Hierarchical Problem Solving with the Linkage Tree Genetic Algorithm

Dirk Thierens
Universiteit Utrecht
Utrecht, The Netherlands
D.Thierens@uu.nl

Peter A.N. Bosman
Centrum Wiskunde & Informatica (CWI)
Amsterdam, The Netherlands
Peter.Bosman@cwi.nl

ABSTRACT

Hierarchical problems represent an important class of nearly decomposable problems. The concept of *near decomposability* is central to the study of complex systems. When little a priori information is available, a black box problem solver is needed to optimize these hierarchical problems. The solver should be able to learn linkage information, and to preserve and test partial solutions at different levels in the hierarchy. Two well known benchmark functions - shuffled Hierarchical If-And-Only-If (HIFF) and shuffled Hierarchical Trap (HTRAP) functions - exemplify the challenges posed by hierarchical problems. Standard genetic algorithms are unable to solve these problems, and specific methods, like SEAM and hBOA, have been designed to address them. In this paper, we investigate how the recently developed Linkage Tree Genetic Algorithm (LTGA) performs on these hierarchical problems. We compare LTGA with SEAM and hBOA on HIFF and HTRAP functions. Results show that, although LTGA is a simple algorithm compared to SEAM and hBOA, it nevertheless is a very efficient, reliable, and scalable algorithm for solving the randomly shuffled versions of HIFF and HTRAP, two hard, hierarchical problems.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Performance, Experimentation

Keywords

Evolutionary Computation, Estimation-of-Distribution Algorithms, Hierarchical Problems, Linkage Learning, Optimal Mixing, Linkage Tree Genetic Algorithm.

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

1. INTRODUCTION

Hierarchical systems are composed of subsystems which are hierarchies by themselves until one recursively reaches a basic element. Interactions within each subsystem at a certain level are stronger than the interactions between the different subsystems. The subsystems can also be viewed as partial solutions to a complex problem. By recursively constructing partial solutions, hierarchy represents an organizational method to address large, complicated problems in a scalable and efficient way. Simon's classical book, *The Sciences of the Artificial*, discusses the role of hierarchy and modularity to achieve what he called *near decomposability* [15]. It is through *near decomposability* that complex systems can be build, whether they are engineering systems, social systems, economical systems, or biological systems.

Since hierarchical systems are so ubiquitous, it is important to study how evolutionary algorithms can solve these types of problems. Watson and Pollack did pioneering work on this subject and introduced the well known HIFF problem. They also designed an evolutionary algorithm, SEAM, based on symbiotic composition - as opposed to sexual recombination - to solve HIFF problems [20][21]. Pelikan and Goldberg also recognized the importance of hierarchy early on, and defined an even more challenging hierarchical problem, HTRAP [9][10][11]. They also designed hBOA which is capable of solving HIFF and HTRAP efficiently. De Jong, Watson and Thierens delineated the class of hierarchical problems and analyzed a framework for hierarchical genetic algorithms [4][5].

When studying computational methods to solve hierarchical systems it is important that we focus on black-box approaches. The problem solvers should be able to identify the near decomposable and hierarchical structure by itself - this is, without being given this problem information a priori. It is only when we treat the problem this way that we obtain computational tools that are widely applicable to Simon's important class of complex problems. Therefore, we focus in the experimental analysis on the shuffled versions of HIFF and HTRAP. In the shuffled version all problem variables are randomly shuffled over the entire problem representation so the problem solver has no clue about what problem variables belong together in a subsystem of the hierarchy. It should be noted that standard genetic algorithms are incapable of solving such systems.

In recent years, we have introduced a very efficient linkage learning evolutionary algorithm, called the Linkage Tree Genetic Algorithm (LTGA) [16][19]. The purpose of this paper is to see whether LTGA can solve the shuffled ver-

sions of the HIFF and HTRAP problems. We also compare LTGA with SEAM and hBOA. The next Section discusses LTGA, SEAM, and hBOA. Section 3 specifies the HIFF and HTRAP hierarchical functions. In Section 4 we show experimental results of LTGA on HIFF and HTRAP, and compare them with SEAM and hBOA. Section 5 discusses how LTGA handles key aspects of hierarchical problems. Finally, Section 6 concludes the paper.

