Discover Gene Specific Local Co-regulations from Time-course Gene Expression Data

JI ZHANG, QIGANG GAO
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia, Canada

HAI WANG
Sobey School of Business
Saint Mary's University
Halifax, Nova Scotia, Canada

[1]

Abstract

Discovering gene co-regulatory relationships is one of most important research in DNA microarray data analysis. The problem of gene specific co-regulation discovery is to, for a particular gene of interest (called target gene), identify the condition subsets where strong gene co-regulations of the target gene are observed and its co-regulated genes in these condition subsets. The co-regulations are local in the sense that they occur in some subsets of full experimental conditions. The study on this problem can contribute to better understanding and characterizing the target gene during the biological activity involved. In this paper, we propose an innovative method for finding gene specific co-regulations using genetic algorithm (GA). A sliding window is used to delimit the allowed length of conditions in which gene co-regulations occur and an *ad hoc* GA, called the progressive GA, is performed in each window position to find those condition subsets having high fitness. It is called progressive because the initial population for the GA in a window position inherits the top-ranked individuals obtained in its preceding window position, enabling the GA to achieve a better accuracy than the non-progressive algorithm. kNN Lookup Table is utilized to substantially speed up fitness evaluation in the GA. Experimental results with a real-life gene expression data demonstrate the efficiency and effectiveness of our technique in discovering gene specific co-regulations.

1 Introduction

DNA microarray technology provides faster, more efficient and accurate way for measuring the relative representation of each mRNA species in the total mRNA population. A microarray experiment involves measuring the relative representation of a large number of mRNA species simultaneously, typically thousands or even tens of thousands, in a set of related biological conditions (e.g., time points taken during a biological process). The experimental results for each condition is compared

[1]Contacting author: Ji Zhang, Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada. Email: jiz@cs.dal.ca

to a common reference condition and the result for each gene is the ratio of the relative abundance of the gene in the experimental condition compared to the reference. Typically, the microarray experiment results are represented in a two-dimensional table or matrix, with each row representing a gene and each column representing an experimental time condition. The entry of the table is the log-transformed expression ratio of the gene under the given time condition. The expression ratio measures the relative expression level of gene in the experimental condition. Positive values indicate higher expression level compared to the reference and vice versa.

DNA microarray provides us with a global view of gene expression and has been used in a number of different ways. One interesting research direction is to study co-regulatory relationships among genes under different temporal conditions. These temporal conditions are the experimental time points along the course of some biological activity when the expression of genes are extracted. It has been known that a gene may be regulated by multiple regulators along the full timeline and the phenomenon of partial (or local) co-expression between genes has been identified, meaning that gene profiles may simultaneously change in a sub-range of the time course rather than the overall time course [24]. An interesting problem is to *find condition subsets for observing strong co-regulations for the target genes and the regulators of target gene in these condition subsets.* This is called *Single Gene Approach* for gene microarray analysis [18]. The discovered co-regulated genes and the associated condition subsets are gene specific. The answer to this question is very helpful for human users to better understand and characterize the target gene by means of its co-regulations with other genes in the discovered sets of experimental conditions during the biological activity involved. As early DNA microarray experiments have shown that genes of similar function yield similar expression patterns [16], gene-specific co-regulations are therefore able to assist in function prediction of unknown genes through in-depth study on its correlated genes whose function has been known. In this paper, we are interested in studying the *local* co-regulations of the target gene that occur in a few neighboring, but not necessarily consecutive, conditions. Those co-regulations among conditions located far apart from each other in the timeline are disregarded. The biological rationale behind this is that genes are more likely to display biologically meaningful co-regulations at neighboring conditions. These co-regulations may experience time-lag [12], but such lagged co-regulation still often occur within a relatively short time period compared to the entire timeline involved.

Even though the problem of gene co-regulation discovery has been studied intensively in recent years, there is rare research work on gene specific co-regulation discovery. The most naive way is to evaluate co-regulation between target gene g and each other gene in the dataset in each possible condition subset. Another way to approach this problem is to use clustering analysis to find gene clusters in each condition subset and evaluate the members of the clusters to which g belongs in order to find its closely co-regulated genes. Unfortunately, the complexities of these two methods are at least $O(NM^2)$, where N and M correspond to the number of genes and number of conditions in the dataset, respectively. Given the fact that M is usually large for gene expression data, therefore these methods are prohibitively expensive. Note that genetic algorithm can be applied to these two native methods to improve their efficiency. However, in pair-wise evaluation method, N runs of genetic algorithm is required to evaluate co-regulation between g and each other gene. This will still be rather slow if there are large number of genes in the dataset. While for the clustering-based methods, the fitness function tailored to clustering purpose will guide the search process towards the condition subsets where the overall quality of gene clusters is better. Quite likely, this may lead to the loss of those condition subsets in which the overall cluster quality is inferior but the target gene is in fact significantly co-regulated with others.

In this paper, we propose an approach for mining local gene-specific co-regulation using genetic algorithm. The basic idea of our approach is to first find the condition subsets in which the target gene g is most significantly co-regulated with others and the co-regulated genes of g are then selected from its nearest neighbors in these condition subsets. Specifically, a sliding window is used to scan all the experimental conditions sequentially and the search of condition subsets is performed within each window position. A progressive genetic algorithm is presented, in which the top-ranked condition subsets obtained in one window position will be used to find good subsets in the subsequent window position. kNN Lookup Table is used to remarkably boost the efficiency of the genetic algorithm by speeding up the fitness evaluation. Experimental results demonstrate the efficiency and effectiveness of the technique we propose.

Roadmap. The remainder of this paper is organized as follows. Section 2 reviews some of the related work. Section 3 presents a formal formulation of local gene-specific co-regulation discovery problem. Our technique for discovering local gene co-regulation using genetic algorithm is discussed in details in Section 4. In Section 5, we elaborate on the technique for speeding up the genetic algorithm. Experimental results are presented in Section 6 and the last section concludes the whole paper.

2 Related Work

Unsupervised learning, i.e. clustering analysis, is currently the most commonly used technique for gene co-regulation study from microarray data. It is able to identify genes that are co-regulated in a similar manner, forming groups or clusters, under a set of specific experimental conditions. The research efforts have mainly been taken in two aspects in clustering analysis of time-course gene expression data, i.e. the *clustering algorithms* and *similarity measures* used in the clustering analysis.

The commonly used clustering algorithms in discovering gene co-regulations include hierarchical clustering method [13], k-means algorithm [13], Self-Organization Maps (SOMs) [14] and SVD-based clustering algorithm [10] that have come to existence in the domain of machine learning and data mining for a long time. Even though they are capable of grouping similar microarray data into clusters, most of them perform clustering based on the entire set of conditions (i.e. full dimensionality). This causes them to miss out those interesting co-regulations embedded in the lower dimensional condition subsets. This renders them incapable to cope with local co-regulation discovery problem.

To find the gene co-regulations in some subsets of conditions, a few subspace clustering methods for gene expression data, such as Coupled Two-Way Clustering [9], bi-cluster [5] and δ-cluster [21], are proposed. They try to find sub-matrices/blocks defined by a subset of genes on a subset of conditions that satisfy some user-defined clustering criterion. Since gene expression dataset is high-dimensional by nature, thus finding all these coherent blocks is a NP problem due to the curse of dimensionality. Therefore, these methods are mostly very slow. Moreover, the condition subsets where clusters are observed are not neighboring; most of them are actually far apart from each other. This is inappropriate when deal with time-course microarray data. The temporal conditions where meaningful co-regulations can be observed should be relatively close in the time frame. To handle local gene co-regulations, Ji *et al.* [12] recently proposed a method for identifying local time-lagged gene clusters. In this method, each gene will be clustered into a few so-called q-clusters whose members share the same local change pattern for q consecutive conditions. Although this method does not suffer the problem of full dimensionality, it can only identify co-regulations occurring in a few consecutive conditions. The conditions in which the gene exhibit co-regulation may not consecutive in the sense that one or a few conditions may be skipped in practice. In addition, the local patterns are rigidly restricted to have a fixed length q so this method cannot find those patterns with smaller variable lengths.

