Dependency Structure Matrix, Genetic Algorithms, and Effective Recombination

Tian-Li Yu tianliyu@cc.ee.ntu.edu.tw
Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan

David E. Goldberg deg@uiuc.edu
Department of Industrial & Enterprise Systems Engineering, University of Illinois at Urbana-Champaign, Illinois 61801

Kumara Sastry kumara@kumarasastry.com
Department of Industrial & Enterprise Systems Engineering, University of Illinois at Urbana-Champaign, Illinois 61801[1]

Claudio F. Lima clima@ualg.pt
Department of Electronics and Computer Science Engineering, University of Algarve Campus de Gambelas, 8000-117 Faro, Portugal

Martin Pelikan pelikan@cs.umsl.edu
Department of Math and Computer Science, University of Missouri at St. Louis, Missouri 63121

Abstract

In many different fields, researchers are often confronted by problems arising from complex systems. Simple heuristics or even enumeration works quite well on small and easy problems; however, to efficiently solve large and difficult problems, proper decomposition is the key. In this paper, investigating and analyzing interactions between components of complex systems shed some light on problem decomposition. By recognizing three bare-bones interactions—modularity, hierarchy, and overlap, facetwise models are developed to dissect and inspect problem decomposition in the context of genetic algorithms. The proposed genetic algorithm design utilizes a matrix representation of an interaction graph to analyze and explicitly decompose the problem. The results from this paper should benefit research both technically and scientifically. Technically, this paper develops an automated dependency structure matrix clustering technique and utilizes it to design a model-building genetic algorithm that learns and delivers the problem structure. Scientifically, the explicit interaction model describes the problem structure very well and helps researchers gain important insights through the explicitness of the procedure.

Keywords

Genetic algorithms, dependency structure matrix, problem decomposition, modularity, hierarchy, overlap.

[1]Currently at Intel Corp.

1 Introduction

In a variety of different areas, researchers have been investigating complex systems, and one of the major directions in scientific research is to solve problems arising from these systems. Here, the concept of problem solving is general and involves many important research topics, including equation solving, search, optimization, and machine learning. In the context of genetic algorithms (GAs; Holland, 1975; Goldberg, 1989), problem solving refers to finding global or near-global optima for a given fitness function. For easy and small-scale problems, simple heuristics or enumeration works quite well. However, to solve the more difficult and larger-scale problems in complex systems effectively and efficiently, one needs to discover the "clouded simplicity" (Simon, 1968) or the "hidden order" (Holland, 1995) of such systems, if there is one.

Once the complexity is better understood, a large problem can be decomposed into several smaller subproblems. The concept of decomposition can be tracked back at least as far as the 1637 publication of Descartes's *A Discourse on Method* (Descartes, 1637/1994) and lays the foundations of many important problem-solving techniques in computer science, including divide-and-conquer, dynamic programming (Cormen et al., 2001), and artificial intelligence planning (Russell and Norvig, 2003). According to Descartes (1637/1994) and Simon (1968), understanding complex systems is possible only through proper decomposition. In other words, difficult problems arising from such systems are solvable only by bootstrapping from simple subsolutions into more complex and complete ones.

Solving problems by decomposition is efficient because a proper decomposition minimizes the *interactions* between subproblems. The interaction is defined by the nature of the problem. Two components interact with each other if and only if the associated subproblem cannot be solved without the information carried by both components. Based on a similar idea, this paper develops techniques to decompose a complex system via detecting interactions between components of the system. Moreover, this paper takes a step further to categorize the interactions into three different types: (1) *modularity* (interactions between components), (2) *hierarchy* (interacting components form modules and interacting modules form higher-level modules), and (3) *overlap* (interactions between modules if they share some components; Figure 1). Here, we borrowed the terms "component" and "module" from the field of organization theory (Christopher, 1964) since we are discussing complex systems. In a GA, a module refers to a group of highly-interacting genes whose information should be transferred together during recombination in order to produce potentially superior offspring.

In this paper, the idea of decomposing complex systems by detecting interactions between components is applied to the GA field to design a powerful stochastic problem solver that solves difficult problems with modularity, hierarchy, and overlap. Particularly, the work in this paper develops a technique to extract the interaction information—the information of highly interacting components—by using a dependency structure matrix (DSM) borrowed from the literature of project management and corporate organization. The DSM clustering technique is then utilized to design a *competent* GA, called the *dependency structure matrix genetic algorithm*, or *DSMGA*.

The primary objective of this paper is to understand bare-bones requirements for solving boundedly difficult problems arising from complex systems with modularity, hierarchy, and overlap. Based on this understanding, this paper also tries to develop component GA techniques to solve such problems via proper problem decomposition.

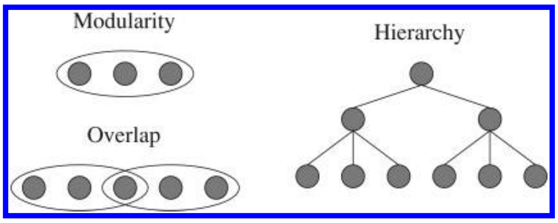


Figure 1: Three bare-bones interactions: modularity, hierarchy, and overlap.

This paper is composed of five primary parts. The first part describes the dependency structure matrix (DSM) and the DSM clustering problem. The second part utilizes the DSM clustering technique to design DSMGA, which solves problems with modularity. The third part extends DSMGA to solve problems via hierarchical decomposition. The fourth part recognizes the keys to conquer the overlap difficulty via investigating the 2D spin-glass problem and designs an effective and efficient recombination method for problems with overlap. The fifth part concludes this paper.

2 Analyzing Complex Systems via DSM Clustering

This section describes how to analyze complex systems by DSM clustering techniques. A DSM is a matrix that contains the information of pair-wise interaction between every pair of the components in a system. The objective of DSM clustering is to transfer the pair-wise interaction information into higher-order interaction information. DSM clustering techniques are known to be extremely useful in architectural improvement in organizations and product design and development (McCord and Eppinger, 1993; Pimmler and Eppinger, 1994). The clustering techniques developed in this section will be used later in this paper as a module detector to design a *competent* GA (Goldberg, 2002) that solves boundedly difficult problems with modules within a subquadratic number of function evaluations.

This section firstly gives a brief introduction to the DSM method, the terminology, and the complexity of DSM clustering. Subsequently, a clustering metric is proposed based on the minimum description length principle (MDL). Combining with a simple GA, the clustering metric is then tested on several manually designed DSMs.

2.1 Dependency Structure Matrix (DSM) Method

A DSM is a matrix representation of a graph. The vertices of the graph, representing the components in a complex system, correspond to the column and row headings in the matrix (Eppinger et al., 1994; Yassine et al., 2003), as shown in Figure 2. The arrows, representing relationships between components, correspond to the $\times$ marks inside the matrix. For example, if there is an arrow from vertex C to vertex A, then an $\times$ mark is placed in row A and column C. Diagonal entries have no significance and are blacked out in the following figures. Alternatively, one can use 1 and 0 to replace $\times$ and blank, respectively, making the DSM a binary matrix $[d_{ij}]$ with entries $d_{ij} = 0$ or 1 for $i \neq j$. For simplicity and consistency, we set $d_{ii} = 1$ in the binary representation.

Once the DSM for a product is constructed, it can be analyzed for identifying modules, a process referred to as clustering. The goal of DSM clustering is to find a clustering arrangement where modules minimally interact with each other while

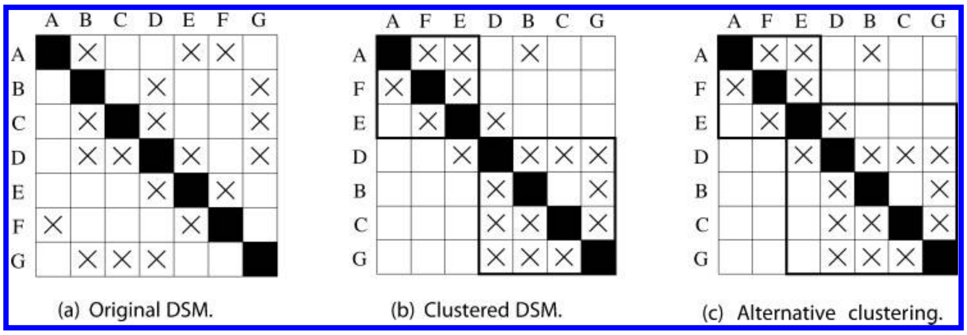


Figure 2: DSM clustering examples.

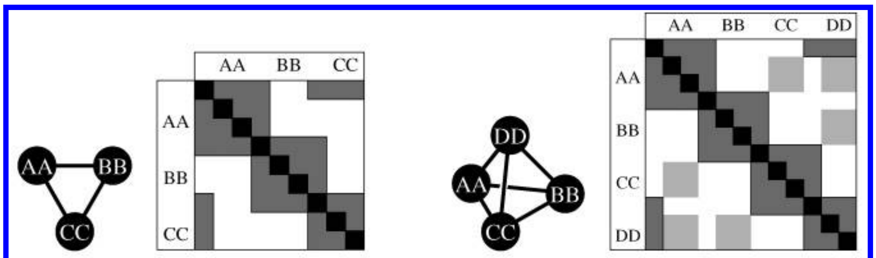
components within a module maximally interact with each other (Fernandez, 1998). As an example, consider the DSM shown in Figure 2(a). One can see from Figure 2(b) that the original DSM was rearranged by permuting rows and columns to contain most of the interactions within two separate modules: $\{A, F, E\}$ and $\{D, B, C, G\}$. However, three interactions are left out of any modules. An alterative arrangement is suggested in Figure 2(c). This arrangement suggests the forming of two overlapping modules: $\{A, F, E\}$ and $\{E, D, B, C, G\}$. It eliminates two left-out interactions by introducing a bigger but sparser module. Generally speaking, a clustering arrangement is considered to be "good" if only few (or none) interactions are left out and clusters are dense.

The DSM representation of a system/product architecture has been shown to be useful because of the visual appeal and simplicity. Numerous researchers have used it to propose architectural improvements by manipulating the order of rows and/or columns in the matrix (McCord and Eppinger, 1993; Pimmler and Eppinger, 1994). In an attempt to automate this manual process of DSM inspection and manipulation, Fernandez (1998) used the simulated annealing search technique to find good DSM clustering arrangements. In his approach, each component starts out by being an individual module and evaluates bids from all the other modules. If any module is able to make a bid that is better than the current base case, then the component is moved inside the module. The objective function is therefore a trade-off between the cost of being inside a module and the overall system benefit. Sharman et al. (2002) attempted using Fernandez's algorithm on an industrial gas turbine. However, they showed that this algorithm is incapable of predicting the formation of good clustering arrangements for complex product architectures due to the oversimplification of the objective function. In a similar venue, Whitfield et al. (2002) used GAs to form product modules. Their algorithm is also built upon Fernandez's concept and consequently suffers from similar problems.

To design an objective function that better captures the characteristics of DSM clustering, we need to first better understand DSM clustering complexities, which will be discussed next.

2.2 DSM Clustering Complexities

The challenge of applying automated clustering algorithms to complex DSMs comes from the difficulty of extracting the relevant information and then conveying the



(a) One module has to be broken in the DSM representation.

(b) In the 2D DSM representation, many modules have to be broken. The overlaps between modules (light gray) make the DSM difficult for humans to understand.

Figure 3: Planar triangular modules and tetrahedron of modules.

information to users. This is most noticeable in the poor handling of the path-dependent situations and modules with a three-dimensional topology.

