA (LTGA) (Thierens, 2010). LTGA uses crossover based on a linkage tree constructed with a hierarchical clustering algorithm. The goal of using the linkage tree is to ensure that crossover will not disrupt important partial solutions corresponding to the highly correlated problem variables and, at the same time, it will ensure that partial solutions are combined effectively.

The purpose of this paper is twofold: (1) Present a thorough empirical evaluation of LTGA on a large set of problem instances of additively decomposable problems. (2) Speed up the clustering algorithm in LTGA by replacing the standard distance metric used in the original LTGA to measure distance of clusters of variables by a distance metric based on pairwise statistics and a problem-specific metric.

The paper is organized as follows. Section 2 describes the linkage tree GA (LTGA). Section 3 presents two alternative metrics that can be used in LTGA for building the linkage tree. Section 4 presents and discusses experimental results. Section 5 summarizes and concludes the paper.

2 Linkage-Tree GA

Linkage-tree genetic algorithm (LTGA) (Thierens, 2010) works with a population of candidate solutions. In this paper, candidate solutions are represented by binary strings of length n, but LTGA can be applied to problems where candidate solutions are represented by strings over any finite alphabet. The initial population is generated at random according to the uniform distribution over all admissible solutions. Before proceeding with the first iteration, local search based on single-bit neighborhoods is applied to every solution in the initial population. In each iteration of the local search, a single bit is flipped that improves the quality of the solution the most until no more improvement is possible. We discuss the reasons for using local search on the initial population as well as the effects of doing this in section 4. In subsequent iterations of LTGA, no local search is used anymore.

In each iteration, a linkage tree is first built using a hierarchical clustering algorithm (see section 2.1) and the new population of candidate solutions is then generated by repeated application of crossover (see section 2.2). The next iteration is then executed unless termination criteria are met; for example, the run may be terminated when a solution of sufficient quality has been found or when a bound on the maximum number of fitness evaluations has been reached.

2.1 Learning the Linkage Tree

LTGA learns linkages between problem variables using a hierarchical clustering algorithm, which is used to create a *linkage tree*. The linkage tree is used to create crossover masks that help crossover to reduce the likelihood of disrupting combinations of values of groups of correlated variables.

The distance of clusters in LTGA is measured using *normalized variation of information*. For clusters C_i and C_j of variables, the distance of C_i and C_j is defined as

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

where $H(C_i, C_j)$ is the entropy of $C_i \cup C_j$, and $H(C_i)$ and $H(C_j)$ are entropies of C_i and C_j, respectively. The entropy for a set S of random variables is defined as $H(S) = -\sum_s p(S = s) \log_2 p(S = s)$, where the sum runs over all instances s of variables in S and $p(S = s)$ denotes the probability of $S = s$. In the context of LTGA, $p(S = s)$ is defined to be the proportion of population members for which $S = s$.

The bottom-up agglomerative hierarchical clustering algorithm used to create a linkage tree starts with each variable forming an individual linkage tree of one node. Each linkage tree defines one cluster. In each iteration, the algorithm computes the distances between pairs of clusters or trees according to eq. (1), and it merges the two linkage trees that have the minimum distance by hanging the two trees under a new binary root node. Therefore, in each iteration, the number of linkage trees decreases by one. The algorithm terminates when there is only one linkage tree, which contains all problem variables. In the resulting linkage tree, each node defines a subset of variables that are located in the subtree starting in this node. Since the trees were merged based on the distance measure defined in eq. (1), correlated variables are likely to be merged in early stages of the algorithm and, consequently, most subtrees should either contain all these correlated variables or none of them. That is why the linkage tree provides useful information for crossover which can be used to prevent disruption of important partial solutions.

2.2 Linkage-Tree Crossover

The set of masks to be used by crossover is defined as the set of variable subsets contained in any subtree of the linkage tree, except for the subset containing all problem variables at the root of the linkage tree. The reason for excluding the root is that it does not make sense to exchange the values of *all* variables simultaneously.

Once the linkage tree is defined, the subsets of variables in its subtrees are used as masks in a two-parent crossover. First, the population of N solutions is shuffled. Next, the $N/2$ pairs of individuals undergo crossover (1st with 2nd, 3rd with 4th, and so on), creating $N/2$ children. Each application of the crossover operator creates one offspring solution and, possibly, modifies parents, as is explained in the next paragraph. After the first round of crossover, the population is shuffled again and another $N/2$ pairs of individuals undergo crossover, creating another $N/2$ offspring.

For each pair X and Y of solutions to which crossover is applied, LTGA tries all masks created from the linkage tree ordered according to their sizes, starting with the shortest masks containing a single variable (leaves of the tree). For each mask, values in string positions corresponding to this particular subset of variables are exchanged between X and Y and the two created solutions X' and Y' are evaluated using the fitness function. If the best of X' and Y' is better than both X and Y, X and Y are replaced by X' and Y' before applying the next mask. In the end, after using all masks, the best of the solutions obtained with different masks is copied to the next population. Of course, the original solutions X and Y might have changed along the way (and most often they do); these changes are retained.

It is of note that the masks containing a single variable each allow LTGA to perform local search based on the simple bit-flip neighborhood. Nonetheless, unlike in a traditional local search, because LTGA uses crossover as the only variation operator, in LTGA it is necessary to ensure that for each position there is at least one parent containing the desired value of the bit.