As for as the similarity measures are concerned, the Euclidean-based distance metrics and Pearson's Correlation Coefficient are among the widely used ones. Recently, there are a number of research work focusing on improving the effectiveness of similarity measures in handling time series of gene expression data. These work take into account of some limitations in the Euclidean-based distance metrics and Pearson's Correlation Coefficient. Feng *et.al* proposed a time-frequency approach for clustering gene expression time series [7]. This method considers time series data as mathematical functions within a larger system and identifies the relationship in these functions by means of time-frequency analysis. Ramoni *et.al* consider the time-series being generated by some stochastic processes and these data can thus be grouped into clusters that corresponding to their generating processes [17]. In this work, each gene expression data is represented as a so-called autoregressive equation and clusters are generated in a way such that the posterior probability of the resulting clusters can be maximized. In [6], Erdal *et.al* convert each microarray time-series data into a binary string. This allows for the modeling of positive and negative co-regulations. The authors then use Longest Common Subsequence (LCS) to measure the similarity between genes expression data in the clustering. Similarly, Kwon *et al.* mark the changes of gene expression as an event (Rising (R), Constant (C) or Falling (F)) by calculating the slope of the expression value at each time interval, resulting in a string of events [15]. A global sequence alignment algorithm, the Needleman−Wunsch algorithm, is then employed to match the corresponding events of two genes, based on which a numerical score is generated as an indicator of the likelihood of a regulatory relationship existing between those two genes. Yeung *et.al* employ dominant spectral component to measure the similarity between microarray time-series data [22]. Amato *et.al* focus on the pre-processing of microarray data for noise removal and feature selection and they use Negentropy as the similarity metric for clustering purpose [1]. Because obvious heuristics are not available, the co-regulation discovery process using these metric will be rather slow in exploring the subsets of experimental conditions across the whole timeline.

Filkov *et.al* address a few fundamental problems in analysis of microarray time-series data including regulation prediction, edge detection, periodicity and phase analysis, correlation comparison of distinct length sequences and correlation significance of small alphabet sequences [8]. However, gene-specific co-regulation discovery problem is untouched in this paper. Interested readers can refer to [4] for a survey of important issues in analysis microarray time-series data.

It is worthy noting that, the most salient drawback of clustering-based analysis methods, regardless of the algorithms and similarity measures they use, lies in that they cannot provide direct and efficient support to the gene specific co-regulation discovery problem. As discussed in the Introduction Section, it will be computationally prohibitive to extract single gene co-regulations directly from the gene clusters. Using clustering in the search method such as genetic algorithm will divert the search direction towards the subsets of conditions that optimized the overall clustering quality instead of those where strong co-regulations of the target gene can be found. Furthermore, the clustering methods are less appropriate when only a few genes are likely to co-regulated [18] as these co-regulated genes may not form a strong cluster. Our approach adopts a more direct manner to deal with this problem. It first finds the condition subsets in which the target gene is most significantly co-regulated with others, and then find the co-regulated genes of the target gene from these condition subsets.

3 Problem Formulation

To define local gene co-regulations, we need to delimit the allowable length of a condition subset. To this end, a *sliding window* with a fixed size, ω, will be used. The size of this window is specified by human users a priori. This may require some biological knowledge to decide the maximum possible number of conditions under which meaningful co-regulations of the target genes are to be studied. A large window allows for a study on gene co-regulations within a wider span of conditions and vice versa. To examine all the possible condition subsets, this window will be slided from the leftmost to the rightmost position, with one condition offset each time. Therefore, for a gene expression dataset with M dimensions, there will be $M - \omega + 1$ different positions for a sliding window with a size of ω. In the extreme case when $M = \omega$, there will be only one position for the window as it now covers the full dimensionality. Hence, the problem studied in this paper is general in the sense that it is applicable to co-regulation discovery in both partial and full dimensionality, depending on the specification of window size ω.

Having discussed the sliding window, we can now formulate the problem to be studied in this paper mathematically. Some of the frequently used symbols in this paper, together their annotations, are presented in table 1. let $D = N \times M$ be the table with N genes and M conditions,

Symbols	Meaning
N	Number of genes in the expression dataset
M	Number of conditions in the expression dataset
ω	Size of the sliding window
n	Top-n subsets of conditions having highest fitness
k	The number of closest neighbors considered
N_s	Number of genes in the sampling dataset
P	Number of individuals in each population of GA
N_g	Number of generations in GA
p_c	Probability of applying crossover
p_m	Probability of applying mutation

Table 1: Notations

representing the given microarray data, and $C = \{d_1, d_2, \ldots, d_M\}$ be the full set of experimental conditions. Given a target gene g, the set of top n condition subsets, denoted as S, in which target gene g is most significantly co-regulated with some other genes are defined as follows:

$$S = \{s_1, s_2, \ldots, s_n\}$$

where each element s_i $(1 \leq i \leq n)$ of S is subject to the following constraints:

1. $s_i \subseteq C$ and $|s_i| \leq \omega$;
2. For any other condition subset $s \notin S$, $s \subseteq C$ and $|s| \leq \omega$, we have $f(g, s_i) \leq f(g, s)$.

where $|s|$ denotes the number of conditions in subset s. $f(g, s)$ is the function computing the normalized distance between g and its k^{th} nearest neighbors in condition subset s, i.e.

$$f(g, s) = \frac{dist(g, g_i, s)}{\sqrt{|s|}}, g_i \text{ is } k^{th}NN(g, s, D) \tag{1}$$

where $k^{th}(g, s, D)$ is the k^{th} nearest neighbors of g from dataset D in s, where k is a human-specified parameter, and $dist(g, g_i, s)$ is the metric used to compute Euclidean distance between g and g_i in s. Suppose $x = \{x_1, x_2, \ldots, x_{|s|}\}$ and $y = \{y_1, y_2, \ldots, y_{|s|}\}$ are the projections of two genes x and y in s. The distance between x and y in s is formulated as follows:

$$dist(x, y, s) = \sqrt{\sum_{i=1}^{|s|} (x_i - y_i)^2} \tag{2}$$

$\sqrt{|s|}$ used in Eq. (1) serves as a normalization factor. It helps render $f(g, s)$ to have comparable magnitude across different condition subsets s. The set of top co-regulated genes for the target gene g in a condition subset $s_i \in S$ $(1 \leq i \leq n)$ are simply the kNNs of g in s_i.

Finally, by putting together S and the co-regulated genes of g in each member of S, we can obtain the *answer set* $\mathcal{A}$ to gene specific co-regulation problem as follows:

$$\mathcal{A} = \{< s_i, g_j > | s_i \in S \text{ AND } g_j \in kNNSet(g, s_i, D)\} \tag{3}$$

The answer to the problem is basically a set of pairs whose first element represents condition subset s_i and the second element represents a top co-regulated gene g_i of g in s_i.

4 Genetic Algorithm for Discovering Gene Co-regulations

The evolutionary algorithm [11], such as genetic algorithm, is inspired by the Darwinian theory of evolution that a competition among the various species lead to survivals of the only fittest after a natural selection process. The fitter individuals tend to mate each other more often, resulting in better individuals [3]. The basic elements of genetic algorithm consist of *individual representation, fitness function, selection operators* and *search operators*. The individuals represent the candidate solutions of the problem. There are various representation of individuals in different problems such as bit vector, LISP program, tree-like structure, etc. The fitness or goodness of an individual is evaluated by means of the fitness function. The fitness value of an individual is similar to the objective function value. A higher fitness value indicates a fitter individual and vice versa. The commonly used fitness function include classification/clustering accuracy, cost of learning algorithm and complexity of individuals. Selection operators are responsible for selecting good individuals in the current population for generating offerings in the subsequent generation. Fitness-proportionate selection, rank-based selection and tournament-based selection are among the most common selection methods. The search operators are then applied on these selected individuals to produce their offerings. Mutation and crossover are two basic search operators in GA applications.

Often, typically general-purpose black-box GA software on straightforward string encodings does not work very well [2]. Therefore, an *ad hoc* genetic algorithm needs to be designed to well suit the specific problem under study basing on a good understanding of the problem. This involves choosing appropriate individual representation, fitness function, selection operators and search operators. In the sequel, we will elaborate on the designing details of the genetic algorithm for gene specific co-regulation discovery.

4.1 Individual Representation

To prevent a terminological ambiguity arisen from the "gene" in the microarray dataset and the "gene" of individual used in GA domain, we will call the the "gene" of individual used in GA domain as "*bit*" instead for the rest of this paper. Our GA technique uses standard binary individual encoding; all individuals are represented by strings with fixed and equal length ω, where ω is the window size. Using binary alphabet $\Sigma = \{0, 1\}$ for gene alleles, each bit in the individual will take on the value of "0" and "1", indicating whether or not its corresponding condition is selected, respectively ("0" indicates the corresponding condition is absent and vice versa for "1"). For a simple example, the individual "100101" when $\omega = 6$ means that the 1st, 4th and 6th conditions in the current window are selected, which is a 3-dimensional subset.

