Molecular Programming: Evolving Genetic Programs in a Test Tube

Byoung-Tak Zhang
Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National University, Seoul 151-742, Korea
btzhang@bi.snu.ac.kr
http://bi.snu.ac.kr/

Ha-Young Jang
Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National University, Seoul 151-742, Korea
hyjang@bi.snu.ac.kr
http://bi.snu.ac.kr/

ABSTRACT

We present a molecular computing algorithm for evolving DNA-encoded genetic programs in a test tube. The use of synthetic DNA molecules combined with biochemical techniques for variation and selection allows for various possibilities for building novel evolvable hardware. Also, the possibility of maintaining a huge number of individuals and their massively parallel manipulation allows us to make robust decisions by the "molecular" genetic programs evolved within a single population. We evaluate the potentials of this "molecular programming" approach by solving a medical diagnosis problem on a simulated DNA computer. Here the individual genetic program represents a decision list of variable length and the whole population takes part in making probabilistic decisions. Tested on a real-life leukemia diagnosis data, the evolved molecular genetic programs showed a comparable performance to decision trees. The molecular evolutionary algorithm can be adapted to solve problems in biotechnology and nano-technology where the physico-chemical evolution of target molecules is of pressing importance.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis*; I.2.6 [**Artificial Intelligence**]: Learning—*induction*; J.3 [**Life and Medical Sciences**]: Biology and Genetics

General Terms

Algorithms, design

Keywords

Molecular programming (MP), genetic programs, *in vitro* evolution, DNA computing, molecular evolutionary computation (MEC)

GECCO'05, June 25–29, 2005, Washington, DC, USA.

1. INTRODUCTION

Genetic programming (GP) has been proposed as a method for automatically constructing computer programs using the principle of natural selection in nature [6]. Typically, tree-structured programs are used to represent the individuals in the population, various mutation and crossover operators are applied to the programs to generate their variations, and the fittest programs are selected to the next generation. The fitness of the individual program is evaluated by test-running it on a training set of example cases. Instead of tree-structured programs, linear-structured representations have also been proposed to evolve machine-code-like genetic programs [12]. Langdon [7] reviews various data structures used as representations for genetic programming. Applications of genetic programming range from controller design, multi-agent programming [13] and modelling biological systems [4] to quantum computer programming [16]. One of the distinguishing features of genetic programming is that it evolves variable-length representations [10, 15]. GP usually tries to find the program structure as well as its parameters. The capability of GP to search the whole space of potential programs based on training examples is very attractive from the machine learning point of view.

In this paper we present a genetic programming method that evolves DNA molecular structures in a test tube. The program in our case represents a DNA sequence representing a combination of markers for diagnosing a disease. For example, a program $(x_1 = 1, x_3 = 1, x_5 = 1, y = 1)$ in the form of "decision lists" or its DNA encoding (Figure 1) denotes a decision rule saying "diagnose the DNA sample as positive for disease y if it contains all the three markers x_1, x_3, and x_5." Based on a training set of input-output pairs (x, y) of DNA samples x associated with their disease labels y, our "molecular" genetic programming method, or "molecular programming" for short, evolves diagnosis rules that best predicts the training samples. The in vitro evolution of DNA-encoded genetic programs opens up a possibility of using GP in bio-technology and nano-technology where DNA is used as the structural material to be designed. Actually, unbeknownst to many of the evolutionary computation researchers, biochemists and bioengineers have been utilizing the concept of evolutionary algorithms for the design of novel molecules [17, 18, 19] and for the study of natural evolution [5].

There are some similarities and differences between the

standard genetic programming and the molecular programming. Molecular programming (MP) is similar to a standard GP in that its representation is of variable-length, which is a defining characteristic that distinguishes GP from other evolutionary computation methods. The use of decision lists as the representation of program structure is distinguished from other GP approaches, including the linear GP [12]. The use of DNA computing technology makes the design of the evolutionary operators very different from the conventional GP and other evolutionary computation methods. The possibility of synthetic DNA molecules and their manipulation by biochemical techniques in a test tube allows for the use of huge population size. Most of the operators, such as reproduction and selection, are massively parallel. In our simulations in this paper we will deal with population sizes of $118,096 \times 10^6 \approx 1.2 \times 10^{11}$ or less for resource limitations on silicon computers. However, in typical biochemical DNA computing experiments 10^{15} molecules or more have been used [8, 9]. Here the number 10^{15} comes from the use of nanomolar DNA, i.e. 6.2×10^{23} (molecules/mol) $\times$ 10^{-9} (mol) $\approx 10^{15}$. This paper aims to investigate the potentials of molecular computing in the context of GP.

The paper is organized as follows. Section 2 describes the DNA-based diagnosis problem and our approach to solving it. The decision-list representation of genetic programs is suggested. Section 3 describes the molecular algorithm for making robust decisions utilizing the molecular-scale huge population size. Section 4 describes the procedure for evolving the DNA-encoded decision lists in the test tube. Section 5 shows the results of evolving the DNA-based genetic programs for solving the diagnosis problem. It also characterizes the behavior of the evolutionary algorithm by analyzing the fitness landscape of the molecular programs. Section 6 draws conclusions.

