Conquering Hierarchical Difficulty by Explicit Chunking: Substructural Chromosome Compression

Tian-Li Yu
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
104 S. Mathews Ave., Urbana, IL 61801, USA
tianliyu@illigal.ge.uiuc.edu

David E. Goldberg
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
104 S. Mathews Ave., Urbana, IL 61801, USA
deg@uiuc.edu

ABSTRACT

This paper proposes a chromosome compression scheme which represents subsolutions by the most expressive schemata. The proposed chromosome compression scheme is combined with the dependency structure matrix genetic algorithm and the restricted tournament replacement to create a scalable optimization tool which optimizes problems via hierarchical decomposition. One important feature of the proposed method is that at the end of the run, the problem structure obtained from the proposed method is comprehensible to human researchers and is reusable for larger-scale problems. The empirical result shows that the proposed method scales sub-quadratically with the problem size on hierarchical problems and is able to capture the problem structures accurately.

Categories & Subject Descriptors

G.1.6 [Mathematics of Computing]: Global Optimization–Analyze.

General Terms

Algorithms, Design.

Keywords

Genetic Algorithms, Linkage Learning, Model Building, Hierarchy, Chunking, Chromosome Compression.

1. INTRODUCTION

In a variety of different areas, researchers are often confronted with complex systems. Complex systems can be either natural, *e.g.*, an atom with complex electric-field interactions between protons and electrons, or artificial, *e.g.*, an organization with complex communications between employees. To solve difficult, large-scale problems in complex systems effectively and efficiently, proper problem decomposition is the key.

The concept of decomposition can be backtracked at least as far as the 1637 publication of Descartes's *A Discourse on Method* [1] and is often referred as *linkage learning* in the genetic algorithm (GA) field. Since Holland [8] addressed the importance of linkage learning for GA success, many such methods have been developed [3, 12].

Hierarchical problems come from hierarchical complex system–a complex system composed of subsystems, and each of which is hierarchical by itself [17]. Many complex systems around us are hierarchical. For example, materials are composed of molecules, molecules are composed of atoms, atoms are composed of electrons, protons, neurons, and so forth. A university is composed of colleges, colleges are composed of departments, and so forth.

Inspired by the fact that many complex systems are hierarchical, Pelikan and Goldberg [14] proposed hBOA, one of few genetic and evolutionary algorithms [22] that is known to optimize problems with random linkage by hierarchical decomposition. hBOA has shown the ability to decompose hierarchical problems which are not fully decomposable in one single level [13, 14].

hBOA has shown an excellent ability of optimizing the given problems; however, the decomposition information is implicitly stored in a Bayesian network, which is usually incomprehensible for human researchers. In many real-world applications, the knowledge of the problem structure is as valuable as a high-quality solution to the problem. For example, in the field of feature selection, one of the most important issues is to discover the dependencies and redundancies among many features.

The objective of this paper is to develop an explicit hierarchical decomposition scheme for GAs. The proposed method is based on the dependency structure matrix genetic algorithm (DSMGA) [24], which utilizes dependency structure matrix clustering techniques for linkage learning. The proposed method optimizes the problem via hierarchical decomposition, and when it finishes, the problem structure is stored in an explicit manner that is transparent to human researchers.

The paper is organized into five main parts. The first part revisits hierarchical difficulty, hierarchical problems, and the keys to conquer them. The second part gives a brief introduction to DSMGA. The third part describes the proposed explicit chunking: substructural chromosome compression, which reduces the search space on-the-fly. The fourth part

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

describes the experiments and demonstrates the results. Finally, the paper is concluded by discussions and some future work.

2. HIERARCHICAL DIFFICULTY AND HIERARCHICAL PROBLEMS

Hierarchical problems come from hierarchical complex systems. They are not fully decomposable in one single level. In hierarchical problems, the signal of the interactions of upper levels is too small to detect unless all lower levels are solved. This section describes the keys to conquer hierarchical difficulty and details three typical hierarchical problems—the hierarchical IFF, the hierarchical XOR, and the hierarchical trap, which are later used as test functions in the paper.

2.1 Keys to conquer hierarchical difficulty

Pelikan and Goldberg [14] recognized three keys to conquer hierarchical difficulty.

Proper decomposition. At each level, the problem needs to be properly decomposed so that the GA can mix subsolutions effectively.

Niching. The GA needs to be able to preserve promising subsolutions to the next level because no correct decision can be made until the GA advances to the upper levels.

Chunking. To prevent the complexity of the hierarchical problem from growing exponentially, the GA needs to represent one block in a lower level as one variable in an upper level.

In hBOA [14], the chunking is implicit and it is achieved by recognizing local substructures in the Bayesian network. In this paper, we propose an explicit chunking scheme which is more comprehensible. It will be detailed in Section 4.