Please note that the locus of a bit in an individual represents only its *relative* position with respect to the window it belongs to. The final answer to our problem may entail a mapping from genotype to phenotype that involves converting a bit's relative locus within an individual into the full dimensionality. This needs the index of the window to which the individual belongs. More precisely, let $Index(W)$ be the index of a window W, $L(b, W)$ be the relative offset of bit b in window W and $L(b)$ be the absolute locus of b w.r.t full dimensionality, then we have the following mapping:

$$L(b) = Index(W) + L(b, W) - 1 \tag{4}$$

where $1 \leq Index(W) \leq M - \omega + 1$ and $1 \leq L(b, W) \leq \omega$, M is the number of conditions in the dataset. For instance, if an individual in the 2nd window position is "100110" (the window size ω is 6), then by applying the above bit-wise mapping, this individual can be converted to a condition set $\{2, 5, 6\}$, meaning that the above individual represents a 3-dimensional condition set containing the 2nd, 5th and 6th conditions in the whole set of experimental conditions of the microarray data.

4.2 Fitness Function

The fitness function in our GA-based teichnique, denoted by $f_{fit}(g, s)$, is defined based on $f(g, s)$ that is given in Eq. (1). Its definition is presented as follows:

$$f_{fit}(g, s) = f(g, s)^{-1} = \left\{ \frac{dist(g, gi, s)}{\sqrt{|s|}} \right\}^{-1}, g_i \text{ is } k^{th} NN(g, s, D) \tag{5}$$

The purpose to make fitness function $f_{fit}(g, s)$ to be the inverse of $f(g, s)$ is to achieve a consistence between fitness function value and fitness of condition subsets; that is, a high fitness function value corresponds to a fitter solution and vice versa. Give the target gene g, our problem can be formulated as a maximization problem that aims to search for those condition subsets s that are able to maximize $f_{fit}(g, s)$.

4.3 Selection

In our work, *fitness-proportionate selection*, also known as roulette-wheel selection, is used to select fitter solutions in each step of the evolution. Fitness-proportionate selection is a stochastic selection method where the selection probability of a condition subset, w.r.t the target gene g, is proportional to the value of its fitness function $f_{fit}(s, g)$, i.e.,

$$Pr(s) = \frac{f_{fit}(s, g)}{\sum_{i=1}^{P} f_{fit}(s_i, g)} \tag{6}$$

where P is the population size. Since $f_{fit}(g, s) > 0$, then each individual stands a chance of being selected for the next generation.

4.4 Crossover and Mutation

Crossover and mutation are two most commonly used search operators in genetic algorithm. Following Holland's canonical GA specification [11], the crossover and mutation used in this paper is *single-point crossover* and *bit-wise mutation*. In single-point crossover, a crossover locus on two parent individuals is selected and all the bits beyond that locus in the strings are swapped between the two parents, producing two new children. The bit-wise mutation involves flipping each bit randomly and leads to generating a new children. In our work, all the new individuals generated by crossover and mutation are of the same length, i.e. ω, as their parent(s). There are two associated probabilities, p_c and p_m, used to determine the frequencies for applying crossover and mutation, respectively. Please note that the application of crossover and mutation is not mutually exclusive in the sense that each selected pair of parents will go through tests of crossover and mutation to decide which search operator(s) is/are to be applied on them. Normally, we have $p_c >> p_m$, meaning that crossover is performed in a much higher frequency than mutation.

4.5 Progressive Genetic Algorithm

The naive GA-based approach to deal with our problem involves multiple steps, with a sliding window position being examined in each step. Genetic algorithm is applied for each sliding window position *independently* in order to identify top condition subsets in the window. The initial population for each window position is generated randomly. The top n condition subsets within the window will be maintained as the *candidate individuals*. Therefore, the total number of condition subsets obtained after a window scan on all conditions will be $n * (M - \omega + 1)$, given that there are $M - \omega + 1$ different window positions. The top n condition subsets are selected from these $n * (M - \omega + 1)$ individual candidates, together with the closely co-regulated genes in respective condition subsets relative to the target gene.

Considering the fact that the windows locating at two consecutive positions are highly overlapped with each other and the results of a window position are very useful for the subsequent one, thus a more effective method is to adapt a *progressive* fashion in the search process. In contrast to the naive GA-based method, the progressive method includes into the initial population for each window position, except the first one, the *modified* individuals produced in its previous window. The initial population comes from two sources, the modified individuals from previous window and some other individuals generated randomly with a bias. Please note that the entire initial population for the window in the first position is generated randomly without any bias. Next, we will elaborate on individual modification and biased random population generation. The detailed description of the progressive genetic algorithm is presented at the end of this subsection.

• Individual Modification

Let us suppose that a top-ranked condition subset s is obtained in the window at the i^{th} position, denoted as W_i. Two cases will be considered here in which different modification schemes will be applied accordingly:

- **Case 1**: The first bit of s is "1", meaning that s contains the first condition in window W_i. An example of such a subset s can be "101101";
- **Case 2**: The first bit of s is "0", suggesting that s does not contain the first condition in window W_i. An example of such a subset s can be "001101".

Two modification operations, *deletion* and *insertion*, will be performed on s to generate new individuals for the next window as follows:

1. The first bit of s will be deleted;
2. If the first bit of s is "1", then *two* new individuals will be generated by inserting "0" and "1" respectively into the tail position of the string obtained in the first substep. If the first bit of s is "0", then only *one* new individual will be generated by inserting "1" into the tail position.

For instance, if a top-ranked condition subset in W_i is "100010", then by deleting the first bit and inserting "0" and "1" respectively at the end, we end up obtaining two new individuals, i.e. "000100" and "000101", for W_{i+1}. Whereas if the top-ranked condition subset in Window i is "000010", then only one new individual will be generated from it by deleting the first bit and inserting "1" at the end for W_{i+1}, which generates "000101".

Now, let we denote by regular expressions $\Omega\{\omega-1\}0$ and $\Omega\{\omega-1\}1$ the strings with the format of $\underbrace{\Omega\ldots\Omega}_{\omega-1}0$ and $\underbrace{\Omega\ldots\Omega}_{\omega-1}1$, respectively. Ω is a "don't care" symbol that can be instantiated by either "0" or "1". Based on the modification scheme, we can see that the number of modified individuals matching $\Omega\{\omega-1\}1$ is no less than the number of those matching $\Omega\{\omega-1\}0$.

The motivation for modifying the top-ranked individuals in a window is that they preserve the segment of strings that are potentially contained in the good individuals in next window due to the high degree of overlap between two consecutive window positions. Thus, these modified individuals, if properly modified, can provide useful guidance to the search process for the next window. Noted that the reason why we do not add "0" to those individuals in Case 2 is because the new modified individuals, with "0" being added at end, has been evaluated in the current window and therefore should be excluded from the initial population of its subsequent window.

- **Random Population Generation with Bias**

For a window position, besides modifying the top-ranked individuals obtained from the preceding window, we will generate a set of individuals randomly to create its initial population. To achieve the necessary diversity of the initial population, it is desired to have approximately equal probability for the presence of each condition in every individual of the initial population for each window. Unfortunately, random population generation without any bias is not able to meet this need due to inclusion of modified individuals from the preceding window. As discussed earlier, among those individuals coming from the preceding window, there is usually a higher number of modified individuals matching $\Omega\{\omega-1\}1$ than those matching $\Omega\{\omega-1\}0$. Therefore, it is necessary to have a mechanism to offset this unbalance by means of introducing some *bias* in the initial population generation for each window. Probabilistically speaking, such bias will help generate more individuals with the format of $\Omega\{\omega-1\}0$ than those with the format of $\Omega\{\omega-1\}1$. To this end, we first generate random binary strings with a length of $\omega-1$ and then add "0" and "1" to the end of these strings with a probability of p_0 and p_1, respectively, $p_0 \geq p_1$. p_0 and p_1 are defined in three cases as follows:

$$\begin{cases} p_0 = \frac{0.5P-n_0}{P-n_0-n_1}, p_1 = \frac{0.5P-n_1}{P-n_0-n_1}, & \text{if} \quad n_0 \leq n_1 < 0.5P \\ p_0 = 1, p_1 = 0 \quad , & \text{if} \quad n_0 < 0.5P \leq n_1 \\ p_0 = 0, p_1 = 0 \quad , & \text{if} \quad n_0 + n_1 = P \end{cases} \tag{7}$$

where n_0 and n_1 denote the number of modified individuals from the preceding window matching $\Omega\{\omega-1\}0$ and $\Omega\{\omega-1\}1$, respectively. n_0 and n_1 are subject to $n_0 \leq n_1$ and $n_0 + n_1 \leq P$.

1. When $n_0 \leq n_1 < 0.5P$, the expected number of individuals matching $\Omega\{\omega-1\}0$ and $\Omega\{\omega-1\}1$ introduced by random population generation should be $0.5P-n_0$ and $0.5P-n_1$, respectively. The total number of individuals to be generated randomly is $P-n_0-n_1$. Therefore, we have $p_0 = \frac{0.5P-n_0}{P-n_0-n_1}$ and $p_1 = \frac{0.5P-n_1}{P-n_0-n_1}$.

