Automatic Clustering of Gene Ontology by Genetic Algorithm

Razib M. Othman, Safaai Deris, Rosli M. Illias, Zalmiyah Zakaria, and Saberi M. Mohamad

***Abstract*—**Nowadays, Gene Ontology has been used widely by many researchers for biological data mining and information retrieval, integration of biological databases, finding genes, and incorporating knowledge in the Gene Ontology for gene clustering. However, the increase in size of the Gene Ontology has caused problems in maintaining and processing them. One way to obtain their accessibility is by clustering them into fragmented groups. Clustering the Gene Ontology is a difficult combinatorial problem and can be modeled as a graph partitioning problem. Additionally, deciding the number *k* of clusters to use is not easily perceived and is a hard algorithmic problem. Therefore, an approach for solving the automatic clustering of the Gene Ontology is proposed by incorporating cohesion-and-coupling metric into a hybrid algorithm consisting of a genetic algorithm and a split-and-merge algorithm. Experimental results and an example of modularized Gene Ontology in RDF/XML format are given to illustrate the effectiveness of the algorithm.

***Keywords*—**Automatic clustering, Cohesion-and-coupling metric, Gene Ontology; Genetic algorithm, Split-and-merge algorithm

I. Introduction

THE Gene Ontology (GO) [1] is an effort done by The Gene Ontology Consortium (www.geneontology.org) to define consistent terminology that describe the attributes of biological process, cellular component, and molecular function of a gene product. The intention of GO is to share common understanding of the meaning of any term used, and therefore could support the database query tool to find functionally equivalent terms in cross-database search. In essence, this will improve retrieval consistency across resources and the recall and precision of the query result within resources.

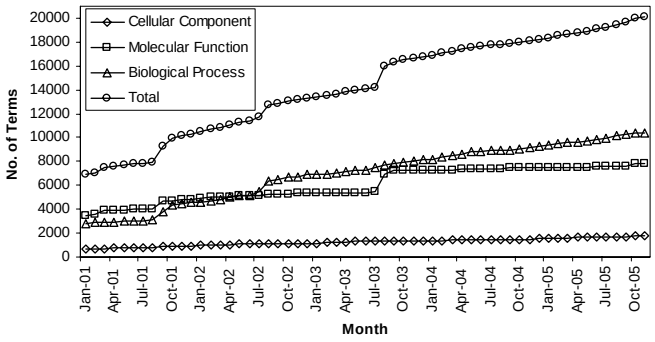


Fig. 1 Growth of GO terms
(source: http://www.geneontology.org/MonthlyReports/)

In conjunction with rapid progress in bioinformatics field, an increasing number of terms are being generated in the GO, see Fig. 1. This is due to the attempt to standardize as many terms as possible in different repositories for plant, animal, and microbial genomes such as The Arabidopsis Information Resource (TAIR)–database for the brassica family plant *Arabidopsis thaliana*, Rat Genome Database (RGD)–database for the rat *Rattus norvegicus*, and GeneDB protozoa–databases for *Plasmodium falciparum*, *Leishmania major*, *Trypanosoma brucei*, and several other protozoan parasites. At this time, the GO contains about 20,069 terms and 29,102 relationships between the terms (as of November 5, 2005). These terms are associated with 1.65 million gene products, 0.23 million amino acid sequences, and 0.25 million species. The high dimension of the GO instances and its monolithic character has caused its maintenance and processing more difficult and challenging.

Therefore, in this study, a hybrid approach consisting of the genetic algorithm and split-and-merge algorithm is applied to automatically cluster the GO terms into smaller and highly intra-related clusters. The hybrid genetic algorithm used software engineering measurements, the cohesion-and-coupling metric, to quantify the quality of clustering (QOC), see (6)–(9). The idea of using these metrics are to produce good clusters by maximizing the degree of interaction between terms in a cluster (high cohesion) and also

Manuscript received December 18, 2005. This work is supported by the Malaysian Ministry of Science, Technology, and Innovation (MOSTI) in part under Intensification of Research in Priority Areas (IRPA) grant (project no. 04-02-06-0057-EA001) and in part under Short Term Research (STR) grant (project no. 75162).

Razib M. Othman is with the Faculty of Computer Science and Information System, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, MALAYSIA (corresponding author; phone: 607-5532358; fax: 607-5565044; e-mail: razib@fsksm.utm.my).

Safaai Deris is with the School of Graduate Studies, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, MALAYSIA (e-mail: safaai@fsksm.utm.my).

Rosli M. Illias is with the Faculty of Chemical and Natural Resources Enginneering, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, MALAYSIA (e-mail: r-rosli@utm.my).

Zalmiyah Zakaria is with the Faculty of Computer Science and Information System, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, MALAYSIA (e-mail: zalmiyah@fsksm.utm.my).

Saberi M. Mohamad is with the Faculty of Computer Science and Information System, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, MALAYSIA (e-mail: saberi@fsksm.utm.my).

minimizing the degree of interaction between terms in different clusters (low coupling). The genetic algorithm is chosen due to its efficient navigation through large search space and good performance as stochastic search procedure. It is used to generate potential clusters by applying standard crossover and mutation operator, together with enforcing the cohesion-and-coupling metric into fitness function. Then, the split-and-merge algorithm is implemented to efficiently estimate the number k of clusters. Learning the k is achieved by the split-and-merge algorithm based on the cohesion-and-coupling metric by improving any infeasible clusters. Furthermore, parallelization of genetic algorithm based on coarse-grained (island) model [2] is considered to reduce time complexity.

Recently, there has been an increasing awareness of the benefits of the GO RDF/XML for biological data mining and information retrieval, integration of biological databases, finding genes, and incorporating knowledge in the GO for gene clustering. But the size and massive nature of the GO RDF/XML cause problems that affect maintaining, publishing, validating, and processing the GO instances. This is due to the fact that the ontology as a whole is too large to handle. Therefore, the purpose of this study is to partition the GO RDF/XML into a set of more accessible and understandable modules. By modularizing this single monolithic file into smaller files will enable amino acid sequences and IEA (Inferred from Electronic Annotation) evidence associations to be included into the GO RDF/XML. With these additions, it would complete and cohere the GO RDF/XML file. Thus, the GO RDF/XML will be more processable and exchangeable by software agent or other machine-readable meta-data.

This paper is arranged as follows. The second section begins with the problem description of clustering the GO terms. The third section discusses related work in the clustering area. The fourth section explains the flow of the proposed genetic algorithm. The fifth section details the split-and-merge algorithm for discovering an appropriate k. The sixth section describes the parallelization process of the hybrid genetic algorithm. The seventh section presents the experimental results of clustering the GO terms and the modularized semantic web of the GO RDF/XML format. Some discussions and the conclusion of the paper are included in the final section.