From the perspective of time complexity, the building of the linkage tree and the evaluation of candidate solutions are the two computational bottlenecks of LTGA. In the construction of the linkage tree, most computational resources go into the computation of cluster distances. For more information about LTGA, please consult Thierens (2010).

3 Measuring Cluster Distance

The previous section pointed out that the procedure for measuring the distance of clusters in LTGA is one of the two computational bottlenecks of LTGA. In this section, we will examine two interesting ways to speed up this component: (1) A distance metric based on pairwise statistics and (2) a fixed distance metric for additively decomposable problems based on problem-specific knowledge.

3.1 Pairwise Metric

As was suggested in the study that introduced LTGA (Thierens, 2010), the distance between the clusters C_i and C_j may be approximated using an average of pairwise distances between all pairs c_i and c_j of variables such that $c_i \in C_i$ and $c_j \in C_j$. Specifically, the distance can be approximated as

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

The main advantage of the pairwise metric $D'(\cdot)$ is that this metric depends only on pairwise statistics and the population of solutions does not have to be parsed after every merge of two clusters, as is the case for $D(\cdot)$. Once the probabilities for pairs of variables have been computed, no further parsing of the population is necessary. This should significantly reduce the time required to learn linkage trees. However, in addition to the reduced time complexity, we will see that using the pairwise metric will sometimes lead to reduced population sizes and the overall number of candidate solutions that must be examined until the global optimum is found.

3.2 Problem-Specific Metric

Even when the problem being solved is difficult, it may be relatively straightforward to gather information about the important correlations of variables that follows directly from the definition of the problem. This is going to be the case for example in the problems being solved in the experimental section of this paper, but more generally, the same approach can be used for many difficult classes of additively decomposable problems, including classes of NP-complete problems. Furthermore, when solving many problem instances of the similar type, it may often be possible to gather information about variable interactions from examining previous runs of the optimization algorithm, and use this information to design a problem-specific heuristic distance metric for future instances of the same problem. This approach is often referred to as *learning from experience* and has been studied especially in the context of estimation of distribution algorithms (EDAs) (Hauschild & Pelikan, 2008; Hauschild & Pelikan, 2009), because many EDAs provide a wealth of information about the problem in the form of graphical and other probabilistic models. In this paper, we focus on the former type of problem-specific metrics, which are applicable to additively decomposable problems.

The fitness in an additively decomposable problem composed of m subproblems is defined as

$$f(X_1, \ldots, X_n) = \sum_{i=1}^{m} f_i(S_i) \tag{2}$$

where $(X_1, \ldots, X_n)$ are input variables, f_i is the ith subfunction, and $S_i \subset \{X_1, X_2, \ldots, X_n\}$. While they may often exist multiple ways of decomposing the problem into an additively decomposable problem, typically one would prefer decompositions which minimize the sizes of subsets $\{S_i\}$.

Given an additively decomposable problem, we may define the distance between two variables using a weighted graph G of n nodes, one node per variable. For any two variables X_i and X_j in the same subset S_k, that is, $X_i, X_j \in S_k$, we create an edge in G between the nodes X_i and X_j with weight $w_{i,j} = 1$. Denoting by $l_{i,j}$ the weight of the shortest path between X_i and X_j in G, we may define the distance between two variables as

$$D''(X_i, X_j) = \begin{cases} l_{i,j} & \text{if a path between } X_i \text{ and } X_j \text{ exists} \\ n & \text{otherwise} \end{cases}$$

The above distance measure makes variables in the same subproblems close to each other, whereas for the remaining variables, the distances correspond to the length of the chain of subproblems that relate the two variables. The distance is maximal for variables that are completely independent.

The overall distance of two clusters C_i and C_j can then be computed as the average distance between pairs $c_i \in C_i$ and $c_j \in C_j$, similarly as with the metric based on pairwise distances.

The main advantage of the problem-specific metric is that the population does not have to be large enough for important correlations to be discovered, because the important correlations are directly reflected in the fixed distance metric. Since the discovery of variable correlations is the key factor for population sizing in many linkage learning evolutionary algorithms (Pelikan, Goldberg, & Cantú-Paz, 2000; Pelikan, Sastry, & Goldberg, 2002; Yu, Sastry, Goldberg, & Pelikan, 2007), this should lead to a significant decrease of the population sizes compared to the exact or pairwise distance metrics based on populations of solutions. On the other hand, it may often be the case that some interactions following from the additive decomposition may not be as important as other interactions, even interactions between variables that are not contained in a single subproblem of the decomposition. That is why the linkage trees discovered using the problem-specific metric may not be the best linkage trees to use. Thus, on one hand, problem-specific distance metrics may be considered as the ideal case, because decompositions will closely correspond to the problem definition regardless of the information obtained from the populations of candidate solutions; on the other hand, better decompositions may be found by examining population of candidate solutions.