2.2.1 Path Dependency in Clustering

Consider a triangular arrangement with symmetrical relationships that can be loosely clustered into three similar modules AA, BB, and CC, as shown in Figure 3(a). The DSM for this physical arrangement can only show two modules and must break up the third. The way the clustering algorithm operates will be path dependent since once the algorithm has started to cluster on any two components, it is unlikely to reverse course to explore a different configuration. This situation may occur depending on how the clustering algorithm perceives the raw data. For example, branch and bound algorithms that are presented with a partially clustered starting point may never branch widely enough to evaluate alternative solutions. For the example in Figure 3(a), module CC has been broken up even though it is identical to the other two clusters in all respects. Module AA or BB could equally likely be broken up when positioned where CC is.

2.2.2 Dimensions and Topology

Consider a simple three-dimensional structure such as a tetrahedron. This is depicted in Figure 3(b) showing four equal modules, each with dense internal interactions and weak external interactions. If all the modules are perfectly equal, it is purely a matter of chance how any clustering algorithm would present a clustering arrangement. In this example, module DD is the one visually disrupted most by being presented last in the sequence. This results in the effect of spreading its inter-module interactions over a wider spatial area, depicted in the DSM in light gray. To an untrained observer, this might be thought of as a bus where module DD is the unique possessor of system-wise integrating functions and some random cross-links in the zone AA-CC.

2.3 MDL-Based DSM Clustering Metric

Existing DSM clustering algorithms can be found elsewhere (McCord and Eppinger, 1993; Fernandez, 1998; Thebeau, 2001; Whitfield et al., 2002). Experiments with these algorithms have shown that those clustering metrics are insufficient for accurately predicting "good" clustering arrangements (Sharman et al., 2002; Sharman and Yassine, 2004). The lack of an effective clustering method motivates the idea of developing an information theoretic clustering metric that suggests an appropriate total number of modules and detects overlapping modules as well as modules with a three dimensional structure.

Yu et al. (2003b) proposed a clustering metric based on the minimum description length principle (MDL; Rissanen, 1978, 1999; Barron et al., 1998; Lutz, 2002), where MDL is adopted to balance the complexity and accuracy of the clustering arrangement. The metric can be expressed as follows:

$$f_{\mathrm{DSM}}(M) = n_M \log n_c + \log n_c \Sigma_{i=1}^{n_M} M_i + (|S_1| + |S_2|)(2 \log n_c + 1), \tag{1}$$

where n_M is the total number of modules, n_c is the total number of components, M_i is the number of components within the ith modules, and S_1 and S_2 are two mismatch sets (Yu et al., 2003b).

With the metric, the DSM clustering problem is converted to an optimization problem: Given a DSM, the objective is to find a DSM clustering arrangement, M, to minimize the metric, f_{DSM}. In other words, f_{DSM} is the length needed to describe the given dataset DSM by using the model M.

We then used a GA to optimize the above DSM-clustering problem. Figure 4 shows the empirical results of two manually designed examples. Those DSMs are designed with known optimal clustering arrangements. They are then randomly reordered and handed to the GA as inputs to test if the GA is able to rediscover the modules. These results show that a GA with the MDL-based clustering metric is able to correctly cluster DSMs with overlapping modules and modules that are broken in the "a" representation.

To conclude, MDL provides a good metric for DSM clustering. The real-world applications of this DSM clustering technique can be found elsewhere (Yu et al., 2007).

3 Finding Extrema of Problems with Modularity

This section proposes the dependency structure matrix genetic algorithm (DSMGA) and demonstrates how DSMGA solves problems with nonoverlapping modularity. DSMGA utilizes DSM clustering techniques to extract building blocks (BBs; Goldberg, 2002) information and uses the information to accomplish BB-wise crossover. Here, a BB is a group of highly interacting genes, and is essentially a module. Three cases, tight, loose, and random modularity, are tested using DSMGA and a simple GA. Empirical results show that DSMGA is able to correctly identify BBs and outperforms a simple GA by using the extracted BB information.

3.1 What Are Modules in GAs?

The concept of modularity in GA is not new. It can be found at least as early as Holland's observation in 1975 (Holland, 1975) under the notion of *building blocks* (BBs). Later, the concept of BBs is refined and developed by Goldberg, addressed as *the building block hypothesis* (BBH; Goldberg, 1989, 2002; Mitchell, 1996; Holland, 2000), and caused a long debate in the GA research history. The goal of this paper is not to argue BBH. Instead, we would like to show that if modularity exists in an optimization problem, learning and utilizing the module information enables GAs to efficiently solve the problem via problem decomposition.

Giving a formal definition of modules in GAs is not an easy task and beyond the scope of this paper. However, to clarify what this paper aims to solve, we give a *loose*

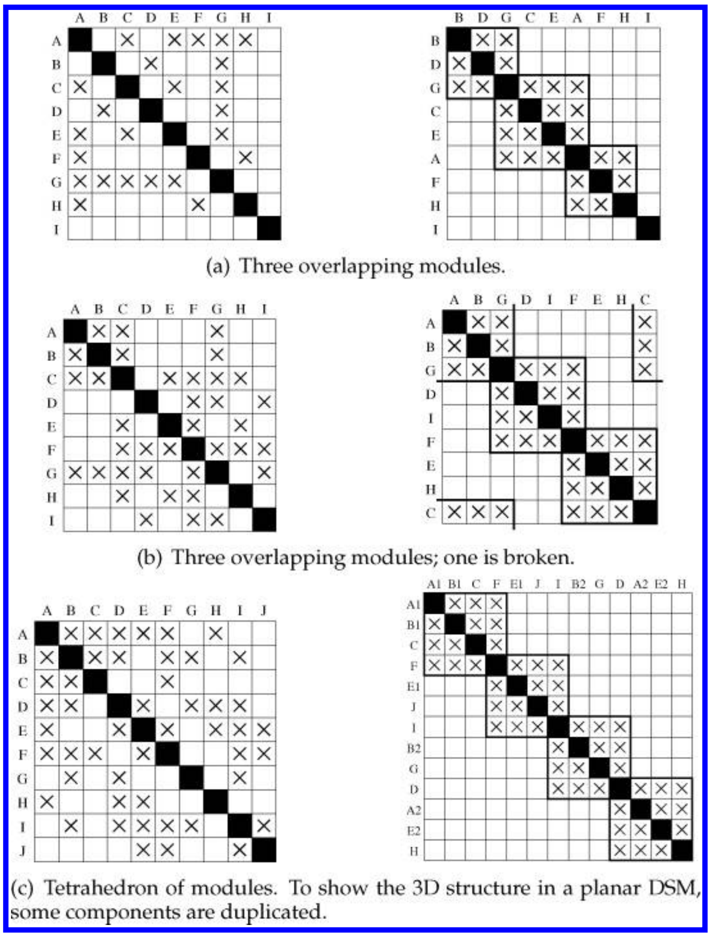

(a) Three overlapping modules.
(b) Three overlapping modules; one is broken.
(c) Tetrahedron of modules. To show the 3D structure in a planar DSM, some components are duplicated.

Figure 4: Illustrative examples of DSM clustering. The left column is the manually designed DSMs given as inputs to the GA. The right column shows how the GA clusters them.

definition of modules here: “two genes belong to same module when the recombination of promising solutions yields lowly-fit offspring with the information of these two genes considered separately.” For more formal definitions, readers are referred to the above references about BBH and references concerning deception (Goldberg, 1987; Deb and Goldberg, 1993).

The concept of modularity also plays an important role in genetic programming (GP; Koza, 1994). The concept is addressed as *automatically defined functions* (ADFs). The concept, however, is different from that in GAs. In GP, an ADF corresponds to a repeatedly reusable program, while in GA, a module corresponds to a group of strongly interacting genes—those genes are not repeatedly reusable, but they are subsolutions that should not be disrupted during recombination. GP researchers have also been wondering if the concept of BBs can help GP development (Haynes, 1997; Sastry et al., 2003).

In short, the modularity mentioned in this paper is similar to the concept of BBs in GA and GP, but without overlapping. The goal of identifying such modules is to achieve problem decomposition. The concept of ADF in GP also comes from modularity, but an

ADF is a component primarily for reusability, while a BB is a component primarily for decomposition.

3.2 Interaction Detection

It has been shown that if the GA is not capable of learning the interaction topology of the problem, the population size required for finding the global optima scales exponentially with the problem size (Thierens and Goldberg, 1993). If the GA is able to learn the problem structure and decomposes the problem accordingly, the population size required is $O(m2^k)$, where m is the number of BBs and k is the order of one BB. In other words, a GA with proper problem decomposition only needs a polynomial number of function evaluations to the problem size for boundedly difficult problems, where k is bounded by a constant. Note that the primary difficulty comes from the fact that the information of which genes form which BBs is unknown.

The key to reduce the run duration of GAs from exponential to polynomial is interaction detection. Many such techniques have since been developed and can be categorized as follows.

Perturbation. GAs in this category detect interactions among genes by perturbing alleles and monitoring the change done by such perturbation. Such GAs usually consist of two phases. The first phase perturbs alleles and detects interactions, and the second phase combines the promising BBs. Typical examples are mGA (Goldberg et al., 1989), fmGA (Goldberg et al., 1993), gemGA (Kargupta, 1996), LINC, and LIMD (Munetomo and Goldberg, 1999).

Interaction Adaptation. GAs in this category utilize a unique fitness metric to detect interactions and solve the problem at the same time. The interaction arrangement is embedded in the encoding. A chromosome with highly interacting genes encoded closer has a higher survival rate under recombination. Typical examples are LEGO (Smith and Fogarty, 1996) and LLGA (Harik, 1997; Chen and Goldberg, 2005).

Model Building. The typical framework of model building GAs consists of the following steps: (1) randomly initializing a new population, (2) selecting a set of promising solutions, (3) model building based on those promising solutions, (4) generating the next generation by utilizing the model, and (5) repeating steps 2–5 until the termination condition is met. Examples are cGA (Harik et al., 1998), ecGA (Harik, 1999), BOA (Pelikan et al., 1999), and hBOA (Pelikan and Goldberg, 2001; Pelikan, 2005). More examples can be found in Larrañaga and Lozano (2002), Goldberg (2002), and Pelikan et al. (2006).

DSMGA developed in this paper belongs to the model building GA category. Among GAs in this category, DSMGA most resembles ecGA and hBOA while it differs from them in the following respects. The ecGA uses an information theoretical metric to detect the interactions among genes. The interaction model used by ecGA is explicit, but does not consider overlap and hierarchy. The hBOA utilizes a Bayesian network to express the interactions among genes and implicitly handles problems with modularity, hierarchy, and overlap. Similar to ecGA, the interaction model in the DSMGA design is also explicit, but the DSMGA design handles problems with modularity, hierarchy (Section 4), and overlap (Section 5). The hBOA also solves problems with all three types

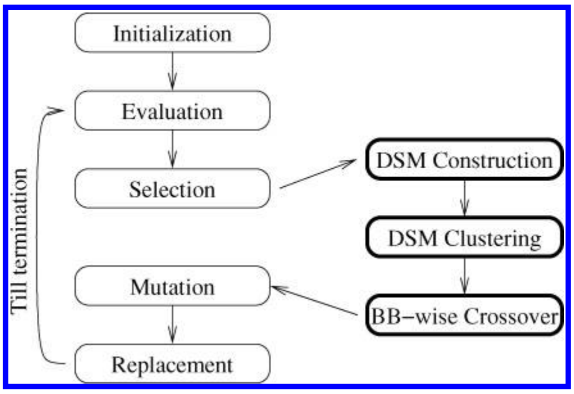


Figure 5: The framework of DSMGA. The major difference between DSMGA and a simple GA is that DSMGA utilizes DSM clustering techniques to obtain BB information and then uses it to achieve BB-wise crossover.

of interactions, but the interaction model in DSMGA is much more comprehensible, so it is easier for researchers to gain knowledge about the problem.

3.3 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. Three primary tasks in DSMGA are: (1) DSM construction from the current population, (2) DSM clustering, and (3) BB-wise crossover. The framework of DSMGA is shown in Figure 5.