2. When $n_0 < 0.5P \leq n_1$, the modified individuals with format of $\Omega\{\omega-1\}1$ has exceeded half of the initial population, so only the individuals that match $\Omega\{\omega-1\}0$ will be generated randomly. Thus, we have $p_0 = 1$ and $p_1 = 0$.

3. $n_0 + n_1 = P$ suggests that the number of modified individuals alone has reached the number of individuals in the initial population, therefore the step of random generation is purely omitted, with a zero probability for both p_0 and p_1.

Based on the above discussions, we can see that the diversity of initial population can only be guaranteed in the first case. This is because, in the second and third cases, the randomly generated individuals cannot fully compensate the unbalance introduced by inheriting the modified individuals from preceding window. If one wishes to ensure that the first case can always occur in the algorithm, then we should have $n < 0.5P$. We prove this in the following lemma.

Lemma 1. If $n < 0.5P$, then we have $n_0 \leq n_1 < 0.5P$.

Proof: Let x_0 and x_1 be number of individuals, out of the top n individuals obtained in the preceding window, whose the first bit is "0" and "1", respectively. Obviously, $x_0 + x_1 = n$. Based on our aforementioned method for modifying individuals, we have $n_0 = x_1$ and $n_1 = x_0 + x_1 = n$. Thus if $n < 0.5P$, then $n_0 \leq n_1 < 0.5P$, as desired. (End of proof)

Lemma 1 has established that, as long as the number of top-ranked individuals n to be obtained in each window position is less than half of the initial population size P, the diversity of initial population can thereby be realized. Since n_0 and n_1 may be changed for different window positions,

```
Algorithm: Progressive_GA(the target gene $g$ )
1. $CandidateSet \leftarrow \emptyset$;
2. WHILE (the window does not reach the last position) DO {
3.   $S_{pop} \leftarrow$ initial population of $P$ strings;
4.   WHILE (evolution_stop_criterion=false) DO {
5.     FOR each individual $s$ in $S_{pop}$ DO
6.       evaluate fitness of $s$;
7.     $CandidateSet \leftarrow CandidateSet \cup$ top $n$ individuals in the current generation;
8.     $S_{pop} \leftarrow selection(S_{pop})$;
9.     $S_{pop} \leftarrow crossover(S_{pop}, p_c)$;
10.    $S_{pop} \leftarrow mutation(S_{pop}, p_m)$; } }
11. $\mathcal{S} \leftarrow$ Top $n$ individuals in $CandidateSet$;
12. $\mathcal{G} \leftarrow$ Co-regulated $k$NNs of $g$ in $\mathcal{S}$;
13. Return ($\mathcal{S}$, $\mathcal{G}$);
```

Figure 1: Progressive Genetic Algorithm

therefore we have different p_0 and p_1 for the initial population generation of different window positions which need to be dynamically updated as the algorithm proceeds.

The detailed algorithm of progressive GA for gene specific co-regulation discovery is given in Figure 1. *CandidateSet* is the set for storing the top n individuals obtained in all generations and it is generation-wisely updated (Line 7-8). There are two nested while loops (Line 2 and 4). The outer while loop examines all the possible window positions, whereas the inner loop performs GA-based subset search within each window. The progressive GA differs the naive GA in the initial population generation for each different window position (Line 3), in which individual modification and biased random population generation are performed.

5 Speed Up Genetic Algorithm

Like many other GA applications, the most computationally expensive step performed in our genetic algorithm lies in the fitness evaluation of individuals. The problem of relative slow fitness evaluation in our work is because the fitness evaluation for each individual (i.e. subset of conditions) involves scanning all the genes in the dataset in order to find the k^{th} nearest neighbors of the target gene for computing fitness function $f_{fit}(g, s)$. Using indexing methods to speed up k^{th}NN search is not efficient in our problem whatsoever due to two major reasons. First, since a large number of condition subsets may be evaluated in the GA, it will be expensive to index the genes in each possible condition subset. Second, for gene-specific co-regulation discovery problem, we may be only interested in a small number of target genes, thus the cost associated in building indexing cannot be amortized by the one-time performance gain resulting from gene indexing.

To speed up fitness evaluation, we draw on the *kNN Lookup Table* proposed in [23] to speed up computation of fitness function in our technique. A kNN Lookup Table for a target gene g, denoted as $\mathcal{T}^g$, is a $M \times k$ table containing information about its k nearest neighbors in *each* single dimension of full data space with M dimensions. The entry x_{ij} of the table represents the j^{th} nearest neighbor of g in the i^{th} dimension, where $1 \leq i \leq M$ and $1 \leq j \leq k$.

The idea of using kNN Lookup Table for speeding up $f_{fit}(g,s)$ is to approximate $f_{fit}(g,s)$ by quickly computing its lower and upper bounds, i.e. $f^k_{min}(g,s)$ and $f^k_{max}(g,s)$. For the details regarding how to compute $f^k_{min}(g,s)$ and $f^k_{max}(g,s)$ efficiently by means of kNN Lookup Table, please refer to [23]. The approximated fitness of s with respect to the target gene g is computed by using the average of $f^k_{min}(g,s)$ and $f^k_{max}(g,s)$ as follows:

$$f_{app}(g,s) = \frac{f^k_{min}(g,s) + f^k_{max}(g,s)}{2} \tag{8}$$

kNN Lookup Table is advantageous in the following two aspects: 1) its construction cost is $O(NM)$, which is linear with respect to both number and dimensionality of genes in the gene expression data set. For the whole gene expression data, only one kNN Lookup Table is needed; 2) The total complexity for computing $f^k_{min}(g,s)$ and $f^k_{max}(g,s)$ is $O(k^2|s|^2)$, which becomes independent of N and M. It has been shown in [23] that this fitness approximation scheme leads to a computation saving by up to a factor of $k|s|N$ compared to the case when no approximation is performed. In our work, a pre-processing step is performed to construct kNN Lookup Table based on the gene expression data set prior to the GA-based condition subset search.

Since we approximate $f_{fit}(g,s)$ in the GA, the accuracy of computation is thus somehow limited. To address this problem, we can perform a refinement step on the top condition subsets we obtain in the GA (stored in $CandidateSet$). Instead of using the lower and upper bounds of $f_{fit}(g,s)$ for a fast fitness approximation, the refinement step computes the accurate fitness for top candidate condition subsets and the top n condition subsets among them will be returned. Admittedly, evaluating each condition subset in this refinement process is more expensive than the approximation as it involves more accurate computations. However, the number of candidate condition subsets is typically much smaller than the total number of condition subsets approximated in the GA. Furthermore, a pruning optimization strategy can be devised to speed up the computation. The basic idea of this pruning optimization strategy is that, after the fitness of n candidate condition subsets have been accurately evaluated, we start to maintain the minimum $f_{fit}(g,s)$ for the top n condition subsets we have found thus far, denoted as $MinFit$. Those unevaluated condition subsets that satisfies $f^k_{max}(g,s) < MinFit$ cannot become the top n condition subsets and can therefore be safely pruned. This is because that $MinFit$ is monotonically increasing as we examine more condition subsets in the refinement step. Specifically, the refinement process takes the following steps:

1. We start the condition subset evaluation with those subset candidates that have the maximum $f^k_{max}(g,s)$ value. After n condition subsets candidates have been evaluated, the minimum $f_{fit}(g,s)$ value for the top n condition subsets we have evaluated thus far is assigned to $MinFit$;

2. For each condition subset s' in $CandidateSet$ that has yet been examined, if $f^k_{max}(g,s') <$

$MinFit$, then s' is pruned away;

3. For a candidate condition subset s that have not been pruned in Step 2, if $f_{fit}(g, s) > MinFit$, then s is included into the top n list and the condition subset that has the smallest value of $f_{fit}(g, s)$ in the current list is removed. $MinFit$ is also updated;

4. Repeat Step 2-3 until all the condition subsets in $CandidateSet$ have been evaluated.

6 Experimental Results and Evaluation

6.1 Experimental Setup

In our experiments, the Spellman's dataset is used that can be downloaded from http://genome-www.stanford.edu/ cellcycle/data/rawdata. This dataset contains all the data for the alpha factor, cdc15, and elutriation time courses, and includes the data for the Clb2 and Cln3 induction experiments. We used only the alpha-factor and CDC28 datasets for our experiments, as did in [12] and [15]. The dataset used for experimental purpose contains 6178 genes at 35 time points, forming a 6178×35 matrix (this dataset can be downloaded at *http://www.comp.nus.edu.sg/ jiliping/p2/YeastData.xls*).