2. BACKGROUND

2.1 LTGA

The Linkage Tree Genetic Algorithm is a population-based, stochastic local search algorithm that identifies which variables should be treated as a dependent set during the exploration phase [16][19][3] . LTGA represents this dependence information in a hierarchical cluster tree of the problem variables. The linkage tree of a population of solutions is the hierarchical cluster tree of the problem variables built by an agglomerative hierarchical clustering algorithm. The distance measure $D(X_1, X_2)$ represents how much dependency there is between two sets of variables X_1 and X_2. We use as distance measure the mutual information I.

An agglomerative hierarchical clustering algorithm proceeds bottom-up. First, each problem variable is assigned to a single cluster. Then, the clustering algorithm recursively joins the closest clusters until only one cluster is left. A fast clustering technique is the average linkage clustering or unweighted pair group method with arithmetic mean (UPGMA) that only looks at distances between pairs of variables: $D(X_1, X_2) = \frac{1}{|X_1||X_2|} \sum_{X \in X_1} \sum_{Y \in X_2} I(X, Y)$. This distance measure can be computed very efficiently because all required information-theoretic measures between pairs of variables can be calculated beforehand [6, 17, 19, 13]. Efficient implementations of hierarchical clustering run in $O(n\ell^2)$ time, with n the population size and ℓ the problem length [7].

The linkage tree has ℓ leaf nodes (the clusters having a single problem variable) and $\ell - 1$ internal nodes. Each node of the linkage tree divides the set of problem variables into two mutually exclusive subsets. One subset is the cluster of variables at that node, while the other subset is the complementary set of problem variables. LTGA uses this division of problem variables as linkage sets and samples their values as a group. Sample values are obtained by taking a random solution of the current population as donor, and copying the corresponding variable values to the current parent solution.

Each generation LTGA builds a linkage tree of a selected set of individual solutions from the current population. The selection process is standard tournament selection with tournament size $s = 2$. Note that the selected solutions are only used to build the linkage tree, the rest of the algorithm proceeds with the current population. New solutions are generated by greedily exploring the neighborhood of each solution in the current population. This neighborhood is defined by the linkage tree. LTGA traverses the tree in the opposite order of the merging of the clusters by the hierarchical clustering algorithm. New solutions are only accepted when they have a bette, or equal, fitness value than the original solution. When the tree is completely traversed, the solution obtained is copied to the population of the next generation. This tree traversal process is done for each solution in the current population.

It should be noted that half of the variable samples are single bit values, so LTGA is also performing a partial 1-bit local search. As an efficiency gain, LTGA always checks whether the offspring solutions are actually different from their parent solution, if not no call to the fitness function is done, and the algorithm simply proceeds with its tree traversal.

2.2 SEAM

The Symbiogenic Evolutionary Adaptation Model (SEAM) differs from standard genetic algorithms in three key components [20][21]. First, SEAM uses symbiotic combination as an alternative to sexual recombination. Whereas sexual recombination extracts and reassembles gene-subsets from the parents in new combinations, symbiotic combination simply joins whole individuals. Second, since the individuals in SEAM are partially specified, a template is needed to fill in the blanks. Sets of other, partially-specified individuals are chosen from the population in order to build a template and to evaluate two competing individuals within this context. Third, SEAM applies Pareto coevolution to preserve population diversity.

2.3 hBOA

The Hierarchical Bayesian Optimization Algorithm (hBOA) and its predecessor, BOA, belong to the class of Estimation-of-Distribution (EDA) algorithms [9][10][11][12].

BOA and hBOA construct, each generation, a Bayesian network to model the probabilistic dependencies between the problem variables of the most promising solutions in the current population. This Bayesian network is then sampled to generate new candidate solutions. In order to address hard, hierarchical functions, BOA was extended to hBOA. One extension was the use of decision trees in order to make the representation of the large number of conditional probabilities tractable. Another extension was the use of restricted tournament replacement as niching technique in order to ensure diversity maintenance in the population.

3. HIERARCHICAL PROBLEMS

There are two well known hierarchically decomposable benchmark functions in the EC literature: the Hierarchical If-And-Only-If (HIFF) function and the Hierarchical Trap (HTRAP) function. In this paper we are only interested in the randomly shuffled versions of HIFF and HTRAP, so even when not explicitly stated, we always refer to these versions when talking about HIFF or HTRAP problems.