II. Statement of the Problem

Automatic clustering is a process of dividing a set of objects into unknown groups, where the best number k of groups is determined by the clustering algorithm. That is, objects within each group should be highly similar to each other than to objects in any other group. Finding the k automatically is a hard algorithmic problem. The automatic clustering problem can be defined as follows:

Let $X = \{X_1, X_2, \ldots, X_n\}$ be a set of n objects. These objects are clustered into non-overlapping groups $C = \{C_1, C_2, \ldots, C_k\}$, where C is called a cluster, k is the unknown number of clusters, $C_i \cap C_j = \emptyset$ for $i \neq j$, $C_1 \cup C_2 \cup \ldots \cup C_k = X$, $C_i \subseteq X$, and $C_i \neq \emptyset$.

In the GO context, the GO terms are structured as Directed Acyclic Graph (DAG). Let GO graph $G = \{V, E\}$, where V is a set of nodes that represent the GO terms and E is a set of directed edges that represent relationships between the GO terms. Clustering the GO graph can be considered as a Graph Partitioning Problem (GPP). The aim of GPP is to cut a vertex set V into k disjoint and non-empty subsets such that the number of edges connecting nodes in different subsets is minimized and the number of edges connecting the nodes in the same subsets is maximized. GPP is a fundamental combinatorial optimization problem that has numerous practical applications in many areas including design of VLSI circuits [3], mesh partitioning in parallel processing [4], image segmentation in computer vision [5], and gene expression analysis in bioinformatics [6].

To partition the GO graph, the following questions need to be answered:

1) What is the most suitable clustering algorithm to find the optimal solution of the GPP, and that offers reasonable amount of execution time to this NP-complete problem?
2) What is the precise criterion for discovering the number k of clusters and for measuring the goodness of the clusters?

In this paper, the first question is answered by aggregating split-and-merge algorithm, which consists of two steps, into the parallel genetic algorithm. At first, the entire node is decomposed into a number of clusters using the split algorithm. These clusters are then automatically combined using the merge algorithm in several iterations until the suitable number k of clusters is obtained. On the other hand, the cohesion-and-coupling metric is used to answer the second question.

III. Related Work

The clustering problem is omnipresent in many fields of science and engineering. It has been solved by various techniques such as k-means [7], genetic algorithm [8], self-organizing map [9], fuzzy c-means [10], and particle swarm optimization [11]. Survey of clustering techniques can be found in [12]–[14]. Recently, the increasing amount of data has made the number k of clusters difficult to guess, and the value supplied by the user based on prior knowledge, presumptions, and practical experiences is often inaccurate. Therefore, reasonable ways of identifying the number k of clusters automatically is required to avoid trial-and-error work. Lately, several techniques have been proposed to determine the number k of clusters. Most of the techniques are wrapped around k-means or genetic algorithm. Split and/or merge rules are the most famous wrapper methods to increase or decrease the number k of clusters while the algorithm continues. Among these techniques are:

1) X-means [15]; in this the splitting decision is performed

by computing the Bayesian Information Criterion (BIC) until the upper bound of k is attained.

2) G-means [16]; it starts with small number of k-means centers and raises the number of centers using Gaussian distribution.
3) CLUSTERING [17]; it is an automatic clustering based on heuristic strategy that uses the nearest neighbor to group those data that are situated close to one and another. Then, genetic algorithm is used to group the smaller clusters into larger ones.
4) Genetic Clustering Algorithm (GCA) [18]; it is basically composed of two steps. First, the data set is divided into a number of clusters using Cluster Decomposition Algorithm (DCA) and at the second step, Hierarchical Cluster Merging Algorithm (HCMA) is used to combine the clusters automatically.
5) S+G [19]; it is also a two stage method, which in the beginning uses a self-organizing feature map to determine the number k of clusters and then employs a genetic algorithm based clustering to find the final solution.

In the case of the GPP, an extensive study of Kerninghan-Lin algorithm, simulated annealing, tabu search, watermarking, and normalized cut have been carried out by [20]–[23], [5] respectively. Review of the GPP techniques can be found in [24], [25]. Several studies using genetic algorithm for the GPP have also been done by:

1) Bui and Moon [26] introduced a schema of preprocessing phase before the initialization of population to ameliorate the quality of the chromosome. The different classes of graphs: random graph, random geometric graph, random regular graph, and caterpillar graph consisting of 134 to 5,252 nodes, were tested with the algorithm.
2) Kaveh and Bondarabady [27] implemented genetic algorithm for finite element decomposition of 1,640 to 6,720 elements. Sequences of coarsening and uncoarsening process are performed to transform the large scale graph G_0 into a smaller size graph G_n and vice versa such that a suitable size of graph can be partitioned by genetic algorithm.
3) Kohmoto *et al.* [28] has incorporated simulated annealing into genetic algorithm to generate feasible solutions. The algorithm is then applied to undirected graph with 124 to 250 nodes.

For the ontology clustering or semantic web modularization, very little effort has been done in this area. Stuckenschmidt and Klein [29] have proposed a method for automatic partitioning of large ontologies based on the structure of the class hierarchy. The method consists of three steps:

1) In the first step, a dependency graph is created from ontology source file using PROLOG-based tool that reads OWL and RDF schema files. It then displays the dependency graph using networks analysis tool Pajek.
2) In the second step, the strength of the dependencies between the concepts in the dependency graph is determined by computing the propositional strength network.
3) In the third step, an island algorithm is used to determine the modules existing in the dependency graph.

IV. Proposed Hybrid Genetic Algorithm

The hybrid genetic algorithm can be initialized with $k_{\min}$ minimum number of clusters that needs to be provided by the user and a DAG graph with i number of nodes and j number of directed edges, where $i, j, k_{\min} \in \{1, 2, \ldots, n\}$. Ab initio, the algorithm starts with initializing few parameters, such as number of generations $t_{\max}$, size of population ps, crossover probability p_c, and mutation probability p_m which can be modified by the user. The subsequent steps in the algorithm can be described as follows:

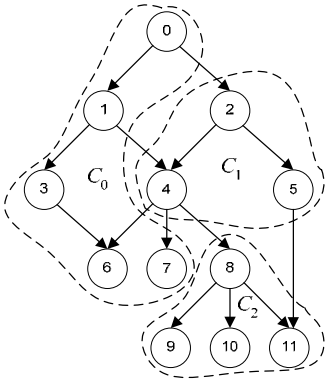
1) Set iteration $t = 0$. Encode the DAG $G = \{V, E\}$ using a cluster-number (see discussion on the chromosome representation) schema and generate the initial chromosomes $x_1^0 \ldots x_{ps}^0$ of population $P(0)$ randomly where the value of genes are between $[1 \ldots k]$. Then, evaluate the fitness for each chromosome $x^0 \in P(0)$ using the fitness function $f(x^0)$ based on the cohesion-and-coupling metric (see discussion on the fitness function).
2) If $t > t_{\max}$, then terminate the process, decode the best chromosome $x_{\max} \in P$, and display the clustering C. Otherwise, go to step 3.
3) Increment $t = t + 1$. Create a new population by selecting good chromosomes from old population (iteration $t - 1$).
4) Perform crossover between two chromosomes x_a^t, $x_b^t \in P(t)$ with probability p_c and then mutate each gene in a single chromosome $x^t \in P(t)$ with probability p_m.
5) Perform split function $S(x^t)$ to increase the k and then decrease the k using merge function $M(x^t)$ for each chromosome $x^t \in P(t)$ such that cohesion score α is maximized and coupling score β is minimized (see discussion on the split-and-merge algorithm).
6) Evaluate the fitness for each chromosome $x^t \in P(t)$ using the fitness function $f(x^t)$ and go to step 2.

