A Bayesian Algorithm for In Vitro Molecular Evolution of Pattern Classifiers

Byoung-Tak Zhang[1,2] and Ha-Young Jang[1]

[1] Biointelligence Laboratory, Seoul National University, Seoul 151-742, Korea
{btzhang, hyjang}@bi.snu.ac.kr
http://bi.snu.ac.kr/
[2] Computer Science and Artificial Intelligence Laboratory (CSAIL), MIT, Cambridge, MA 02139

Abstract. We use molecular computation to solve pattern classification problems. DNA molecules encode data items and the DNA library represents the empirical probability distribution of data. Molecular bio-lab operations are used to compute conditional probabilities that decide the class label. This probabilistic computational model distinguishes itself from the conventional DNA computing models in that the entire molecular population constitutes the solution to the problem as an ensemble. One important issue in this approach is how to automatically learn the probability distribution of the DNA-based classifier from observed data. Here we develop a molecular evolutionary algorithm inspired by directed evolution, and derive its molecular learning rule from Bayesian decision theory. We investigate through simulation the convergence behaviors of the molecular Bayesian evolutionary algorithm on a concrete problem from statistical pattern classification.

1 Introduction

Pattern classification is a classical and fundamental problem in artificial intelligence and machine learning with a wide range of applications including computer vision, data mining, information retrieval, and bioinformatics. The task of a pattern classifier is to assign a class to an input pattern. A variety of pattern classification techniques have been developed so far (see, for example, [4]). This paper explores the potential of molecular computing to solve the computational problems involved with pattern classification. We focus on the probabilistic formulation of the pattern classification problem. The objective is to build a joint probability $P(X, Y)$ of input pattern X and output class Y. Once this model is constructed, class decisions can be made by computing the conditional probabilities, such as $P(Y|X)$. In doing so, we make use of DNA-based molecular computation.

We use a library (as test tube or in some other format) of DNA molecules to represent the probability distribution of data. Each molecule encodes an instance of training data and the frequency of molecules is proportional to the probability of observing the patterns stored in the library. This makes the whole library of

molecules a probabilistic pattern classification device. In this paper we develop a molecular algorithm for learning probabilistic pattern classifiers. It is motivated by in vitro evolution [15] and we derive from Bayesian decision theory a rule for setting the learning parameters for evolving the classifier using a probabilistic DNA-library.

The paper is organized as follows. In Section 2 we describe the probabilistic approach to pattern classification and provide the probability-theoretical basis of using DNA molecules for solving this problem. Section 3 discusses molecular computation of probability functions related with pattern classification. Section 4 addresses the learning problem and presents a molecular algorithm inspired by in vitro evolution. Section 5 shows the mathematical background of the molecular algorithm by deriving its learning rule. Section 6 presents and discusses simulation results. Section 7 draws conclusions.

2 Pattern Classification and DNA Molecules

The aim is to build a pattern classification system f that outputs a label y given an input pattern $\mathbf{x} = (x_1, ..., x_n)$, i.e. $f(\mathbf{x}) = y$. In this paper we restrict ourselves to the case of binary variables. It is convenient to assume there exists a (unknown) target system f^* as an ideal model for f. We also assume f^* behaves according to the probability distribution $P^*(Y|\mathbf{x})$, but the exact form of probability model is unknown. The only information we have is data collected from the input-output pairs of f^*.

DNA computing provides a promising approach to realizing the pattern classifier. We can represent each input pattern as a sequence of A, T, G, and C. The output label can also be encoded as a DNA sequence. For example, if we use 10-mer to encode each binary variable and if there are 30 variables for input pattern and one variable for class label, then DNA molecules of length 310-mer can represent an instance of the training example. The training patterns can be stored as a DNA library and given a query pattern $\mathbf{x}_q$, the molecular pattern classifier compares every library element against $\mathbf{x}_q$ to make class-label decisions. The use of DNA molecules as a memory device and the use of biochemical techniques for memory storage and retrieval provides interesting properties, such as massive parallelism, associative search, fault-tolerance, and miniaturization [2, 3, 6].