3.1 HIFF

The Hierarchical If-And-Only-If function is structured as a balanced binary tree. Each node X in the tree is at a distance *height(X)* from its leaf nodes. Leaf nodes contribute 1 to the fitness and have a value 0 or 1 as specified by the input string. Actually, we could as well assign no fitness contributions to leaf nodes, but we stick here to the original formulation. Internal or parent nodes contribute to the fitness by $2^{height(X)}$ if and only if their children have both a value either 0 or 1. Otherwise, the fitness contribution is 0. It should be noted that due to the exponentially scaled fitness contribution each level in the tree has the same magnitude to the fitness contribution. The value of a parent

node is 0 (resp. 1) when both its children are 0 (resp. 1), and value NIL otherwise.

3.2 HTRAP

The Hierarchical Trap function is structured as a balanced k-ary tree with $k \geq 3$. Similarly to HIFF, internal nodes have a value 0 (resp. 1) if all their children nodes have value 0 (resp. 1), and value NIL otherwise. The contribution of the internal nodes to the overall fitness is a function of unitation, meaning it is depending on the number of ones and zeroes in the children nodes. If there is one or more child node with value NIL, then the fitness contribution of the parent node is 0. The fitness contribution function of an internal node X is $f(u(X)) \times k^{height(X)}$ where $f(u(X))$ is the classical trap function with u the number of ones (and $k-u$ zeroes) at the children nodes of X:

$$f(u) = \begin{cases} f_{high} & \text{if } u = k \\ f_{low} - u\frac{f_{low}}{k-1} & \text{otherwise.} \end{cases}$$

At the root node $f_{high} = 1$ and $f_{low} = 0.9$, while for all other internal nodes $f_{high} = f_{low} = 1$. The global optimal solution is thus the string of all ones. However, HTRAP biases the search to the solution of all zeroes on each but the top level. In addition, the top level is fully deceptive with the all zeroes solution as deceptive attractor. In the shuffled version all problem variables are randomly positioned on the string representation. Clearly, this is a very hard function to optimize. Not only should the linkage groups be identified, but also the alternative partial solutions at each level need to be preserved in the population until the problem can be solved at the highest level.

4. EXPERIMENTAL RESULTS

The experimental results reported here are all measuring the first hitting time of the global optimal solution - this is, the number of fitness evaluations required to generate the global optimum for the first time during a run. The associated population size is determined by starting from a very small population and letting the algorithm run until the population is fully converged. If the optimum has not been found, the next trial will use a population size twice as large. When the optimum is encountered the minimal population size is searched for by performing a bisection search between the current population size and the previous size which is half as big [14]. All experimental results are averaged over 30 runs, and the success criterion is that all 30 runs find the global optimum. Reported experimental results of SEAM [20] and hBOA [10] use the same success criterion which facilitates the comparison.

4.1 HIFF

We have run experiments on the HIFF function for string lengths respectively $\ell = 16, 32, 64, 128, 256, 512$. Figure 1 plots the number of function evaluations and the associated, minimal population size. The smallest HIFF problem, $\ell = 16$, can be solved reliably by LTGA in about 500 function evaluations with an average population size of 24, while the largest HIFF problem, $\ell = 512$, is solved in a little less than 300000 function evaluations with an average population size of 112.

The SEAM algorithm, as reported in [20], required about 220000 function evaluations for the HIFF problem of length $\ell = 64$. The population size was started at 1000 and is reduced during run time. No experimental results for larger string lengths are reported.

Figure 1 also shows the results for hBOA taken from [9]. It is truly remarkable that the number of function evaluations needed by hBOA are almost indistinguishable from those of LTGA. Although both algorithms learn linkage structure while optimizing the problem, they are fundamentally different in the way they learn linkage information, and in the way they use this structural information to generate new solutions. As a result of these differences LTGA requires much smaller populations sizes than hBOA. Unfortunately, no population sizes have been reported in [9], but in general hBOA - and other entropy-based model building EDAs - require a order of magnitude larger population sizes than LTGA [22].