A. Chromosome Representation

A good chromosome representation is crucial to the convergence velocity of the hybrid genetic algorithm and the quality of the solution obtained. Therefore, the cluster-number scheme is used to ensure that the gene values can be simply assigned and interpreted even for large graphs. In addition, it makes it more possible to relate each chromosome to a solution for the GPP. The cluster-number scheme represents a clustering of n objects as an array of n integers where the value at ith subscript denotes the cluster number which holds the ith object.

To partition the DAG graph, the graph is represented by a single chromosome using 1D array of integers as follows:

1) Genes are integer values that represent the cluster number that each particular node belongs to.
2) Loci are mapped to the node number.



$C_0 = \{n_0, n_1, n_3, n_6, n_7\}$

$C_1 = \{n_2, n_4, n_5\}$

$C_2 = \{n_8, n_9, n_{10}, n_{11}\}$

Fig. 2 Example graph G_1

node	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	loci
cluster	0	0	1	0	1	1	0	0	2	2	2	2	gene

Fig. 3 Chromosome representation of the graph G_1

3) Chromosome length is the number of nodes in the graph.

Edges between nodes are input to the algorithm as a $n \times 2$ matrix, with n rows corresponding to number of edges and 2 columns associated with a pair of nodes. Fig. 3 shows a chromosome representation of the graph G_1 (see Fig. 2) with 12 nodes and 3 clusters.

B. Reproduction

During the reproduction phase, two classical and most often-used genetic operators are employed, i.e., the crossover and the mutation operators. These operators are chosen due to their effectiveness with the 1D array of integers representing a chromosome and the cohesion-and-coupling metric based fitness function. The crossover operator creates new offspring by combining features of their parents. In the meantime, the mutation operator arbitrarily alters one or more genes produced from the crossover process. The reason for using these operators in the hybrid genetic algorithm is to generate new population with higher total fitness in each generation.

Although such operators are effective, the resulting solutions do not guarantee feasibility. In order to increase the feasibility and optimality of the solution, the offsprings go through alteration process by the split function $S(x)$ and then the merge function $M(x)$ after every reproduction by the genetic operators. The transformation is based on a cluster-by-cluster basis by making modification in a single chromosome ($S(x)$, $M(x) : x \rightarrow x'$), which is then evaluated by the fitness function $f(x')$. Even though the purpose of these functions are to determine the best number k of clusters, indirectly the solutions will be improved and be repaired by shifting to a better neighbor solution until no improvement can be made. The split function $S(x)$ and the merge function $M(x)$ are discussed elaborately in the split-and-merge algorithm section.

C. Fitness Function

The optimization of the GPP can be stated as optimizing a function f that partitions the graph G into k subgraphs G_1, G_2, …, G_k, where k is the best value which generates highly cohesive clusters. On the dot, the main objective of partitioning the DAG graph is to find feasible and near-optimal solution that maximizes the preference for cohesion between nodes in a cluster and minimizes the preference for coupling between different clusters.

The cohesion α_i of the cluster i of the DAG graph can be calculated by:

$$\alpha_i = \frac{\mu_i}{\frac{N_i(N_i-1)}{2}} \tag{1}$$

where N_i is the number of nodes in the cluster i and μ_i is the number of its internal edges.

The coupling $\beta_{i,j}$ between clusters i and j is given by:

$$\beta_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \dfrac{\varepsilon_{ij}}{N_i N_j} & \text{if } i \neq j \end{cases} \tag{2}$$

where N_i and N_j are number of nodes in the clusters i and j respectively and ε_{ij} is the number of edges from cluster i to cluster j.

The initial fitness function $f^0(x)$ of the DAG graph partitioning is measured by constituting a trade-off between cohesion score α and coupling score β. This trade-off is computed by subtracting the average cohesion from the average coupling. The initial fitness function $f^0(x)$ is given as follows:

$$f^0(x) = \begin{cases} \dfrac{\sum_{i=1}^{k} \alpha_i}{k} - \dfrac{\sum_{i,j=1}^{k} \beta_{i,j}}{\frac{k(k-1)}{2}} & \forall k > 1 \\ \alpha_i & k = 1 \end{cases} \tag{3}$$

The values of the initial fitness function $f^0(x)$ vary between [-1…1]. A good quality cluster has a high value of $f^0(x)$. However, to ensure the algorithm obtains a balanced clustering, standard deviation of dependency index $stdev(\gamma)$ is considered, see (5). Therefore, a feasible and near-optimal solution is searched by maximizing the result of subtracting the standard deviation of the dependency index $stdev(\gamma)$ from the initial fitness function $f^0(x)$:

