Hierarchical Problem Solving and the Bayesian Optimization Algorithm

Martin Pelikan
Illinois Genetic Algorithms Laboratory
University of Illinois
Urbana, IL 61801
pelikan@illigal.ge.uiuc.edu

David E. Goldberg
Illinois Genetic Algorithms Laboratory
University of Illinois
Urbana, IL 61801
deg@illigal.ge.uiuc.edu

Abstract

The paper discusses three major issues. First, it discusses why it makes sense to approach problems in a hierarchical fashion. It defines the class of hierarchically decomposable functions that can be used to test the algorithms that approach problems in this fashion. Finally, the Bayesian optimization algorithm (BOA) is extended in order to solve the proposed class of problems.

1 INTRODUCTION

Recently, the connection between human innovation and genetic algorithms has been discussed (Goldberg, 2000; Holland, 1995; Koza, 1994; Koza, Bennett III, Andre, & Keane, 1999). There are two important implications of this result: the innovation can be thought of as a model of genetic algorithms and the genetic algorithms can be thought of as a model of innovation. Moreover, in the genetic and evolutionary computation community there has been a growing interest in what we call *hierarchical problem solving*.

The purpose of this paper is threefold. The paper discusses why it makes sense to approach problems in a hierarchical fashion and combine promising solutions from lower levels to form the solutions on a higher level. The class of hierarchically decomposable problems, as an extension of widely discussed, used, and analyzed additively decomposable problems, is defined. Finally, the Bayesian optimization algorithm (BOA) is extended in order to solve the described class of problems.

Section 2 provides the background and motivation. The class of hierarchically decomposable problems is defined in Section 3. The Bayesian optimization algorithm is briefly described in Section 4. Section 5 discusses possible extension of models used in the BOA to guide the search in order to adjust the model-building to hierarchical problems. The directions of future research are outlined in Section 6. The paper is summarized and concluded in Section 7.

2 GENETIC ALGORITHMS, INNOVATION, AND HIERARCHY

A genetic algorithm (Holland, 1975; Goldberg, 1989) evolves a population of potential solutions to a given problem. The first population of solutions is generated at random. By means of a measure of quality of solutions given by a user, usually expressed in the form of one or multiple functions, better solutions are selected from the current population. The selected solutions undergo the operators of mutation and crossover in order to create the population of new solutions (the offspring population) that fully or in part replace the original (parent) population. The process repeats until the termination criteria (e.g., convergence to a singleton) given by the user are met.

As it was argued recently (Goldberg, 2000), selection, crossover, and mutation are not very interesting operators when acting alone. By repeatedly applying selection alone, the best solution of the initial population would simply overtake the entire population. More importantly, nothing but the small, randomly generated, region of the search space (the initial population of solutions), would be explored. By repeatedly applying crossover alone, the final effect would be the one of shuffling parts of a set of randomly generated solutions. By repeatedly applying mutation, the neighborhood of randomly generated solutions would only be explored. Thus, the effects of applying either operator by itself would be no better than the one of generating a number of solutions at random with no bias whatsoever.

Important features of these operators emerge when using a combination of these. By using the selection and mutation together, the initial population of solutions is *continually improved* by selecting better solutions and exploring their close neighborhood. By introducing crossover, the solutions are no longer only improved by slight perturbations but pieces of solutions are combined together to form new solutions. This process is not unlike a human *cross-fertilizing innovation* (Goldberg, 2000).

One of the implications of this argument is that the results of genetic and evolutionary computation can be used as a tool for understanding and modeling human innovation. On the other hand, the achievements and experience from human innovation and engineering design can be seen as yet another source of inspiration for genetic and evolutionary computation in order to design methods that solve hard problems of our interest quickly, accurately, and reliably. This paper investigates on using *hierarchical problem solving* as one of the cornerstones of engineering design in order to improve current genetic and evolutionary optimization methods.