As the experimental setup, we set the size of the sliding window $\omega = 10$, the nearest neighbors considered $k = 10$, the number of top condition subsets returned in the end (as well as the number of top individuals kept as individual candidates in each window) $n = 10$, the number of generations for the GA in each window position $N_g = 20$, the population size in each generation $P = 30$, the frequency of applying crossover $p_c = 0.8$ and the frequency of applying mutation $p_m = 0.2$. The same setting is applied for all experiments except that some major parameters, such as the window size, number of genes, number of conditions and total number of individuals to be evaluated, will be varied when experiments are performed to evaluate their respective effect on the performance. To create test datasets with desired number of gene and/or conditions for our experimental purpose, random sampling is performed horizontally and vertically on the aforementioned real-life dataset. The running time reported in the experiments are averages over 5 samples with the same number of genes and/or conditions. All the experimental evaluations are carried out on a Pentium 4 PC with 512MB RAM.

6.2 Efficiency Study

We start the performance evaluation with the efficiency study of our technique. Specifically, we will investigate the effect of parameters such as the number of condition, the number of genes, the window size and the number of individuals needs to be evaluated on the efficiency of our method.

- **Effect of number of conditions**

Given that the number of conditions for most gene expression data sets is large, we thus first evaluate the effect of number of conditions on the performance. Figure 2 presents the running time

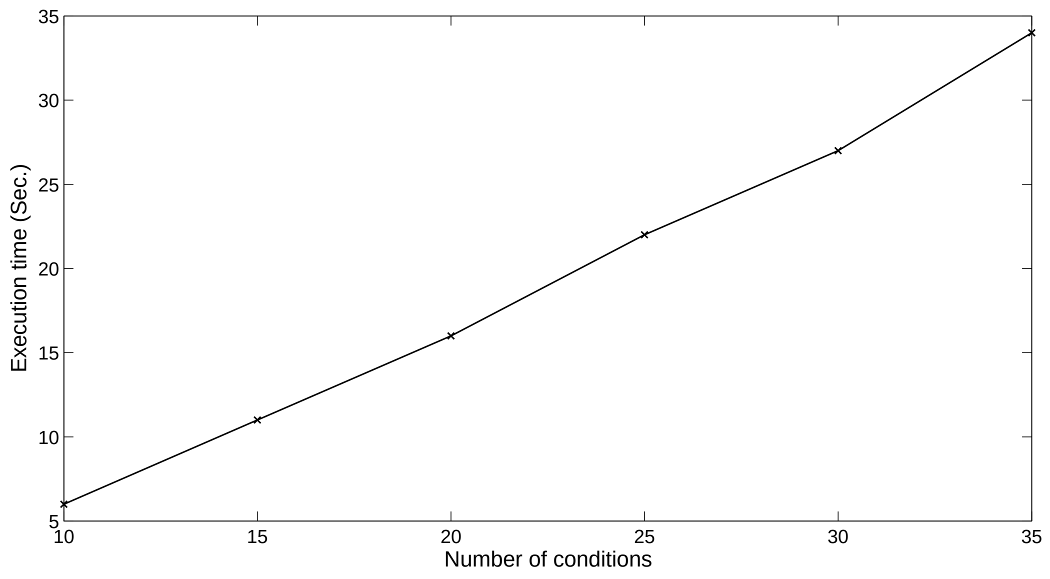

Figure 2: Running time under varying number of conditions

of our technique when varying the number of conditions. Opposite to what is normally perceived, the time spent grows linearly, rather than exponentially, when the number of conditions increases. The underlying reason is that the increase in the number of conditions will only lead to an linear increase of the number of window positions (recall that the number of window positions is $M-\omega+1$). Hence, the time complexity scales up in a linear manner, providing that the search workload for each window remains unchanged. This property enables the good scalability of our method with regard to number of conditions.

- **Effect of number of genes**

Figure 3 presents the running time of kNN Lookup Table construction and our progressive genetic algorithm under different number of genes. The number of genes only affects the efficiency of constructing kNN Lookup Table. For each single-dimensional subspace, we need to find the kNNs of the target gene. Thus, the complexity of kNN Lookup Table construction is in a linear order with regard to the number of genes. When kNN Lookup Table has been constructed, we no longer need to scan all the genes in the data set for k^{th} NN search in fitness computation. As such, the cost of fitness computation is independent of the number of gene; the complexity curve in Figure 3 becomes roughly a horizontal line.

- **Effect of window size**

The window size determines the size of the search space for condition subsets within each window, which is in an exponential order of the window size. This does not necessarily mean that the running time of our method will be exponential with respect to the window size whatsoever. The actual

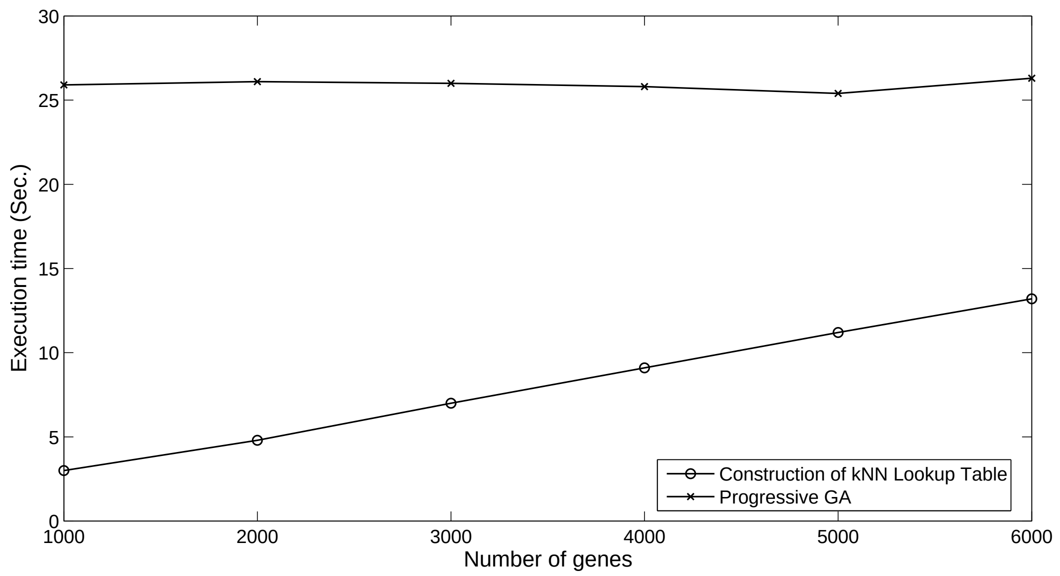

Figure 3: Running time under varying number of genes

running time is depended on how the search workload within each window is specified. More precisely, if we use the fixed number of generations and population size for each generation in the GA, i.e. *fixed number of individuals* to be evaluated in each window, then the total search workload for each window will be the same. In this case, increase in window size will consequently lead to a decrease, rather than an increase, in the number of window positions and therefore a drop of running time. Yet, using a fixed search workload for windows with different sizes is obviously not an effective strategy. A *fixed ratio search* scheme, in contrast, performs a search workload that is proportional to the size of search space in this window. The time complexity for each window now becomes quadratic with respect to the window size. Even though the number of window positions is decreased, the overall complexity is dominated by the quadratic complexity increase within each window. The running time of these two search schemes are presented in Figure 4. For each window position, the search workload with a fixed number of individuals is set to be 400 and that of a fixed ratio is 30% of the search space of the window.

- **Effect of search workload**

The search workload, i.e. the number of individuals to be evaluated, in each window position is mainly decided by two factors in the GA, the number of generations N_g and the population size P in each of the generations. The total workload is equal to $P * N_g$. As a result, under a fixed number of window positions, any change in either P or N_g will give rise to a linear change in the total running time of the algorithm accordingly. We test five workloads for each window position in this experiment, ranging from 500 to 1500, and the results are presented in Figure 5.