DSMGA constructs the DSM by using mutual information metric. Entry d_{ij} of the DSM is given by the *sampled* mutual information between the ith and the jth genes from the population after selection. We call it sampled because the mutual information is calculated based on the current population instead of the whole search space. Similar entropy-based metrics have been commonly used in other types of GAs. The ecGA (Harik, 1999), BMDA (Pelikan and Mühlenbein, 1999), and BOA (Pelikan et al., 1999) are typical examples. The calculation will be detailed later.

The DSM clustering was detailed in the previous section. The only difference in DSMGA is that based on the research by Yu and Goldberg (2004), it is beneficial to adopt a hill-climber for the DSM clustering to save computational time while not degrading model quality too much.

BB-wise crossover (Harik, 1999) is performed after DSM clustering obtains BB information. The BB-wise crossover is similar to a regular allele-wise crossover when there are no overlapping BBs. The only difference is that the exchange of information is performed at the BB level. If the BB information is perfect, no BB disruption occurs. DSMGA adopts BB-wise uniform crossover since it provides a higher mixing rate than one-point crossover or two-point crossover.

3.3.1 Interaction-Detection Threshold

To alleviate the computational burden, after the sampled mutual information is calculated, it is transferred into the binary domain where the metric indicates whether or not the pair of genes interact with each other. Once an appropriate threshold is calculated, a pair of genes is considered as interacting with each other if and only if the calculated mutual information is greater than the threshold. The threshold can be decided by investigating the distribution of mutual information.

Mutual information is defined as the Kullback-Leibler distance (Kullback and Leibler, 1951) between the joint distribution and the product distribution:

$$\begin{aligned}\mathbb{I}(X;Y) &= D(p(x,y)||p(x)p(y)) \\ &= \Sigma_x \Sigma_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)},\end{aligned} \tag{2}$$

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

According to Kleiter (1999) and Hutter and Zaffalon (2005), if two random variables X and Y are independent, the distribution of the sampled mutual information $\mathbb{I}_n(X;Y)$ calculated over n samples can be approximated as a Gaussian distribution with mean $\mu \approx \frac{1}{2n}$ and variance $\sigma^2 \approx \frac{1}{2n^2}$. Both μ and σ^2 tend to zero when the sample size n is large.

Given a threshold θ, the decision error ϵ is given by $1 - \Phi(z)$, where Φ is the cumulative standard Gaussian probability function and $z = \frac{\theta-\mu}{\sigma}$.

For a problem with m modules, identifying at least $(m-1)$ modules correctly is preferred. Thus the overall decision accuracy needs to be greater than or equal to $\frac{m-1}{m}$. In a DSM, there are approximately $\frac{l^2}{2}$ pairs of genes. Therefore, the following relation holds.

$$(1-\epsilon)^{\frac{l^2}{2}} \geq \frac{m-1}{m}. \tag{3}$$

For a large m and a large l, the inequality in Equation (3) can be approximated as $\epsilon \leq \frac{2}{ml^2}$.

When the decision variable z is large, the cumulative standard Gaussian function can be approximated using a Taylor expansion yielding the following inequality.

$$\frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}z} \leq \frac{2}{ml^2}. \tag{4}$$

Recall that $z = \frac{\theta-\mu}{\sigma}$. Given $l = km$, the inequality in Equation (4) can be solved for z as

$$z \geq \sqrt{\mathbb{W}(cl^6)}, \tag{5}$$

where $\mathbb{W}$ is Lambert's W function[2] and $c = \frac{1}{8\pi k^2}$. For problems with unknown k, it is conservative to calculate z by assuming $m = l$, and hence $c \approx 0.04$. Given a population size

[2]Lambert's W function is defined as the inverse function of $f(W) = We^W$.

of n, a threshold that ensures $(m-1)$ modules are identified correctly can be decided as

$$\theta = \frac{1}{2n} + \sqrt{\frac{\mathbb{W}(cl^6)}{2n^2}}. \tag{6}$$

The mutual information metric can also be transferred into probabilistic domain instead of binary domain. In this case, the transferred metric indicates the probability that the pair of genes interact with each other. However, this is not necessary given that the variance is approximately $\frac{1}{2n^2}$, which decreases rapidly with increasing population size n. In a regular GA experiment, the population size is usually large enough such that the gray area is small, and the information from the binary domain is accurate enough to make the correct decision.

3.4 A Modularity Identification Test

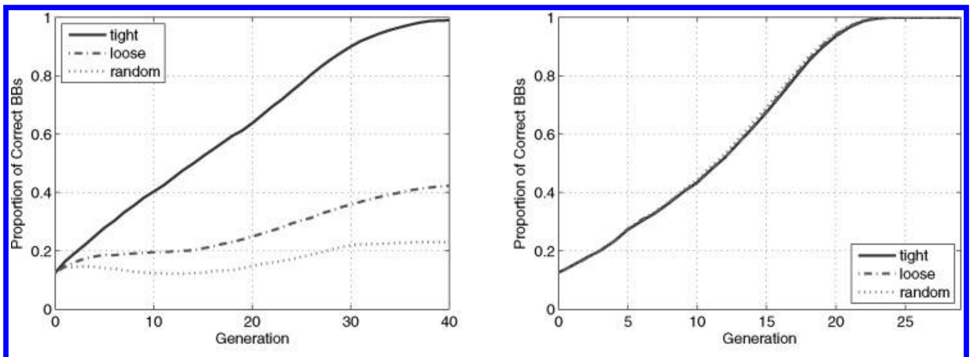
We used an (m, k) trap, where $m = 10$ and $k = 3$ as a test function. The 3-bit trap is given by $f^3_{\text{trap}}(u = 0) = 0.9$, $f^3_{\text{trap}}(u = 1) = 0.45$, $f^3_{\text{trap}}(u = 2) = 0.0$, and $f^3_{\text{trap}}(u = 3) = 1.0$, where u is the number of 1s. The reason for adopting the trap function is to emphasize the problem decomposition. If the problem is not properly decomposed, any hill-climbing methods tend to give a solution containing all 0s (Goldberg, 1987).

Three cases were tested: tight, loose, and random modularity. Define $U(x)$ as a counting function that counts the number of 1s in x. In the tight modularity test, interacting genes are arranged next to each other, and the fitness function is defined as $f^3_{\text{trap}}(U(x_1, x_2, x_3)) + f^3_{\text{trap}}(U(x_4, x_5, x_6)) + f^3_{\text{trap}}(U(x_7, x_8, x_9)) + \cdots$. In the loose modularity test case, interacting genes are arranged away from each other as far as possible, and the fitness function is defined as $f^3_{\text{trap}}(U(x_1, x_{11}, x_{21})) + f^3_{\text{trap}}(U(x_2, x_{12}, x_{22})) + f^3_{\text{trap}}(U(x_3, x_{13}, x_{23})) + \cdots$. In this arrangement, the crossover operator easily disrupts BBs if interactions are not detected correctly. For more details about this test scheme, readers are referred to Goldberg et al. (1989).

Given a failure rate of $\frac{1}{10}$, the population size is set to 182 according to the gambler's ruin model (Harik et al., 1997; Miller, 1997). Binary tournament selection is adopted, and no mutation is used. Without prior knowledge, the maximal number of clusters is set to 30, assuming that the number of clusters would be lesser than or equal to the chromosome length. Finally, a bit-wise hill climber is adopted in the DSM clustering phase.

Figure 6(a) shows the performance of the simple GA using two-point crossover. The simple GA works only for the tight modularity case. For the loose and random modularity cases, the simple GA does not work because of BB disruption. Correspondingly, Figure 6(b) illustrates the performance of DSMGA using BB-wise two-point crossover. DSMGA converges for all three tests. Even in the tight modularity test case, DSMGA (converges at the 28th generation) outperforms the simple GA (converges after the 40th generation) because DSMGA disrupts fewer BBs. Figure 7 shows the DSM created by DSMGA for the tight modularity case. The perfect result is 10, 3-bit clusters located on the diagonal. At the fifth generation, DSMGA identifies eight BBs correctly; at the tenth generation, DSMGA has successfully identified all 10 BBs.

These results indicate that simple GAs work with problems that have tight modularity, but not on problems with random modularity. In other words, to ensure the performance of simple GAs, either modularity does not play an important role in the



(a) Simple GA with two-point crossover.

(b) DSMGA with BB-wise two-point crossover.

Figure 6: The performance comparison of a simple GA with two-point crossover and DSMGA with BB-wise two-point crossover.

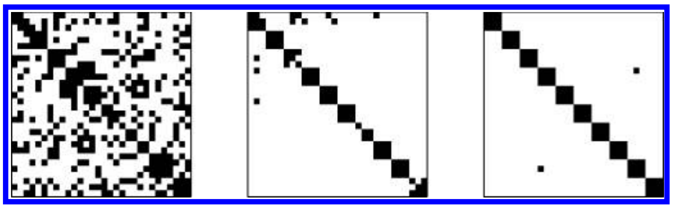


Figure 7: The DSMs created by the DSMGA in the tight modularity test. From left to right, the DSMs are created at generation 0, 5, and 10, respectively. The perfect result should be 10, 3-bit clusters on the diagonal. Results of the random and loose modularity test are similar, but with columns and rows reordered.

problem, or we encode the problem in a way that interacting genes are next to each other based on prior knowledge.

4 Finding Extrema for Problems with Hierarchy

The previous section shows that DSMGA is capable of solving problems with modularity through proper decomposition. This section extends the algorithm to another level of optimization by considering another class of problems—hierarchical problems.

Hierarchical problems come from a hierarchical complex system—a system composed of subsystems, each of which is hierarchical by itself (Simon, 1968). Many complex systems around us are hierarchical. Materials are composed of molecules, molecules are composed of atoms, atoms are composed of electrons, protons, neutrons, and so forth. A book is composed of chapters, chapters are composed of sections, 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 (2001) proposed hBOA, one of the few genetic and evolutionary algorithms that are known to optimize problems with random modularity by hierarchical decomposition. hBOA has shown the ability to decompose hierarchical problems that are not fully decomposable in one single level (Pelikan, 2002; Pelikan and Goldberg, 2001). Readers who are interested in hierarchical problems in the context of GAs are also referred to the work of Watson and Pollack (2005).

The hBOA has demonstrated excellent optimization ability; 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 section is to develop an explicit hierarchical decomposition scheme for GAs. The proposed method optimizes the problem via explicit hierarchical decomposition, and the stored problem structure is transparent to human researchers. In this section, we assume no overlaps among modules. We will discuss problems with overlapping modules in the next section.

The section firstly revisits hierarchical difficulty, hierarchical problems, and the keys to conquer them. Then it describes the proposed explicit chunking—substructural chromosome compression, which reduces the search space on the fly. Finally, experiments and discussion conclude this section.

4.1 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 interactions in an upper level are too weak to detect unless all lower levels are solved. This section describes the keys to conquer hierarchical difficulty and details a hierarchical problem—the hierarchical trap, which is used as a test function later in this section.

4.1.1 Keys to Conquer Hierarchical Difficulty

Pelikan and Goldberg (2001) 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, the chunking is implicit and is achieved by recognizing local substructures in the Bayesian network. This section proposes an explicit chunking scheme, which is more comprehensible. Details will be described in Section 4.2.

4.1.2 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. The test problem—the hierarchical trap (Pelikan and Goldberg, 2001)—is detailed as follows.