2. REPRESENTING THE MOLECULAR GENETIC PROGRAMS

The aim is to build a decision-making system f that outputs a label y given an input pattern $\mathbf{x} = (x_1, ..., x_n)$, i.e.

$$f(\mathbf{x}) = y \quad \text{for all } (\mathbf{x}, y). \tag{1}$$

It is convenient to assume there exists a (unknown) target system f^* as an ideal model for f. However, we do not know the exact form of f^* and the only information we have to build f is data collected from the input-output pairs of f^*, i.e. a training data set.

To be more specific, consider a DNA-based diagnosis problem. Given a training set D of K labelled DNA samples in the form

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{K} \tag{2}$$

$$\mathbf{x}_i = (x_{i_1}, x_{i_2}, ..., x_{i_n}) \in \{0,1\}^n \tag{3}$$

$$y_i \in \{0,1\}. \tag{4}$$

Here $\mathbf{x}_i$ represents the DNA markers (subsequences of genes) in sample i and y_i is its associated diagnosis. For example, a training example $(10101, 1)$ means the sample is diagnosed positive ($y = 1$) if it contains the DNA markers numbered 1, 3, and 5 ($x_1 = 1, x_3 = 1, x_5 = 1$) and does not contain the rest ($x_2 = 0, x_4 = 0$).

To solve the diagnosis problem, a test tube of DNA molecules representing the genetic programs or diagnosis rules is maintained. Given is a set D of training data consisting of pairs of DNA-sample and its associated label, i.e. in the form of Equation (2). The goal is to find a population of genetic programs (or a single genetic program, depending on the interpretation as discussed below) that can predict the correct diagnosis label for a future DNA-sample, i.e. a decision maker f in Equation (1).

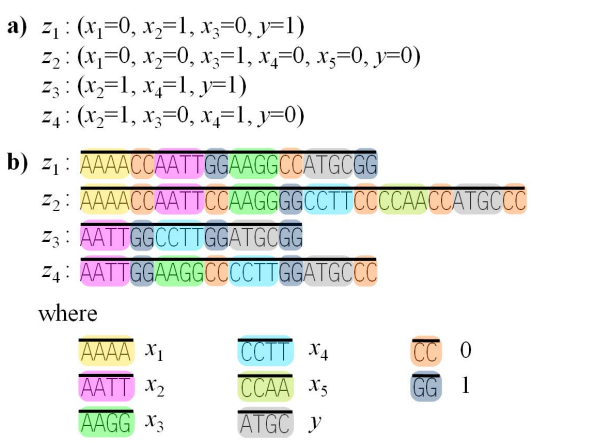

Figure 1: Population of genetic programs in two different representations: (a) set of decision lists, (b) library of DNA molecules corresponding to (a). The DNA code shown are illustration-purposes only and this design does not fully reflect the biochemical properties of the sequences.

Each individual genetic program is represented as a conjunction of binary variables x_i and a class label y what we shall refer to as a "decision list". The generic form of a decision list is $(x_1 = 1, x_3 = 1, x_5 = 1, y = 1)$, where the commas are interpreted as logical ANDs. The *order* of a decision list is defined as the number of input variables in it. Thus, the decision list $\mathbf{z} = (\mathbf{x}, y) = (x_1 = 1, x_3 = 1, x_5 = 1, y = 1)$ is of order 3. The population consists of decision lists of variable orders as illustrated in Figure 1(a).

The decision lists in DNA oligomers (i.e., short single-stranded DNA sequences) are represented as shown in Figure 1(b). Each attribute-value pair is encoded as a sequence of nucleotides (A, T, G, and C). The output label can also be encoded as a DNA sequence. For example, if 6-mer (i.e., a poly*mer* consisting of 6 monomers) is used as in the figure to encode each binary variable with its value (e.g., $(x_1 = 0)$ as AAAACC) and if a decision list contains 10 input variables and one output variable, then it can be encoded as a DNA molecule of length $6 \times 11 = 66$-mer.

The whole population consists of multiple copies of the decision lists and the number of copies is proportional to the importance of the decision list. The goal of molecular genetic programming is to find the probabilistic distribution of the decision lists to solve the diagnosis problem. Since each decision list represents a conjunction, the population represents a disjunction of conjunctions where each conjunction may have multiple copies. This representation has some similarity with the decision tree [11] in that the whole population represent a disjunction of conjunctions of attribute-value pairs. However, our ensemble representation allows for probabilistic computation of decision labels rather than deterministic as in conventional decision tree methods.

3. EXECUTING THE MOLECULAR GENETIC PROGRAMS

In the previous section it is described how to represent the decision lists using DNA molecules. Essentially, the DNA test tube (also called the DNA library) represents the joint probability $P(X, Y)$ of the input pattern X (DNA sample) and the output class Y (diagnosis). In this section it is discussed how the class label can be computed using the library.