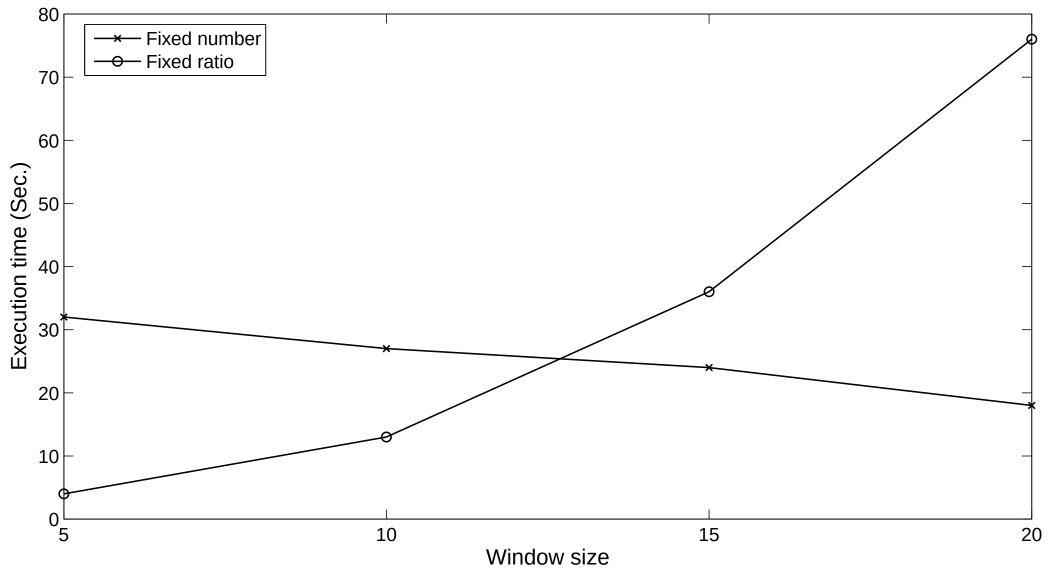

Figure 4: Running time under varying window sizes

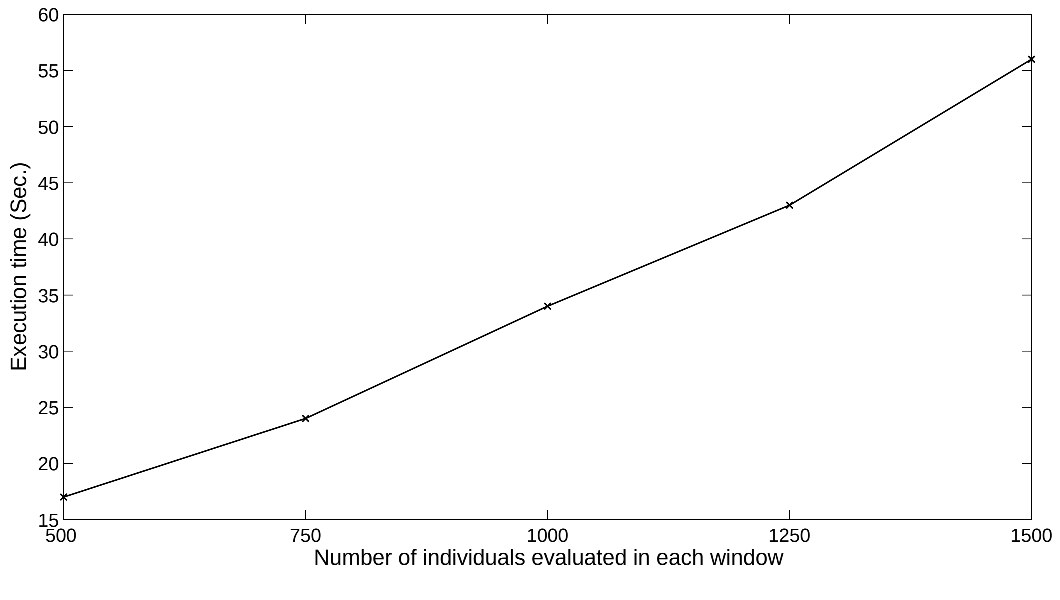

Figure 5: Running time under varying number of individuals to be evaluated

Methods	5	10	15	20
Brute forth	39	71	219	624
GA-based method (Fixed ratio)	12	24	69	210
GA-based method (Fixed number)	41	38	34	28

Table 2: Efficiency comparison between brute force method and our method

- **Comparison with Brute Force Method**

The naive brute force method searches exhaustively the subsets of conditions in each window position for the optimal solutions. It can be considered as the baseline method for solving our problem. Apparently, the search complexity depends entirely on the window size that determines the whole search space lattice. In this experiment, we demonstrate the efficiency of our method by comparing the time spent in our method with that in the native brute force method. Different window sizes, ranging from 5 to 20, are studied. The results are tabulated in Table 2. We can see that our method, by using GA, is more efficient than the brute force method. The fix ration GA-based method is faster than the brute force method by roughly a factor of $\frac{1}{\alpha}$, where α is the ratio of computational workload in GA against the total workload in each window. Our fix-number GA-based method, which is independent of the window size, is faster than the brute force method by a significantly large margin, especially when the window size is large.

6.3 Effectiveness Study

For effectiveness analysis, experiments are performed to test fitness enhancement of progressive GA versus non-progressive scheme, the convergence and finally the accuracy of our approach.

- **Fitness enhancement by using progressive GA**

For effectiveness study, we first investigate the contribution of progressive GA used in our method to enhancing the fitness of individuals, compared to the case when non-progressive GA is used. They primarily differ in that the progressive GA inherits the top-ranked individuals obtained from the previous window position with appropriate modifications and introduces bias in the initial random population generation, while the non-progressive GA evaluates each window independently and the entire initial population for each window position is generated randomly. Figure 6 presents the *averaged* fitness of top 10 individuals for each window position (from No. 1 to 26). The result demonstrates that the progressive GA outperforms non-progressive scheme in term of fitness in up to 88% of the window positions and fitness improvement by over 10% is observed at about 25% of the window positions. This result indicates that progressive GA is more capable of finding fitter individuals than the non-pregressive GA under the same computaional workload.

- **Convergence study**

GA tends to produce an increasing number of fitter individuals as evolution proceeds, referring

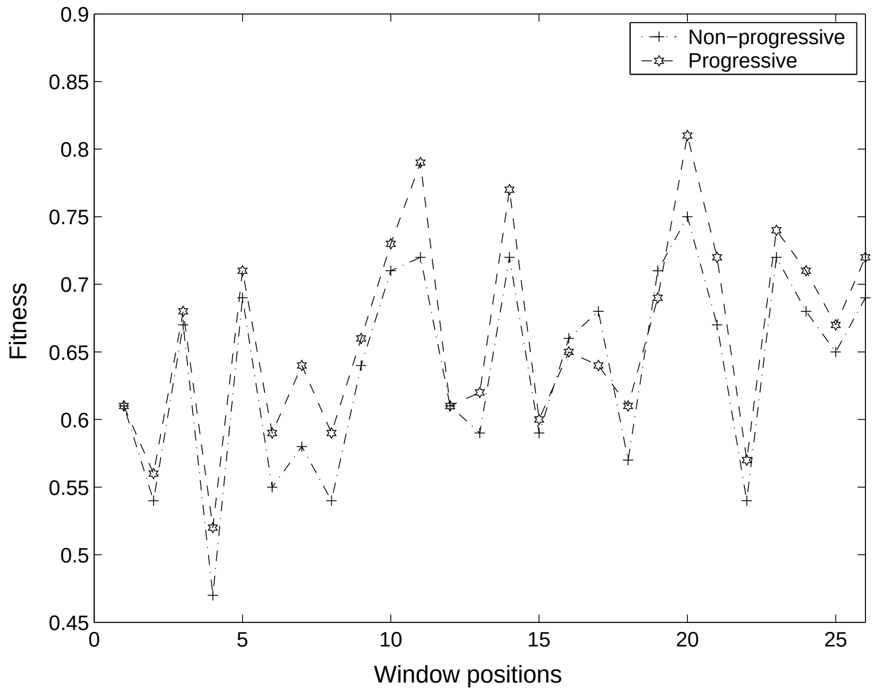

Figure 6: Fitness of non-progressive and progressive GA

to as the phenomenon of convergence. In this experiment, we investigate the convergence of our technique. Without losing generality, three window positions, the first, middle and last (i.e. 1^{st}, 13^{th} and 26^{th}), are picked up for this study. For a better representation, we normalize the fitness of condition subsets by dividing them with the best fitness we observe in the GA. In this way, we convert the fitness of condition subsets into the range of (0,1]. For each generation, the number of individuals with relative good fitness ($\geq$ 0.7 in our experiment) are counted. As we can see from Figure 7 that the number of individuals with good fitness is increased as the GA evolves, which indicates a good convergence of our method. In addition, the good individuals do not only appear in the last generation, though the overall convergence of the GA has been observed. A small number of good individuals have been observed in the earlier generations of the GA. This verifies the validity of keeping track of the top-ranked individuals in each generation of GA in our approach to prevent the loss of good individuals appearing in different, particularly the early, generations of the GA.