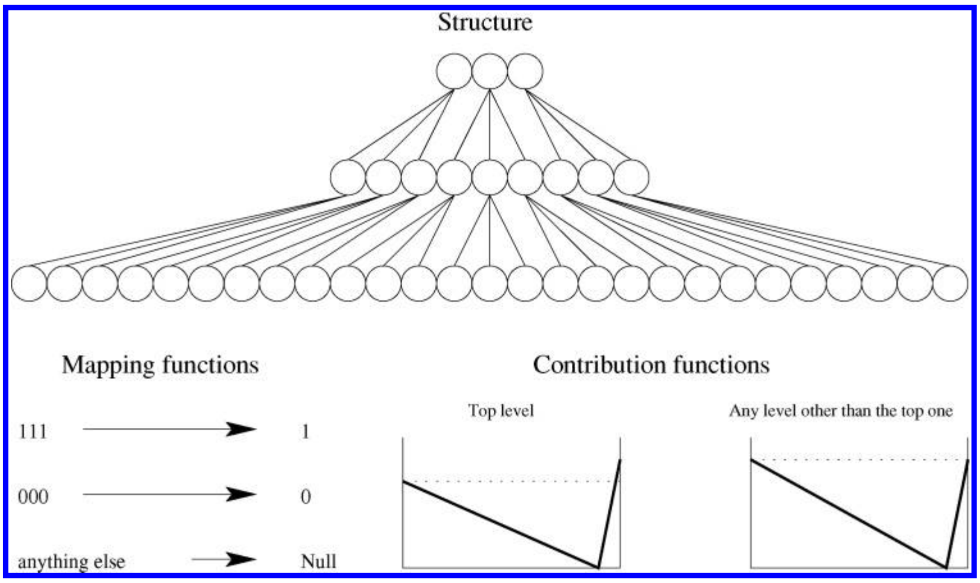


Figure 8: The structure, mapping function, and contribution functions of the hierarchical trap. This example has three levels and $k = 3$. Note that 111 and 000 are equally good except for the top level.

Pelikan and Goldberg (2001) proposed a scalable hierarchical problem called the *hierarchical trap* (hTrap). The hTrap is composed of three major components (see Figure 8).

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 all 0s and 1s is mapped to 0 and 1 respectively, and everything else is mapped to "-" which does not contribute to the next level.
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:

$$trap_k(u) = \begin{cases} f_{\text{high}} & \text{if } u = k \\ f_{\text{low}} \times \dfrac{k-1-u}{k-1} & \text{otherwise,} \end{cases} \tag{7}$$

   where u is the unitary of the input string. If any position of the string is "-", then the contribution of the string to the fitness is zero.

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

4.2 Substructural Chromosome Compression

This section proposes an explicit chunking scheme, called the substructural chromosome compression. The substructural chromosome compression scheme expresses a BB by one single variable when the BB nearly converges to only few expressive schemata.

Apart from hBOA, one of the early efforts on solving hierarchical problems via the idea of compression is HGA (DeJong et al., 2004). Their identification and compression mechanism is different, and HGA takes roughly $O(l^k)$ time. The idea of using standard text compression techniques in GAs has also been explored elsewhere (Toussaint, 2005), but the method to date only works with explicit prior knowledge of model boundaries, an assumption that makes the technique virtually unusable in most applications. Nevertheless, the idea of compression is sound and the goal of this section is to realize a practical and broadly applicable technique to solve problems on a larger scale. More recently, the hierarchical trap was claimed to be solvable in linear time (Iclanzan and Dumitrescu, 2007). However, one of the assumptions is that the BB information is given. Otherwise, it will require $O(l^2)$ number of function evaluations to retrieve the BB information. In this paper, the BB information is not assumed to be known. Instead, the GA utilizes DSM clustering to identify BBs.

The key idea of substructural chromosome compression is to represent a nearly-converged BB by only π of the most expressive schemata. The gambler's ruin population-sizing model (Harik et al., 1997) estimates the population size to be $\Theta(2^k m)$ for a binary problem with m BBs of order k. In typical hierarchical problems, such as hIFF (Watson et al., 1998), hXOR (Watson and Pollack, 1999), and hTrap, the second level has (m/k) BBs with order k. Since the complexity of a higher level is expected to decrease, the following condition should hold assuming the leading constant is virtually problem independent.

$$\pi^k \left(\frac{m}{k}\right) \le 2^k m, \tag{8}$$

or simply

$$\pi \le 2\sqrt[k]{k}. \tag{9}$$

Note that π decreases when k increases, and for $k \ge 3$, π is less than 3. Therefore this paper only focuses on the case where $\pi = 2$ although the method is not limited to this special case. In other words, when a BB nearly converges, it is compressed to a 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 an upper level. Otherwise, the number of schemata in a BB would be

$$2^{k^2}$$

in the next level, and the GA fails because of insufficient BB supply. To prevent this 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 interaction of an upper level is not expressed before all BBs of the lower level are compressed. The methods are described in detail in following sections.

4.2.1 Compression Criterion

As mentioned before, a nearly converged BB is compressed to one single bit. Here, we derive a reasonable criterion to decide when to perform the compression.

Assuming that the most expressive schema in the current population is the correct one, the following derivation is based on making a good decision between the second (H_2) and the third (H_3) most expressive schemata. Note that the derivation is similar to that of the decision-making population-sizing model (Goldberg et al., 1992).

Assume that the proportions of H_2 and H_3 in the current population are p_2 and p_3, respectively, and by definition, $p_2 \geq p_3$. If the GA reaches a steady state, the proportions are of binomial distribution with n samples, where n is the population size. The probability of making an error can be calculated as:

$$\epsilon = 1 - \Phi\left(\frac{p_2 - p_3}{\sqrt{\sigma_{H_2}^2 + \sigma_{H_3}^2}}\right), \tag{10}$$

where Φ is the standard cumulative Gaussian function and $\sigma_{H_2}^2$ and $\sigma_{H_3}^2$ are variances of p_2 and p_3, respectively. Assuming steady state, the variances can be approximated as

$$\sigma_{H_2}^2 \simeq \frac{p_2(1-p_2)}{n},$$

and similarly,

$$\sigma_{H_3}^2 \approx \frac{p_3(1-p_3)}{n}.$$

Since there is no turning back once a BB is compressed, the room for decision-making error is small. Here the derivation adopts an error tolerance similar to the GA convergence condition. For a problem with m BBs, at most one BB can be wrong. Define a decision variable z by:

$$z = \frac{p_2 - p_3}{\sqrt{\sigma_{H_2}^2 + \sigma_{H_3}^2}}. \tag{11}$$

For $z \gg 1$, the decision error can be approximated as

$$\epsilon \approx \frac{1}{\sqrt{2\pi} z e^{\frac{z^2}{2}}}. \tag{12}$$

Since at most one error can occur among all m BBs, the following relation holds.

$$(1-\epsilon)^m \geq \frac{m-1}{m}. \tag{13}$$

For a large m, the above inequality can be approximated as $\epsilon \leq \frac{1}{m^2}$, which yields

$$\frac{1}{\sqrt{2\pi} z e^{\frac{z^2}{2}}} \leq \frac{1}{m^2}. \tag{14}$$

The above inequality can be solved as:

$$z \geq \sqrt{\mathbb{W}\left(\frac{m^4}{2\pi}\right)}, \tag{15}$$

where $\mathbb{W}$ is Lambert's W function. For a problem with unknown m, conservatively assuming $m = l$ yields

$$z = \sqrt{\mathbb{W}\left(\frac{l^4}{2\pi}\right)}.$$

Finally, a BB is compressed when the signal difference satisfies

$$p_2 - p_3 \geq z\sqrt{\sigma_{H_2}^2 + \sigma_{H_3}^2}, \tag{16}$$

where

$$\sigma_{H_2}^2$$

and

$$\sigma_{H_3}^2$$

can be approximated by the binomial distribution as

$$\frac{p_2(1-p_2)}{n}$$

and

$$\frac{p_3(1-p_3)}{n},$$

respectively, given the population size of n.

After the interaction information is retrieved from the DSM clustering, the GA checks if the inequality in Equation (16) is satisfied for every BB. If so, the BB is compressed to 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.

4.2.2 Interaction Detection Threshold

For hierarchical problems, a threshold is needed to prevent the upper level interactions from expressing before all BBs in the lower level are compressed.

Equation (6) gives a threshold for noise resistance that we call θ_1 in this section. If the sampled mutual information of two genes is less than θ_1, these two genes are considered to be independent, and the corresponding DSM entry is 0.

Now consider those entries where the sampled mutual information is greater than θ_1. Another threshold is required to distinguish the interactions of the current lowest level from that of all other upper levels. For a hierarchical problem, the strengths of the interactions are stronger in lower levels and weaker in upper levels. The task here is equivalent to finding the cluster with the greatest mean in a set of numbers. Since the signal strength of the interactions from upper levels is significantly weaker than that from the current lowest level, a simple k mean clustering algorithm where $k = 2$ should be able to distinguish these interactions. The splitting point from the k mean algorithm gives the desired threshold θ_2.

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

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

The DSM constructed above should mainly contain the dependency information from the current lowest level, which has the strongest dependencies by definition.

4.2.3 Putting It All Together

Recall that to conquer the hierarchical difficulty, a GA needs to maintain (1) proper decomposition at each level, (2) preservation of alternative solutions, and (3) representation of a chunk at the lower level as one single variable at the upper level. Now that all tools are available for the three key points, a GA that solves problems via hierarchical decomposition is developed on the basis of DSMGA (Yu et al., 2003a).

The proposed method utilizes DSM clustering techniques to decompose the hierarchical problem at each level. It adopts restricted tournament replacement (RTR; Harik, 1994; Pelikan and Goldberg, 2001) to preserve promising subsolutions just like hBOA does. Unlike hBOA, however, DSMGA takes advantage of the explicit interaction model to achieve an explicit chunking scheme, the substructural chromosome compression.

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

4.3 Empirical Results

The scalability of DSMGA was tested on hTrap with the result shown in Figure 9. For problems with sizes smaller than or equal to 729, the results are averaged over 30 independent runs. Due to limited computational resources, for problems with sizes greater than 729, DSMGA was performed only once on an extrapolated population size according to the results for smaller problems. The number of function evaluations of DSMGA scale as $\Theta(l^{1.56}\log(l))$ on hTrap, where l is the problem size. The scaling order is similar to that of hBOA (Pelikan, 2002) and implies that DSMGA scales subquadratically on these hierarchical problems. The results indicate that the proposed substructural chromosome compression scheme works well since it is known that the number of function evaluations scales exponentially to the problem size without a proper chunking mechanism (Pelikan, 2002).

As mentioned, one of the advantages of DSMGA over hBOA is that DSMGA adopts an explicit interaction model and delivers the problem structure in a comprehensible manner to human researchers. Figure 10 illustrates the Bayesian networks of generations 1, 3, 5, and 7 of hBOA performing on a $3 \times 3 \times 3$ hTrap. The population size is set at 1000. In generations 1 and 3, one might recognize some BBs of size 3; however, it is not obvious which genes form BBs without further analysis in generations 5 and 7. For

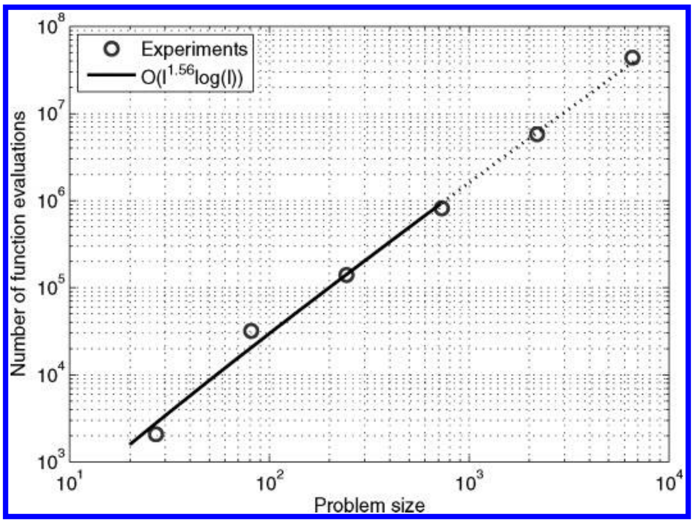


Figure 9: Scalability of DSMGA on hTrap with problem size up to 6,561.