2.2 The design of hierarchical problems

Several hierarchical problems were designed to test the methodology. The design is guided by the three keys described in the previous section: (1) proper decomposition, (2) niching, and (3) chunking. Three test functions used in this paper are the hierarchical IFF [20], the hierarchical XOR [21], and the hierarchical trap [14]. They are detailed as follows.

Hierarchical if and only if (hIFF)

The hierarchical IFF problem is defined on a binary string $x \in \{0,1\}^{2^\lambda}$, where λ is the number of hierarchical levels. First, define a boolean function $h(x)$ to determine if x is *valid* or not. Let $L = x_1 x_2 \cdots x_{2^{\lambda-1}}$, and $R = x_{2^{\lambda-1}+1} x_{2^{\lambda-1}+2} \cdots x_{2^\lambda}$.

$$h_{iff}(x) = \begin{cases} 1 & \text{, if } \lambda = 0 \\ 1 & \text{, if } h_{iff}(L) = 1, h_{iff}(R) = 1, \text{ and } L = R \\ 0 & \text{, otherwise.} \end{cases} \tag{1}$$

level	global optimum	# of optima
0	1	2
1	10	2
2	1001	4
3	10010110	16
4	1001011001101001	256
5	10010110011010010110100110010110	65536

Table 1: One of the global optima and the number of optima of hXOR for level 0 to 5. There are half 0's and 1's in the global optima, and the number of optima grows exponentially with the problem size.

Based on $h_{iff}(x)$, for $\lambda > 0$, the fitness of hIFF is defined recursively as:

$$H_{iff}(x) = H_{iff}(L) + H_{iff}(R) + \begin{cases} \text{length}(x) & \text{, if } h_{iff}(x) = 1 \\ 0 & \text{, otherwise.} \end{cases} \tag{2}$$

The base case is that for $\lambda = 0$, $H_{iff}(x) = 1$. hIFF has two global optima: a string with all 0's and with all 1's. hIFF has $2^{l/2}$ local optima at the lowest level for the problem size of l.

Hierarchical exclusive or (hXOR)

The global optima of hIFF are all 1's and all 0's, which might not be too hard to find for an algorithm biased (or drifted) to some particular allele value. To prevent the search algorithm from exploiting the problem property, the hierarchical XOR problem is designed.

The definition of hXOR is very similar to that of hIFF, but with an alternation in the validation function by the complement check.

$$h_{xor}(x) = \begin{cases} 1 & \text{, if } \lambda = 0 \\ 1 & \text{, if } h_{xor}(L) = 1, h_{xor}(R) = 1, \text{ and } L = \overline{R} \\ 0 & \text{, otherwise.} \end{cases} \tag{3}$$

Here $\overline{R}$ is the bitwise negation of R. Similar, for $\lambda > 0$, the fitness of hXOR is defined recursively as:

$$H_{xor}(x) = H_{xor}(L) + H_{xor}(R) + \begin{cases} \text{length}(x) & \text{, if } h_{xor}(x) = 1 \\ 0 & \text{, otherwise.} \end{cases} \tag{4}$$

The base case is that for $\lambda = 0$, $H_{xor}(x) = 1$. Similar to hIFF, hXOR also has two global optima and $2^{l/2}$ local optima at the lowest level for the problem size of l. However, there are exactly half of 1's and half of 0's in the global optima. Table 2.2 lists one of the global optima for level 0 to 5.

Hierarchical trap (hTrap)

Pelikan and Goldberg [14] proposed a scalable hierarchical problem called the *hierarchical trap* (hTrap). hTrap is composed of three major components (Figure 1):

1. **Structure.** The hierarchical trap structure is a balanced k-ary tree.
2. **Mapping function.** The mapping function maps genes from lower levels to upper levels. A block of

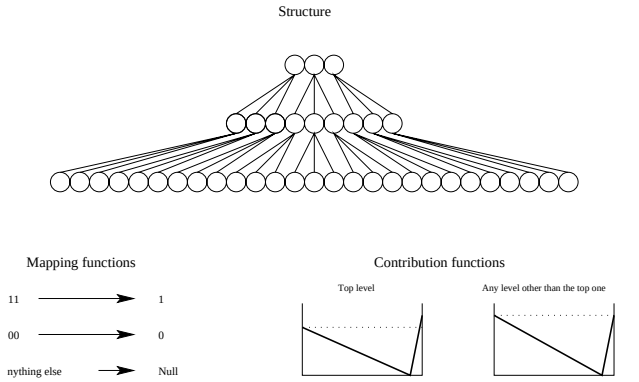


Figure 1: The structure, mapping functions, and contribution functions of the hierarchical trap. The function has three levels and $k = 3$. Note that 111 and 000 are equally good except for the top level.

all 0's and 1's is mapped to 0 and 1 respectively, and everything else is mapped to '-'.

3. **Contribution functions.** The contribution function is based on trap functions of order k. There are two parameters in the trap functions: f_{high} and f_{low}, which control the degree of deception. The trap function is defined as:

$$trap_k(u) = \begin{cases} f_{high} & , \text{ if } u = k \\ f_{low} \times \frac{k-1-u}{k-1} & , \text{ otherwise.} \end{cases} \quad (5)$$

Here u is the unitary of the input string. If any position of the string is a '-' symbol, the contribution of the string to the fitness is zero.