The above problem-specific measure for additively decomposable problems is only one possibility, and other approaches may be envisioned. For example, the weight of an edge between the variables in G may further decrease with the number of subsets in which the variables are contained, or the distance of variables which are not mutually contained in one subset may increase exponentially fast with the shortest path. An even more interesting possibility would be to consider the subfunctions themselves in measuring the distances, so that only correlations that lead to nonlinearities are considered or that some correlations are given a priority over others. Finally, as was already mentioned, the problem-specific distance metric can be based on previous runs of LTGA or another linkage learning evolutionary algorithm (e.g. an EDA) on other problem instances of the similar type. The key is to use problem-specific information to specify the distance metric so that variables that are likely to be correlated will be closer to each other.

4 Experiments

4.1 Test Problems

To test the original LTGA and its modifications based on the two distance metrics discussed in section 3, we consider three problem classes: (1) Concatenated traps of order k, (2) nearest-neighbor NK landscapes, and (3) 2D spin glasses. Concatenated traps of order k are decomposable into separable subproblems of order k so that the subproblems in an optimal additive decomposition

do not overlap. Nonetheless, for trap problems, variation operators must take into account important correlations between variables; otherwise, these problems become intractable (Thierens & Goldberg, 1993; Thierens, 1999). The last two problem classes are also additively decomposable; however, for these problems the optimal decomposition contains much overlap between the different subproblems, which makes these problems a lot more challenging than concatenated traps. All included test problems assume that admissible solutions are encoded as binary strings of length n and all these functions can be solved in polynomial time.

4.1.1 Trap-k

In concatenated traps of order $k \geq 3$ (trap-k) (Ackley, 1987; Deb & Goldberg, 1991), the input string is first partitioned into independent groups of k bits each. This partitioning is unknown to the algorithm and it does not change during the run. A k-bit deceptive trap function is applied to each group of k bits and the fitness is the sum of the contributions of all trap functions. The contribution of each group of k bits is computed as

$$trap_k(u) = \begin{cases} k & \text{if } u = k \\ k-1-u & \text{otherwise} \end{cases},$$

where u is the number of 1s in the input string of k bits. The task is to maximize the function. An n-bit trap-k function has one global optimum in the string of all 1s and $(2^{n/k}-1)$ other local optima. Traps of order k necessitate that all bits in each group are treated together, because statistics of lower order are misleading. Since LTGA performance is invariant with respect to the ordering of string positions (Pelikan, 2005), it does not matter how the partitioning into k-bit groups is done. In this paper, we consider mainly $k \in \{5, 6, 7\}$ although in one of the cases we also consider $k = 8$.

4.1.2 Nearest-neighbor NK landscapes

An NK fitness landscape (Kauffman, 1989; Kauffman, 1993) is fully defined by the following components: (1) The number of bits, n, (2) the number of neighbors per bit, k, (3) a set of k neighbors $\Pi(X_i)$ for the i-th bit for every $i \in \{1, \ldots, n\}$, and (4) a subfunction f_i defining a real value for each combination of values of X_i and $\Pi(X_i)$ for every $i \in \{1, \ldots, n\}$. Typically, each subfunction is defined as a lookup table. The objective function f_{nk} to maximize is defined as $f_{nk}(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} f_i(X_i, \Pi(X_i))$. The difficulty of optimizing NK landscapes depends on all components defining an NK problem instance. In this paper, we consider nearest-neighbor NK landscapes, in which neighbors of each bit are restricted to the k bits that immediately follow this bit. The neighborhoods wrap around; thus, for bits which do not have k bits to the right, the neighborhood is completed with the first few bits of solution strings. The reason for restricting neighborhoods to nearest neighbors was to ensure that the problem instances can be solved in polynomial time even for $k > 1$ using dynamic programming (Pelikan, 2010). The subfunctions are represented by look-up tables (a unique value is used for each instance of a bit and its neighbors), and each entry in the look-up table is generated with the uniform distribution from $[0, 1)$. The used class of NK landscapes with nearest neighbors is thus the same as that in Pelikan (2010). In all experiments, we use $k = 5$ and we consider n from 20 to 100 with step 10. For each n, we use 1,000 unique, independently generated instances. The reason for using such a large number of instances is that for NK landscapes algorithm performance often varies substantially across different instances and some algorithm properties may not be apparent if only a few instances are used.

4.1.3 2D spin glass

Ising spin glasses are prototypical models for disordered systems (Young, 1998). A simple model to describe a finite-dimensional Ising spin glass is typically arranged on a regular 2D or 3D grid where each node i corresponds to a spin s_i and each edge $\langle i, j\rangle$ corresponds to a coupling between two spins s_i and s_j. Each edge has a real value $J_{i,j}$ associated with it that defines the relationship between the two connected spins. To approximate the behavior of the large-scale system, periodic boundary conditions are often used that introduce a coupling between the first and the last elements in each row along each dimension. For the classical Ising model, each spin s_i can be in one of two states: $s_i = +1$ or $s_i = -1$. Given a set of coupling constants $J_{i,j}$, and a configuration of spins C, the energy can be computed as $J_{i,j}$, and a configuration of spins C, the energy can be computed as $E(C) = -\sum_{\langle i,j\rangle} s_i J_{i,j} s_j$, where the sum runs over all couplings $\langle i, j\rangle$. Here the task is to find a spin configuration for a given set of coupling constants that minimizes the energy of the spin glass. The states with minimum energy are called *ground states*. The spin configurations are encoded with binary strings where each bit specifies the value of one spin (0 for a spin +1, 1 for a spin -1). One generally analyzes a large set of random spin glass instances for a given distribution of the spin-spin couplings. In this paper we consider the $\pm J$ spin glass, where each spin-spin coupling is set randomly to either $+1$ or -1 with equal probability. We use instances arranged on square grids of sizes 8×8, 10×10, 12×12, 14×14, 16×16, and 18×18 spins; that is, the problem sizes range from 64 to 324 spins. We consider periodic boundary conditions. For each problem size, we use 1,000 unique, independently generated problem instances. All instances were obtained from the Spin Glass Ground State Server at the Univ. of Cologne. Also for spin glasses, the number of instances is relative large because small sets of random instances provide often only limited insight.