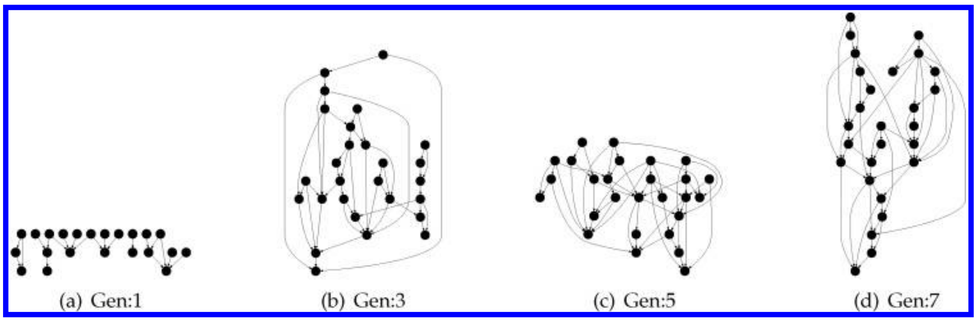


Figure 10: Interaction model of hBOA on 3 × 3 × 3 hTrap problem.

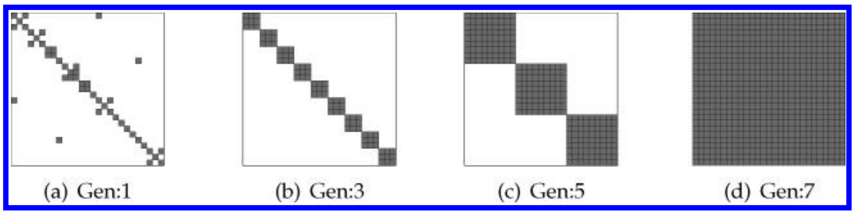


Figure 11: Interaction model of DSMGA on 3 × 3 × 3 hTrap problem.

comparison, Figure 11 shows the DSMs of DSMGA performing on the same problem with the same population size. From the figure, one can easily tell that the problem is decomposed into nine BBs of size 3 at generation 3, three BBs of size 9 at generation 5, and one BB of size 27 at generation 7.

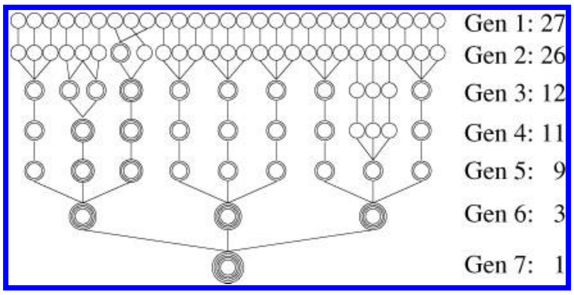


Figure 12: Problem structure obtained by DSMGA for the hTrap function with three levels and $k = 3$ (chromosome length $= 3^3 = 27$). The nodes represents genes of chromosomes. The number of circles represents the compression level. The descriptions on the right show the generation number and the chromosome length. The ideal case would be a complete 3-ary tree with three layers (Figure 8).

If we record the generations in which when BBs are compressed, we can reconstruct the problem structure, as shown in Figure 12. Despite some imperfections due to sampling noise, we can see that DSMGA rediscovers the problem structure of hTrap as a 3-ary tree with three layers.

The scalability test indicates that DSMGA scales subquadratically on the tested hierarchical problem. It also demonstrates that DSMGA is capable of capturing the problem structures for this test function. It has also been shown that DSMGA has the same capability on hIFF and hXOR (Yu and Goldberg, 2006).

The way that DSMGA optimizes problems is to automatically create a customized recombination operator (BB-wise crossover) 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. Such reusability of the DSM analysis will be demonstrated in the next section.

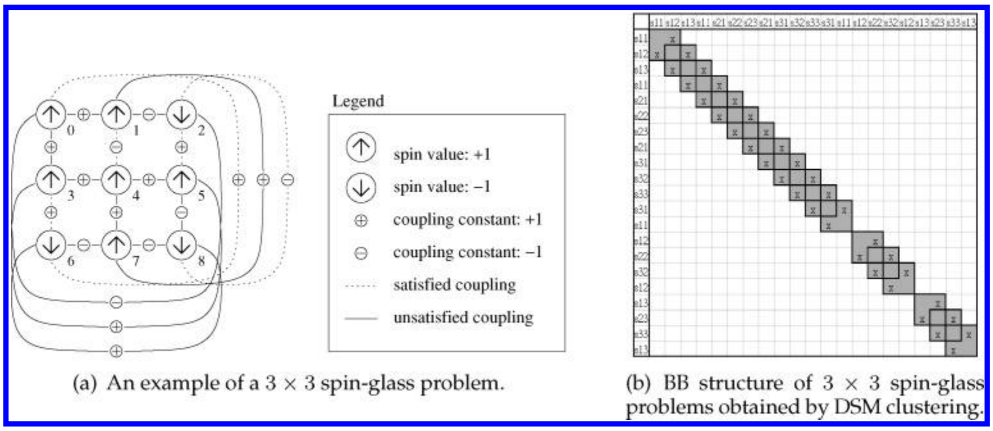
5 Finding Extrema for Problems with Overlap

This section investigates problems with overlap and attempts to develop recombination methods to conquer the overlap difficulty. In particular, a problem with numerous overlapping BBs—2D spin glasses—is investigated. This section proposes several essential modifications of the recombination method and details the ideas behind these modifications as well.

5.1 2D Ising Spin Glasses

The study of spin glasses (Fischer and Hertz, 1991) has attracted considerable attention in statistical physics and condensed matter physics. Researchers have been studying the optimization of the Ising spin-glass problem by using GAs because of interesting properties such as the symmetry, large number of local optima, scalability of the problem size, and overlapping BBs (Pelikan et al., 2004; Pelikan and Goldberg, 2003; Pelikan and Mühlenbein, 1999; Naudts and Naudts, 1998; van Hoyweghen et al., 2002; van Hoyweghen, 2001; Mühlenbein et al., 1999).

The physical state of an Ising spin-glass system is defined by (1) a set of spins $\{\sigma_0, \sigma_1, \ldots, \sigma_{l\text{-}1}\}$, where the σs are Ising variables with values -1 or $+1$, and (2) a set of



(a) An example of a 3 × 3 spin-glass problem.
(b) BB structure of 3 × 3 spin-glass problems obtained by DSM clustering.

Figure 13: 3 × 3 spin-glass problem and its BB structure.

coupling constants J_{ij} relating spins σ_i and σ_j. The simplest spin-glass Hamiltonian is defined as:

$$H(\sigma) = -\Sigma_{i,j=0}^{n-1} J_{ij}\sigma_i\sigma_j. \tag{18}$$

The objective is to find the ground state of the spin-glass system. In other words, given a set of coupling constants J_{ij}, the task is to find a set of spin values $\{\sigma_i\}$ that maximizes Equation (18).

Here the coupling constants are restricted to binary, $J_{ij} \in \{-1, +1\}$. Also, this paper only considers the two dimensional case where the spins are arranged in a 2D lattice and every spin couples with its four neighbors. To approximate the behavior of a large-scale system, the periodic boundary condition is adopted, making the 2D lattice a torus. Figure 13(a) shows an example of a 3 × 3 spin-glass system. The example contains 13 satisfied couplings and 5 unsatisfied couplings, which yields a fitness value of 8.

For the Ising spin-glass problem in its general form, a set at spin values at the ground state is known to be NP-complete (Monien and Sudborough, 1988). However, in the special case of graphs with a bounded genus, several polynomial algorithms exist, and the current best one scales as $\Theta(l^{3.5})$, where l is the problem size (Galluccio and Loebi, 1999a,b). Note that the 2D toroid version is a special (and easier) case of graphs with a bounded genus.

5.1.1 The BB Structure of the 2D Spin-Glass Problem

In this section, we analyze the interactions of 2D spin-glass problem off-line. The interaction model can be obtained from either arithmetic manipulation or off-line DSM analysis. Two reasons led to the decision to perform interaction detection off-line: (1) to focus on the recombination strategy and ignore the factor of modularity identification error, and (2) to demonstrate how to utilize DSM analysis off-line on smaller scale problems and apply the obtained knowledge to larger scale problems.

The fitness function of the spin-glass instance shown in Figure 13(a) can be expressed as:

$$H(\sigma) = \sigma_0\sigma_1 - \sigma_1\sigma_2 + \cdots. \tag{19}$$

Algorithm 1 The `minimalcut` recombination strategy.

```
1. Perform a BB identifying algorithm to capture the overlapping topology.
2. Construct a graph $G = (V, E)$ where the vertices are BBs, and the edges are
   overlapping relations between BBs. There is an edge between two vertices if and
   only if the two corresponding BBs overlap.
3. Randomly choose two nodes $n_1$ and $n_2$. Then partition the graph $G$ into two
   subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ that satisfy the following conditions:
   $n_1 \in V_1, n_2 \in V_2$, and $|E| - |E_1| - |E_2|$ is minimal.
```

If $(\sigma_0, \sigma_1) = (+1, +1)$ or $(-1, -1)$, the fitness increases by one from the first term. Similarly, $(\sigma_1, \sigma_2) = (-1, +1)$ or $(+1, -1)$ contributes one to the fitness value. According to Goldberg (2002), BBs are groups of minimal, sequential, and superior genes. Thus, σ_0 and σ_1 form one BB while σ_1 and σ_2 form another one. Therefore, every two adjacent spins form a BB in the 2D spin-glass. A 2D spin-glass problem with l spins contains $2l$ BBs; every gene belongs to four different BBs, and every BB overlaps with six other adjacent BBs.

The BB structure can also be obtained via DSM analysis. Given a 2D spin-glass problem with randomly assigned coupling constants, generate a population of chromosomes with randomly initialized spin values. Evaluate every chromosome using the fitness function, perform selection, and then create a DSM over those selected chromosomes. The average neighborhood information can be retrieved by averaging the DSMs over many spin-glass instances. Finally, experiments on applying the DSM clustering technique yielded the same BB structure (Figure 13[b]).

5.2 Trial One: `MinimalCut`

Yu et al. (2005) proposed a recombination strategy yielding minimal BB disruption that we call `minimalcut` in this paper. The `minimalcut` recombination strategy when dealing overlapping BBs is briefly described in Algorithm 1.

It has been demonstrated that `minimalcut` works well on problems with cyclicly overlapping BBs. As a first attempt, `minimalcut` is applied to the 2D spin-glass problem. In general, finding the minimal cut of a graph is NP-hard. However, for the 2D spin-glass problem, finding a minimal cut is not difficult.

Two possible ways of minimally cutting the overlapping graph of the 2D spin-glass problem are shown in Figure 14. The first cutting method, `minimalcut1`, is not too different from two-point crossover when spins are arranged in a specific order (Figure 14[c]). In `minimalcut1`, the number of BB disruptions is always $2\sqrt{l}$, and the number of spins mixed during the recombination (the effective exchange length, or *EEL*) is $\frac{l}{2}$ on average. In the second cutting method, `minimalcut2`, the number of BB disruptions is $4r$, and EEL is r^2, where r is a controllable parameter. Yu and Goldberg (2004) indicated that the number of BB disruptions needs to be less than $2I\sqrt{m}$ for a successful GA convergence, where I is the selection intensity and $m = 2l$ in this case. By constraining the number of BB disruptions to be $\sqrt{l}$, calculation shows that approximately $r = \frac{\sqrt{l}}{4}$ and $\text{EEL} = \frac{l}{16}$.