In engineering design, the problems are often solved in a hierarchical fashion. New designs or ideas are composed of other designs or ideas without having to reinvent these. Many sub-parts of our new design can be created separately and the final result is produced by combining the alternatives. For example, when designing a car, the car stereo and the engine can be designed separately and combined together to form a part of a new car design. Various alternatives can be tried and the final choice can be done by comparing different combinations of reasonable car stereos and engines. When designing an engine, there is no need to reinvent the carburetor, and one can simply choose one from a set of reasonable carburetors that we have already designed. When completing the design, we can simply use an appropriate engine in combination with the remaining parts (e.g., the car stereo). To put all the parts together, we need not reinvent nuts and bolts each time we modify some part of the engine (e.g., the size of cylinders) but simply use some reasonable ones we have designed along the way. In general, higher-level knowledge can be obtained at much lower price when we approach the problem at lower level first, and use the results of this in order to compose higher-order solutions.

Next section describes a general class of hierarchically decomposable functions. With this definition at hand, the subsequent section continues by proposing an extension of the recently proposed Bayesian optimization algorithm to the class of hierarchically decomposable problems which are an extension of additively decomposable problems discussed in our earlier work.

In the following text, the solutions are represented by binary strings of a fixed length, but the results can be easily extended to fixed-length strings over any finite base alphabet. Each string position represents a (binary) random variable and the set of promising solutions selected according to their fitness represents a multivariate random sample.

3 HIERARCHICALLY DECOMPOSABLE FUNCTIONS

Hierarchically decomposable functions (HDFs) were first presented by Goldberg (1998) who designed the so-called *Tobacco Road Function* which combined deception and multimodality up a number of levels (also in Goldberg (1997)). The class of hierarchically consistent functions was later presented by Watson, Hornby, and Pollack (1998). In HDFs, the *fitness contribution* of each building block (an intact sub-part of the solution quasi-separable from its context) is separated from its *interpretation* (meaning) when it is used as a building block for constructing the solutions on a higher level. The overall fitness is defined as the sum of fitness contributions of each building block.

In this paper, we will consider a more general class of hierarchically decomposable functions than the one introduced by Watson et al. (1998) that allows any order and interpretation of every building block on each level. Furthermore, the fitness contribution of each considered vector of interpretations will not be assigned uniformly but an arbitrary function for each block of interpretations can be used. Finally, the individual contributions will not be automatically multiplied by the length of the input vector.

Let us consider a hierarchical function on input vectors $X = (X_0, \ldots, X_{n-1})$ of n variables defined on $L \leq n$ levels. The value of variable X_i is denoted by x_i. On each level $i \in \{1, \ldots, L-1\}$, let us define m_i functions that contribute to the overall fitness. On input, each of these functions gets a vector of building-block interpretations (meanings) from a lower level. On each level i there will be m_i recursively computed interpretations.

The interpretations on a 0th level are simply the values of input variables, i.e. $v_{0,k} = x_k$ for all k. There are $m_0 = n$ such interpretations. On level $i > 0$, the jth contributory function $f_{i,j}$ is defined on a subset of interpretations from the lower level, with the indices from $S_{i,j} \subset \{0, \dots, m_{i-1}-1\}$. These interpretations will be joint together to form a higher-level interpretation by function $T_{i,j}$. We denote the vector of interpretations with the indices from $S_{i,j}$ by $V_{i,j}$, i.e. $V_{i,j} = \{v_{i-1,k} | k \in S_{i,j}\}$. The jth interpretation on the level i, denoted by $v_{i,j}$, is then given by the recursive function

$$v_{i,j} = \begin{cases} T_{i,j}(V_{i,j}) & \text{if } i > 0 \\ x_j & \text{otherwise} \end{cases}, \tag{1}$$

where $i \in \{0, \dots, L-1\}$, and $j \in \{0, \dots, m_j - 1\}$. A simple example of the recursive computation of the interpretations for a vector of $n = 9$ variables and $L = 3$ levels is shown in Figure 1.