The hierarchical trap functions used in this paper have k set to 3. Both f_{high} and f_{low} are set to 1 for all but the highest level. In the highest level, $f_{high} = 1$ and $f_{low} = 0.9$. So the decision between competing BBs cannot be made correctly until the GA reaches the highest level.

3. INTRODUCTION TO DSMGA

This section gives a brief introduction to the dependency structure matrix genetic algorithm (DSMGA), which is later used in the paper as a basis of the proposed method. DSMGA utilizes the dependency structure matrix (DSM) clustering techniques to extract the information of building blocks (BBs) and uses the information to accomplish BB-wise crossover. This section first introduces the concept of DSM and the DSM clustering problem. It then describes the metric to cluster DSMs. Finally, the framework of DSMGA is detailed.

3.1 Dependency structure matrix (DSM) and the clustering problem

A dependency structure matrix is essentially an adjacency matrix representation of a graph where each entry d_{ij} represents the dependency between node i and node j [18, 23]. Entries d_{ij} can be real numbers or integers. The larger the d_{ij} is, the higher the interaction is between node i and node j. If we focus on the 0-1 domain, then $d_{ij} = 0$ means that node i and node j do not interact, and $d_{ij} = 1$ means that

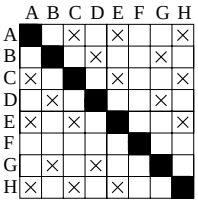


Figure 2: A DSM where clustering is not so obvious at the first glance. "x" means that dependency exists; the blank means no dependency.

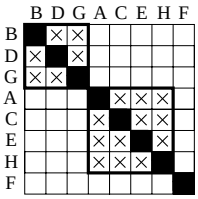


Figure 3: The same DSM but after reordered. The DSM can be cleanly clustered as (BDG, ACEH, F).

node i and node j interact with each other. The diagonal entries (d_{ii}) have no significance and are usually set to zero or blacked-out. Figures 2 and 3 show examples of DSMs where an "x" symbol stands for 1, and a blank square stands for 0. From Figure 2, for example, we know that A and C are dependent; A and F are independent. For elaborate exposition of DSM, please see MIT DSM web site: `http://www.dsmweb.org/`.

The goal of DSM clustering is to find subsets of DSM elements (*i.e.*, clusters) so that nodes within a cluster are maximally interacting, and clusters are minimally interacting. For example, the DSM in Figure 2 can be clustered as {BDG, ACEH, F} (Figure 2). In a typical DSM clustering problem, overlapping clusters (clusters that share same nodes) are permissable.

DSM clustering is a sophisticated task which requires expertise. For example, not too many people know how to cluster the DSM in Figure 2. However, after nodes are reordered, it is easily seen that the DSM can be cleanly clustered into three parts (Figure 3). In addition, a criterion that evaluates clustering arrangement is essential for automated clustering algorithms.

3.2 MDL-based DSM clustering metric

Previous DSM clustering algorithms can be found elsewhere [2, 16]. Their results showed that the objective function used was short of accurately predicting "good" clustering because of the oversimplified objective function.

Here we use a DSM clustering metric based on the minimal description length principle (MDL) [15]. Suppose that we have a model which describes a given data set, $DSM = [d_{ij}]$. Here, the model means a description that specifies which node belongs to which cluster. Usually, the model does not completely describe the given data; otherwise, the model would be too complex to use. Therefore, the description length that the model needs to describe the given data consists of two parts: the model description and the mismatched data description.

The minimum description length principle (MDL) [15] sat-

isfies our needs for dealing with the above trade-off. The MDL can be interpreted as follows: among all possible models, choose the model that uses the minimal length for describing a given data set (that is, model description length plus mismatched data description length). There are two key points that should be noted when MDL is used: (1) the encoding should be uniquely decodable, and (2) the length of encoding should reflect the complexity. For example, the encoding of a complicated model should be longer than that of a simple model. Next, we define the MDL clustering metric in detail.

Model Encoding. The way we encode the model is straightforward. The description of each cluster starts with a number which is sequentially assigned to each cluster, and then this is followed by a sequence of nodes in the cluster. Figure 4 shows a DSM clustering arrangement and the corresponding model description. It is easily seen that the length of this model description is as follows:

$$\Sigma_{i=1}^{n_c} \left(\log_2 n_n + cl_i \cdot \log_2 n_n \right), \quad (6)$$

where n_c is the number of clusters in the model, n_n is the number of nodes, cl_i is the number of nodes in the i-th cluster. In the example of Figure 4, $n_c = 2$ clusters, $n_n = 8$ nodes, $cl_1 = 4$, and $cl_2 = 3$. The table in the figure reads as follows: "cluster 1 has 4 nodes: B, D, G, and A; cluster 2 has 3 nodes: C, E, and H."