$$f(x) = f^0(x) - stdev(\gamma) \tag{4}$$

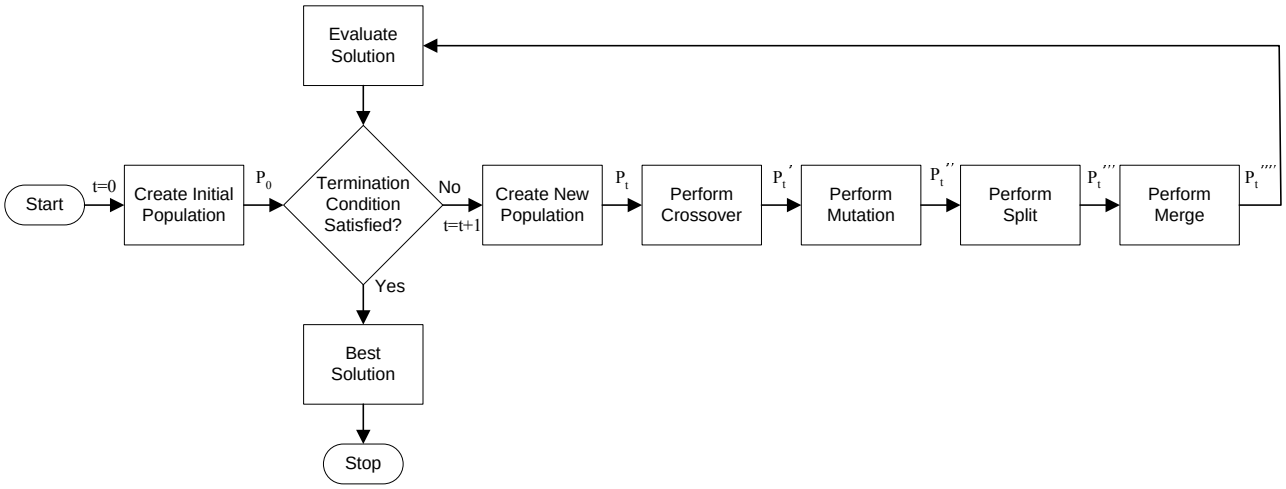


Fig. 4 Flowchart of the proposed hybrid genetic algorithm

```
Algorithm Split (x);
Input: x (a chromosome)
Output: x′ (a modified chromosome)
begin
  for p := 1 to x.NoOfClusters() do
    x_split := x;
    for q := 1 to x_split.Length() do
      if x_split.Gene(q) > p then x_split.Gene(q) := x_split.Gene(q) + 1; end-if
    end-for
    for q := 1 to s do
      x_q := x_split;
      for r := 1 to x_q.Length() do
        if x_q.Gene(r) = p then x_q.Gene(r) := Random(p, p + 1); end-if
      end-for
      if x_q.QOC(C_p, C_p+1) > x.QOC(C_p) and x_q.DependencyIndex(C_p) >
        I_min and x_q.DependencyIndex(C_p+1) > I_min
        then x := x_q; p := p + 1; end-if
    end-for
  end-for
end
```

Fig. 5 Function for splitting clusters

```
Algorithm Merge (x, k_min);
Input: x (a chromosome), k_min (a minimum number of clusters)
Output: x′ (a modified chromosome)
begin
  n := x.NoOfClusters();
  if n ≠ 1 then
    for p := 1 to n do
      for q := p + 1 to n do
        if x.NoOfClusters() > k_min and x.Coupling(p, q) ≠ 0 then
          x_merge := x;
          for r := 1 to x_merge.Length() do
            if x_merge.Gene(r) = q then x_merge.Gene(r) := p; end-if
          end-for
          if x_merge.QOC(C_p) > x.QOC(C_p, C_q) and
            x_merge.DependencyIndex(C_p) < I_max
            then x := x_merge; end-if
        end-if
      end-for
    end-for
  end-if
end
```

Fig. 6 Function for merging clusters

V. The Split-and-Merge Algorithm

By the embedment of the split-and-merge algorithm into the genetic algorithm, the k value which is held by each gene in the chromosomes will be refined and fixed. Through this method, chromosomes with best number k of clusters and high fitness are reproduced in each generation. Hence, it eliminates the process of producing solutions with unsuitable number k of clusters and accelerate the pace for convergence. The detailed steps of these algorithms are shown in Fig. 4, Fig. 5, and Fig. 6. After undergoing the repairing process, any illegal chromosome will be adjusted and then be evaluated by the fitness function $f(x)$. The illegal chromosome represents a partition in which some clusters are empty. For example, given $k = 3$, the chromosome $x = (1\ 1\ 3\ 1\ 3\ 3)$ is illegal because cluster number two is empty.

Definition 1. Legal and Illegal Chromosome. Given a chromosome $x = g_1, g_2, \ldots, g_n$, let $e(x)$ be the number of nonempty clusters in x divided by k, $e(x)$ is called legality ratio. The chromosome x is legal if $e(x) = 1$ and illegal otherwise.

Unfortunately, in some cases the repairing process can cause clusters to further split or merge due to strong internal dependencies. This phenomenon creates unbalanced subgraphs and reflects the aim of creating modular ontology. Therefore, dependency index γ is introduced to stabilize the split-and-merge algorithm and to forbid it from producing micro or giant clusters during splitting or merging process. The dependency index γ_i of the cluster i is given by:

$$\gamma_i = \frac{N_i - k}{\sum_{j=1}^{k} N_j - 1} \tag{5}$$

The target value for dependency index γ_i of a cluster i is 0. The maximum value is 1 which represents the worst case where most of the nodes form a large cluster. Meanwhile, negative value indicates pathological clusters with undersized

number of nodes.

A. Dividing of Clusters with Split Algorithm

The main objective of the split function $S(x)$ is to decompose each cluster in chromosome x into reasonable fragmented clusters. Detailed split function $S(x)$ is shown in Fig. 5. This function works by creating clone chromosomes $x_1^c \ldots x_n^c$ from the chromosome $x \in P(t)$. For each cluster $C_1 \ldots C_p$ in the clone chromosome x^c, divide the cluster C_p into two clusters C_p and C_{p+1}. The chromosome x will be replaced by the best clone chromosome x^c that satisfies the following criteria:

1) The QOC of the clusters C_p and C_{p+1} in the clone chromosome x^c is higher than the QOC of the cluster C_p in the chromosome x.
2) The dependency index γ of the clusters C_p and C_{p+1} in the clone chromosome x^c must be greater than the dependency index threshold for small cluster I_{min}.

The QOC of the clusters C_p and C_{p+1} in the clone chromosome x^c is computed as follows:

$$x^c.QOC(C_p, C_{p+1}) = \frac{\sum_{i=p}^{p+1} \alpha_i}{2} - \frac{\sum_{i=p,j=1}^{p+1,k} \beta_{i,j}}{2k-3} \tag{6}$$

The QOC of the cluster C_p in the chromosome x is calculated with the following equation:

$$x.QOC(C_p) = \alpha_p - \frac{\sum_{i=p,j=1}^{p,k} \beta_{i,j}}{k-1} \tag{7}$$

B. Combining of Clusters with Merge Algorithm

The merge function $M(x)$ is carried out to merge the isolated clusters by repairing genes in the chromosome x when necessary. The goal is to guarantee that all the chromosomes repaired by the split function $S(x)$ are genuinely fit to be feasible and near optimal solution. As shown in Fig. 6, the merge function $M(x)$ is invoked to combine clusters C_p and C_q in the chromosome $x \in P(t)$. If the trial consolidation fulfills the following conditions, then permanently merge clusters C_p and C_q:

1) The QOC of the merged clusters C_p and C_q is higher than the QOC of the cluster C_p as alone.
2) The dependency index γ of the merged clusters C_p and C_q must be less than the dependency index threshold for large cluster I_{max}.