Since each genetic program (i.e. decision list) has a class label, given a query the decision can be made based on each individual program. An alternative interpretation, we choose here, is to consider the whole population of decision lists as a single genetic program. In this ensemble approach the final decision making is performed by a consensus of the decisions of the individuals in the population. Since each decision list is labelled either 1 or 0, the whole population is partitioned into the two clusters. Given a query (DNA sample of a patient), its class (disease diagnosis) is determined by matching it against each and every decision list in the population and taking its majority class. As we shall see, the ensemble machine naturally makes use of the huge number of decision lists produced by the molecular genetic programming process to make decisions robust. A similar ensemble method has been proposed previously in [20] in the context of standard genetic programming.

The class label is determined by computing the probability of each class conditional on the input pattern $\mathbf{x}$, and then determining the class whose conditional probability is the highest, i.e.

$$y^* = \arg \max_{Y \in \{0,1\}} P(Y|\mathbf{x}) \quad (5)$$

$$= \arg \max_{Y \in \{0,1\}} \frac{P(Y, \mathbf{x})}{P(\mathbf{x})}, \quad (6)$$

where $P(Y, \mathbf{x}) = P(Y|\mathbf{x})P(\mathbf{x})$ was used and Y represents the candidate classes.

A method for realizing Equation (5) is to initialize the library with nth order decision lists and evolve their distributions. That is, the empirical probability distribution $P(X, Y)$ can be represented by a set of point estimators that constitute the DNA library L of decision lists:

$$P(X, Y) \approx \frac{1}{|L|} \sum_{i=1}^{|L|} f_i^{(n)}(X_1, X_2, ..., X_n, Y), \quad (7)$$

where $f_i^{(n)}(X_1, X_2, ..., X_n, Y)$ is the ith decision list of order n and $|L|$ is the library size. This is reasonable since the probability of hybridization between two DNA strands (i.e., Watson-Crick complementary binding of A-T and G-C nucleotides) can be expressed as the Boltzmann distribution

$$P(\mathbf{x}_i, y_i|\mathbf{x}_q, y_q) = \frac{\exp\left(-\Delta G(\mathbf{x}_i, y_i|\mathbf{x}_q, y_q)\right)}{\sum_j \exp\left(-\Delta G(\mathbf{x}_j, y_j|\mathbf{x}_q, y_q)\right)}, \quad (8)$$

where ΔG is the free energy in the hybridization between the DNA strands representing the library element $(\mathbf{x}_i, y_i)$ and the query sample $(\mathbf{x}_q, y_q)$ and each decision list is designed to represent the probability factor of matching:

$$f_i^{(k)}(\mathbf{x}_q, y_q) = \exp\left(-\Delta G(\mathbf{x}_i, y_i|\mathbf{x}_q, y_q)\right), \quad (9)$$

where $\Delta G(\mathbf{x}_i, y_i|\mathbf{x}_q, y_q)$ is the free energy between the ith library element and the given query sample. The matching

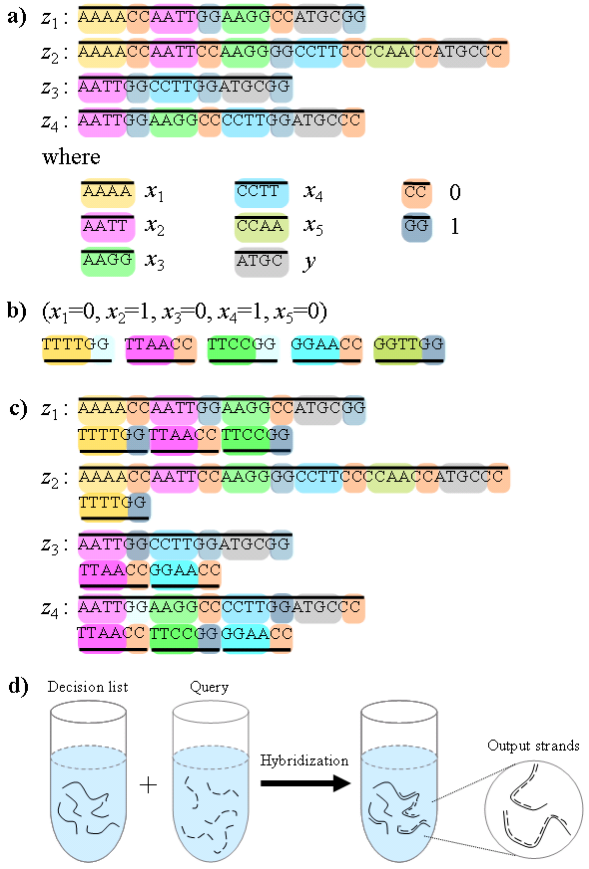

Figure 2: Illustration of the decision-making procedure using the population of DNA-encoded genetic programs: (a) Library of decision lists, (b) query sample (in multiple copies), (c) decision lists hybridized with query samples,(d) schematic for illustrating the whole decision procedure.

probability can be calculated by normalizing over the whole library. This approximation in Equation (7) can be made arbitrarily accurate by increasing the library size $|L|$.