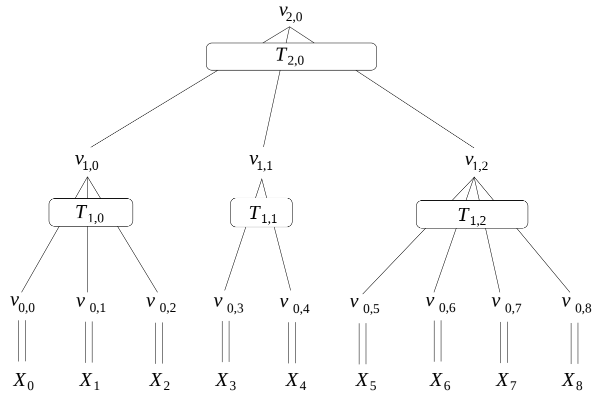


Figure 1: An example interpretation for $n = 9$ variables on $L = 3$ levels.

The total fitness is defined as the sum of functions defined on the subsets of interpretations that are interpreted together in order to get the interpretation on a higher level. For jth interpretation $T_{i,j}$, the corresponding function with the same inputs is denoted by $f_{i,j}$. The overall value of the fitness function is thus given by

$$f(X) = \sum_{i=1}^{L-1} \sum_{j=0}^{m_i - 1} f_{i,j}(V_{i,j}). \tag{2}$$

To solve HDFs, two issues must be addressed. The models must allow groups of variables to be merged into a single unit that will be further treated as a new ultimate variable. Moreover, niching becomes an important issue because in order to have enough material to combine, we need to preserve diversity; to combine solution from a certain level to form a solution of a higher order, we want to have a sufficient number of the parts of solutions we want to combine. Niching methods were frequently discussed in recent work (Goldberg, 1989; Oei, Goldberg, & Chang, 1991; Mahfoud, 1995; Mengshoel & Goldberg, 1999). Here we focus on the modeling part, i.e. on how the models to be used should look like and how these can be learned given the set of promising solutions.

Example:
The following function is defined by using bipolar fully deceptive functions of order 6. A bipolar function of order 6 is constructed from a deceptive function of order 3 which is defined on binary vectors $X = (X_0, X_1, X_2)$ of order 3 as

$$f_{3deceptive}(u) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases},$$

where u is the number of one's in the input vector (string) X. The bipolar function of order 6 is defined on binary vectors of length 6 as

$$f_{6bipolar}(u) = f_{3deceptive}(|3 - u|).$$

The function is defined on L levels. The input vector is of size $n = 6^l$. All interpretation functions will be defined in the same way and they will interpret a block of 6 bits according to the major occurrence of either bit (in case of tie, we interpret the block as 0), i.e.

$$v_{i,j} = \begin{cases} 0 & \text{if } u \leq 3 \\ 1 & \text{otherwise} \end{cases},$$

where u is the number of ones in the input vector of interpretations (each of which is a binary number), $i \in \{0, \dots, L-1\}$, and $j \in \{0, \dots, m_j - 1\}$. The contributory functions $f_{i,j}$ simply return the value of the bipolar function $f_{bipolar}$, i.e.

$$f_{i,j}(V_{i,j}) = f_{bipolar}(u),$$

where $V_{i,j}$ is the input vector of interpretations (each of which is again a binary value), and u is the number of ones in the input vector $V_{i,j}$. The function has two global optima in points $000\dots0$ and $111\dots1$. Similarly as functions additively composed of a bipolar function, it has a great number of deceptive local optima. Moreover, the optima on the higher levels aggravate the deception of functions on each level. To scale each function according to their "importance", measured for instance by the number of input bits that affect its value, the function contribution can be multiplied by a factor of 6^j, where j is the number of the level.