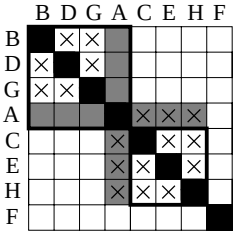
If n_n and n_c are known, it is not difficult to see that the above model description is uniquely decodable. When n_n is given, and by assuming $n_c \leq n_n$, then $\log n_n$ bits are needed to describe n_c. It is a constant for all models, and therefore they are omitted without loss of accuracy.

Mismatched Data Description. Based on the model, we first construct another DSM (call it $DSM' = [d'_{ij}]$), where each entry d'_{ij} is 1 if and only if some cluster contains both node i and node j simultaneously.

Then, we compare DSM' with the given DSM. For every mismatched entry, where $d'_{ij} \neq d_{ij}$, we need a description to indicate where the mismatch occurred (i and j) and one additional bit to indicate whether the mismatch is zero-to-one or one-to-zero. Define a mismatch set $S = \{(i,j) | d'_{ij} \neq d_{ij}\}$. The mismatched data description length is given by:

$$\Sigma_{(i,j)\in S} \left(\log n_n + \log n_n + 1 \right). \quad (7)$$

The first $\log n_n$ in the bracket indicates i, the second one indicates j, and the additional one bit indicates the type of mismatch.



Length	$\log_2 n_n$	$4 \log_2 n_n$	$\log_2 n_n$	$3 \log_2 n_n$
Description	4	B, D, G, A	3	C, E, H

Figure 4: The above is a clustering arrangement of a DSM. The below is the associated model description. The shadowed cells represent mismatches. There are 12 mismatches in this model.

The MDL clustering metric is given by the summation of the model description length and the mismatched data description. With some arithmetic manipulations, the metric can be expressed as follows:

$$f_{DSM}(M) = \log n_n \Sigma_{i=1}^{n_c} (cl_i + 1) + |S|(2 \log n_n + 1), \quad (8)$$

where n_c is the number of clusters, n_n is the number of nodes in the DSM, cl_i is the size of the i-th cluster, and S is a mismatch set.

With the above metric, the DSM clustering problem is converted into an optimization problem: Given a DSM, the objective is to find a DSM clustering arrangement (model, M) to minimize the above metric (f_{DSM}).

3.3 The Framework of DSMGA

There are two optimization problems in DSMGA: one is the given optimization problem, and the other is the DSM clustering problem. The key idea of DSMGA is to use an auxiliary search algorithm to extract the BB information by using the DSM clustering technique, and then solve the given optimization problem more effectively and efficiently by using the extracted BB information. The main task in DSMGA can be decomposed into three subtasks: (1) DSM construction from the current population, (2) DSM clustering, and (3) BB-wise crossover.

After the BB information is obtained from the auxiliary search algorithm, BB-wise crossover can be achieved. The BB-wise crossover is like an ordinary allele-wise crossover, but instead of mixing genes, the BB-wise crossover mixes BBs and will not disrupt BBs. The order of BBs is not significant because BBs have no (or little) interaction with each other by definition. For more details about BB-wise crossover, refer to Harik's work [6].

In this paper, the pair-wise dependency (the entries of the DSM) is calculated by mutual information. A steepest descent algorithm is adopted to optimize the DSM clustering problem. Based on the MDL metric, it add/remove one node to/from one cluster at each iteration. The steepest descent algorithm stops when no further improvement is possible.

4. SUBSTRUCTURAL CHROMOSOME COMPRESSION

This section proposes an explicit chunking scheme, which we call the *substructural chromosome compression*. The substructural chromosome compression scheme expresses a building block by one single variable when the alleles in the building block nearly converge to only few most expressive schemata.

The idea of using standard text compression techniques in GAs has been explored elsewhere [19], but the method to date only works with explicit prior knowledge of model boundaries on an assumption that makes the technique is virtually unusable in most applications. Nevertheless, the idea of compression is sound and the goal of this portion

of the work is to realize a practical and broadly applicable technique in a larger solver.

The key idea of substructural chromosome compression is to represent a nearly-converged BB by only π of the most expressive schemata. Gambler's ruin population-sizing model [7] estimates the population size of $O(2^k m)$ for a binary problem with m BBs of order k. In a typical hierarchical problem, the second level has (m/k) BBs with order k. Since we expect the complexity of an upper level to be less than that of an lower level, the following condition should be maintained:

$$\pi^k \left(\frac{m}{k}\right) \leq 2^k m, \tag{9}$$

or simply

$$\pi \leq 2\sqrt[k]{k}. \tag{10}$$

It can be seen that π is decreasing with k increasing, and with $k \geq 3$, π should be less than 3. Therefore, in this paper, we only focus on the case where $\pi = 2$ although the method is not limited to this special case. In other words, when a BB is nearly converged, we compress it into one single bit, where 1 maps to the most expressive schema and 0 maps to the second most expressive schema. The information of the other $(2^k - 2)$ less expressive schemata are discarded.