However, the use of nth order decision lists only has some weaknesses. It is well known that higher-order features are too specific and not always useful in practice. It tends to result in overfitting problem and lose the generality. Thus, in this paper, we use the decision lists of variable length and let evolution find the appropriate complexity of the programs. This is where GP comes in. That is, our approach tries to approximate the probability distribution by k-th order decision lists $f_i^{(k)}(X_{j_1}, X_{j_2}, ..., X_{j_k}, Y)$ by

$$P(X, Y) \approx \frac{1}{|L|} \sum_{i=1}^{|L|} f_i^{(k)}(X_{j_1}, X_{j_2}, ..., X_{j_k}, Y), \quad (10)$$

where $|L|$ is the library size. Note that there are $N(k)$ copies of the k-th order library elements with $\sum_{k=1}^{n} N(k) = |L|$.

Hence, the above equation can be rewritten as

$$\begin{aligned} P(X,Y) &\approx \frac{1}{|L|}\sum_{k=1}^{n} N(k) \times f^{(k)}(X_{j_1}, X_{j_2}, ..., X_{j_k}, Y) \\ &= \sum_{k=1}^{n} \left(\frac{N(k)}{|L|} \times f^{(k)}(X_{j_1}, X_{j_2}, ..., X_{j_k}, Y) \right) \\ &= \sum_{k=1}^{n} P(X^{(k)}, Y), \end{aligned} \quad (11)$$

where $X^{(k)} = X_{j_1}, X_{j_2}, ..., X_{j_k}$ and $\frac{N(k)}{|L|}$ is the weighting factor or strength of the library elements of order k.

Figures 2 and 3 summarize the procedure for decision making using the molecular genetic programs in the test tube. Given a query pattern $\mathbf{x}$ all the molecules that match the query is extract from the library. These molecules will have class labels from which the majority label is decided as the class of the query pattern. A class label is a sequence appended to denote the class to which the pattern belongs. In in-silico implementation of this method, the given query has to be matched against each and every element of the library. In in-vitro molecular computation this can be done in a massively parallel fashion. Instead of a single query $\mathbf{x}$, multiple copies (up to the number of population size) of it is used so that they can be matched with library elements in parallel. The decision can be made by comparing the number of elements in class 1 with those in class 0.

The molecular algorithm for computing the class labels is summarized in Figure 3. It should be mentioned that there are some technical issues to be considered before this algorithm is efficiently realized using biochemical techniques. For example, in Step 2, the input query example can be amplified (by, for example, polymerase chain reaction or PCR) so that they can be matched in a massively parallel fashion against library elements. In addition, the query instance $\mathbf{x} = (x_1, x_2, ..., x_n)$ can be chopped into n DNA pieces representing x_1, x_2, ..., x_n, respectively, so that each of them can be matched separately to decision-list elements.

The extraction may involve some mismatches due to the potential for formation of double-stranded DNA duplexes. There is a lot of work going on to design the sequences and codeword sets (see, for example, [14] and references therein). From the machine learning point of view, the small error occurred by DNA mismatches offers the possibility of generalization by allowing unobserved patterns to be classified. The decision-making can still be robust because it is based on the statistics of the huge number of molecular samples.

It is useful to check the decision criterion the above molecular algorithm is computing. To see this, note that, in Step 3.1, the count or concentration $c(\mathbf{x})$ of $\mathbf{x}$ in M approximates the probability of observing the pattern which is called evidence:

$$c(\mathbf{x})/|L| = |M|/|L| \approx P(\mathbf{x}). \quad (12)$$

Step 3.2 essentially computes the frequencies $c(Y|\mathbf{x})$ of molecules belonging to different classes Y. These are an approximation of the conditional probabilities given the pattern, i.e. a posteriori probabilities:

$$c(Y|\mathbf{x})/|M| = |M^Y|/|M| \approx P(Y|\mathbf{x}). \quad (13)$$

Thus, in effect, the protocol computes the maximum a posteriori (MAP) criterion:

$$\begin{aligned} y^* &= \arg \max_{Y \in \{0,1\}} c(Y|\mathbf{x})/|M| \\ &= \arg \max_{Y \in \{0,1\}} c(Y|\mathbf{x}) \\ &\approx \arg \max_{Y \in \{0,1\}} P(Y|\mathbf{x}) \end{aligned} \quad (14)$$

which validates our objective set out in Equation (5). It is worth noting that for classification purposes only the relative frequency or concentration of the molecular labels are important.

- 1. Let the library L represent the current empirical distribution $P(X,Y)$ as in Equation (10).
- 2. Present an input (query) pattern $\mathbf{x}$.
- 3. Classify $\mathbf{x}$ using L as follows:
  - 3.1 Extract all library molecules matching with $\mathbf{x}$ into M.
  - 3.2 From M separate the molecules into classes:
    - Extract the molecules with label $Y = 0$ into M^0.
    - Extract the molecules with label $Y = 1$ into M^1.
  - 3.3 Compute $y^* = \arg\max_{Y \in \{0,1\}} |M^Y|/|M|$.

Figure 3: The molecular algorithm for decision-making based on DNA-encoded genetic programs.