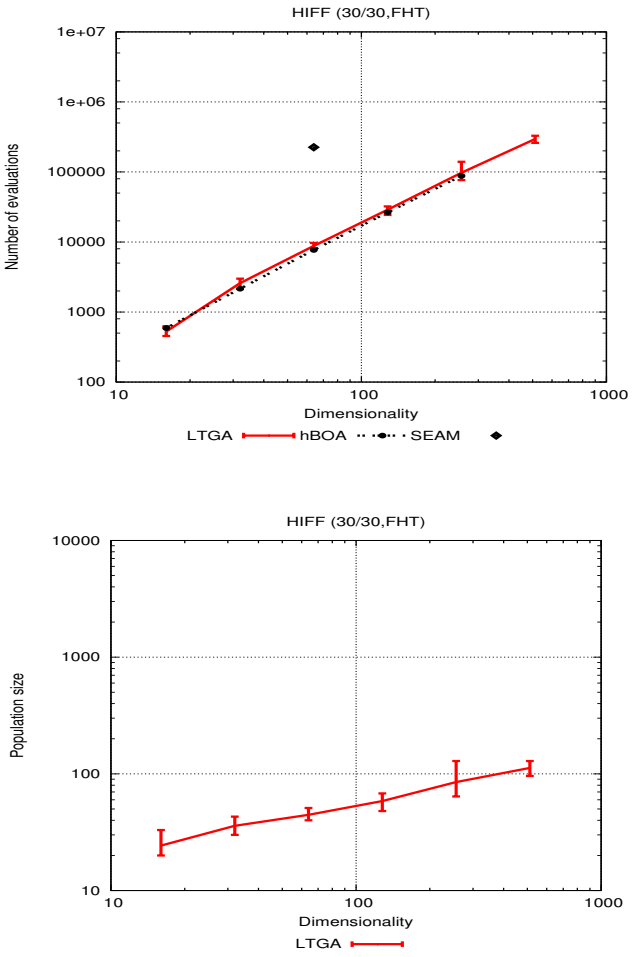


Figure 1: Results on HIFF problems of different length with LTGA, hBOA, and SEAM.

4.2 HTRAP

We have run experiments on the HTRAP function with $k = 3$ and string lengths respectively $\ell = 27, 81, 243, 729$. Figure 2 plots the number of function evaluations and the associated, minimal population size. The smallest HTRAP problem, $\ell = 27$, can be solved reliably by LTGA in about

8500 function evaluations with an average population size of 100, while the largest HTRAP problem, $\ell = 729$, is solved in about 1130000 function evaluations with an average population size of 320. It can also be seen that the variation of the first hitting time is small, indicating a consistent convergence behavior.

Figure 2 also shows the results for hBOA taken from [9]. For the smaller HTRAP function hBOA needs around half the number of function evaluations than LTGA. However, the number of function evaluations seems to scale worse for hBOA than for LTGA. No experimental results where reported for hBOA on a problem with length $\ell = 729$, but by extrapolating the curve in Figure 2 we can see that hBOA appears to need about double the amount of function evaluations than LTGA. It should also be noted that the computational run time for building a hierarchical tree is much lower than for building a Bayesian network.

SEAM has not been applied on the HTRAP function, but considering the result on HIFF it is safe to assume that it would be significantly less efficient than hBOA and LTGA. Due to the deceptive nature of HTRAP it is even doubtful if SEAM could solve these functions at all.