4.2 Experimental Setup

The maximum number of iterations for each problem instance is set to the overall number of bits in the problem; according to preliminary experiments, this upper bound substantially exceeds the actual number of iterations required to solve each problem. Each run is terminated either when the global optimum has been found, when the population consists of copies of a single candidate solution, or when the maximum number of iterations has been reached. For each problem instance, we use bisection (Sastry, 2001; Pelikan, 2005) to ensure that the population size is within 5% of the minimum population size to find the global optimum in 10 out of 10 independent runs. For concatenated traps, 10 bisection runs are executed for each problem size. To make the results more stable and increase efficiency of LTGA, the probabilities of 0s and 1s in each position are enforced to be as close to 50% as possible.

As was discussed in section 2, local search based on the bit-flip neighborhood was used on every string in the initial population. The reason for using local search on the initial population is that otherwise the initial linkage tree is fully random, and the effects of the first round of crossover are often significantly disruptive. Another possibility to eliminate these disruptive effects would be to use one of the common selection operators, such as tournament selection, prior to the first iteration of LTGA.

To evaluate performance of each algorithm, we focus on the number of steps of the local search ran on the initial population, the overall number of fitness evaluations, and the CPU time. The steps of the local search are not counted as evaluations in order to distinguish between evaluations and local search steps, because for many additively decomposable problems, performing a local search step is much less computationally expensive than evaluating a solution. In most cases, the growth of the number of local search steps, the numbers of evaluations, or the CPU times are

approximated by a function an^b where $a, b > 0$, which provides a polynomial approximation of these statistics. All experiments were done on AMD Opteron 2360 SE processors of 1300 MHz. LTGA was implemented in C/C++ using the GNU Compiler Collection 4.3.1. Although the code could be further optimized for efficiency, the primary focus of our experiments concerning the CPU times was on the *growth* of the CPU times rather than their absolute values.

4.3 Results

4.3.1 Pairwise vs. exact metric

In the first set of experiments, we examine and compare performance of LTGA with the original, exact metric for measuring cluster distances (section 2) and LTGA with the pairwise metric based on distances between pairs of variables (section 3.1). The results for all test problems are shown in figure 1. It is of note that largest problem sizes are omitted for LTGA with the exact metric due to the limited computational resources and the substantial computational cost of running experiments with this version of LTGA.

For trap-5, trap-6 and trap-7 the growth of the number of local search steps and the number of evaluations are nearly identical for both the pairwise and the exact metric. This indicates that for traps, the growth of the number of evaluations and the number of steps of the local search are not affected by the use of pairwise statistics. With respect to the CPU time, the rate of growth for the exact metric is much worse (faster) than that for the pairwise one, and the differences in the CPU time become even more substantial as k grows. The advantages of using the pairwise metric rather than the exact one grow with problem size in terms of the overall CPU time. It is also of note that the population sizes (not shown) for all separable problems are much smaller than those required by most other linkage learning GAs, both in terms of their absolute values and the rate of growth. It is also interesting to mention that without running the local search on the initial population (these results are not shown in this paper), the rate of growth of all examined statistics deteriorates as k grows. Nonetheless, this is not the case when local search is used.

The results for nearest-neighbor NK landscapes and 2D spin glasses are somewhat surprising because in both cases, the rates of growth of the number of local search steps and the number of evaluations are substantially greater for the exact metric than for the pairwise metric. That means that using the metric based on pairwise statistics appears to be beneficial not only in terms of the CPU time required for computing the distance metric, but it also leads to qualitatively better (more effective) linkage trees. The differences are further magnified in terms of the CPU time. Comparing the obtained results to those obtained with other advanced linkage learning GAs such as the hierarchical BOA (Pelikan, 2010; Pelikan, 2005) indicates that while LTGA appears to be competitive for separable problems, the presence of overlap between subproblems has a significant negative effect on LTGA performance, making it much less competitive than its peers.

It is important to note that while the polynomial fits for the number of local search steps, the number of evaluations, and the CPU times on 2D spin glasses appear to be quite accurate, the log-scale plots indicate that the growth of these statistics may be faster than polynomial. On the other hand, for NK landscapes, the polynomials grow slightly faster than the actual values. Evaluating LTGA on spin glass and NK landscape instances of even larger sizes may help to shed light on this topic, although due to the resource constraints we do not have such results at the moment.

In summary, the results confirm the hypothesis that the pairwise metric outperforms the exact metric in terms of the overall CPU time due to the faster computation of cluster distances. Nonetheless, the results indicate that for decomposable problems with overlap between subproblems the pairwise metric outperforms the exact metric even with respect to the number of local

search steps and the number of evaluations both in terms of the magnitude and the rate of growth.