Figure 15 shows the empirical scalability test of a GA with these two cutting methods. The data points are medians of the results for 50 independent 2D spin-glass problem

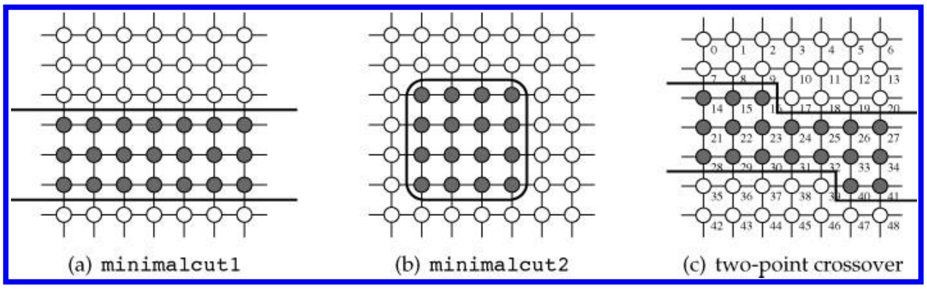


Figure 14: Two possible cutting ways for the `minimalcut` method.

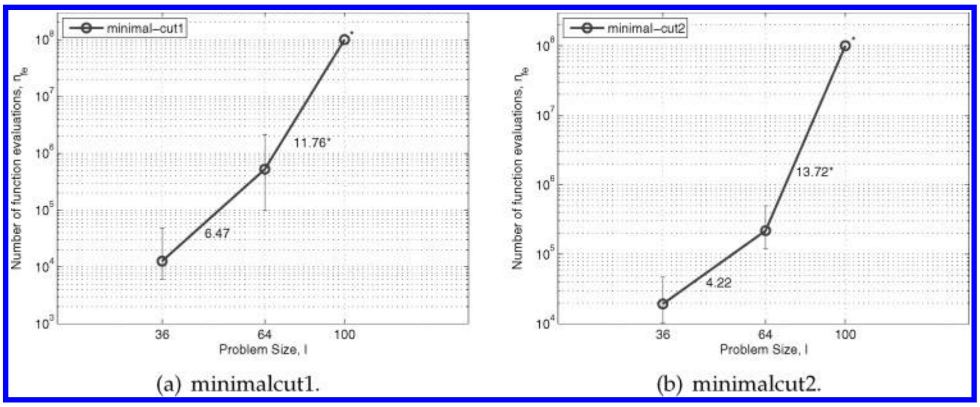


Figure 15: Scalability of `minimal-cut` on the 2D spin-glass problem.

instances with randomly initialized coupling constants. For each problem, 10 bisection runs are performed to find the minimal population size for 10 consecutive successes of finding the global optima. Using the averaged minimal population size yields the number of function evaluations that the GA needs for the 2D spin-glass problem. The lower error bar stands for the first quartile, and the upper error bar stands for the third quartile. Basically, none of these two cutting methods scales well on the 2D spin-glass problem. For most of the problems with 10×10 spins, the GA does not converge even after 10^8 function evaluations.

5.3 Trial Two: `MinimalCut` + Niching

`Minimalcut` does not scale well on the 2D spin-glass problem due to several reasons, and one of them is the lack of the ability to preserve alternative solutions.

Figure 16 shows a 2D spin-glass instance with six spins. The global optimal solution is that all spins are of the same direction, which yields six satisfied couplings and one unsatisfied coupling. Note that the local information suggests that σ_1 and σ_4 should spin in the opposite direction, which contradicts the global information. Therefore, the decision of the correct BB among all four possible combinations $(\sigma_1, \sigma_4) = (1, 1), (1, -1), (-1, 1), (-1, -1)$ cannot be made correctly in the early stage of the GA

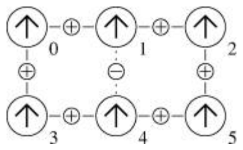


Figure 16: A 2D spin-glass problem to which the global optimal solution has one unsatisfied coupling.

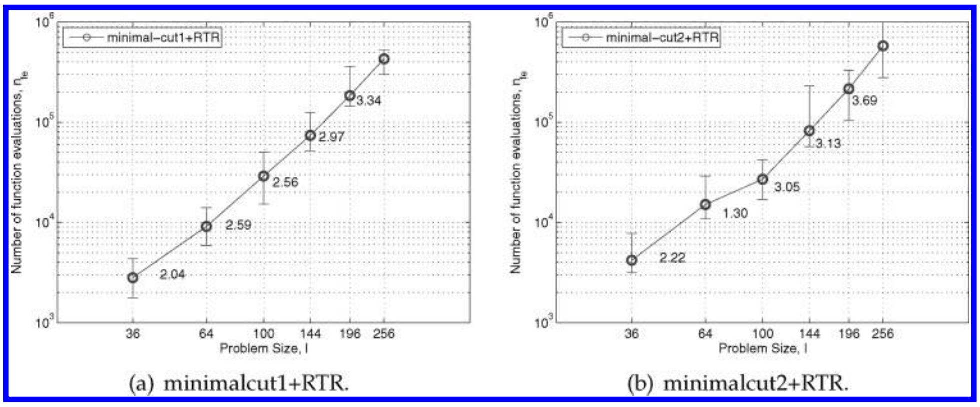


Figure 17: Scalability of `minimalcut+RTR` on the 2D spin-glass problem.

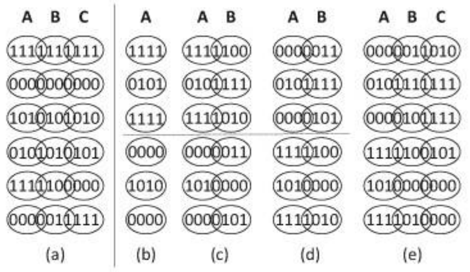
run. Before the decision is made, the GA needs to preserve alternative solutions. This observation suggests adopting niching methods in the GA.

The empirical scalability result for `minimalcut` with RTR shows a significant improvement (Figure 17). With RTR, the GA is able to find a global optimum for the problem with 16×16 spins within 10^6 function evaluations. Nevertheless, in the log-log plot, the slopes of the data points increase rapidly with problem size.

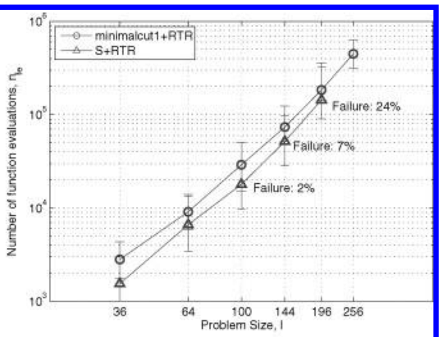
5.4 Trial Three: Sequencing + Niching

Based on the facet-wise models developed by Yu and Goldberg (2004), the problem that occurred for `minimalcut1` is due to too many BB disruptions, $\sqrt{2m}$ in total; the problem for `minimalcut2` is due to a too small EEL, $\frac{1}{16}$ of the total chromosome length, and moderate disruptions, $\sqrt{\frac{m}{2}}$ in total. Possible improvements for the recombination fall into two aspects: (1) decrease the number of disruptions by considering the alleles of the overlapped genes, and (2) every BB has different *strength*. If a certain number of BBs have to be disrupted, the weaker ones are preferred. We will define the strength later.

Suppose a problem with a population shown in Figure 18(a) has three overlapping BBs with no cycle. Recombination can be performed by considering the alleles of those overlapped genes to avoid disruptions. First, partition the population by the alleles of the overlapped genes. In the example, BB_A overlaps BB_B with one gene, and therefore, partition the population into two sets where the overlapped gene of the chromosomes has a value of 0 in one set and a value of 1 in the other set. Now that the allele of the



(a) Non-disruptive population wise crossover with non-cyclic overlaps.



(b) Scalability of S+RTR on the 2D spin-glass problem.

Figure 18: Sequencing and niching.

overlapped gene is coherent in both sets, population-wise shuffling can be performed between BB_A and BB_B within each set without disruption. Then a similar procedure is carried out to shuffle BB_B and BB_C.

This modification should have a similar effect, if not better, as in the work of Tsuji et al. (2006), where they adopt pair-wise crossover and treat two BBs as nonoverlapping when the overlapped genes of the BBs have the same alleles. The crossover operator presented here examines the whole population to find those chromosomes with overlapped genes having the same alleles, and then performs a population-wise shuffling on those chromosomes, which does not cause any disruption when the overlap is noncyclic.

With this modification, however, disruptions still occur when the overlap is cyclic. The appropriate strategy is to preserve *stronger* BBs and disrupt *weaker* ones. Here, the strength of a BB is defined by how severe the disruption can be if the BB is crossed somewhere in the middle. Entropy has shown to be a decent indicator for the disruptions (Harik, 1999; Pelikan et al., 1999). Define the strength of a BB as:

$$Stength(\vec{x}) = \Sigma_i Entropy(x_i) - Entropy(\vec{x}), \tag{20}$$

where $\vec{x}$ is a BB, x_i is the ith gene of the BB, and $Entropy(\vec{x})$ is the joint entropy of the BB.

The nondisruptive population-wise shuffling starts from the strongest BB. Among the BBs that overlap with the first BB, the strongest is chosen as the next candidate. The procedure continues until the value of every gene is decided. Note that those BBs that cause a cycle are discarded. The crossover topology is a tree where every node has only one single parent. In a 2D spin-glass problem with l spins and $2l$ BBs, $(l-1)$ stronger BBs are preserved while $(l+1)$ weaker BBs are disrupted.

Figure 18(b) shows the scalability results of the `sequencing` recombination method with RTR (`S+RTR`) on the 2D spin-glass problems. For comparison, the result of `minimalcut1+RTR` is also plotted in the figure. `S+RTR` consumes a smaller number of function evaluations than `minimalcut1+RTR` does, although the slope increases in a similar trend. One important thing to note is that when the problem size becomes larger, more often the GA with `S+RTR` cannot find any global optima within 10^8 function evaluations.[3]

[3]The failure rates are shown in Figure 18(b).

Algorithm 2 The S+W recombination algorithm.

```
1. $S_{\text{decided}} = \phi$.
2. Randomly select a gene $x$ from the strongest BB. Calculate $Pr(x)$ from the parent
   population. The value of gene $x$ of chromosomes in the offspring population is
   decided by sampling $Pr(x)$. $S_{\text{decided}} \leftarrow S_{\text{decided}} \cup x$.
3. Select the strongest BB among all the BBs where some genes are in $S_{\text{decided}}$
   and some other genes are not. $index = \text{argmax}_i Strength(\{BB_i | \exists x, x \in S_{\text{decided}}, x \in BB_i \text{ and } \exists y, y \notin S_{\text{decided}}, y \in BB_i\})$.
4. Randomly select a gene $y$ from $BB_{\text{index}}$.
5. Create a set of parents of $y$. $\mathcal{P} = \{p_i | p_i \in S_{\text{decided}} \text{ and } \exists BB, p_i \in BB, y \in BB\}$.
6. Calculate the conditional probability $Pr(y|\mathcal{P})$ from the parent population. If the
   allele pattern of $\mathcal{P}$ does not exist in the parent population, use the marginal
   probability $Pr(y)$ instead.
7. The value of gene $y$ of chromosomes in the offspring population is decided by
   sampling $Pr(y|\mathcal{P})$ or $Pr(y)$.
8. $S_{\text{decided}} \leftarrow S_{\text{decided}} \cup y$. Repeat steps 3 to 8 until all genes are in $S_{\text{decided}}$.
```

5.5 Trial Four: Sequencing + Well-Informed Decision + Niching

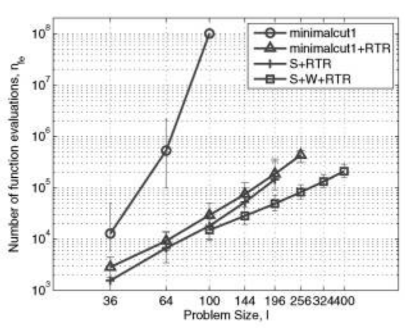
S+RTR does not yield satisfactory scalability due to BB disruptions, even though disruptions only happen to weaker BBs. As indicated before, in the 2D spin-glass problem, every gene belongs to four different BBs. Among these four BBs, S+RTR preserves at most two while it disrupts at least two others.