4 PROBABILISTIC MODEL-BUILDING GENETIC ALGORITHMS

Probabilistic model-building genetic algorithms (PMBGAs), also called the estimation of distribution algorithms (Mühlenbein & Paaß, 1996), replace genetic recombination of the genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989) by building an explicit model of promising solutions and using the constructed model to guide the further search. As models, probability distributions are used. For an overview of recent work on PMBGAs, see Pelikan, Goldberg, and Lobo (2000).

The Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1998) uses Bayesian networks to model promising solutions and subsequently guide the further search. In the BOA, the first population of strings is generated at random. From the current population, the better strings are selected. Any selection method can be used. A Bayesian network that fits the selected set of strings is constructed. Any metric as a measure of quality of networks and any search algorithm can be used to search over the networks in order to maximize/minimize the value of the used metric. Besides the set of good solutions, prior information about the problem can be used in order to enhance the estimation and subsequently improve convergence. New strings are generated according to the joint distribution encoded by the constructed network. The new strings are added into the old population, replacing some of the old ones.

As a model of the selected strings, a Bayesian network is used in the BOA. A Bayesian network is a directed acyclic graph with the nodes corresponding to the variables in the modeled data set (in our case, to the positions in the solution strings). Mathematically, a Bayesian network encodes a joint probability distribution given by

$$p(X) = \prod_{i=0}^{n-1} p(X_i|\Pi_{X_i}), \tag{3}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of all the variables in the problem, Π_{X_i} is the set of parents of X_i in the network (the set of nodes from which there exists an edge to X_i) and $p(X_i|\Pi_{X_i})$ is the conditional probability of X_i conditioned on the variables Π_{X_i}. A directed edge relates the variables so that in the encoded distribution, the variable corresponding to the terminal node will be conditioned on the variable corresponding to the initial node. More incoming edges into a node result in a conditional probability of the corresponding variable with conjunctional condition containing all its parents.

To construct the network given the set of selected solutions, various methods can be used. All methods have two basic components: a scoring metric which discriminates the networks according to their quality and the search algorithm which searches over the networks to find the one with the best scoring metric value. The BOA can use any scoring metric and search algorithm. In our recent experiments, we have used the Bayesian-Dirichlet metric (Heckerman, Geiger, & Chickering, 1994). The complexity of the considered models was bounded by the maximum number of incoming edges into any node denoted by k. To search the space of networks, a simple greedy algorithm was used due to its efficiency. For further details, see Pelikan, Goldberg, and Cantú-Paz (1999).