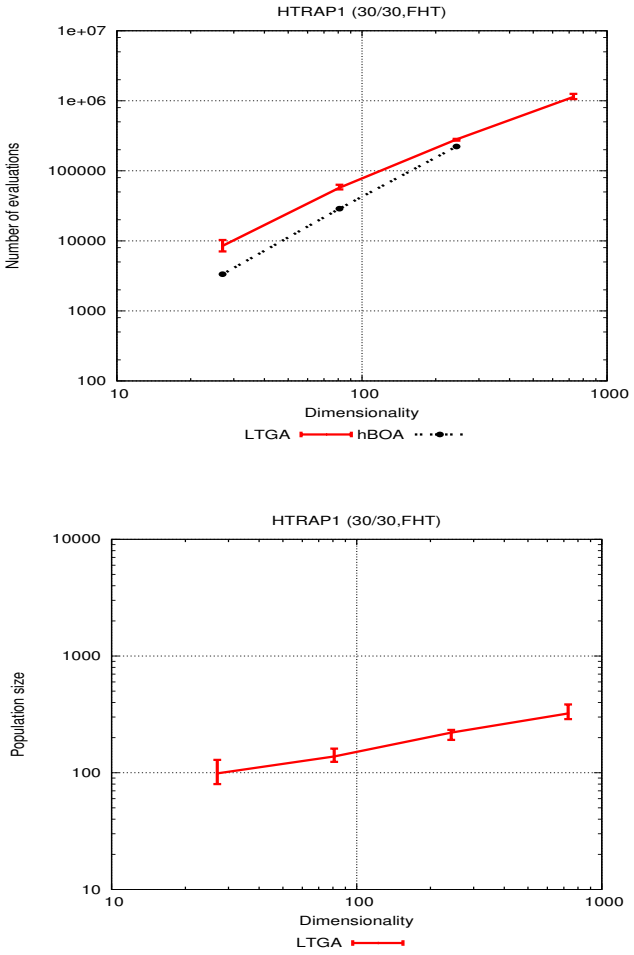


Figure 2: Results on HTRAP problems of different length with LTGA and hBOA.

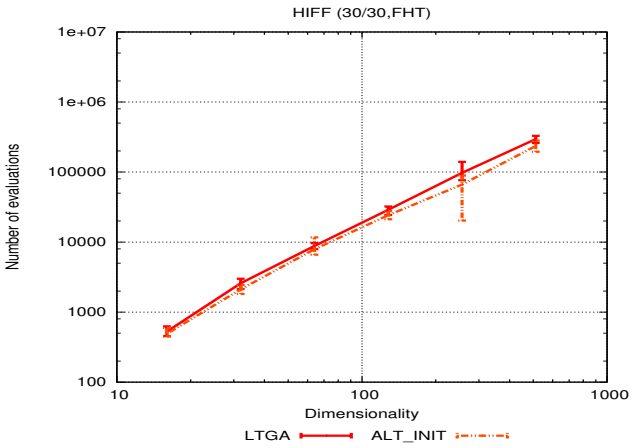
4.3 Alternative Initialization

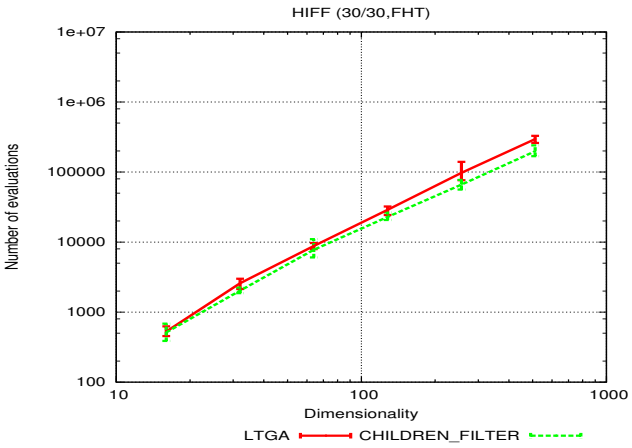
LTGA requires a much smaller population size than the typical Genetic Algorithm or Estimation-of-Distribution Algorithm. For small population sizes, the way the initial, random population is constructed becomes an influencing factor. Although it gets hardly ever reported in the literature, it is safe to assume that most researchers create an initial random population by flipping a fair coin for each bit value of each solution in the population, and thus approximating a binomial distribution. However, for small populations it is more beneficial to initialize the population by enforcing that for each problem variable there are exactly half of the population size zeroes and the other half ones. Such a random population can easily be constructed by starting from a population with half of the solutions consisting of all zeroes while the other half are all ones, and then shuffle the bit values. Care should be taken that enough shuffling iterations are performed to completely decorrelate the bit values (here we applied 15 shuffling iterations). Figure 3 shows experimental results on the HIFF problem. The minimal required population size and the number of function evaluations are now both noticeably smaller. For the HTRAP function, the required population sizes are larger, therefore the random initialization method has less impact.