The QOC of the cluster C_p in the chromosome x is computed by (8) as shown below:

$$x.QOC(C_p) = \alpha_p - \frac{\sum_{i=p,j=1}^{p,k} \beta_{i,j}}{k-2} \tag{8}$$

The QOC of the merged clusters C_p and C_q in the chromosome x is calculated as follows:

$$x.QOC(C_p, C_q) = \frac{\alpha_p + \alpha_q}{2} - \frac{\sum_{i=p,j=1}^{p,k} \beta_{i,j} + \sum_{i=q,j=1}^{q,k} \beta_{i,j}}{2k-3} \tag{9}$$

VI. The Parallelization Process

When the hybrid genetic algorithm is employed to cluster the GO, it becomes computationally intensive. This is due to the fact that the GO graph have a large number of nodes and many directed edges. In addition, it demands a multitude of chromosomes and many generations of population in order to obtain good solutions. This scenario becomes deteriorated when population for each generation is required to go through the reproduction process on which the crossover, mutation, split, and merge functions, as shown in Fig. 4, are applied.

To resolve this problem, an efficient and affordable parallel hybrid genetic algorithm is developed by exploiting the advantages of island model. It is implemented on a low-cost PC cluster using message passing interface libraries. Island model is used to dissever the single large population into a number of subpopulations in order to allow each subpopulation to evolve their solutions autonomously. This parallelization model is chosen since it permits each subpopulation to be assigned to each processor of the low-cost PC cluster. Therefore, the computation load can be shared among processors, and it indirectly reduces the computation time. Moreover, the inter-processor communication between processors is lessened because the interaction happens when some chromosomes are migrated from one subpopulation to another. The migration process is done by moving a number of emigrants from the source subpopulation to replace the worst chromosomes in the target subpopulation. The emigrant is randomly selected among the best chromosomes in the source subpopulation. The parallelization of the hybrid genetic algorithm can be explained as follows:

1) Set global iteration $t = 0$. Encode the DAG $G = \{V, E\}$ and generate the initial population $P(0)$ of random chromosomes $x_1^0 \ldots x_{ps}^0$.
2) Divide the population $P(0)$ into nsp number of subpopulations $SP_0(0) \ldots SP_{nsp-1}(0)$.
3) Distribute the subpopulations $SP_0(0) \ldots SP_{nsp-1}(0)$ to $nproc$ number of processors $Proc_0 \ldots Proc_{nproc-1}$, one processor is assigned to one subpopulation. For each processor $Proc_n$, $\forall n \in \{0, 1, \ldots, nproc\text{-}1\}$, set local iteration $t = 0$ and perform local computation (step 4 to 11).
4) Evaluate the fitness for each chromosome $x^0 \in SP_n(0)$ using the fitness function $f(x^0)$.
5) If local $t > t_{\max}$, then terminate the process on the processor $Proc_n$, decode the best chromosome $x_{\max} \in SP_n$, and display the clustering C of the subpopulation SP_n. Otherwise, go to step 6.
6) Increment local $t = t + 1$. Create a new subpopulation

$SP_n(t)$ by selecting good chromosomes from old subpopulation $SP_n(t-1)$.

7) Perform crossover between two chromosomes x_a^t, $x_b^t \in SP_n(t)$ with probability p_c and then mutate each gene in a single chromosome $x^t \in SP_n(t)$ with probability p_m.
8) Perform split function $S(x^t)$ to increase the k and then decrease the k using merge function $M(x^t)$ for each chromosome $x^t \in SP_n(t)$.
9) Compute the fitness for each chromosome $x^t \in SP_n(t)$ by applying the fitness function $f(x^t)$.
10) If $t = t_M$, where t_M is an isolation time to perform the migration operation at every M generation, then select a target subpopulation SP_{target} and replace i number of worst chromosomes in the target subpopulation $a_1 \ldots a_i$ with j number of best chromosomes from this subpopulation $b_1 \ldots b_j$, where a is $x_{\min} \in SP_{target}$, b is $x_{\max}^t \in SP_n(t)$, and $i = j$.
11) Proceed to step 5.

VII. Computational Results

The parallel hybrid genetic algorithm discussed in the previous section has been tested using GO data in MySQL format as released on November 2005 (available online at www.godatabase.org/dev/database/archive/). The algorithm is implemented by enhancing the GAlib C++ libraries [30]. The basic information of the GO graph is shown in Table 1. There are 20,069 nodes representing the GO terms and 29,102 directed edges corresponding to the relationships between the terms.

The parameters used to run the parallel hybrid genetic algorithm are shown in Table 2. The computer used is a low-cost PC cluster, HP d530 with 25 processors. Each processor is assigned to one subpopulation consisting of 4 chromosomes. The low-cost PC cluster is implemented using MPICH libraries [31] developed by Argonne National Laboratory under Fedora Core 2 running on Pentium 4 processor 2.8 GHz, 512 MB RAM, and 100 Mbps NIC.

The evolution of the 25 subpopulations is shown in Fig. 7. The stability of the parallel hybrid genetic algorithm can be seen in Table 3 and Fig. 8, where results of 5 separate runs are compared by taking the best individual from the 25 subpopulations in each run. The convergence appeared as early as after 230 generations. The optimal value of the fitness function is in the interval 130.5×10^{-6} to 135.4×10^{-6}. The time taken varied from $152.8\text{s} \times 10^3$ to $231.9\text{s} \times 10^3$. The clustering utilization is depicted in Fig. 9, where the range of the dependency index $\Re(\gamma_l - \gamma_s)$ is between 0.005 and 0.008.

To test the consistency of the number of clusters found by the parallel hybrid genetic algorithm, different minimum numbers of clusters $k_{\min}$ are given to the algorithm as shown in Table 4. The results show that if the minimum number of clusters $k_{\min}$ provided by the user is greater than the best number k of clusters, then the number of clusters found is bound to it.

TABLE I
Basic Information of GO Graph

Items	Data
Number of nodes	20,069
Number of directed edges	29,102

TABLE II
Parameters of Parallel Hybrid Genetic Algorithm

Items	Parameter
Number of population	100
Number of generation	400
Crossover probability	0.8
Mutation probability	0.01
Size of genome	20,069
Replacement percentage	0.5
Type of crossover	Partial match crossover
Type of mutation	Swap mutation
Type of genetic algorithm	Steady-state genetic algorithm
Scaling	Sigma truncation scaling
Fitness function	Maximizing preferences
Minimum number of clusters	5
Number of clone chromosomes	5
Dependency index threshold for small cluster	0.1
Dependency index threshold for large cluster	0.3
Number of subpopulations	25
Isolation time	10 generations
Number of emigrants	1
Type of replacement	Bad by best
Type of migration	Stepping stone