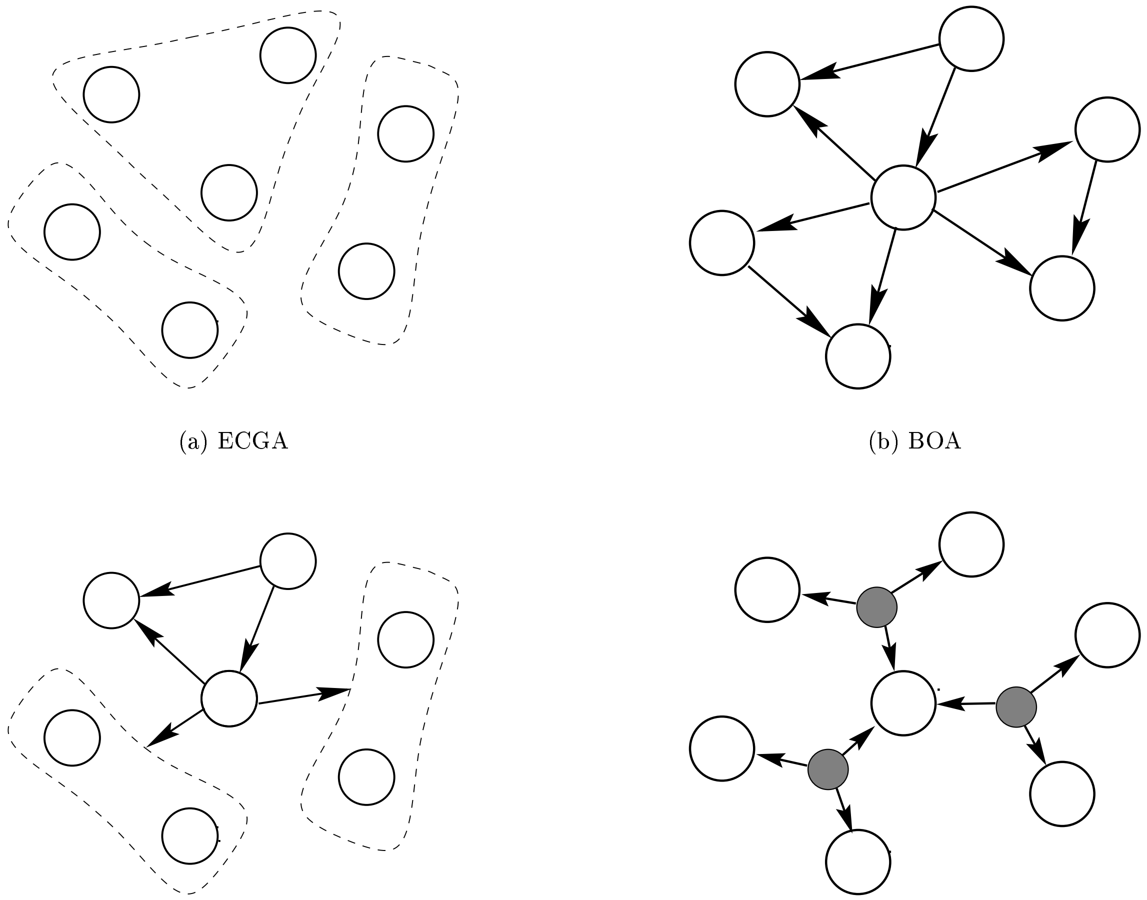
5 HIERARCHICAL MODEL BUILDING

To hierarchically solve a problem, we need to incrementally find important low-order partial solutions and combine these to create the solutions of higher order. Starting with single bits (symbols of base alphabet), once we get top high-quality solutions of some order we simply treat these solutions as the building blocks to be used to construct solutions of higher order. In this fashion, the order of partial solutions we get gradually grows over time.

5.1 HIERARCHICAL MODELS

In order to adjust modeling to hierarchical problems, we will use models that, among estimating the joint distribution between single variables, also allow multiple variables to be *merged* together and form a *new variable*. This variable will be further treated as a single unit. In this fashion the solutions of higher order can be formed by using groups (clusters) of variables as basic building blocks.

The idea of clustering the input variables and treating each cluster as an intact building block comes from learning used in the extended compact genetic algorithm (ECGA) (Harik, 1999). For each group of variables only instances that are in the modeled data set will be considered like in learning Bayesian networks



(a) ECGA

(b) BOA

(c) hBOA

(d) Hidden variables

Figure 2: Models used in a) ECGA, b) BOA, c) hierarchical BOA (Huffman networks), and d) an alternative model based on using hidden variables.

with local structure (Friedman & Goldszmidt, 1999). The clusters (groups) of variables are related as in classical directed-acyclic-graph (DAG) Bayesian networks used in the original BOA algorithm. This class of hybrid models was first introduced by Davies and Moore (1999) who called these models *Huffman networks*.

Let us, for example, at certain point in time, have three positions with only two values in the entire population: 000 and 111. Then, instead of working with each of these positions separately, these can be merged into a single binary variable with two new values $0'$ and $1'$, where $0'$ corresponds to 000 and $1'$ corresponds to 111. In this fashion, both the model complexity as well as the model expressiveness improve. Moreover, by reducing the number of variables, the search for good networks becomes more efficient and accurate. Each group of merged variables represents an intact part of the solutions from lower-level that is to be treated as a single variable on a higher level.