4. EVOLVING THE MOLECULAR GENETIC PROGRAMS

In the previous section it is assumed that the library represents the proper joint-probability distribution $P(X,Y)$ of patterns X and their class Y. Here we describe how the library is revised from observed data.

We start with a random collection of DNA strands. Each DNA sequence represents an instance $(\mathbf{x}, y)$ of a vector (X,Y) of random variables of interest in the problem domain. Without any prior knowledge the DNA sequences are generated to represent uniform distribution of the data variables. As a new training example $(\mathbf{x}, y)$ is observed, the patterns matching $\mathbf{x}$ is extracted from the library. The class y^* of $\mathbf{x}$ is determined by the classification procedure described in the previous section. Then, the matching patterns are modified in their frequency depending on their contribution to the correct or incorrect classification of $\mathbf{x}$. If the label v of the library pattern $(\mathbf{u}, v)$ matching $\mathbf{x}$ is correct, i.e. $v = y$, it is reproduced:

$$L \leftarrow L + \{(\mathbf{u}, v)\}. \quad (15)$$

Optionally, if the label v is incorrect, i.e. $v \neq y$, the matching library pattern is removed from the library:

$$L \leftarrow L - \{(\mathbf{u}, v)\}. \quad (16)$$

The update of the library in this way is more or less like evolutionary computation with the additional feature that the presentation of a training example proceeds one generation of the library (as a population). This is also a learning

- 1. Let the library L represent the current empirical distribution $P(X, Y)$.
- 2. Get a training example $(\mathbf{x}, y)$.
- 3. Classify $\mathbf{x}$ using L as described in the previous section. Let this class be y^*.
- 4. Update L
  - If $y^* = y$, then $L_n \leftarrow L_{n-1} + \{\Delta c(\mathbf{u}, v)\}$ for $\mathbf{u} = \mathbf{x}$ and $v = y$ for $(\mathbf{u}, v) \in L_{n-1}$,
  - If $y^* \neq y$, then $L_n \leftarrow L_{n-1} - \{\Delta c(\mathbf{u}, v)\}$ for $\mathbf{u} = \mathbf{x}$ and $v \neq y$ for $(\mathbf{u}, v) \in L_{n-1}$.
- 5. Goto step 2 if not terminated.

Figure 4: The molecular algorithm for evolving the population of DNA-encoded genetic programs.

procedure since the library improves its classification performance as new examples are presented.

The molecular algorithm for the whole evolutionary learning procedure is summarized in Figure 4. In Step 4, $\Delta c(\mathbf{u}, v)$ denotes the number of copies of $(\mathbf{u}, v)$ to be added or subtracted. Addition operation can be implemented by PCR and removal can be done by extraction of the corresponding molecules. The update process relies upon the reliability of DNA extraction technology. Note also that the learning rule has a parameter Δc that reflects the strength of learning for each training example. This is also related to the reproduction rate. Big Δc imposes high reproduction rate while small Δc forces a low reproduction rate. How to set this parameter is an important issue for the stability and the adaptability of the algorithm.

To see the quantitative relationship between the parameter Δc and reproduction rate, the Bayesian framework for evolutionary computation is used [21]. In view of Bayesian evolution, the evolution from L_{n-1} to L_n can be rewritten as

$$P_n(X, Y|\mathbf{x}, y) = (1 + \delta)P_{n-1}(X, Y|\mathbf{x}, y), \quad (17)$$

where δ is a learning rate determining the strength of reproduction, hence also called *reproduction rate*. Using Bayes rule, we can derive [22]

$$\delta = \frac{P(\mathbf{x}, y|X, Y) - P(\mathbf{x}, y)}{P(\mathbf{x}, y)}. \quad (18)$$

This indicates that the molecular algorithm follows the Bayesian evolutionary update rule [21]. Also it was shown that δ is expressed as the amplification ratio of the number $\Delta c(\mathbf{x}, y)$ of additional copies of molecules to the number of current copies $c_{n-1}(\mathbf{x}, y)$, i.e.

$$\delta = \frac{\Delta c(\mathbf{x}, y)}{c_{n-1}(\mathbf{x}, y)}. \quad (19)$$

Since $\Delta c(\mathbf{x}, y)$ is determined by the number of PCR cycles for signal amplification, the reproduction rate δ can be set indirectly by controlling the number of PCR cycles or its fraction.

5. SIMULATION RESULTS AND DISCUSSION

Microarray gene expression data are used to evolve molecular genetic programs for making diagnosis based on DNA. Since molecular programming is based on DNA molecules, it is very natural to solve the problem based on DNA. Microarrays or DNA chips are a new technology for measuring gene expression intensities at the cDNA (i.e. the DNA sequence complementary to the mRNA sequence) level. Gene expression data are collected from microarray experiments for ALL/AML leukemia [3]. It should be noted that the microarray gene expression data contain much noise and this application can be a good test bed problem for the molecular programming approach against uncertainty.

The microarray data are preprocessed and 10 genes were selected out of 12600 genes. The genes are chosen according to the information gain measure for extracting features [11]. The training set consists of 120 examples each composed of 10 genes plus the associated leukemia class which is AML or ALL. A 6-fold cross-validation is used for testing the performance. That is, the whole data set of 120 examples is partitioned into 6 subsets and a total of six sessions were run, where each run used a subset of 20 examples for test and the remaining 100 examples (5 subsets) for training.