To avoid disrupting too many BBs, the recombination needs to incorporate more information when deciding the values of genes. Ideally, the recombination method should decide the allele of gene x_i by utilizing the information of all the genes of the BBs containing x_i. For instance, if genes x and p_j form a BB for $j = 1, 2, 3, 4$, the allele of gene x should be decided by sampling from the conditional probability $Pr(x|p_1, p_2, p_3, p_4)$.

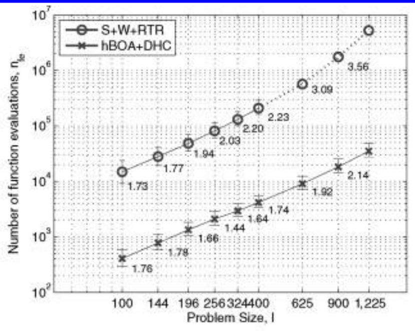
A recombination method is constructed based on the above concept. The pseudocode of the algorithm, called the `sequencing + well-informed decision` recombination algorithm (S+W), is shown in Algorithm 2.

S+W+RTR shows a significant improvement in terms of number of function evaluations consumed and the scaling order (Figure 19[a]). To obtain a more accurate comparison, S+W+RTR is performed on 1,000 independent problem instances for each different problem size. Within 2×10^5 function evaluations, S+W+RTR is able to solve problems up to 20×20 spins.

The scaling order for S+W+RTR still increases with the problem size (Figure 19[b]), while the increasing rate is much slower than exponential. On the contrary, hBOA+HDC (Pelikan and Goldberg, 2003) consumes a small number of function evaluations and scales better. The following discusses possible reasons and remedies. In reviewing the crossover shown in Figure 18(a), there are different recombination probabilities between the BBs. For example, the probabilities that $BB_A = 1111$ recombines with $BB_B = \{1111, 1010, 1100, 0000, 0101, 0011\}$ can be calculated as $\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0, 0\}$. Similarly, the recombination probabilities for $BB_A = 1111$ and $BB_C = \{0000, 0101, 0000, 1111, 1010, 1111\}$) is $\{\frac{2}{9}, \frac{2}{9}, \frac{2}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}\}$. The above calculations indicate that BB_C has a more uniformly



(a) Comparison of `minimalcut`, `minimalcut+RTR`, `S+RTR`, and `S+W+RTR` on the 2D spin-glass problem. `S+W+RTR` outperforms others in terms of number of function evaluations consumed and the scaling order.



(b) However, S+W+R does not scale polynomially. The slopes (noted on the figure) slowly increase with the problem size. For problems larger than 20x20, only one problem instant is tested due to computational power limitation. On the other hand, hBOA+DHC scales better.

Figure 19: The performance of S+W+RTR.

distributed probability to recombine with BB_A than BB_B does. For the `sequencing` recombination, the root has the most uniformly distributed recombination probability with the furthest leaf nodes. In order to have a more uniformly distributed recombination probability among BBs, a longer path in the sampling sequence is preferred. However, note that all the conditional probabilities are merely estimations. A longer path also amplifies estimation errors. A metric is needed to balance the trade-off; however, the investigation is beyond the scope of this paper.

5.6 Discussion of Results

From the series of experiments, three keys to conquer overlap difficulty can be recognized:

1. **Preservation of Alternative Solutions.**
2. **Proper Sequencing.** Recombination should start from the strongest BB, and disrupt weaker BBs if necessary.
3. **Well-Informed Decision.** When weaker BBs have to be disrupted, its alleles should be decided by all its neighbors.

This paper does not yet fully answer the question of how to achieve proper sequencing. Nevertheless, it acknowledges that a proper sequencing should respect the strengths of BBs and nondeterministic mixing. If a recombination algorithm respects these key issues, it scales much better than traditional crossover operators.

Combined with local searchers, hBOA solves the spin-glass problem in polynomial time with an order that competes the current best algorithm. Compared with the proposed method in Figure 19(b), hBOA does scale better. However, the reason for hBOA's success is not fully understood.

With the crossover modifications in this section, DSMGA was also able to solve 2D spin-glasses up to 1,225 spins within $O(l^{3.6})$ function evaluations. Experiments in this paper suggest that hBOA's success may rely on adopting RTR and Bayesian networks. RTR preserves alternative solutions, while Bayesian networks sample alleles in a proper sequence and make well-informed decisions.

6 Summary and Conclusions

This paper develops advanced GA techniques for problems with modularity, hierarchy, and overlap. It first motivates the importance of problem decomposition for solving large-scale problems arising from complex systems. Interaction detection using a mutual information metric emerges as the main mechanism for problem decomposition. Inspired by organization theory, this paper utilizes the DSM clustering technique to identify modularity for GAs. Combined with BB-wise crossover, DSMGA is constructed to solve boundedly difficult problems with nonoverlapping modules within subquadratic number of function evaluations. DSMGA is further extended with an explicit chunking scheme to solve problems via hierarchical decomposition. Through a series of experiments on the 2D spin-glass problem, this paper recognizes keys to conquer overlap difficulty, and the arguments are empirically verified.

Understanding problem structures and decomposing complex systems lay the foundation of this paper. Compared to existing GAs, the work presented in this paper distinguishes itself in two respects: (1) it is capable of handling problems with modularity, hierarchy, and limited overlap, and (2) it adopts explicit interaction models.

Among many other GAs, only hBOA is capable of handling problems with all three types of interaction. However, the interaction model in hBOA is opaque to users. On the contrary, the interaction model (DSM) in this paper is transparent to users. In many real-world applications, the explicit knowledge of the problem structure is as valuable as finding a high-quality solution to the problem. Further, as presented in this paper, the explicit interaction model can be used to develop advanced techniques.

Compared with hBOA, DSMGA delivers explicit interaction models to users. However, as a trade-off, hBOA has better performance than DSMGA while dealing with problems with many overlapping modules. It has been shown that DSMGA scales subquadratically for problems with circularly overlapping traps (Yu et al., 2005), which is a problem with limited overlap; however, DSMGA fails to scale polynomially for the 2D spin-glass problem, which is a problem with overwhelming overlap. On the contrary, hBOA with a local searcher solve 2D spin glasses up to 1,225 spins within $O(l^{2.2})$ function evaluations ($O[l^{3.6}]$ for DSMGA), where l is the number of spins. As mentioned in Section 5.6, although we identified three keys to solve a problem with overlap as (1) preservation of alternative solutions, (2) proper sequencing, and (3) well-informed decision, we did not yet fully answer in this paper the question of how to achieve proper sequencing.

Acknowledgments

This work was sponsored by Taiwan National Science Council under grant NSC97-2218-E-002-020-MY3, U.S. Air Force Office of Scientific Research, Air Force Material Command, USAF, under grants FA9550-06-1-0370 and FA9550-06-1-0096, U.S. National Science Foundation under CAREER grant ECS-0547013, ITR grant DMR-03-25939 at Materials Computation Center, grant ISS-02-09199 at US National Center for Supercomputing Applications, UIUC, and the Portuguese Foundation for Science and Technology under grants SFRH/BD/16980/2004 and PTDC/EIA/67776/2006.

References

Barron, A., Rissenen, J., and Yu, B. (1998). The MDL principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760.

Chen, Y.-P., and Goldberg, D. E. (2005). Convergence time for the linkage learning genetic algorithm. *Evolutionary Computation*, 13(3):279–302.

Christopher, A. (1964). *Notes on the synthesis of form*. Boston, MA: Harvard Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to algorithms* (2nd ed.). Cambridge, MA: MIT Press.

Deb, K., and Goldberg, D. E. (1993). Analyzing deception in trap functions. In *Foundations of Genetic Algorithms 2*, pp. 93–108.

DeJong, E. D., Thierens, D., and Watson, W. (2004). Hierarchical genetic algorithms. In *Parallel Problem Solving from Nature (PPSN VIII)*, pp. 232–241.

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

Eppinger, S. D., Whitney, D. E., Smith, R. P., and Gebala, D. A. (1994). A model-based method for organizing tasks in product development. *Research in Engineering Design*, 6(1):1–13.

Fernandez, C. (1998). Integration analysis of product architecture to support effective team co-location. Masters thesis, Massachusetts Institute of Technology, Cambridge.

Fischer, K. H., and Hertz, J. A. (1991). *Spin glasses*. New York: Cambridge University Press.

Galluccio, A., and Loebi, M. (1999a). A theory of Pfaffian orientations. I. Perfect matchings and permunents. *Electronic Journal of Combinatorics*, 6(1):Research Paper 6.

Galluccio, A., and Loebi, M. (1999b). A theory of Pfaffian orientations. II. t-joins, k-cuts, and duality of enumeration. *Electronic Journal of Combinatorics*, 6(1):Research Paper 7.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In *Genetic Algorithms and Simulated Annealing*, Chap. 6 (pp. 74–88) Pitman Publishing, London.

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

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

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

Goldberg, D. E., Deb, K., Kargupta, H., and Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 56–64.

Goldberg, D. E., Korb, B., and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530.

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

Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Doctoral dissertation, University of Michigan, Ann Arbor.

Harik, G. R. (1999). Linkage learning via probabilistic modeling in the ecGA. IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Urbana, IL.

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

Harik, G. R., Lobo, F. G., and Goldberg, D. E. (1998). The compact genetic algorithm. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 523–528.

Haynes, T. (1997). Phenotypical building blocks for genetic programming (pp. 26–33). San Mateo, CA: Morgan Kaufmann.

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

Holland, J. H. (1995). *Hidden order—How adaptation builds complexity*. Reading, MA: Perseus Books.

Holland, J. H. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary Computation*, 8(4):373–391.

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

Iclanzan, D., and Dumitrescu, D. (2007). Overcoming hierarchical difficulty by hill-climbing the building block structure. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007)*, pp. 1256–1263.

Kargupta, H. (1996). The performance of the gene expression messy genetic algorithm on real test functions. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pp. 631–636.

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

Koza, J. (1994). *Genetic programming II: Automatic discovery of reusable programs*. Cambridge, MA: MIT Press.

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

Larrañaga, P., and Lozano, J. (Eds.). (2002). *Estimation of distribution algorithms*. Boston, MA: Kluwer Academic Publishers.

Lutz, R. (2002). Recovering high-level structure of software systems using a minimum description length principle. In *Proceedings of the 13th Irish International Conference on Artificial Intelligence and Cognitive Science (AICS)*, pp. 61–69.

McCord, K., and Eppinger, S. D. (1993). Managing the integration problem in concurrent engineering. Working Paper 3594, MIT Sloan School of Management, Cambridge, Massachusetts.

Miller, B. L. (1997). *Noise, sampling, and efficient genetic algorithms*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana.

Mitchell, M. (1996). *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.

Monien, B., and Sudborough, I. H. (1988). MIN-CUT is NP-complete for edge weighted trees. *Theoretical Computer Science*, 58(1–3):209–229.

Mühlenbein, H., Mahnig, T., and Rodriguez, A. O. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5:215–247.

Munetomo, M., and Goldberg, D. E. (1999). Identifying linkage groups by nonlinearity/non-monotonicity detection. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, 1:433–440.

Naudts, B., and Naudts, J. (1998). The effect of spin-flip symmetry on the performance of the simple GA. *Parallel Problem Solving from Nature*, pp. 67–76.