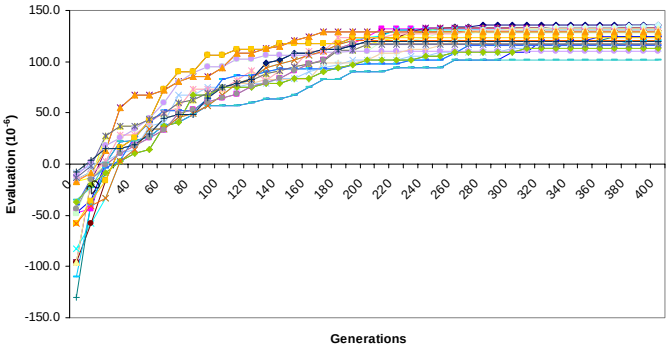


Fig. 7 Evolution of 25 subpopulations

In order to assess the performance of the parallel hybrid genetic algorithm, its behavior is compared with the parallel standard genetic algorithm. The results are shown in Table 5, where $k = 5$ is examined. The integration of the split-and-merge algorithm into the genetic algorithm produced higher optimal value and resolved the hard algorithmic problem in estimating the number k of clusters. Due to additional processing requirements to filter the chromosomes in the population in order to find the best number k of clusters, the results in Table 5 show an increase of CPU time for the parallel hybrid genetic algorithm. The clustering utilization between these algorithms can be found in Fig. 10. The results

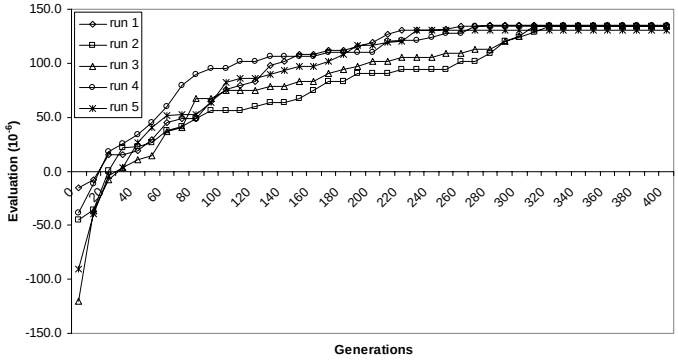


Fig. 8 Evolution of 5 runs

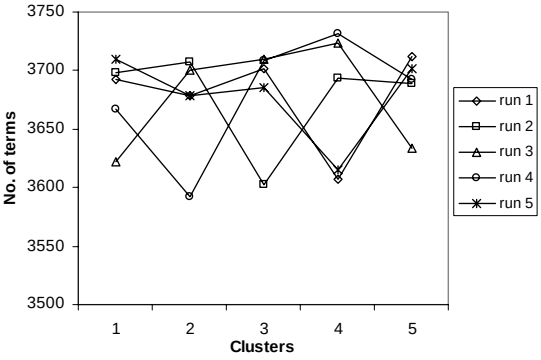


Fig. 9 Clustering utilization of 5 runs

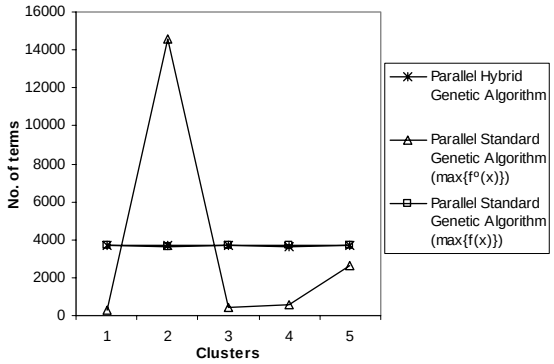


Fig. 10 Clustering utilization of parallel hybrid genetic algorithm and parallel standard genetic algorithm

show that the standard deviation of the dependency index *stdv*(γ) plays an important part to create a balanced clustering.

Fig. 11 shows an example of GO:0006631 that includes GO:0019752 from the cluster C_0 (line 8) and GO:0044255 from the cluster C_1 (line 9). The figure also depicts the encompassment of amino acid sequence (line 13-27) of IPR006180 from the InterPro database and IEA evidence association (line 30-39) with gene BC4V2_0_00031. The example shows that by modularizing the monolithic GO

TABLE III
RESULTS OF FIVE RUNS

Items	Run 1	Run 2	Run 3	Run 4	Run 5
CPU time (seconds 10^3)	199.9	231.9	215.1	188.7	152.8
Number of generation to converge	280	320	310	270	230
Number of clusters found	5	5	5	5	5
Maximum value of fitness function (10^{-6})	135.4	133.9	134.5	134.3	130.5
Dependency index for the largest cluster γ_l	0.202	0.201	0.202	0.203	0.201
Dependency index for the smallest cluster γ_s	0.196	0.196	0.197	0.195	0.196
Range $\Re(\gamma_l - \gamma_s)$	0.006	0.005	0.005	0.008	0.005

TABLE IV
NUMBER OF CLUSTERS FOUND BY PARALLEL HYBRID GENETIC ALGORITHM

Minimum number of clusters	CPU time (seconds 10^3)	Number of generation to converge	Number of clusters found	Maximum value of fitness function (10^{-6})
1	210.4	310	5	130.9
2	203.6	310	5	131.6
3	202.5	300	5	131.8
4	197.9	280	5	133.1
5	199.9	280	5	135.4
6	200.7	270	6	138.4
7	205.8	280	7	161.0
8	298.0	320	8	185.2
9	331.5	320	9	190.8
10	353.2	330	10	217.4

TABLE V
COMPARISON OF PARALLEL HYBRID GENETIC ALGORITHM WITH PARALLEL STANDARD GENETIC ALGORITHM

Items	Parallel Hybrid Genetic Algorithm ($k_{min} = 5$)	Parallel Standard Genetic Algorithm ($k = 5$ and $\max\{f^0(x)\}$)	Parallel Standard Genetic Algorithm ($k = 5$ and $\max\{f(x)\}$)
CPU time (seconds 10^3)	199.9	106.2	125.2
Number of generation to converge	280	340	340
Number of clusters found	5	-	-
Maximum value of fitness function (10^{-6})	135.4	4048.5	98.1
Dependency index for the largest cluster γ_l	0.202	0.793	0.202
Dependency index for the smallest cluster γ_s	0.196	0.014	0.196
Range $\Re(\gamma_l - \gamma_s)$	0.006	0.779	0.006

RDF/XML file, the smaller GO RDF/XML files can be easily maintained and made more thoroughgoing.