For the simulation of in vitro evolution of the molecular genetic programs, the population size of $118,096 \times 10^6 \approx 1.2 \times 10^{11}$ was used, where 118,096 is the number of different library elements and 10^6 is the number of their copies. The library was initialized to contain each and every conjunction of order 1 through 10. These include $(x_1 = 0, y = 0), (x_1 = 0, y = 1), (x_1 = 1, y = 0), (x_1 = 1, y = 1), (x_1 = 0, x_2 = 0, y = 0), (x_1 = 0, x_2 = 0, y = 1), (x_1 = 1, x_2 = 0, y = 0), ...$ Thus, the total number of the different library elements is

$$N = \sum_{k=1}^{10} {}_{10}\mathrm{C}_k \cdot 2^k \cdot 2 = 118,096 \quad (20)$$

where the notation ${}_{10}\mathrm{C}_k$ denotes the number of combinations to choose k variables out of 10. Each of the elements is our genetic program.

The questions we are interested to address in the simulations are:

- Does the molecular GP process converge to the best solution available by the training data?
- If yes, how fast is the convergence? What's the effect of reproduction rate?
- What is the effect of program size and its variability on accuracy?
- Can the method evolve compact genetic programs if they exist?
- Does the huge population size really contribute to the accuracy and robustness of the genetic programs?

Figure 5 shows the evolution of the fitness as generation goes on. Fitness was measured each generation when a new training example was observed. One sweep through the training set constitutes an *epoch* which is equivalent to 100 generations in this experiment. The best accuracy of approximately 90% was obtained in 7 epochs.

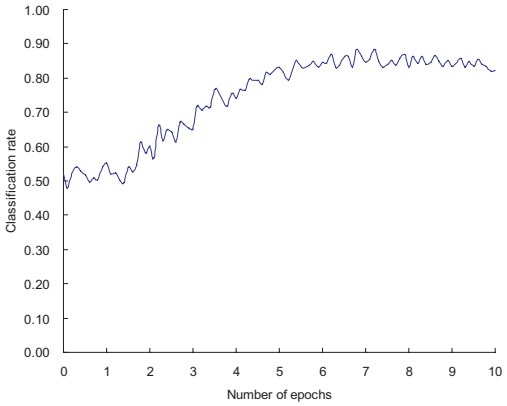

Figure 5: Fitness evolution of the population of molecular genetic programs. Shown are the average classification rates over the 6-fold cross-validation runs. Though there are fluctuations the fitness values tend to converge 90 % accuracy. The reproduction rate was 0.01.

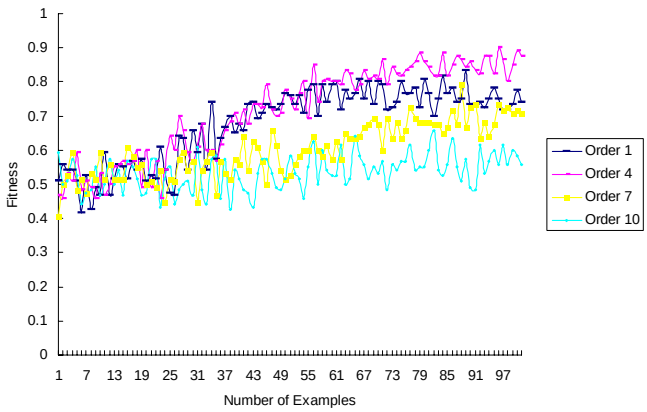

Figure 6: Fitness curves for runs with fixed-size programs. Shown are average fitness values for runs with programs of fixed-order 1, 4, 7, and 10.

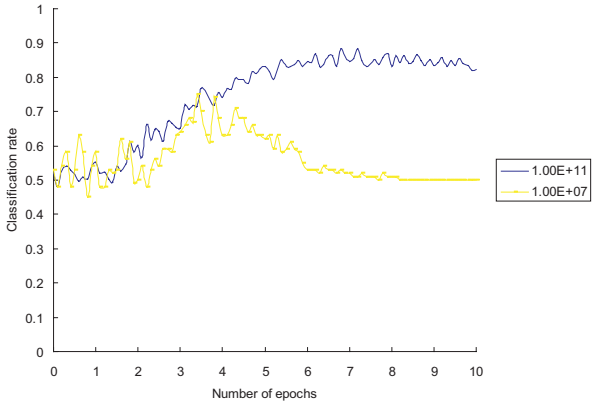

Figure 7: Effect of population size on ensemble performance. Shown are the best-fitness curves for population sizes of 10^{11} (in our experiments) and 10^7 (subsampling case for testing). The results show that too much subsampling degrades the performance.

The effect of size-variability was investigated by comparing the result above against the GP runs with fixed-size programs. Figure 6 shows the fitness curves for the 4 runs with programs of fixed order 1, 4, 7, and 10. It is observed that most of the cases the fixed-length programs achieve lower-accuracy. Only the run for fixed-order of 4 obtained a relatively good accuracy which is still inferior to the run for variable-length programs. This seems attributed to the fact that the population of variable-length programs has stronger expressive power than that of fixed programs.