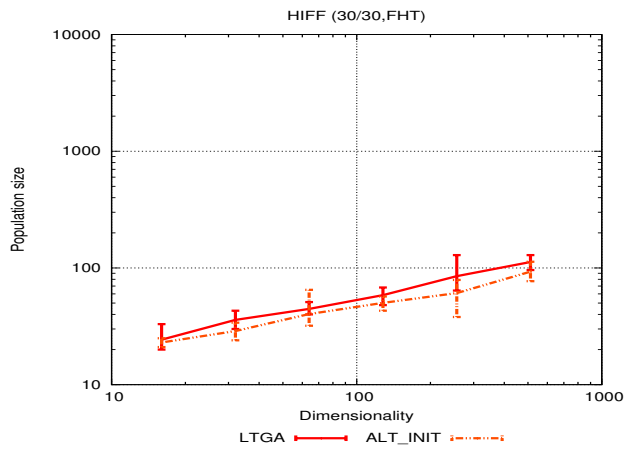
4.4 Child Node Filtering

LTGA builds a linkage tree by using a bottom-up, hierarchical clustering algorithm. Cluster trees have ℓ terminal nodes and $\ell - 1$ internal nodes, so the number of linkage groups of size at least 2 is always $\ell-2$ (note that the top node in the cluster tree covers all problem variables and therefore is not used as a linkage group). However, enforcing this fixed amount of linkage groups can cause some redundant fitness function evaluations and therefore decrease the performance. To see when this is the case let us consider a small example. Assume we have a problem with length 6 and a population of size 5, $P = \{000000, 000111, 0000111, 111000, 111000\}$, resulting in a linkage tree $T = \{(0), (1), (2), (3), (4), (5), (01), (012), (34), (345)\}$. The linkage group (012) has only the partial solutions {000***} and {111***} present in the population, similarly the linkage group (345) has only {***000} and {***111}. Since the population has converged to this state, we can assume that the missing partial solutions {100***, 010***, 001***, 110***, 101***, 011***} and {***100, ***010, ***001, ***110,***101, ***011} are deemed no longer useful in the search process. Therefore, it would be inefficient to let the exploration operators create these partial solutions anew. Clearly, this is exactly what LTGA would do when using one of the linkage groups $(0), (1), (2), (3), (4), (5), (01), (34)$ during the mixing process. We can eliminate this redundant exploration by recognizing that this situation occurs whenever we have a linkage group G with entropy $H(G)$ and all its children linkage groups have the same entropy $H(G)$. In the example above the linkage group (012) has entropy $H(012) = -\frac{2}{3}\ln\frac{2}{3} - \frac{1}{3}\ln\frac{1}{3} = 0.6365$. All its children linkage groups (0),(1),(2), and (01) have the same entropy value and are therefore redundant.

During the construction of the linkage tree we are already computing the entropy values of the linkage groups, so it requires little extra effort to check whether some linkage groups can be eliminated. When the hierarchical clustering algorithm wants to merge two clusters F_i and F_j with equal entropy $H(F_i) = H(F_j)$, and the entropy of the joint cluster







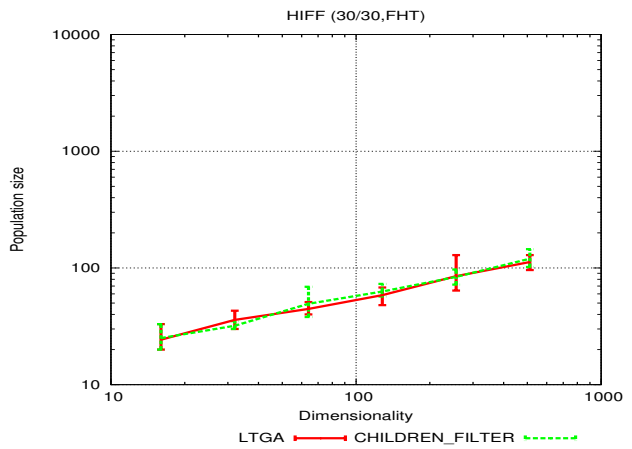


Figure 3: Results on HIFF problems of different length with LTGA using different initialization procedures.

Figure 4: Results on HIFF problems of different length with LTGA using child node filtering.

F_{ij} is also equal to this value $H(F_{ij}) = H(F_i) = H(F_j)$, then we remove the clusters F_i and F_j from the linkage tree and add the joined cluster F_{ij}.

Figure 4 shows the reduction in function evaluations due to the filtering of redundant linkage subsets in case of the HIFF function. For the largest problem considered, $\ell = 512$, the number of function evaluations is reduced from approximately 300000 to 200000. The filtering does not have any effect on the minimal population size required. For the HTRAP function the gain is relatively less significant.

It should be noted that the filtering criterion we use here is rather stringent: the entropies of the merged clusters and their joined cluster need all to be equal. This only happens during the last generations so the impact of the filtering is somewhat limited. A more relaxed way of filtering, and additionally, ways to filter parent nodes, are discussed in [2].