VIII. CONCLUSION

The aim of this work is to automatically partition the humongous GO RDF/XML file into smaller files in order to

```
<go:term rdf:about="http://www.geneontology.org/go#GO:0006631"
n_associations="0">
  <go:accession>GO:0006631</go:accession>
  <go:name>fatty acid metabolism</go:name>
    <go:definition>The chemical reactions involving fatty acids, aliphatic
    monocarboxylic acids liberated from naturally occurring fats and oils by
    hydrolysis.</go:definition>
  <go:is_a rdf:resource="&cluster0;http://www.geneontology.org/go#GO:0019752"
/>
  <go:is_a rdf:resource="&cluster1;http://www.geneontology.org/go#GO:0044255"
/>
  <go:dbxref rdf:parseType="Resource">
    <go:database_symbol>InterPro</go:database_symbol>
      <go:reference>IPR006180</go:reference>
      <go:sequence>AEYLRLPHSLAMIRLCNPPVNAISPTVITEVRNGLQKASL
      DHTVRAIVICGANDNFCAGADIHGFKSPTGLTLGSLVDEIQRYQKPVVA
      AIQGVALGGGLELALGCHYRIANAKARVGFPEVMLGILPGARGTQLLP
      RVVGVPVALDLITSGRHISTDEALKLGILDVVVKSDPVEEAIKFAQTVIG
      KPIEPRRILNKPVPSLPNMDSVFAEAIAKVRKQYPGRLAPETCVRSVQA
      SVKHPYEVAIKEEAKLFMYLRGSGQARALQYAFFAEKSANKWSTPSG
      ASWKTASAQPVSSVGVLGLGTMGRGIAISFARVGIPVVAVESDPKQLD
      TAKKIITSTLEKEASKSGQASAKPNLRFSSSTKELSSVDLVIEAVFEDMN
      LKKKVFAELSALCKPGAFLCTNTSALDVDDIASSTDRPQLVIGTHFFSP
      AHIMRLLEVIPSRYSSPTTIATVMSLSKRIGKIGVVVGNCYGFVGNRML
      APYYNQGYFLIEEGSKPEGVDGVLEEFGFRMGPFRVSDLAGLDVGWK
      VRKGQGLTGPSLPPGTPTRKRGNTRYSPIADMLCEAGRFGQKTGKGW
      YQYDKPLGRIHKPDPWLSEFLSQYRETHHIKQRSISKEEILERCLYSLINE
      AFRILEEGMAASPEHIDVIYLHGYGWPRHVGGPMYYAASVGLPTVLEK
      LQKYYRQNPDIPQLEPSDYLRRLVAQGSPPLKEWQSLAGPHSSKL</go:s
      equence>
  </go:dbxref>
  <!-- more dbxref -->
  <go:association rdf:parseType="Resource">
    <go:gene_symbol>BC4V2_0_00031</go:gene_symbol>
    <go:type>gene</go:type>
    <go:datasource>DDB</go:datasource>
    <go:evidence>IEA</go:evidence>
      <go:full_name>P45954 Acyl-CoA dehydrogenase, short/branched chain
      specific, mitochondrial precursor (EC 1.3.99.-) (SBCAD) (2-methyl branched
      chain acyl-CoA dehydrogenase) (2-MEBCAD) (2-methylbutyryl-coenzyme A
      dehydrogenase) (2-methylbutyryl-CoA dehydrogenase).</go:full_name>
  </go:association>
  <!-- more association -->
</go:term>
```

reduce difficulties in maintaining, publishing, validating, and processing them. This study has shown that clustering the GO can be modeled as the GPP. The model is then solved by a parallel hybrid of the genetic algorithm and the split-and-merge algorithm. The genetic algorithm is used to find a combination of node-cluster and the split-and-merge algorithm is applied to build a feasible clustering and also to automatically search for the most suitable number k of clusters. During the clustering process, the algorithm has employed cohesion-and-coupling metric as criterion to discover the best number k of clusters and to measure the quality of the clusters. The dependency index γ is then introduced to prevent the algorithm from producing problematic clusters with either undersized or oversized number of nodes. Since clustering the GO involves large graph and demands high computing resources, the island model is incorporated into the algorithm. The message passing interface libraries are used as the parallel programming interface and the algorithm is executed on a low-cost PC cluster.

Unlike any other graph partitioning algorithms, the proposed algorithm with the split-and-merge strategy can automatically find the appropriate number k of clusters. Moreover, compared to other automatic clustering algorithms, the proposed algorithm is capable of generating balanced subgraphs and does not rely on distance calculations to measure the strength between cluster centroid to each object. Furthermore, users are allowed to set the minimum number of clusters they wish to maintain and supply the dependency index threshold in order to control the size of the clusters. In fact, the algorithm does not require modifications to the components of the genetic algorithm and the split-and-merge algorithm procedures. Thus, its design is generic and domain independent. Consequently, the algorithm can be fitted to different sorts of problems with minimum changes as long as the problems can be modeled as the GPP.

The experimental results show that the algorithm is effective, stable, and thus, it requires reasonable amount of execution time. In fact, the parallelization process can be implemented with minimum hardware specifications. The proposed algorithm is also capable of finding near-optimal solution among the feasible solutions. Possible directions for further research would be on including the functional interactions between GO terms and on developing a component-based GO. At present, we may be able to get more meaningful and reusable clusters. In future, the clustering results will be applied for predicting protein functions according to the GO information. Moreover, the research will be continued on developing techniques for retrieval and classification of the GO.

REFERENCES

[1] The Gene Ontology Consortium, "Gene ontology: tool for the unification of biology," *Nat. Genet.*, vol. 25, no. 1, pp. 25-29, May 2000.

[2] M. Kaneko, M. Miki, and T. Hiroyasu, "A parallel genetic algorithm with distributed environment scheme," in *Proc. 4th. Int. Conf. Parallel & Distributed Processing Techniques & Applications*, Las Vegas, NV, 2000, pp. 619-625.

[3] S. Dutt and W. Deng, "Probability-based approaches to VLSI circuit partitioning," *IEEE Trans. Computer-Aided Design of Integrated Circuits & Systems*, vol. 19, no. 5, pp. 534-549, May 2000.

[4] C. Walshaw and M. Cross, "Parallel optimisation algorithms for multilevel mesh partitioning," *Parallel Computing*, vol. 26, no. 12, pp. 1635-1660, Nov. 2000.

[5] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Analysis & Machine Intelligence*, vol. 22, no. 8, pp. 888-905, Aug. 2000.

[6] G. Getz, H. Gal, I. Kela, D.A. Notterman, and E. Domany, "Coupled two-way clustering analysis of breast cancer and colon cancer gene expression data," *Bioinformatics*, vol. 19, no. 9, pp. 1079-1089, Jun. 2003.

[7] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Trans. Pattern Analysis & Machine Intelligence*, vol. 24, no. 7, pp. 881-892, Jul. 2002.

[8] C.H. Cheng, W.K. Lee, and K.F. Wong, "A genetic algorithm-based clustering approach for database partitioning," *IEEE Trans. Systems, Man, & Cybernetics*, vol. 32, no. 3, pp. 215-230, Aug. 2002.

[9] S. Günter and H. Bunke, "Self-organizing map for clustering in the graph domain," *Pattern Recognition Letters*, vol. 23, no. 4, pp. 405-417, Feb. 2002.