One thing very critical for the substructural chromosome compression scheme is that every BB of lower levels needs to be compressed **before** the GA can advance to a higher level. Otherwise, the BB size would be 2^{k^2} in the next level and the GA fails because of insufficient BB supply. To avoid that situation from happening, two things need to be taken care of: (1) compress a BB as soon as the decision can be made with high confidence, and (2) make sure that linkage of an upper level is not expressed before BBs of the lower level are compressed. The methods are described in detail in following subsections.

4.1 Compression criterion

As mentioned before, we compress a BB into one single bit when it is nearly converged. Neglecting the probability that the most expressive schema in the current population could be wrong (*i.e.*, not really the most expressive schema), we need to make good decision between the second most expressive schema (H_2) and the third most expressive schema (H_3). The decision-making derivations below is very similar to that of the decision-making population-sizing model [4].

Assume that the proportion of H_2 and H_3 in the current population are p_2 and p_3 respectively, and by definition, $p_2 > p_3$. If the GA reaches the steady state (*i.e.*, p_2 and p_3 are the proportion when the GA is converged with the nicher), the proportion is of binomial distribution with n samples, where n is the population size. The probability of making an error may be calculated as:

$$p = \mathbb{C}_{\mathbb{N}}\left(\frac{p_2 - p_3}{\sqrt{\sigma_{H_2}^2 + \sigma_{H_3}^2}}\right), \tag{11}$$

where $\mathbb{C}_{\mathbb{N}}$ is the cumulative Gaussian function with zero mean and unit standard deviation, and $\sigma_{H_2}^2$ and $\sigma_{H_3}^2$ are variances of p_2 and p_3, respectively. Assuming steady-state, the variance can be approximated as $\sigma_{H_2}^2 \simeq \frac{p_2(1-p_2)}{n}$, and similarly, $\sigma_{H_3}^2 \simeq \frac{p_3(1-p_3)}{n}$.

We would like the error of the decision-making to be very small since once a BB is compressed, there is no turning back. In this paper, we compress a BB when the signal difference is greater than five standard deviations:

$$p_2 - p_3 \geq 5\sqrt{\frac{p_2(1-p_2) + p_3(1-p_3)}{n}}. \tag{12}$$

After the linkage information is retrieved from the DSM clustering, the algorithm checks if Equation 12 is satisfied for every BB. If so, the BB is then compressed as one single bit by converting the most expressive schema to 1, the second most expressive schema to 0, and any other schemata randomly to 1 or 0.

4.2 Linkage detection thresholds

Now we need to compute a proper threshold for linkage detection so that linkage of an upper level is not expressed before the BBs of the lower level are compressed.

In this paper, the pair-wise linkage detection is done via calculating the mutual information between two genes. Mutual information is defined as the Kullback-Leibler distance [11] between the joint distribution and the product distribution:

$$\begin{aligned} I(X;Y) &= D(p(x,y)||p(x)p(y)) \\ &= \Sigma_x \Sigma_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)}, \end{aligned} \tag{13}$$

where D is the the Kullback-Leibler distance, X and Y are two random variables, and x and y are outcomes of the random variables. Note that if X and Y are independent, $p(x,y) = p(x)p(y)$, and hence $I(X;Y) = 0$.

Kleiter [10] approximated the mean and variance of the mutual information by Monte-Carlo studies. Suppose that X and Y are binary random variables (*i.e.*, the possible outcomes are 0 and 1). If we sample (x,y) pairs n times from (X,Y), and let I be the sampled mutual information calculated by Equation 13 from these n samples, the variance of the mutual information can be approximated as:

$$Var[I] \simeq \frac{2E[I]}{n}, \tag{14}$$

where $E[I]$ is the expected value of I.

If X and Y are independent, the expected value of I can be approximated as:

$$E[I] \simeq \frac{1}{2n}. \tag{15}$$

It can been seen that both $E[I]$ and $Var[I]$ are close to zero if the sample size n is large. In addition, Hutter and Zaffalon [9] showed that the distribution of the sampled mutual information has a similar shape of the Beta distribution.

Provided all the above tools, we decide that if two genes X and Y are independent, the sampled mutual information should be less than the mean given by Equation 15 plus ten standard deviations (since the beta distribution has a long tail, and we would like the threshold to be noise-resistant), and that gives us the first threshold:

$$\theta_1 = \frac{21}{2n}, \tag{16}$$

where n is the sample size, or, the size of the population used for building the linkage model. If the sampled mutual information of two genes is less than θ_1, we consider that these two genes are independent and put a "zero" entry in the corresponding DSM cell.

Now consider those sampled mutual information which are greater than the threshold θ_1. We need another threshold

Let M be the set of sampled mutual information, and initialize θ_2 to be 0.

(1) Compute the mean of the data.
$\hat{\mu} = \text{mean}[M]$.