5. DISCUSSION

LTGA and hBOA are both capable of solving difficult hierarchical and nearly decomposable problems in a scalable way. Pelikan and Goldberg discussed three important challenges that must be considered for the design of robust and scalable solvers for these problems [11][12]. They identified 'Three Keys to Hierarchy Success':

1. *Decomposition.* A competent hierarchical optimizer must be capable of decomposing the problem at each level properly by identifying most important interactions between the problem variables and modeling them appropriately.

2. *Chunking.* A competent hierarchical optimizer must be capable of representing partial solutions at each level compactly to enable the algorithm to effectively process partial solutions of larger order (this becomes most important for highest levels).

3. *Diversity maintenance.* A competent hierarchical optimizer must be capable of effective diversity maintenance to preserve alternative partial solutions until it becomes clear which partial solutions may be eliminated.

In hBOA, *decomposition* is obtained by learning, each generation, a Bayesian network from a selected set of good solutions. *Chunking* is achieved by the use of decision trees to

compactly represent the conditional probability tables. Finally, *diversity maintenance* is ensured by applying a niching technique: restricted tournament replacement (RTR). In RTR, a newly generated solution competes directly against its most similar solution in a randomly chosen set W of solutions from the current population. If the new solution is better it replaces the similar solution, otherwise it is discarded. Similarity is based on the Hamming distance between the solutions.

An interesting question is how LTGA handles these 'Three Keys of Hierarchy Success' ?

1. *Decomposition:* the hierarchical cluster tree uses the mutual information between (sets of) problem variables to identify and represent important interactions between variables at each level.

2. *Chunking:* learning a Bayesian network requires building a dependency graph and computing the conditional probability values. The latter can lead to a very large number of parameters which could make the learning of interactions between a large set of problem variables intractable. That is why hBOA needs a chunking mechanism in the form of decision trees. LTGA however does not build a probability model and therefore does not need to estimate conditional probabilities. The linkage tree is simply the result of the agglomerative hierarchical clustering. This cluster tree is a very compact way of representing linkage information so no specific chunking mechanism is required.

3. *Diversity maintenance:* LTGA has no explicit mechanism to preserve niches in the population, and it might look surprising that it nevertheless can preserve alternative partial solutions in the population. The reason this happens is due to the optimal mixing technique: change in the population only comes from replacing a partial solution by another partial solution from the current population as specified by the linkage sets in the linkage tree. These local changes are immediately evaluated and if the result is a decrease in fitness then they are undone. This mechanism is sufficient to maintain the required diversity in the population.

Both HIFF and HTRAP have a tree structure. LTGA also represents the linkage it induces from the promising solutions as a tree. Obviously, this raises the question whether this structural correspondence is responsible for the excellent performance of LTGA. Clearly, there must be some influence, although it is not clear how big this influence is. It is instructive to recall that LTGA has shown excellent performance on the nearest-neighbor NK-landscape, even though a tree cannot represent the overlapping dependencies present in this problem [19]. In addition, it has been shown that a static linkage grouping that mirrors the overlapping interaction structure of the nearest-neighbor NK-landscape is outperformed by LTGA even though it can only represent linkage information as a tree [18]. On the other hand, experimental results on 2D spin glass problems do indicate that the inability of representing overlapping interactions by a tree can hinder the performance of LTGA [13]. Therefore, we also looked at alternative linkage models like the Linkage Neighbor GA [1]. There are currently no studies on hierarchical problems with overlapping sub-functions, but this is definitely an interesting future research topic.

6. CONCLUSIONS

Hierarchical decomposable systems are at the heart of Simon's *near decomposability* view of complex systems, both natural and man-made. Black-box optimization methods that solve such systems are therefore of the uttermost importance. We have shown that the Linkage Tree Genetic Algorithm is a very efficient, reliable, and scalable algorithm for solving the shuffled versions of HIFF and HTRAP, two hard, hierarchical problems. On the HIFF function, LTGA behaves remarkably similar than hBOA. Both algorithms are much more efficient than SEAM. On the HTRAP function, LTGA needs more function evaluations than hBOA on small problems, however LTGA appears to have better scaling behavior, and requires less function evaluations than hBOA for larger problem lengths. The key advantage of LTGA however is that it is an extremely fast and simple algorithm, but nevertheless capable of solving some very hard, nearly decomposable hierarchical problems, like HIFF and HTRAP.