- **Comparative study**

Finally, we explore the accuracy of our method in detecting co-regulated condition subsets for the target gene. Comparative study are first carried out between our method and random search method. A *reference result* for comparative study purpose needs to be obtained and appropriate accuracy metric should be derived. To get the reference result, we perform an exhaustive search for all the possible condition subsets in each of possible window positions for 10 test genes. In this

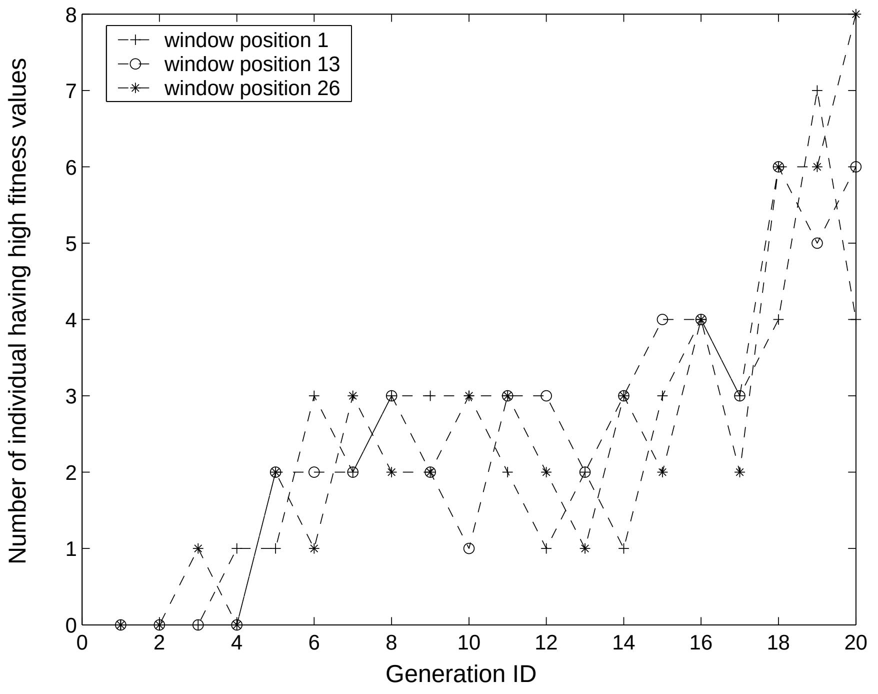

Figure 7: Number of individuals having high fitness values in three window positions

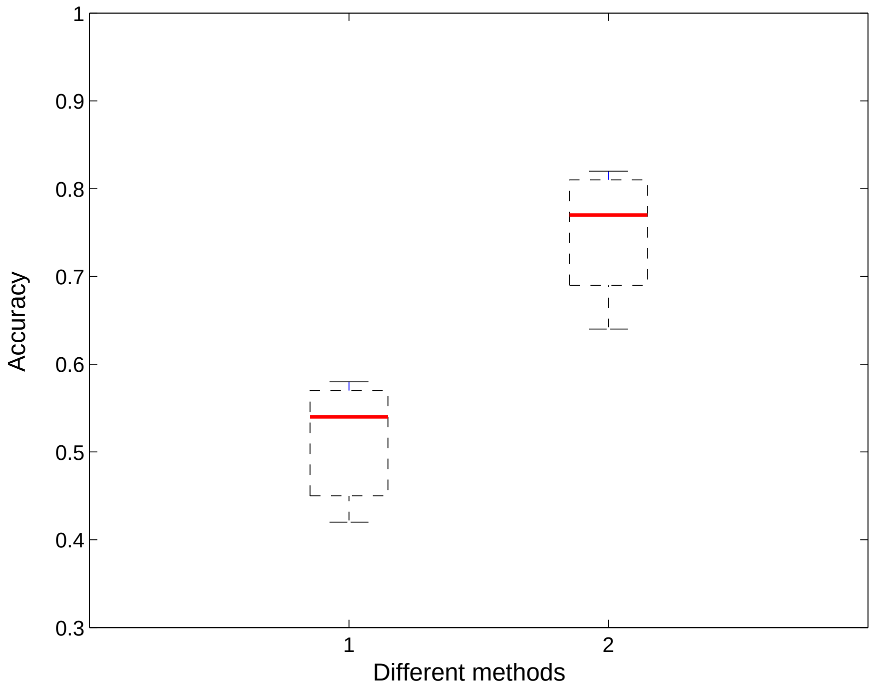

Figure 8: Accuracy of random search method and our method

search process, we only maintain the top-ranked condition subsets, rather than their closely co-regulated genes, for each of the test genes. This is because once one method can correctly identify a condition subset, then it becomes trivial to find the closely co-regulated genes for the test genes in this condition subset. Thus, the focus of our study is the accuracy evaluation on the methods in detecting correct co-regulated condition subsets. The results obtained by the exhaustive search method, used as the reference, is apparently the best possible result we can expect to achieve. The results of our method and the random search method will both be compared with the reference result. As far as the accuracy metric is concerned, we define *accuracy* to measure the robustness of the method in correctly identify the condition subsets. It is defined as follows:

$$Accuracy = \frac{N_a}{n} \times 100\% \tag{9}$$

where N_a denotes the number of correctly identified condition subsets by a method a and n denotes the number of top subsets users are interested in obtaining. Here, the instances of method a are random search method and our method. The correctly identified condition subsets by a method is the intersection of this method's result and that of the exhaustive search method. The number of condition subsets returned by the exhaustive search, random search and our method are all fixed to n. The key considerations in designing this experiment are 1) allowing the same search workload for random search method and our method to ensure a fair comparison and 2) the search workload for the two methods should be considerably smaller than that required in the exhaustive search method to prevent them from reducing to the exhaustive search. In this experiment, the search workload for random search and our method is set to be 50% of that of the exhausitve search method. Figure 8 is the boxplot of the accuracy for the 10 test genes. The first and second columns in the boxplots present the results for the random search and our method, respectively. It illustrates that our method is able to achieve an accuracy of 79%, which is remarkably higher than the random search method's 54%. This is due to the convergence property of GA in our method. Also as expected, the accuracy of the random search method (54%) is approximately equal to the workload allocated (50%) simply because there is no involvement of heurisitcs to positively guide the search process in random search. Of course, the accuracy is largely depended on the search workload allocated; a large workload will certainly lead to a higher accuracy but a more expensive computation is required and vice versa.

Besides conducting comparative study between our method and the random search method, we further explore the capability of clustering methods in tackling gene-specific co-regulation discovery problem. Instead of directly finding the top n condition subsets, by whichever means, in which the best overall quality of gene clusters are observed, we choose the following way for our evaluation purpose that is more economic to implement. First, we find the top n condition subsets returned by the exhaustive search method and mix them with a number of randomly generated condition subsets that are considered to be noise. Then, clustering is performed and top n condition subsets are found that feature the best clustering quality. The result of clustering will be compared with that of the exhaustive search method to compute its accuracy. The rationale of this experiment

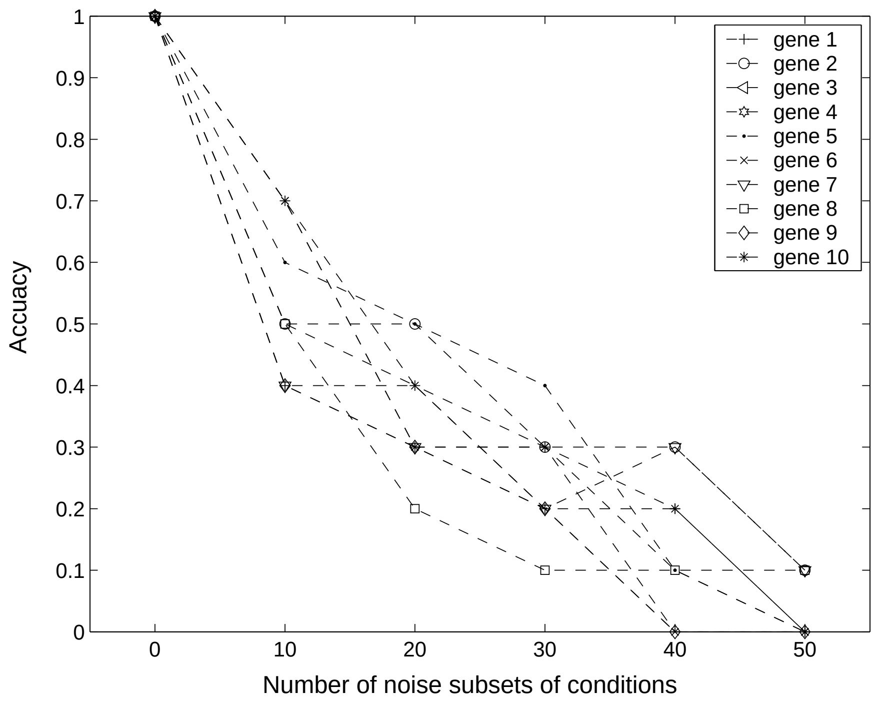

Figure 9: Accuracy of clustering method

is that if clustering methods cannot find all or most of the correct top n condition subsets from the abovementioned mixture of condition subsets (i.e. the correct top n and noise subsets), then we can conclude that the clustering methods are sensitive to the noise we introduce and are not able to correctly identify the true top n condition subsets. This renders them fail to cope with gene-specific co-regulation discovery problem successfully. The reason is that searching a larger number of condition subsets, rather than a small number of them as we do in this experiment, will only increase the likelihood of end up with an even lower accuracy level.