An example model with a few groups of variables is shown in Figure 2c. For comparison, similar examples of models in the BOA and ECGA are shown in parts a) and b) of the same figure. The use of Huffman networks does not require sacrificing modeling generality as in the ECGA. All relationships expressed by DAG models can be covered. On the other side, the overly complex DAG models used in the original BOA can be significantly simplified by “crossing over” the two approaches.

Similar reduction of total model complexity can be achieved by using hidden variables often used in Bayesian networks. In fact, using hidden variables is an alternative and more general approach to the problem of hierarchical model building. We believe that using these models would further improve model-building for problems of a very complex structure. A similar model to the one shown in Figure 2c, based on using hidden variables, is shown in Figure 2d.

5.2 SCORING METRIC FOR HUFFMAN NETWORKS

To learn a model of solutions on a certain level, we will use a combination of the learning methods used in the original Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1999), the extended compact genetic algorithm (ECGA) (Harik, 1999), and Bayesian networks with local structure (Friedman & Goldszmidt, 1999). To discriminate the networks, a minimum description length (MDL) metric will be used. The BDe metric with additional term preferring simpler networks (Friedman & Goldszmidt, 1999) can be used, too. However, simpler models must be preferred to more complex ones, since the clusters tend to grow indefinitely and the boundary on the complexity of models can not be directly applied without weakening the modeling capabilities on hierarchical problems.

To store data according to a particular model, we need to store (1) the definition of clusters (groups) of variables in the model, (2) the probabilistic relationships between the groups of variables (edges between the groups in the model), and (3) the data set (the set of selected solutions) compressed according to the model. Each variable (bit position) is in exactly one of the clusters. The description of data will contain the following fields: (1) the number of clusters in the model, (2) an array of cluster definitions, and (3) the population compressed according to the model.

In further text we will use the following notation: n denotes the number of variables; N denotes the number of instances in the modeled data set; m denotes the number of clusters (groups of variables); $G = (G_0, \dots, G_{m-1})$ denotes the set of clusters G_i; $|G_i|$ denotes the number of variables in G_i; $||G_i||$ denotes the number of instances of variables G_i; Π_i denotes the set of parent groups of G_i; $|\Pi_i|$ denotes the number of parent groups in Π_i; and $||\Pi_i||$ denotes the number of instances of the set of groups Π.

There can be at most n groups of variables, i.e. $m \leq n$, and therefore in order to store the number m of groups, at most $\log_2 n$ bits can be used.

The definition of each group contains (1) the size of the group, (2) the indices of the variables contained in the group, (3) the set of instances of this group, (4) the set of this group's parent identifiers, and (5) the set of conditional probabilities of the instances in this group given all the instances of its parent groups. There can be at most n variables in each group, and therefore the size of each group can be stored by using $\log_2 n$ bits. This boundary could be further reduced by analyzing the entire description at once. There are $\binom{n}{|G_i|}$ possibilities to choose variables to form G_i. Thus, to identify the set of variables in G_i, we need to store only the order of this subset in some ordering of all possible subsets of this size, i.e. we need at most $\log_2 \binom{n}{|G_i|}$ bits. Assuming that we use binary variables, the set of instances of G_i can be stored by using $\log_2 2^{|G_i|} = |G_i|$ bits for the number of instances and $|G_i|.||G_i||$ bits for the specification of all bits in these instances.

Each group can have at most $n - 1$ parents in the network. Thus, the number of parents can be stored by using $\log_2(n-1)$ bits. The number of bits needed to store the components of Π_i is $\log_2 \binom{m}{|\Pi_i|}$.