One of the natural properties we get through molecular computing technology in our genetic programming is the huge population size. Baeck et al. discusses the effect of population size in a simple GA context [1]. To test the effect of large population sizes on our variable-length EC, various runs with varying library sizes were compared. We set all the other parameters the same as the other experiments and changed the population size by resampling (undersampling or oversampling, depending on the necessity) the programs in the population. Figure 7 compares the fitness curves when the library size was reduced from 10^{11} to 10^7. It indicates that too much reduction in population size hurts the stability of the evolutionary algorithm. To put in other way, a large population size helps improve the performance. This seems because in our ensemble approach the number of copies reflects the strength of each genetic program in the voting for the classification (diagnosis in this case). Larger population has better chance of fine-controlling the probability distribution of genetic programs.

Figure 8 shows the change of program sizes during a run. Shown are the distributions of program sizes at generations of 0, 5, 10. The population at generation 0 is bell-shaped with peak at size $k = 7$ where the total number of variable combinations is the largest. Note that there are only very small number of small programs at generation 0. This is because the population was initialized according to the possible number of programs, i.e. ${}_{10}\mathrm{C}_k \cdot 2^k \cdot 2$ with order k. As generation goes on, the number of large programs decreases and the number of small programs increases. This shows the tendency of the evolutionary process utilizing the lower-order decision lists. Note that this size shift is relatively smooth. This seems because the ensemble approach tries to find the best population as opposed to the best individual.

It should be mentioned that the control of evolution might be trickier if the evolution allows for variable-size programs. In our experiments, this has to do with the initialization of population. If the population is initialized with the same copies of each distinct length of program, it will have more of large programs and less of small ones. For example, there are ${}_{10}\mathrm{C}_1 \cdot 2^1 \cdot 2 = 40$ programs of order 1 and there are ${}_{10}\mathrm{C}_4 \cdot 2^4 \cdot 2 = 6720$ programs of order 4.

6. CONCLUSION

We presented an evolutionary method, called molecular programming (MP), for learning genetic programs using DNA computing technology. A novel representation method is introduced that makes use of the molecular DNA structures

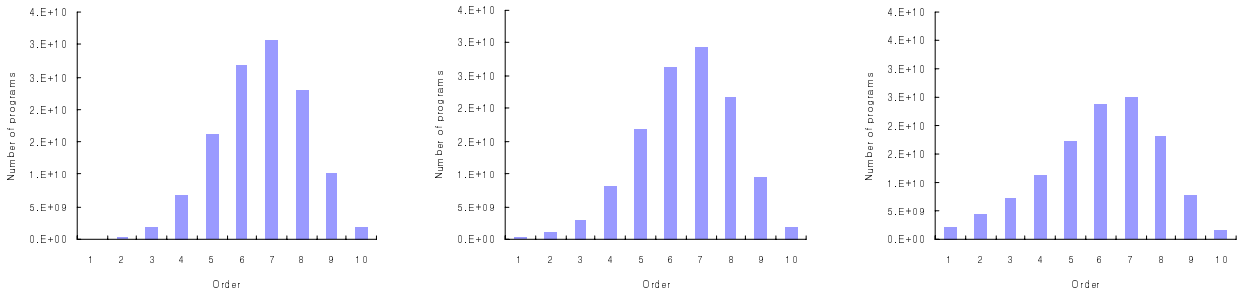

Figure 8: Distribution of the size of genetic programs. Shown are the number of programs of each size in the final population in a run. It shows the tendency that, as generation goes on, smaller programs are used more frequently than larger ones. From left to right the epoch number is 0, 5 and 10 where one epoch consists of 100 generations. The reproduction rate was 0.01.

while maintaining the advantages of variable-length encoding capability of genetic programming (in contrast to other evolutionary computation methods). A molecular evolutionary algorithm is described that makes use of the biochemical techniques for in vitro molecular computing. These include, for example, the massively parallel matching and selection based on the A-T and G-C molecular recognition capability of DNA molecules, the PCR-based exponential reproduction of fitter programs, and the global search capability coming from the population size. Also a DNA-computing-based voting method that allows for robust decision-making on the basis of the huge population is presented. This additional feature does not involve much overhead since the huge number of simple random genetic programs can be obtained for almost free in DNA computing.

The simulation results on the leukemia cancer diagnosis problem show that effective learning is possible using the molecular "genetic" programming method. This is in some sense surprising considering the fact that each genetic program is of simple structure. Our analysis shows that even though the individual programs are simple, their collection as a whole has a powerful representation capability equivalent to the disjunctive normal form. This explains the high accuracy of the diagnosis results. It is also remarkable that the genetic programs are composed of boolean variables and still the results are relatively robust against noise and incomplete information. This seems attributed to the redundancy of the library representation of DNA-coded decision lists. This redundancy comes naturally from the big population size. It seems an interesting future work to study the theoretical connection between the molecular programming algorithms and the bagging algorithms [2] in the sense both algorithms make use of a large number of weak learners to make robust decisions.