Pelikan, M. (2002). *Bayesian optimization algorithm: From single level to hierarchy*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana.

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

Pelikan, M., and Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 511–518.

Pelikan, M., and Goldberg, D. E. (2003). Hierarchical BOA solves Ising spin glasses and MAXSAT. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, pp. 1271–1282.

Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, I:525–532.

Pelikan, M., and Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In *Proceedings of Advances in Soft Computing—Engineering Design and Manufacturing*, pp. 521–535.

Pelikan, M., Ocenasek, J., Trebst, S., Troyer, M., and Alet, F. (2004). Computational complexity and simulation of rare events of Ising spin glasses. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, pp. 36–47.

Pelikan, M., Sastry, K., and Cantú-Paz, E. (Eds.). (2006). *Scalable optimization via probabilistic modeling from algorithms to applications*. New York: Springer-Verlag.

Pimmler, T., and Eppinger, S. D. (1994). Integration analysis of product decompositions. In *Proceedings of the ASME International Conference on Design Theory and Methodology*, DE-68:343–351.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465–471.

Rissanen, J. (1999). Hypothesis selection and testing by the MDL principle. *Computer Journal*, 42:260–269.

Russell, S., and Norvig, P. (2003). Planning. In *Artificial intelligence: A modern approach* (2nd ed.), Chap. 11–12 (pp. 375–461). Upper Saddle River, NJ: Prentice Hall.

Sastry, K., O'Reilly, U., Goldberg, D., and Hill, D. (2003). Building block supply in genetic programming. In R. L. Riolo and B. Worzel (Eds.), *Genetic programming theory and practice*, Chap. 9 (pp. 137–154). Boston, MA: Kluwer.

Sharman, D., and Yassine, A. (2004). Characterizing complex product architectures. *Systems Engineering Journal*, 7(1):35–60.

Sharman, D., Yassine, A., and Carlile, P. (2002). Architectural optimization using real options theory and dependency structure matrices. In *Proceedings of the ASME 28th Design Automation Conference*, pp. DAC–34119.

Simon, H. A. (1968). *The science of artifical*. Cambridge, MA: MIT Press.

Smith, J., and Fogarty, T. C. (1996). Recombination strategy adaptation via evolution of gene linkage. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pp. 826–831.

Thebeau, R. (2001). Knowledge management of system interfaces and interactions for product development processes. Masters thesis, Massachusetts Institute of Technology, Cambridge.

Thierens, D., and Goldberg, D. E. (1993). Mixing in genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 38–45.

Toussaint, M. (2005). Compact genetic codes as a search strategy of evolutionary processes. In *Foundations of Genetic Algorithms (FOGA 2005)*, pp. 75–94.

Tsuji, M., Munetomo, M., and Akama, K. (2006). A crossover for complex building blocks overlapping. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006)*, pp. 1337–1344.

van Hoyweghen, C. (2001). Detecting spin-flip symmetry in optimization problems. In *Theoretical aspects of evolutionary computing* (pp. 423–437). New York: Springer-Verlag.

van Hoyweghen, C., Goldberg, D. E., and Naudts, B. (2002). From TwoMax to the Ising model: Easy and hard symmetrical problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pp. 626–633.

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

Watson, R. A., and Pollack, J. B. (1999). Hierarchically consistent test problems for genetic algorithms: Summary and additional results. In *Late Breaking Papers at the Genetic and Evolutionary Computation Conference*, pp. 292–297.

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

Whitfield, R., Smith, J., and Duffy, A. (2002). Identifying component modules. In *Proceedings of the Seventh International Conference on Artificial Intelligence in Design AID02*, pp. 571–592.

Yassine, A., Jogleker, N., Braha, D., Eppingher, S., and Whitney, D. (2003). Information hiding in product development: The design churn effect. *Research in Engineering Design*, 14(3):145–161.

Yu, T.-L., and Goldberg, D. E. (2004). Quality and efficiency of model building for genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pp. 367–378.

Yu, T.-L., and Goldberg, D. E. (2006). Conquering hierarchical difficulty by explicit chunking: Substructural chromosome compression. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006)*, pp. 1385–1392.

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

Yu, T.-L., Sastry, K., and Goldberg, D. E. (2005). Linkage learning, overlapping building blocks, and systematic strategy for scalable recombination. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)*, pp. 1217–1224.

Yu, T.-L., Yassine, A., and Goldberg, D. E. (2003b). A genetic algorithm for developing modular product architectures. In *Proceedings of the ASME 2003 International Design Engineering Technical Conferences*, pp. DTM–48657.

Yu, T.-L., Yassine, A., and Goldberg, D. E. (2007). An information theoretic method for developing modular architectures using genetic algorithms. *Research in Engineering Design*, 18:91–109.

This article has been cited by:

1. Amir H Gandomi, Kalyanmoy Deb, Ronald C Averill, Shahryar Rahnamayan, Mohammad Nabi Omidvar. 2019. Using semi-independent variables to enhance optimization search. *Expert Systems with Applications* **120**, 279-297. [Crossref]
2. Michał Witold Przewoźniczek, Krzysztof Walkowiak, Arunabha Sen, Marcin Komarnicki, Piotr Lechowicz. 2019. The transformation of the k-Shortest Steiner trees search problem into binary dynamic problem for effective evolutionary methods application. *Information Sciences* **479**, 1-19. [Crossref]
3. Chung-Lun Li, Nicholas G. Hall. 2019. Work Package Sizing and Project Performance. *Operations Research* **67**:1, 123-142. [Crossref]
4. Oana Vuculescu. 2017. Searching far away from the lamp-post: An agent-based model. *Strategic Organization* **15**:2, 242-263. [Crossref]
5. Yuan Sun, Michael Kirley, Saman K. Halgamuge. 2017. Quantifying Variable Interactions in Continuous Optimization Problems. *IEEE Transactions on Evolutionary Computation* **21**:2, 249-264. [Crossref]
6. Michal Witold Przewozniczek, Krzysztof Walkowiak, Michal Aibin. 2017. The evolutionary cost of Baldwin effect in the routing and spectrum allocation problem in elastic optical networks. *Applied Soft Computing* **52**, 843-862. [Crossref]
7. Kei Ohnishi, Chang Wook Ahn. Simple Linkage Identification Using Genetic Clustering 373-384. [Crossref]
8. Peng Yang, Ke Tang, Xin Yao. 2017. Turning High-dimensional Optimization into Computationally Expensive Optimization. *IEEE Transactions on Evolutionary Computation* 1-1. [Crossref]
9. Michal Przewozniczek. 2016. Active Multi-Population Pattern Searching Algorithm for flow optimization in computer networks – The novel coevolution schema combined with linkage learning. *Information Sciences* **355-356**, 15-36. [Crossref]
10. Xiaoliang Ma, Fang Liu, Yutao Qi, Xiaodong Wang, Lingling Li, Licheng Jiao, Minglei Yin, Maoguo Gong. 2016. A Multiobjective Evolutionary Algorithm Based on Decision Variable Analyses for Multiobjective Optimization Problems With Large-Scale Variables. *IEEE Transactions on Evolutionary Computation* **20**:2, 275-298. [Crossref]
11. Chalermsub Sangkavichitr, Prabhas Chongstitvatana. 2016. The use of explicit building blocks in evolutionary computation. *International Journal of Systems Science* **47**:3, 691-706. [Crossref]
12. Samin Shokri, Carl T. Haas, Ralph C. G. Haas, Sang Hyun Lee. 2016. Interface-Management Process for Managing Risks in Complex Capital Projects. *Journal of Construction Engineering and Management* **142**:2, 04015069. [Crossref]
13. Zhao Wang, Maoguo Gong, Tian Xie. Decision Variable Analysis Based on Distributed Computing 447-455. [Crossref]
14. Yiqiao Cai, Jiahai Wang. 2015. Differential evolution with hybrid linkage crossover. *Information Sciences* **320**, 244-287. [Crossref]
15. Eman Sayed, Daryl Essam, Ruhul Sarker, Saber Elsayed. 2015. Decomposition-based evolutionary algorithm for large scale constrained problems. *Information Sciences* **316**, 457-486. [Crossref]
16. Alberto Borboni. 2015. Optimization of Radial Cams with Translating Roller Follower through Genetic Algorithms. *Applied Mechanics and Materials* **783**, 83-94. [Crossref]
17. Xiang-wei Zheng, Dian-jie Lu, Xiao-guang Wang, Hong Liu. 2015. A cooperative coevolutionary biogeography-based optimizer. *Applied Intelligence* **43**:1, 95-111. [Crossref]

18. Shih-Huan Hsu, Tian-Li Yu. Optimization by Pairwise Linkage Detection, Incremental Linkage Set, and Restricted / Back Mixing 519-526. [Crossref]
19. Martin Pelikan, Mark W. Hauschild, Fernando G. Lobo. Estimation of Distribution Algorithms 899-928. [Crossref]
20. Mohammad Nabi Omidvar, Xiaodong Li, Yi Mei, Xin Yao. 2014. Cooperative Co-Evolution With Differential Grouping for Large Scale Optimization. *IEEE Transactions on Evolutionary Computation* **18**:3, 378-393. [Crossref]
21. Simon Li, Li Chen. 2014. Identification of Clusters and Interfaces for Supporting the Implementation of Change Requests. *IEEE Transactions on Engineering Management* **61**:2, 323-335. [Crossref]
22. B. Petraus, R. Arnold, R. Riedel, E. Mueller. Matrices-based modeling of communication within planning projects 1248-1252. [Crossref]
23. Alexander E. I. Brownlee, John A. W. McCall, Qingfu Zhang. 2013. Fitness Modeling With Markov Networks. *IEEE Transactions on Evolutionary Computation* **17**:6, 862-879. [Crossref]
24. Yixin Li, Nan Ren, Mengwan Cao. 2013. WBS Data Analysis Method Based on Information Supply Chain. *Journal of Applied Sciences* **13**:12, 2355-2358. [Crossref]
25. Kai-Chun Fan, Tian-Li Yu, Jui-Ting Lee. 2013. Linkage learning by number of function evaluations estimation: Practical view of building blocks. *Information Sciences* **230**, 162-182. [Crossref]
26. Amin Nikanjam, Adel Rahmani. 2012. Exploiting Bivariate Dependencies to Speedup Structure Learning in Bayesian Optimization Algorithm. *Journal of Computer Science and Technology* **27**:5, 1077-1090. [Crossref]
27. Simon Li, Mehrnaz Mirhosseini. 2012. A matrix-based modularization approach for supporting secure collaboration in parametric design. *Computers in Industry* **63**:6, 619-631. [Crossref]
28. Simon Li. 2011. A matrix-based clustering approach for the decomposition of design problems. *Research in Engineering Design* **22**:4, 263-278. [Crossref]
29. Claudio F. Lima, Fernando G. Lobo, Martin Pelikan, David E. Goldberg. 2011. Model accuracy in the Bayesian optimization algorithm. *Soft Computing* **15**:7, 1351-1371. [Crossref]
30. Yoshitaka Kameya, Chativit Prayoonsri. Pattern-based preservation of building blocks in genetic algorithms 2578-2585. [Crossref]
31. Daniel Selva, Edward F. Crawley. Exploring packaging architectures for the Earth Science Decadal Survey 1-13. [Crossref]
32. Martin Pelikan. Genetic Algorithms . [Crossref]
33. Kai- Fan, Jui-Ting Lee, Tian-Li Yu, Tsung-Yu Ho. Interaction-detection metric with differential mutual complement for dependency structure matrix genetic algorithm 1-8. [Crossref]
34. Amin Nikanjam, Hadi Sharifi, B. Hoda Helmi, Adel Rahmani. Enhancing the efficiency of genetic algorithm by identifying linkage groups using DSM clustering 1-8. [Crossref]