4.3.2 Pairwise vs. problem-specific metric

This section compares performance of LTGA with the metric based on pairwise statistics (section 3.1) and LTGA with the problem-specific metric (section 3.2).

Although we would intuitively expect that the problem-specific metric would be far superior to the pairwise metric on at least separable problems of bounded order, such as trap-5, trap-6, and trap-7, the results indicate that the two metrics perform almost identically in terms of both the number of local search steps and the number of evaluations. Furthermore, in terms of the growth of the CPU time, the problem-specific metric is outperformed by the pairwise metric, although for the problem sizes tested in this paper the runs with the problem-specific metric are still less time consuming than those with the pairwise metric.

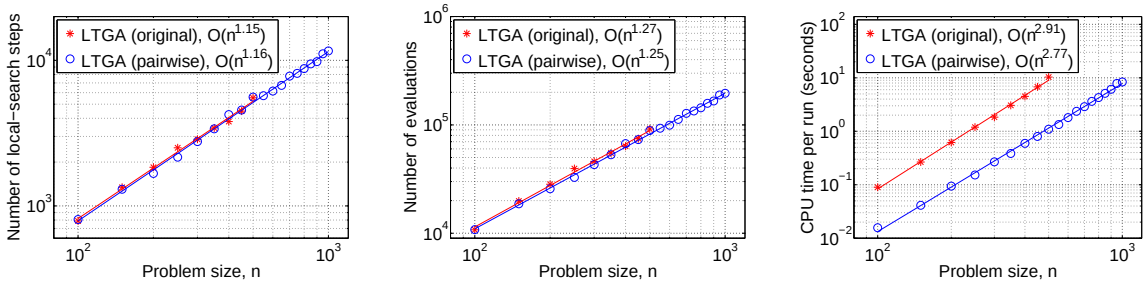
For nearest-neighbor NK landscapes, the problem-specific metric outperforms the pairwise metric in terms of all three statistics, the number of local search steps, the number of evaluations, and the CPU time. The problem-specific metric outperforms the pairwise one both in the magnitude and the rate of growth. Improvement of LTGA performance for decomposable problems with overlap is certainly good news due to the fact that LTGA performance on such problems appears to be significantly worse than that of some other competent linkage learning evolutionary algorithms, such as the hierarchical BOA (Pelikan, 2010; Pelikan, 2005).

The bad news is that the performance gains due to the use of problem-specific metric do not necessarily generalize to other decomposable problems with overlap. Specifically, the results for 2D spin glasses indicate that the problem-specific metric performs worse than the approximate metric in terms of both the magnitude as well as the rate of growth, whether we look at the number of local search steps, the number of evaluations, or the overall CPU time. This reinforces the observation made in the previous section that LTGA in its current form does not seem to deal well with the complex interaction structure of 2D spin glasses.

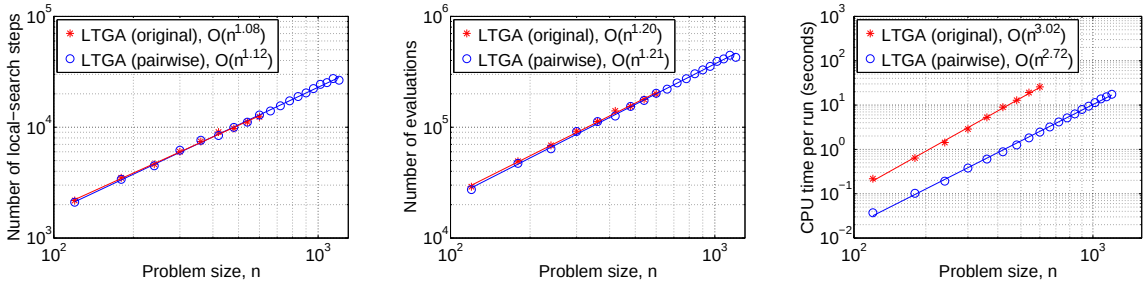
It is also of note that without using local search on the initial population, performance gains of using the problem-specific metric appear to be much more substantial, especially for trap problems (these results not included in the paper). This is most likely due to the fact that with the problem-specific metric, the initial linkage tree corresponds closely to the actual problem structure, whereas for LTGA the initial linkage tree is random (assuming a random initial population). Due to the deceptive nature of trap problems, using a random linkage tree can bias the search away from the global optimum.