It should be noted that the presented algorithms are designed to be implemented eventually in wet DNA technology. In fact, our Molecular Evolutionary Computing (MEC) project (years 2000-2010) aims to advance this technology and its applications using microfluidics-based lab-on-a-chip. There remains several details at the biochemical and physicochemical level that should be taken into account in simulations to reflect the wet-lab realities. These include, for example, the undesirable mismatches in DNA hybridization and the control of temperatures and concentrations for reaction.

7. ACKNOWLEDGEMENTS

This research was supported by the National Research Laboratory Program of the Ministry of Science and Technology, by the Next Generation Technology Program of the Ministry of Industry, Commerce and Energy, and by the BK-21 Program of the Ministry of Education. The authors thank Sung-Kyu Kim and Jinhan Kim for assistance in preparing the manuscript.

8. REFERENCES

[1] T. Baeck, J. N. Kok, G. Rozenberg, Evolutionary computation as a paradigm for DNA-based computing, *Evolution as Computation*, L. F. Landweber and E. Winfree (Eds.), Springer-Verlag, pages 15-40, 2002.

[2] L. Breiman, Bagging predictors, *Machine Learning*, 24:123-140, 1996.

[3] M. H. Cheok, W. Yang, C.H. Pui, J. R. Downing, C. Cheng, C. W. Naeve, M. V. Relling, and W. E. Evans, Treatment-specific changes in gene expression discriminate in vivo drug response in human leukemia cells, *Nature Genetics*, 34:85-90, 2003.

[4] D. Kalyanmoy et al. (Eds.), *Genetic and Evolutionary Computation - GECCO 2004: Genetic and Evolutionary Computation Conference*, LNCS 3103, 2004.

[5] M. Kloster and C. Tang, Simulation and analysis of in vitro DNA evolution, *Phys. Rev. Lett.*, 92(3), 2004.

[6] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA, 1992.

[7] W. B. Langdon, *Data Structures and Genetic Programming*, Kluwer, 1998.

[8] J.-Y. Lee, S.-Y. Shin, T.-H. Park, and B.-T. Zhang, Solving traveling salesman problems using DNA molecules encoding numerical values, *BioSystems*, 78:39-47, 2004.

[9] H.-W. Lim, J.-E. Yun, H.-M. Jang, Y.-G. Chai, S.-I. Yoo, and B.-T. Zhang, Version space learning with

DNA molecules, *DNA Computing 8*, LNCS 2568:143-155, 2003.

[10] S. Luke and L. Panait, Fighting bloat with nonparametric parsimony pressure, *Parallel Problem Solving from Nature - PPSN VII*, LNCS 2439, Springer-Verlag, pages 411-421, 2002.

[11] T. M. Mitchell, *Machine Learning*, The McGrow-Hill Companies, Inc., 1997.

[12] P. Nordin, W. Banzhaf, and F. Francone, Efficient evolution of machine code for CISC architectures using blocks and homologous crossover, *Proc. Third Annual Genetic Programming Conference (GP-99)*, L. Spector, W. Langdon, U.-M. O'Reilly and P. Angeline (eds.), pages 275-299, Morgan Kaufmann, 1999.

[13] U.-M. O'Reilly, P. Testa, S. Greenwold, and M. Hemberg, Agency-GP: Agent-based genetic programming, *GECCO-2001 Late Breaking Papers*, 2001.

[14] S.-Y. Shin, I.-H. Lee, D. Kim, and B.-T. Zhang, Multi-objective evolutionary optimization of DNA sequences for reliable DNA computing, *IEEE Transactions on Evolutionary Computation*, 2004 (to appear).

[15] T. Soule and J. Foster, Effects of code growth and parsimony pressure on populations in genetic programming, *Evolutionary Computation*, 6(4):293-309, 1998.

[16] L. Spector, *Automatic Quantum Computer Programming: A Genetic Programming Approach*, Boston, MA: Kluwer Academic Publishers, 2004.

[17] D. S. Wilson and J. W. Szostak, In vitro selection of functional nucleic acids, *Ann. Rev. Biochem.*, 68:611-647, 1999.

[18] M. C. Wright and G. F. Joyce, Continuous in vitro evolution of catalytic function, *Science*, 276:614-617, 1997.

[19] R. Yokobayashi, R. Weiss, and F. H. Arnold, Directed evolution of a genetic circuit, *Proc. Natl. Acad. Sci. USA*, 99(26):16587-16591, 2002.

[20] B.-T. Zhang and J.-G. Joung, Building optimal committees of genetic programs, *Parallel Problem Solving from Nature 2000*, LNCS 1917:231-240, 2000.

[21] B.-T. Zhang, A unified Bayesian framework for evolutionary learning and optimization, *Advances in Evolutionary Computation*, Chapter 15, pages 393-412, Springer-Verlag, 2003.

[22] B.-T. Zhang and H.-Y. Jang, A Bayesian algorithm for in vitro molecular evolution of pattern classifiers, *DNA Computing 10*, LNCS 3384:458-467, 2005.