7. REFERENCES

[1] P. A. Bosman and D. Thierens. Linkage neighbors, optimal mixing and forced improvements in genetic algorithms. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 585–592, New York, NY, USA, 2012. ACM.

[2] P. A. Bosman and D. Thierens. More concise and robust linkage learning by filtering and combining linkage hierarchies. In *Proceedings of the fifteenth international conference on Genetic and evolutionary computation conference*, GECCO '13, New York, NY, USA, 2013. ACM.

[3] P. A. N. Bosman and D. Thierens. The roles of local search, model building and optimal mixing in evolutionary algorithms from a BBO perspective. In N. Krasnogor and P. L. Lanzi, editors, *GECCO (Companion)*, pages 663–670. ACM, 2011.

[4] E. D. de Jong, D. Thierens, and R. A. Watson. Hierarchical genetic algorithms. In X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. M. Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, editors, *PPSN*, volume 3242 of *Lecture Notes in Computer Science*, pages 232–241. Springer, 2004.

[5] E. D. de Jong, R. A. Watson, and D. Thierens. On the complexity of hierarchical problem solving. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1201–1208, New York, NY, USA, 2005. ACM.

[6] T. S. Duque and D. E. Goldberg. A new method for linkage learning in the ECGA. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1819–1820, New York, NY, USA, 2009. ACM.

[7] I. Gronau and S. Moran. Optimal implementations of UPGMA and other common clustering algorithms. *Inf. Process. Lett.*, 104(6):205–210, 2007.

[8] N. Krasnogor and P. L. Lanzi, editors. *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*. ACM, 2011.

[9] M. Pelikan and D. E. Goldberg. Hierarchical problem

solving and the bayesian optimization algorithm. In L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. C. Parmee, and H.-G. Beyer, editors, *GECCO*, pages 267–274. Morgan Kaufmann, 2000.

[10] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001*, pages 511–518. Morgan Kaufmann, 2001.

[11] M. Pelikan and D. E. Goldberg. A hierarchy machine: Learning to optimize from nature and humans. *Complexity*, 8(5):36–45, 2003.

[12] M. Pelikan and D. E. Goldberg. Hierarchical Bayesian optimization algorithm. In M. Pelikan, K. Sastry, and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling*, volume 33 of *Studies in Computational Intelligence*, pages 63–90. Springer, 2006.

[13] M. Pelikan, M. Hauschild, and D. Thierens. Pairwise and problem-specific distance metrics in the linkage tree genetic algorithm. In Krasnogor and Lanzi [8], pages 1005–1012.

[14] K. Sastry. Evaluation-relaxation schemes for genetic and evolutionary algorithms. Technical Report Master's Thesis. IlliGAL Report No. 2002004, Department of General Engineering, University of Illinois at Urbana-Champaign, January 2002.

[15] H. Simon. *The Sciences of the Artificial*. MIT Press, 1968.

[16] D. Thierens. The linkage tree genetic algorithm. In R. Schaefer et al., editors, *Parallel Problem Solving from Nature — PPSN XI*, pages 264–273, Berlin, 2010. Springer–Verlag.

[17] D. Thierens. Linkage tree genetic algorithm: first results. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, GECCO '10, pages 1953–1958, New York, NY, USA, 2010. ACM.

[18] D. Thierens and P. Bosman. Predetermined versus learned linkage models. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 289–296, New York, NY, USA, 2012. ACM.

[19] D. Thierens and P. A. N. Bosman. Optimal mixing evolutionary algorithms. In Krasnogor and Lanzi [8], pages 617–624.

[20] R. A. Watson and J. B. Pollack. Symbiotic combination as an alternative to sexual recombination in genetic algorithms. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. M. Guervós, and H.-P. Schwefel, editors, *PPSN*, volume 1917 of *Lecture Notes in Computer Science*, pages 425–434. Springer, 2000.

[21] R. A. Watson and J. B. Pollack. A computational model of symbiotic composition in evolutionary transitions. *Biosystems*, 69:187–209, 2003.

[22] T.-L. Yu, K. Sastry, D. E. Goldberg, and M. Pelikan. Population sizing for entropy-based model building in discrete estimation of distribution algorithms. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 601–608, New York, NY, USA, 2007. ACM.