4.3.3 Dependence on k

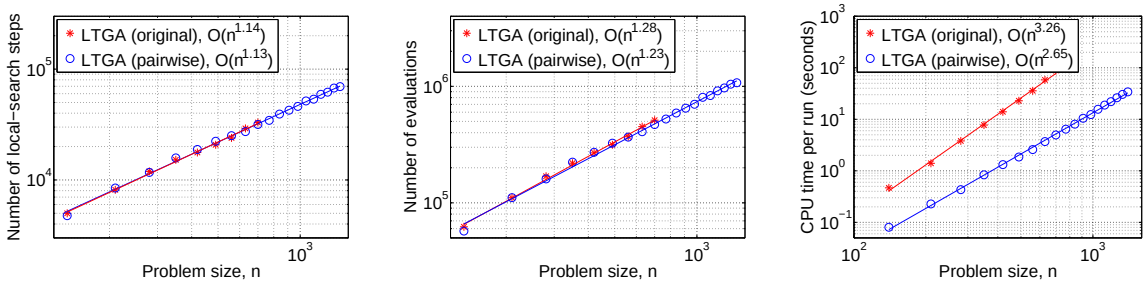
To further examine performance of the two metrics described in sections 3.1 and 3.2, we applied LTGA to a series of trap-k problems of $n = 840$ bits and $k \in \{5, 6, 7, 8\}$. The results for LTGA with the approximate and problem-specific metrics are shown in figure 3. Like for most other linkage learning GAs (Yu, Sastry, Goldberg, & Pelikan, 2007; Pelikan, Sastry, & Goldberg, 2002), LTGA performance degrades exponentially fast with k. It is of note that without local search on the initial population, performance degrades somewhat more slowly with k but, at the same time, LTGA scalability deteriorates with k (the order of the polynomial capturing the time complexity grows with k).