Now, we discuss the clustering quality metric. The intra-cluster sum of distance of a cluster j, $1 \leq j \leq k$, in condition subset s, denoted as E_j^s, is defined as the sum of distance between each gene in cluster j and its center, i.e.

$$E_j^s = \sum_{i=1}^{n_j^s} dist(g_{i,j}^s, m_j^s, s), \text{ where } m_j^s = \frac{\sum_{i=1}^{n_j^s} g_{i,j}^s}{n_j^s} \tag{10}$$

where n_j^s, $g_{i,j}^s$ and m_j^s correspond to the number of genes within cluster j in s, the i^{th} gene within cluster j in s and the center of cluster j in s, respectively. The overall quality of clustering result in s is quantitized by the total intra-cluster sum of distance of all the k clusters we obtain in s. That is,

$$E^s = \sum_{j=1}^{k} E_s^j \tag{11}$$

A lower value of E^s indicates a better clustering quality in s.

In this experiment, we employ k-means clustering method for clustering genes due to its widespread use in microarray data analysis. Since cluster quality in k-means clustering is largely depended on the choice of pre-defined cluster number k, we therefore test 8 different values for k, ranging from 3 to 10, and pick up the best cluster quality from these 8 different settings. We vary the number of randomly generated condition subsets from 0 to 50 and mix them with the 10 top-ranked subsets return by the exhaustive search method.

Figure 9 illustrates the results of the clustering method. 10 genes are tested in this experiment. As the number of noise condition subsets increases, the capability of the clustering method in returning the correct condition subsets drops substantially. Therefore, it is predictable that, as the number of condition subsets we examine is increased to some large number, the accuracy of clustering method will tend to become 0 for all the genes under study. The underlying reason is because the quality of clusters is not compatible with the objective for strong co-regulations with respect to a particular gene. In contrast, our GA-based method is able to achieve a high accuracy (79%) (as presented in Figure 8) even when working on a much larger number of subsets. Its accuracy is able to be further improved as long as a larger computational workload (still much smaller than that of exhustive search) is allowed.

7 Conclusions and Discussions

This paper investigates the problem of gene co-regulation discovery problem in DNA microarray data. Unlike most of the existing methods that find gene co-regulations utilizing clustering analysis, our approach aims to discover co-regulations from the single gene perspective. The basic idea of our approach is to first find the subsets of conditions in which the given gene g is most significantly co-regulated with other genes and the co-regulated genes of g are reported by selecting from its nearest neighbors in these subsets of conditions.

Considering the search space of subsets of conditions is typically large for microarray dataset, genetic algorithm (GA) is employed. Due to inapplicability of the general-purpose black-box GA for our problem, an ad-hoc GA-based algorithms that adopt a progressive paradigm is proposed. Its salient feature is that they try to make use of the top-ranked solutions found in each window position in guiding the search process for its subsequent window position to boost the overall accuracy of the algorithm.

A wide spectrum of experiments are conducted for performance evaluation. The major experimental results suggest that our method is linearly scalable with major parameters such as the number of conditions, the number of genes, the total search overload executed in each window position, etc. The only exception is the window size. It quadratically determines the search workload in each window position if the fixed-ratio search workload is chosen. Given its typical small value (10 in our work) relative to the number of conditions in the expression data set, the window size will not give rise to a dramatic blowout of the search space. In terms of effectiveness, our method features a good convergence and has a higher accuracy compared to the naive random search method. We have also shown that clustering method is not able to successfully cope with gene-specific co-regulation discovery problem. We believe that our method can serve as a promising tool for discovering new and interesting gene co-regulations in time-course gene expression data.

Acknowledgement

The authors would like to thank Dr. Malcolm I. Heywood from Evolutionary Algorithm Research Group and Dr. Christian Blouin from Computational Biology Research Group, both from Faculty of Computer Science at Dalhousie University, for their useful suggestions on the draft of this paper.

References

[1] R. Amato, A. Ciaramella, N. Deniskina, C. Del Mondo, D. di Bernardo, C. Donalek, G.Longo, G. Mangano, G. Miele, G. Raiconi, A. Staiano and R. Tagliaferri. 2006. A Multi-step Approach to Time Series Analysis and Gene Expression clustering. *Bioinformatics*, 22(5): 589-596.

[2] C. C. Aggarwal, J. B Orlin and R. P Tai. 1997. Optimized Crossover for the Independent Set Problem. *Operational Research* 45(2):226-234.

[3] C. C. Aggarwal and P.S. Yu. 2005. An Effective and Efficient Algorithm for High-dimensional Outlier Detection. *VLDB Journal*, 14, pp 211-221.

[4] Z. Bar-Joseph. Analyzing time series gene expression data.2004. *Bioinformatics*, 20(16):2493-2503.

[5] Y. Cheng and G.M. Church, Biclustering of Expression Data. 2000. In *Proc. International Conference on Intelligent Systems for Molecular Biology (ISMB)*, vol. 8, pp. 93-103.

[6] S. Erdal, O. Ozturk, D. Armbruster, H. Ferhatosmanoglu, and W. Ray. 2004. A time series analysis of microarray data. In *4th IEEE International Symposium on Bioinformatics and Bioengineering.*

[7] J. Feng, P. E. Barbano, and B. Mishra. 2004. Time-frequency feature detection for timecourse microarray data. In *2004 ACM Symposium on Applied Computing (SAC'04).*

[8] V. Filkov, S. Skiena, and J. Zhi. 2001. Analysis techniques for microarray time-series data. In *5th Annual International Conference on Computational Biology.*

[9] G. Getz, E. Levine, and E. Domany. 2000. Coupled Two-Way Clustering Analysis of Gene Microarray Data, in *Proc. Natioal Academy of Science*, vol. 97, no. 22, pp. 12079-12084.

[10] G. H Golub and C. F. Van Loan. 1983. *Matrix Computations*, Johns Hopkins University Press.

[11] J. Holland. 1992. *Adaption in Natural and Artificial Systems.* MIT Press, Cambridge.

[12] L. Ji and K. L. Tan. 2005. Identifying Time-Lagged Gene Clusters on Gene Expression Data. *Bioinformatics*, Vol. 21, No. 4, pp. 509-516.

[13] R. A. Johnson and D. W. Wichern. 1998. *Applied Multivariate Statistical Analysis*, Prentice Hall International, USA.

[14] T. Kohonen. 1995. *Self-Organization Maps.* Springer-Verlag, Berlin Heidelberg.

[15] A. T. Kwon, H. H. Hoos, and R. Ng. 2003. Inference of transcriptional regulation relationships from gene expression data. *Bioinformatics*, 19, 905-912.

[16] C. A. Orengo, D. T. Jones and J. M. Thornton. 2003. *Bioinformatics. Genes, proteins and Computers.* BIOS Scientific Publishers ltd, Oxford, UK.

[17] M. F. Ramoni, P. Sebastiani, and I. S. Kohane. Cluster analysis of gene expression dynamics. Proceedings of *the National Academy of Sciences*, USA, 99(14):9121-9126, July 2002.

[18] T. Speed, J. Fridlyand, Y. H. Yang and S. Dudoit. 2001. Discrimination and clustering with microarray gene expression data. *2001 Spring Meeting of International Biometric Society Eastern North American Region (ENAR'01)*, Charlotte NC.

[19] P. K. Sharpe and R. P. Glover. 1999. Efficient GA-based Technique for Classification. *Applied Intelligence* 11, pp 277-284.

[20] R. Tibshirani, T. Hastie, M. Eisen, D. Ross, D. Bostein and P. Brown. 1999. Clustering Methods for the Analysis of DNA Microarray Data. *Technical Report*, Stanford.

[21] J. Yang, W. Wang, H. Wang, and P.S. Yu. 2002. δ-Cluster: Capturing Subspace Correlation in a Large Data Set. In *Proc. 18th International Conference on Data Engineering (ICDE'02)*, pp 517-528, San Jose, CA, USA.

[22] L. K. Yeung, H. Yan, A. W. Liew, L. K. Szeto, M. Yang, R. Kong. 2004. Measuring Correlation between Microarray Time-series Data using Dominant Spectrum Component. In *Second Asia-Pacific Bioinformatics Conference (APBC 2004)*, 309-314, Dunedin, New Zealand.

[23] J. Zhang, Q. Gao, and H. Wang. 2006. A Novel Method for Detecting Outlying Subspaces in High-dimensional Databases Using Genetic Algorithm. *International Conference on Data Mining (ICDM'06)*, pp 731-740, Hong Kong, China.

[24] Y. Zhang, H. Zha, J. Z. Wang, C. Chu. 2004. Gene Co-regulation vs. Co-expression. 8^{th} *Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004)*, poster, San Diego, CA.