(2) Compute the variance of the data.
$\hat{\sigma}^2 = \text{var}[M]$.

(3) Compute the theoretical variance.
$\sigma^2 = \frac{2\hat{\mu}}{n}$.

(4) Check if the data is coherent enough to be one cluster.
if $(\hat{\sigma}^2 < 2\sigma^2)$ terminate and return θ_2.

(5) If not, execute the k-mean algorithm to partition the data into two clusters, M_L (the clustering with the smaller mean) and M_U (the clustering with the large mean), and also get the partition boundary θ_2.
$(M_L, M_U, \theta_2) = k\text{-mean}(M)$.

(6) Replace the data and repeat the process.
$M = M_U$, and go to step (1).

Figure 5: The pseudo code of the algorithm that determines the second threshold θ_2.

to separate the interactions of the lowest level from that of all other upper levels. For a hierarchical problem, the dependencies are different for each level. The task here can be also addressed as finding the cluster with the largest mean in a set of numbers. Since we know the variance of the sampled mutual information, we can simply partition the sampled mutual information until one cluster is coherent enough.

In this paper, we adopt the k-mean clustering algorithm where $k = 2$ to partition the data. We keep performing the partitioning procedure to the cluster with a higher mean until the sampled variance is less than twice of the theoretical variance. The algorithm is shown in Figure 5.

Now that both thresholds, θ_1 and θ_2, are computed, the DSM can be created as $DSM = [d_{ij}]$ where

$$d_{ij} = \begin{cases} 1, & \text{if } I(\text{gene}_i; \text{gene}_j) > \max(\theta_1, \theta_2). \\ 0, & \text{otherwise.} \end{cases} \quad (17)$$

The DSM constructed in this way should mainly contain the dependency information from the currently lowest level, which has the strongest dependencies by definition.

4.3 Putting it all together: DSMGA++

Recall that to conquer the hierarchical difficulty, a GA needs to maintain (1) proper decomposition at each level, (2) preservation of promising subsolutions, and (3) representation of a chunk at the lower level as one single variable at the upper level. Now we have every tool ready to design such a GA. Sine the linkage learning is based on DSMGA [24], we call the newly designed GA as DSMGA++.

DSMGA++ utilizes DSM clustering techniques to decompose the hierarchical problem at each level. It adopts restricted tournament replacement (RTR) [5, 14] to preserve promising subsolutions just like hBOA does. Unlike hBOA, however, in DSMGA++ we take the advantage of the explicit linkage model and achieve an explicit chunking scheme, the substructural chromosome compression.

To conclude, the substructural chromosome compression scheme expresses a building block by one single bit when the alleles in the building block nearly converge to only two niches. It reduces the problem complexity on the fly, and shrinks the search space when parts of the problem are solved.

	Mean	Standard Deviation
Level 1 Dependency	0.09064	0.02200
Level 2 Dependency	0.00291	0.00355
Level 3 Dependency	0.00151	0.00212
Threshold (θ_2)	0.04661689	

Table 2: The second threshold computed from the algorithm in Figure 5 and the means and variances of the measured mutual information from different levels of a 3-level hTrap with $k = 3$ at the first generation. The algorithm is able to distinguish level-1 dependency from dependencies of other levels.

5. EMPIRICAL RESULTS

First, we test if the algorithm shown in Figure 5 is able to find the threshold that we desire. The test function is a 3-level hTrap function with $k = 3$, and population size n is set to 2000. The sampled mutual information should come from three different distributions (level-1, level-2, and level-3 dependencies) and each of which can be approximated by a Gaussian distribution [9]. Table 2 shows the means and variances of the measured mutual information from three different levels as well as the second threshold (θ_2) that the algorithm in Figure 5 computes. As the results indicate, the algorithm can distinguish the level-1 dependency from

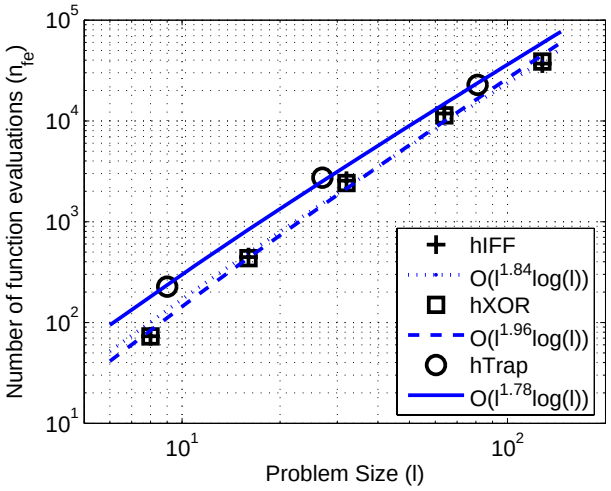


Figure 6: Preliminary scalability test of DSMGA++ on hIFF, hXOR, and hTrap. The number of function evaluations of DSMGA++ scales as $O(l^{1.84}\log(l))$ on hIFF, $O(l^{1.96}\log(l))$ on hXOR, and $O(l^{1.78}\log(l))$ on hTrap, where l is the problem size.