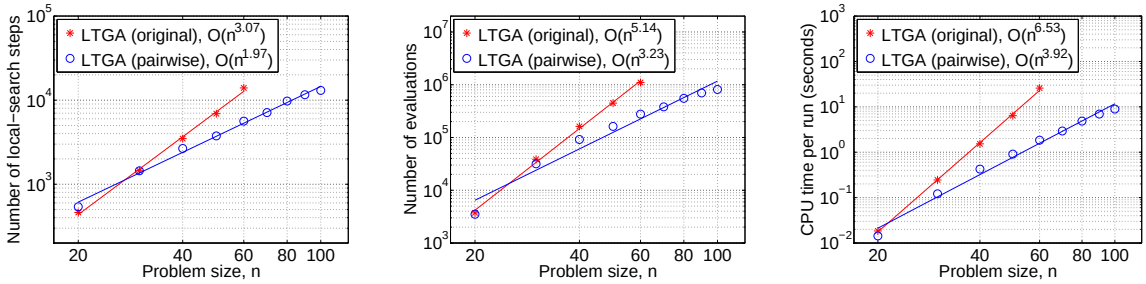
(a) Trap-5



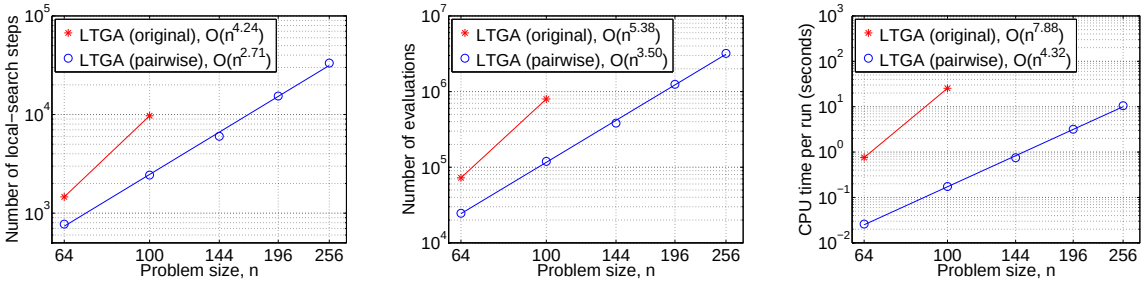
(b) Trap-6



(c) Trap-7

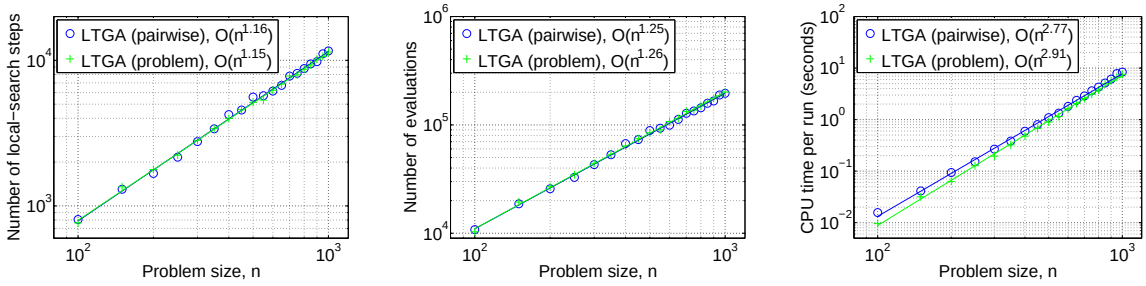


(d) Nearest-neighbor NK landscapes

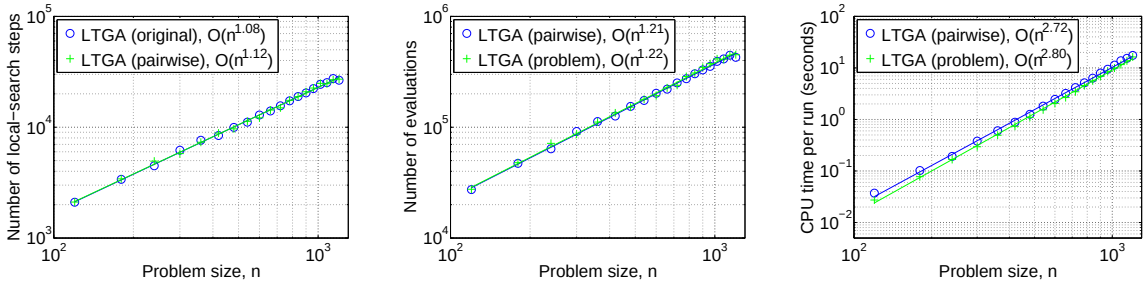


(e) 2D spin glass

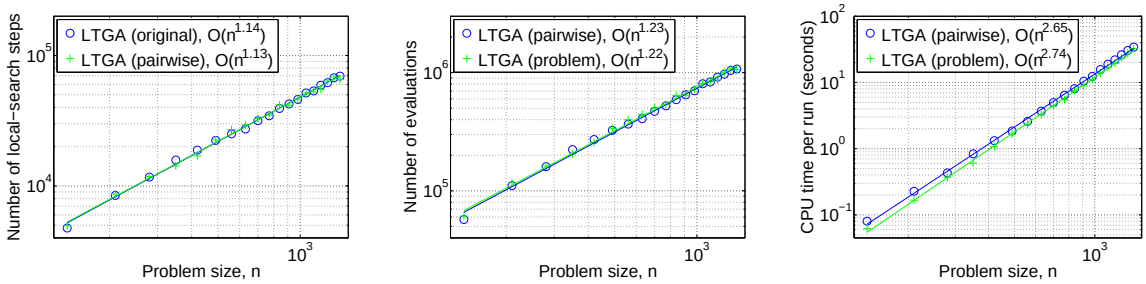
Figure 1: Comparison of LTGA with the exact metric and LTGA with the approximate metric based on pairwise statistics.



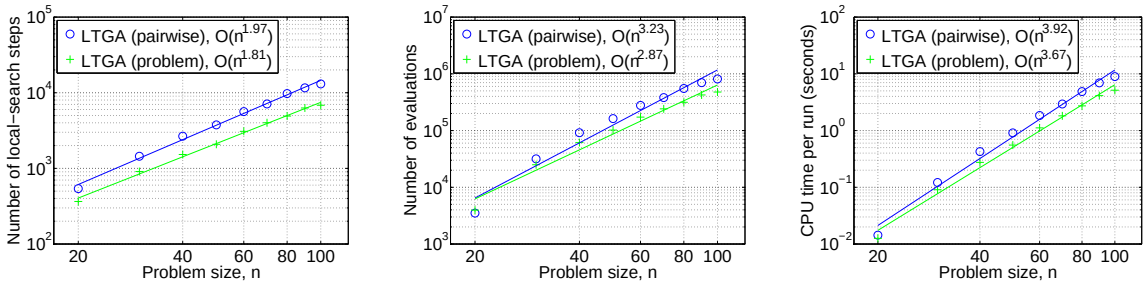
(a) Trap-5



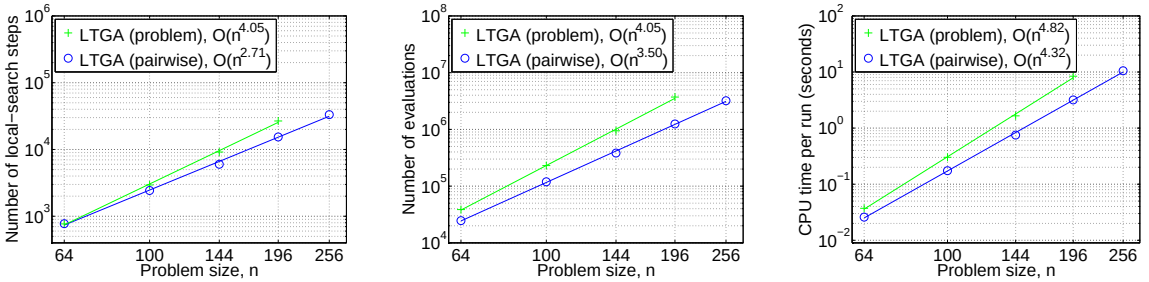
(b) Trap-6



(c) Trap-7



(d) Nearest-neighbor NK landscapes



(e) 2D spin glass

Figure 2: Comparison of LTGA with the approximate metric based on pairwise statistics and LTGA with the problem-specific metric based on the problem definition.

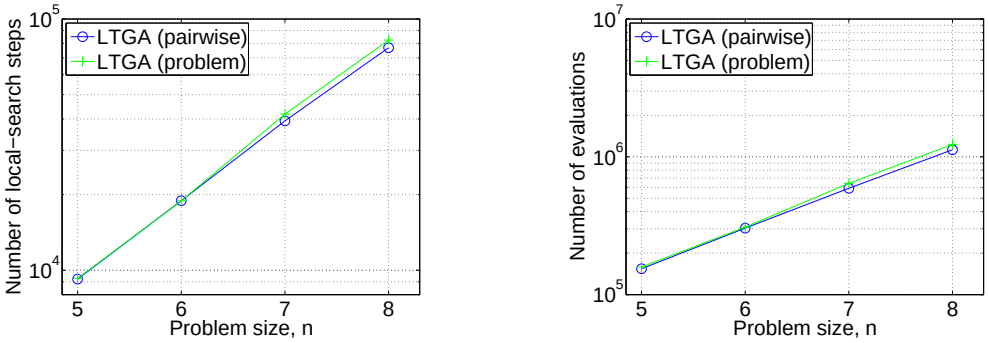


Figure 3: Performance of LTGA with respect to k for concatenated traps of order k. It is of note that unlike other figures in this paper, here the x-axis is linear and y-axis is logarithmic; therefore, exponential growth is represented by a straight line.