To store conditional probabilities for G_i, we will store a frequency of each combination of instances of the variables in G_i and its parents. There are at most

$$||G_i||.||\Pi_i||$$

possible instances. However, this number might be further reduced by using local structures (Friedman & Goldszmidt, 1999) or considering only instances that really appear in the modeled data set. Each frequency can be stored in $0.5 \log_2 N$ bits with a sufficient degree of accuracy (Friedman & Yakhini, 1996). Thus, to store the conditionals corresponding to G_i, we need at most

$$\frac{|G_i| \log_2 N}{2} \prod_{G_j \in \Pi_i} (||G_j|| - 1)$$

bits, since the last frequency can be computed from the remaining ones.

To store the data compressed according to the above model, we need at most

$$-N \sum_{i=0}^{|G|-1} \sum_{g_i, \pi_i} p(g_i, \pi_i) \log p(g_i|\pi_i)$$

bits (Friedman & Goldszmidt, 1999), where the inner sum runs over all instances g_i and π_i of variables in G_i and Π_i respectively, $p(g_i, \pi_i)$ is the probability of the instance with the variables in G_i and Π_i set to g_i and π_i respectively, and $p(g_i|\pi_i)$ is the conditional probability of the variables in G_i set to g_i given that the variables in Π_i are set to π_i.

The overall description length is then computed as the sum of all terms computed above. The lower the metric, the better the model.

5.3 BUILDING A HUFFMAN NETWORK

A method for building Huffman networks for compression of large data sets was presented in Davies and Moore (1999). This method proceeds similarly as other search methods commonly used for learning Bayesian networks by incrementally performing elementary graph operations on the model to improve the value of the scoring metric. This algorithm is often used for its efficiency. A general scheme of the greedy search method used in the original BOA as well as in Davies and Moore (1999) follows:

(1) Initialize the network (to an empty, random, or the best network from the last generation).

(2) Pick an elementary graph operation that improves the score of the current network the most.

(3) If there is such operation, perform it, and go to step 2.

(4) If no operation improves the score, finish.

In addition to usually used operations as edge addition, edge removal, and edge reversal, we can use a new operation that can either (1) join two of the groups of variables to form a single cluster or (2) move one variable from one cluster to another one (and deleting clusters that has become empty, if any). In (Davies & Moore, 1999), the second operation was used. In both cases, the conflicts appearing with existence of cycles must be resolved. When joining two groups, the edges can be either conservatively rearranged so that only edges that coincided with both of the groups will be considered or that all edges to and from either of the groups will be considered, if possible.

6 FUTURE WORK

We are currently implementing the extended hierarchical BOA (hBOA) in order to test the algorithm on various hierarchically decomposable problems. In order to analyze the performance of our algorithm and compare it to alternative methods, a set of test problems will be designed. The results of our recent research at the Illinois Genetic Algorithms Laboratory, focusing on various aspects of problem difficulty, will be used in order to design a rigorous test-suite for methods that approach the problem in a hierarchical fashion.

Alternative solutions to the problem of hierarchical modeling will be compared on the designed test-suite. One of the alternatives, based on using hidden variables, was outlined above. There are other methods that may be used and the question of suitability of each approach still remains open.

The paper did not discuss niching, which becomes a very important issue when solving hierarchical problems. Although it may not be necessary to use niching for solving additively decomposable problems, when solving hierarchical problems it becomes a necessity since many alternative low-order solutions should be preserved before we can find the best way of juxtaposing these. In other words, the importance of the notion of minimal sequential non-inferior BB of a solution dominates the one of the minimal sequential superior BB of a solution (Goldberg, 1997). Therefore, one of the most important issues of future research on this topic should include the use of niching in the methods based on probabilistic modeling, as the BOA algorithm. More advanced niching techniques can be designed by using the constructed model as a hint on the structure of the problem at hand.

7 SUMMARY AND CONCLUSIONS

Recently, Watson et al. (1998) suggested that the simple genetic algorithm can solve some hierarchically decomposable problems quite efficiently. On the other side, anomalous behavior of the simple GA on problems with rewards for various combinations of building block was observed (Forrest & Mitchell, 1993). We believe that the algorithms that use Huffman networks to model promising solutions will offer an efficient and very robust method to solve the class of hierarchical problems. Even though the hierarchical BOA is aimed to solve hierarchically decomposable problems, we expect that its overall performance on other problems will also improve. In fact, by simplifying the used class of models without sacrificing their generality, modeling capabilities of the BOA should improve what should result in that the BOA will solve a more general class of problems efficiently and reliably.