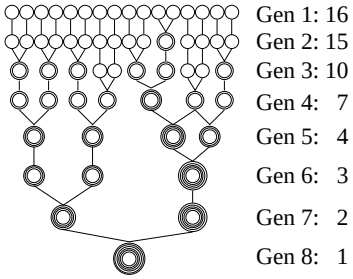


Figure 7: Problem structure obtained by the DSMGA++ for the hXOR function with 4 levels (chromosome length= $2^4 = 16$). The nodes represents genes of chromosomes. The number of circles represents the compression level. The descriptions on the right show the generation number and the chromosome length. The ideal case would be a complete binary tree with 4 layers. The problem structure obtained by DSMGA++ for hIFF is very similar to this.

dependencies of other levels.

The scalability of DSMGA++ is preliminarily tested on hIFF, hXOR, and hTrap. The result is shown in Figure 6. The number of function evaluations of DSMGA++ scales as $O(l^{1.84}\log(l))$ on hIFF, $O(l^{1.96}\log(l))$ on hXOR, and $O(l^{1.78}\log(l))$ on hTrap, where l is the problem size. The scaling order is not as small as that of hBOA [13] probably because the Bayesian network is a more powerful learning model. Nevertheless, DSMGA++ still scales subquadratically on those hierarchical problems. The preliminary results are reasonable and promising. The results well indicate that the proposed substructural chromosome compression scheme functions as expected since it is known that the number of function evaluations scales exponentially of the problem size without a proper chunking mechanism [13].

As mentioned, one of the advantages of DSMGA++ over hBOA is that DSMGA++ delivers the problem structure in a comprehensible manner to human researchers. The problem structures obtained by DSMGA++ for hXOR and hTrap are shown in Figures 7 and 8 respectively. The problem structure obtained from DSMGA++ on hIFF is very similar to that on hXOR, and hence is omitted. The tested hXOR has 4 levels, and the chromosome length is $2^4 = 16$. The tested hTrap has 3 levels, and the chromosome length is $3^3 = 27$. The perfect problem structure for hXOR would be a complete binary tree with 4 layers, and that for hTrap would be a complete 3-ary tree with 3 layers. In the figures, we can see (1) that sometimes some genes in a BB were clustered first then the other genes was clustered in later, (2) that some BBs converged faster than others, and (3) that sometimes overlaps occur. These imperfections are due to the sampling noise. Nevertheless, DSMGA++ still captures the problem structures pretty well.

Figure 9 illustrates the behavior of the substructural chromosome compression scheme on hIFF, hXOR, and hTrap. The tested hIFF and hXOR have 6 levels and chromosome length of $2^6 = 64$. The tested hTrap has 4 levels and chromosome length of $3^4 = 81$. The results are averaged over 100 independent runs. It can be seen that the chromosome lengths keep reducing during the GA runs, which helps the GA to solve the problem more efficiently.

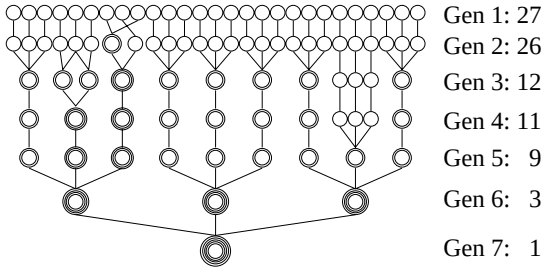


Figure 8: Problem structure obtained by the DSMGA++ for the hTrap function with 3 levels and $k = 3$ (chromosome length= $3^3 = 27$). The nodes represents genes of chromosomes. The number of circles represents the compression level. The descriptions on the right show the generation number and the chromosome length. The ideal case would be a complete 3-ary tree with 3 layers (Figure 1).

6. CONCLUSIONS

This paper proposes a genetic algorithm, namely DSMGA++, which optimizes problems via hierarchical decomposition. DSMGA++ utilizes DSMGA to decompose the problem at each level, restricted tournament replacement (RTR) to preserve alternative promising subsolutions, and a proposed substructural chromosome compression scheme to represent subsolutions at lower levels as decision variables at upper levels.

Preliminary scalability test indicates that the proposed scheme is promising. DSMGA++ scales sub-quadratically on the tested hierarchical problems, including hIFF, hXOR, and hTrap. It is also demonstrated that DSMGA++ is capable of capturing the problem structures for those test functions.