5 Summary and Conclusions

This paper presented a thorough empirical evaluation of the linkage tree genetic algorithm (LTGA) on a large set of additively decomposable problems, including concatenated traps of order k, nearest-neighbor NK landscapes, and 2D spin glasses. The paper also examined two efficient distance metrics which can replace the standard distance metric used in LTGA to create linkage trees. The first metric computes the distance between two clusters as the average distance between pairs of variables in these two clusters. The second metric is based on problem-specific information.

LTGA was shown to perform great on separable problems of bounded order, solving all these problems in low-order polynomial time that is in fact better than that for some other advanced linkage learning GAs, such as BOA. The good news is also that the population sizes were much smaller than those for other advanced linkage learning GAs. However, the results also indicated that LTGA performance deteriorates in presence of overlap between subproblems in an additive problem decomposition, making this algorithm much less competitive for additively decomposable problems with overlapping subproblems.

The pairwise and the problem-specific metrics outperformed the exact metric in most scenarios, both in the magnitude and the rate of growth, whether the complexity is measured by the number of local search steps, the number of evaluations, or the CPU time. The results thus provide a clear support for the superiority of the approximate and problem-specific metrics. With respect to the comparison of the pairwise and problem-specific metrics, the results are not as clear. On concatenated traps and NK landscapes, the problem-specific metric performed as well or better than the pairwise metric, although on 2D spin glasses, the situation reversed.

Future work should focus especially on the design of problem-specific metrics that are more robust and effective than the variant proposed in this paper. The use of problem-specific metrics should be useful in many problem classes, including difficult graph problems and other additively decomposable functions. Problem-specific metrics based on previous runs on similar problems should also be examined. It is also important to investigate the reasons for the rapid deterioration of LTGA performance on problems of complex structure with substantial overlap between subproblems and the ways in which these difficulties can be alleviated. Finally, there are many lessons from the design of advanced estimation of distribution algorithms and other linkage learning genetic algorithms which can be used to further broaden applicability and to improve efficiency and scalability of LTGA.

Acknowledgments

This project was sponsored by the National Science Foundation under CAREER grant ECS-0547013, and by the Univ. of Missouri in St. Louis through the High Performance Computing Collaboratory sponsored by Information Technology Services. Most experiments were performed on the Beowulf cluster maintained by ITS at the Univ. of Missouri in St. Louis. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Ackley, D. H. (1987). An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, 170–204.

Deb, K., & Goldberg, D. E. (1991). *Analyzing deception in trap functions* (IlliGAL Report No. 91009). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Kluwer.

Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. *Foundations of Genetic Algorithms*, *4*, 247–262.

Hauschild, M., & Pelikan, M. (2008). Enhancing efficiency of hierarchical BOA via distance-based model restrictions. *Parallel Problem Solving from Nature*, 417–427.

Hauschild, M., & Pelikan, M. (2009). Intelligent bias of network structures in the hierarchical BOA. *Genetic and Evol. Comp. Conf. (GECCO-2009)*, 413–420.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Kauffman, S. (1989). Adaptation on rugged fitness landscapes. In Stein, D. L. (Ed.), *Lecture Notes in the Sciences of Complexity* (pp. 527–618). Addison Wesley.

Kauffman, S. (1993). *The origins of order: Self-organization and selection in evolution*. Oxford University Press.

Pelikan, M. (2005). *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Springer.

Pelikan, M. (2010). NK landscapes, problem difficulty, and hybrid evolutionary algorithms. *Genetic and Evol. Comp. Conf. (GECCO-2010)*, 665–672.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000). Bayesian optimization algorithm, population sizing, and time to convergence. *Genetic and Evol. Comp. Conf. (GECCO-2000)*, 275–282.

Pelikan, M., Sastry, K., & Goldberg, D. E. (2002). Scalability of the Bayesian optimization algorithm. *International Journal of Approximate Reasoning*, *31*(3), 221–258.

ping Chen, Y., Yu, T.-L., Sastry, K., & Goldberg, D. E. (2007). *A survey of genetic linkage learning techniques* (IlliGAL Report No. 2007014). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Sastry, K. (2001). *Evaluation-relaxation schemes for genetic and evolutionary algorithms*. Master's thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL.

Thierens, D. (1999). Scalability problems of simple genetic algorithms. *Evolutionary Computation*, *7*(4), 331–352.

Thierens, D. (2010). The linkage tree genetic algorithm. *Parallel Problem Solving from Nature*, 264–273.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proc. of the Int. Conf. on Genetic Algorithms (ICGA-93)*, 38–45.

Young, A. (Ed.) (1998). *Spin glasses and random fields*. Singapore: World Scientific.

Yu, T.-L., Sastry, K., Goldberg, D. E., & Pelikan, M. (2007). Population sizing for entropy-based model building in estimation of distribution algorithms. *Genetic and Evol. Comp. Conf. (GECCO-2007)*, 601–608.