[10] R.J. Hathaway and J.C. Bezdek, "Fuzzy c-means clustering of incomplete data," *IEEE Trans. Systems, Man, & Cybernetics*, vol. 31, no. 5, pp. 735-744, Oct. 2001.

[11] C.Y. Chen and F. Ye, "Particle swarm optimization algorithm and its application to clustering analysis," in *Proc. 1st. Int. Conf. Networking, Sensing, & Control*, Taipei, Taiwan, 2004, pp. 789-794.

[12] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264-323, Sep. 1999.

[13] P. Berkhin, *Survey of Clustering Data Mining Techniques*, Accrue Software Inc., San Jose, CA, 2002.

[14] S. Kotsiantis and P. Pintelas, "Recent advances in clustering: a brief survey," *WSEAS Trans. Information Science & Applications*, vol. 1, no. 1, pp. 73-81, Jul. 2004.
[15] D. Pelleg and A. Moore, "X-means: extending k-means with efficient estimation of the number of clusters," in *Proc. 17th. Int. Conf. Machine Learning*, Stanford, CA, 2000, pp. 727-734.
[16] G. Hamerly and C. Elkan. (2003, Dec.). Learning the k in k-means. in *Proc. 17th. Conf. Neural Information Processing Systems*. Vancouver, Canada. Available: http://books.nips.cc/nips16.html.
[17] L.Y. Tseng and S.B. Yang, "A genetic approach to the automatic clustering problem," *Pattern Recognition*, vol. 34, no. 2, pp. 415-424, Feb. 2001.
[18] G. Garai and B.B. Chaudhuri, "A novel genetic algorithm for automatic clustering," *Pattern Recognition Letters*, vol. 25, no. 2, pp. 173-187, Jan. 2004.
[19] R.J. Kuo, K. Chang, and S.Y. Chien, "Integration of self-organizing feature maps and genetic-algorithm-based clustering method for market segmentation," *J. Organizational Computing & Electronic Commerce*, vol. 14, no. 1, pp. 43-60, Jan. 2004.
[20] C. Walshaw and M. Cross, "Mesh partitioning: a multilevel balancing and refinement algorithm," *SIAM J. Scientific Computing*, vol. 22, no. 1, pp. 63-80, Jan. 2000.
[21] S.J. D'Amico, S.J. Wang, R. Batta, and C.M. Rump, "A simulated annealing approach to police district design," *Computers & Operations Research*, vol. 29, no. 6, pp. 667-684, May 2002.
[22] Y.G. Saab, "An effective multilevel algorithm for bisecting graphs and hypergraphs," *IEEE Trans. Computers*, vol. 53, no. 6, pp. 641-652, Jun. 2004.
[23] G. Wolfe, J.L. Wong, and M. Potkonjak, "Watermarking graph partitioning solutions," *IEEE Trans. Computer-Aided Design of Integrated Circuits & Systems*, vol. 21, no. 10, pp. 1196-1204, Oct. 2002.
[24] U. Elsner, "Graph partitioning: a survey," Technische Universitat Chemnitz, Chemnitz, Germany, Tech. Rep. 393, Dec. 1997.
[25] P.O. Fjällström. (1998, Sep.). Algorithms for graph partitioning: a survey. *Linköping Electronic Articles Computer & Information Science*. *3(10)*. Available: http://www. ep.liu.se/ea/cis/1998/010.
[26] T.N. Bui and B.R. Moon, "Genetic algorithm and graph partitioning," *IEEE Trans. Computers*, vol. 45, no. 7, pp. 841-855, Jul. 1996.
[27] A. Kaveh and H.A.R. Bondarabady, "A hybrid graph-genetic method for domain decomposition," *Finite Elements in Analysis & Design*, vol. 39, no. 13, pp. 1237-1247, Oct. 2003.
[28] K. Kohmoto, K. Katayama, and H. Narihisa, "Performance of a genetic algorithm for the graph partitioning problem," *Mathematical & Computer Modelling*, vol. 38, no. 11-13, pp. 1325-1332, Dec. 2003.
[29] H. Stuckenschmidt and M. Klein, "Structure-based partitioning of large concept hierarchies," in *Proc. 3rd. Int. Conf. Semantic Web*, Hiroshima, Japan, 2004, pp. 289-303.
[30] M. Wall. (1996, Aug.). GAlib: a C++ library of genetic algorithm components. Available: http://lancet.mit.edu/ga/dist/galibdoc.pdf.
[31] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message-passing interface standard," *Parallel Computing*, vol. 22, no. 6, pp. 789-828, Sep. 1996.

Razib M. Othman is a doctoral candidate at the Faculty of Computer Science and Information System, the Universiti Teknologi Malaysia. He received the BSc and MSc degrees in Computer Science both from the Universiti Teknologi Malaysia, in 1999 and 2003 respectively. Currently, he is working for his PhD in Computational Biology. He also has interests in artificial intelligence, software agent, parallel computing, and web semantics. In March 2005 he was awarded the Young Researcher award by the Malaysian Association of Research Scientists (MARS). One of his inventions, a software product named *2D Engineering Drawing Extractor,* has recently won 5 awards including the Best Invention of the Pacific Rim at the 21st Invention and New Product Exposition (INPEX) held in Pittsburgh, USA.

Safaai Deris is a Professor of Artificial Intelligence and Software Engineering at the Faculty of Computer Science and Information System, Deputy Dean at the School of Graduate Studies, and Director of Laboratory of Artificial Intelligence and Bioinformatics at the Universiti Teknologi Malaysia. He received the MEng degree in Industrial Engineering, and the DEng degree in Computer and System Sciences, both from the Osaka Prefecture University, Japan, in 1989 and 1997 respectively. His recent academic interests include the application and development of intelligent techniques in planning, scheduling, and bioinformatics.

Rosli M. Illias is an Associate Professor at the Faculty of Chemical and Natural Resources Engineering at the Universiti Teknologi Malaysia. Rosli received the PhD degree in Molecular Biology from the Edinburgh University, UK in 1997, and the BSc degree in Microbiology from the Universiti Kebangsaan Malaysia in 1992. His research interests are in the areas of microbial technology, molecular enzymology, and molecular genetics.

Zalmiyah Zakaria is a lecturer at the Faculty of Computer Science and Information System, the Universiti Teknologi Malaysia. She received the BSc and MSc degrees in Computer Science both from the Universiti Teknologi Malaysia, in 1999 and 2001 respectively. Her research interests focus on artificial intelligence, web-based software engineering, and bioinformatics.

Saberi M. Mohamad is a lecturer at the Faculty of Computer Science and Information System, the Universiti Teknologi Malaysia. He received the BSc and MSc degrees in Computer Science both from the Universiti Teknologi Malaysia, in 2002 and 2005 respectively. His expertise includes gene expression analysis, parallel computing, and computational methods such as genetic algorithms and support vector machines.