The paper discussed three major issues. It provided the reasons for approaching problems in a hierarchical fashion. The class of hierarchically decomposable problems which extends additively decomposable problems in order to test for hierarchical capabilities of optimization algorithms was defined. Possible extensions of the original Bayesian optimization algorithm were outlined, and the direction of future research in the discussed area was drawn.

Acknowledgments

The authors would like to thank Erick Cantú-Paz for many useful discussions.

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-97-1-0050. Research funding for this work was also provided by the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U. S. Government.

References

Davies, S., & Moore, A. (1999). Using bayesian networks for lossless compression in data mining. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-99)* (pp. 387–391). San Diego, CA: ACM Press.

Forrest, S., & Mitchell, M. (1993). What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning*, *13*(2/3), 285–319.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (1 ed.). (pp. 421–459). Cambridge, MA: MIT Press.

Friedman, N., & Yakhini, Z. (1996). On the sample complexity of learning Bayesian networks. In Horvitz, E., & Jensen, F. (Eds.), *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)* (pp. 274–282). San Francisco: Morgan Kaufmann Publishers.

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

Goldberg, D. E. (1997, November). *The design of innovation: Lessons from genetic algorithms*. Unpublished manuscript.

Goldberg, D. E. (1998, June 15). Four keys to understanding building-block difficulty. Presented in Projet FRACTALES Seminar at I.N.R.I.A. Rocquencourt, Le Chesnay, Cedex.

Goldberg, D. E. (2000). The design of innovation: Lessons from genetic algorithms, lessons for the real world. *Technological Forecasting and Social Change*. In press.

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL Report No. 99010). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Heckerman, D., Geiger, D., & Chickering, M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data* (Technical Report MSR-TR-94-09). Redmond, WA: Microsoft Research.

Holland, J. (1995). *Hidden order: How adaptation builds complexity,*. Addison Wesley.

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

Koza, J. (1994). *Genetic programming II: Automatic discovery of reusable programs*. Cambridge, Massachusetts: Massachusetts Institute of Technology.

Koza, J. R., Bennett III, F. H., Andre, D., & Keane, M. A. (1999). *Genetic programming III: Darwinian invention and problem solving*. San Fransisco, CA: Morgan Kaufmann Publishers.

Mahfoud, S. W. (1995, May). *Niching methods for genetic algorithms*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. Also IlliGAL Report No. 95001.

Mengshoel, O. J., & Goldberg, D. E. (1999). Probabilistic crowding: Deterministic crowding with probabilisitic replacement. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Volume I (pp. 409–416). Orlando, FL: Morgan Kaufmann Publishers, San Fransisco, CA.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. In Eiben, A., Bäck, T., Shoenauer, M., & Schwefel, H. (Eds.), *Parallel Problem Solving from Nature - PPSN IV* (pp. 178–187). Berlin: Springer Verlag.

Oei, C. K., Goldberg, D. E., & Chang, S.-J. (1991). *Tournament selection, niching, and the preservation of diversity* (IlliGAL Report No. 91011). Urbana, IL: University of Illinois at Urbana-Champaign.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1998). *Linkage problem, distribution estimation, and Bayesian networks* (IlliGAL Report No. 98013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Volume I (pp. 525–532). Orlando, FL: Morgan Kaufmann Publishers, San Fransisco, CA.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2000). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*. In press.

Watson, R. A., Hornby, G. S., & Pollack, J. B. (1998). Modeling building-block interdependency. In *Parallel Problem Solving from Nature, PPSN V* (pp. 97–106). Springer Verlag.