We would like to emphasize two things about DSMGA++: (1) DSMGA++ is capable of solving problems via hierarchical decomposition and it scales sub-quadratically with

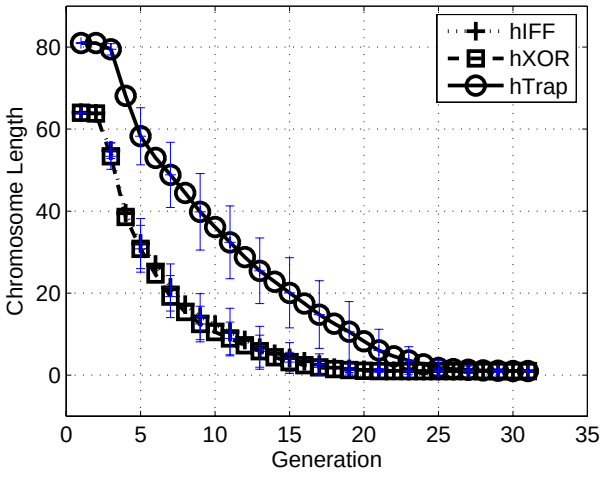


Figure 9: Chromosome length reduction for a level-6 hIFF, a level-6 hXOR, and a level-4, order-3 hTrap. The results are averaged over 100 independent runs.

the problem size, and (2) once the problem is solved, DSMGA++ delivers the problem structure as well. Compared to hBOA, the optimization procedure of DSMGA++ is more transparent to human researchers. The way that DSMGA++ optimizes problems is to automatically create a customized recombination operator for the problem. After applying DSMGA++ to a small-scale problem, one should be able to apply the customized recombination operator to a larger-scale problem of similar structure without the expense of model building.

As for future work, we would like to investigate larger-scale problems to verify the order of the scalability of DSMGA++. Also, we are interested in applying DSMGA++ to real-world problems and demonstrate the reusability of the recombination operator constructed by DSMGA++.

Acknowledgements

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096. 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, or the U.S. Government.

7. REFERENCES

[1] R. Descartes. A discourse on the method of rightly conducting the reason, and seeking truth in the sciences [Veitch, J. (trans.)]. In T. Sorell, editor, *A Discourse on Method: Meditations and Principles*, pages 3–57. Everyman, London, UK, 1994. Original work published 1637.

[2] C. Fernandez. Integration Analysis of Product Architecture to Support Effective Team Co-location. Master thesis, Massachusetts Institute of Technology, 1998.

[3] D. E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Boston, MA, 2002.

[4] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362, 1992.

[5] G. R. Harik. Finding multiple solutions in problems of bounded difficulty. IlliGAL Report No. 94002, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1994.

[6] G. R. Harik. Linkage Learning via Probabilistic Modeling in the ECGA. IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Urbana, IL, Feb. 1999.

[7] G. R. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Proceedings of 1997 IEEE International Conference on Evolutionary Computation*, pages 7–12, 1997.

[8] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[9] M. Hutter and M. Zaffalon. Distribution of mutual information from complete and incomplete data. *Computational Statistics and Data Analysis*, 48(3):633–657, 2005.

[10] G. D. Kleiter. The posterior probability of Bayes nets with strong dependences. *Soft Computing*, 3:162–173, 1999.

[11] S. Kullback and R. A. Leibler. On information and sufficiency. *Annual Mathematical Statistics*, 22:79–86, 1951.

[12] P. Larrañaga and J. Lozano, editors. *Estimation of Distribution Algorithms*. Kluwer Academic Publishers, Boston, MA, 2002.

[13] M. Pelikan. *Bayesian Optimization Algorithm: From Single Level to Hierarchy*. Doctoral dissertation, University of Illinois at Urbana-Champaign, 2002.

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

[15] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[16] D. Sharman, A. Yassine, and P. Carlile. Characterizing modular architectures. *ASME 14th International Conference*, pages DTM–34024, Sept. 2002.

[17] H. A. Simon. *The Science of Artificial*. The MIT Press, Cambridge, Massachusetts, 1968.

[18] D. V. Steward. The design structure system: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management*, 28:77–74, 1981.

[19] M. Toussaint. Compact genetic codes as a search strategy of evolutionary processes. *Foundations of Genetic Algorithms 8 (FOGA 2005)*, pages 75–94, 2005.

[20] R. A. Watson, G. S. Hornby, and J. B. Pollack. Modeling building-block interdependency. *Parallel Problem Solving from Nature, PPSN-V*, pages 97–106, 1998.

[21] R. A. Watson and J. B. Pollack. Hierarchically consistent test problems for genetic algorithms: Summary and additional results. *Late breaking papers at the Genetic and Evolutionary Computation Conference*, pages 292–297, 1999.

[22] R. A. Watson and J. B. Pollack. Modular interdependency in complex dynamical systems. *Artificial Life*, 11(4):445–457, 2005.

[23] A. Yassine, D. R. Falkenburg, and K. Chelst. Engineering design management: An information structure approach. *International Journal of production research*, 37(13):2957–2975, 1999.

[24] T.-L. Yu, D. E. Goldberg, A. Yassine, and Y.-p. Chen. Genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm. *Proceedings of Artificial Neural Networks in Engineering 2003 (ANNIE 2003)*, pages 327–332, 2003.