The method we present here is in some sense an extension of the memory-based learning (MBL) approach [8]. In MBL, the training examples are stored one copy for each instance. Here, we use many copies for each training example. The number of copies of library elements is updated as new training examples are observed so that their frequency is proportional to their probability of observation. MBL does rote-learning and thus very fast in storage, but very slow in recall since classification computation is done from scratch. Here we update the probabilistic library on learning. On recall our method works like a look-up table, but the probabilistic distribution of the data in the library facilitates clas-

sification computation. Keeping multiple copies of data items can also contribute to the robustness and fault-tolerance of the molecular computing system [9, 13]

The frequentist interpretation of probability builds the theoretical basis of our probabilistic molecular pattern classification model. The basic event in molecular pattern classification involves DNA-hybridization reactions. If n is the total number of DNA strands in the library and n_A is the number of strands with pattern (or hybridization event) A, then the probability of A is defined as the limit

$$P(A) = \lim_{n \to \infty} \frac{n_A}{n}. \tag{1}$$

In molecular computation, the accuracy and reliability of the probability values are supported by the Avogadro-scale number of molecules for representing the population. This offers a novel way of representing the probability distribution of data.

3 Molecular Computing for Pattern Classification

In the previous section we described how to represent the probability distribution using DNA molecules. Essentially, the DNA library represents the joint probability $P(X, Y)$ of the input pattern X and the output class Y. In this section we discuss how the class label can be computed.

One criterion to determine the class label is the maximum a posterior (MAP) decision rule. Here, the classifier computes the probability of each class conditional on the input pattern $\mathbf{x}$, and decides as output the class whose conditional probability is the highest, i.e.

$$y^* = \arg \max_{Y \in y_0, y_1} P(Y|\mathbf{x}) \tag{2}$$

where y_0 and y_1 are the candidate classes (for simplicity, we deal with here the case of binary classes, however, the method is generalizable to an arbitrary number of classes). Specific techniques differ in the methods how to model the probability distribution $P(Y, \mathbf{x})$ and how efficiently to compute the necessary probability values.

Here we use a molecular computational method for pattern classification using the probabilistic DNA-library. Given a query pattern $\mathbf{x}$ we extract from the library all the molecules that match the query. These molecules will have class labels from which we decide the majority label as the class of the query pattern. A class label is a sequence appended to denote the class to which the pattern belongs. The extraction may involve some mismatches due to the potential for formation of double-stranded DNA duplexes. There are a lot of work going on to design the sequences and codeword sets (see, for example, [11] 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.

The molecular algorithm for computing the class labels can be summerized as follows.

- 1. Let the library L represent the current empirical distribution $P(X, Y)$.
- 2. Present an input (query) pattern $\mathbf{x}$.
- 3. Classify $\mathbf{x}$ using L as follows:
  - 3.1 Extract all molecules matching $\mathbf{x}$ into M.
  - 3.2 From M separate the molecules into classes:
    - Extract the molecules with label $Y = y_0 = 0$ into M^0.
    - Extract the molecules with label $Y = y_1 = 1$ into M^1.
  - 3.3 Compute $y^* = \arg\max_{Y \in \{0,1\}} |M^Y|/|M|$.

In Step 3.1, note that the count $c(\mathbf{x})$ of $\mathbf{x}$ in M approximates the probability of observing the pattern which is called evidence:

$$c(\mathbf{x})/|L| = |M|/|L| \approx P(\mathbf{x}). \tag{3}$$

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:

$$c(Y|\mathbf{x})/|M| = |M^Y|/|M| \approx P(Y|\mathbf{x}). \tag{4}$$

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

$$y^* = \arg \max_{Y \in \{0,1\}} c(Y|\mathbf{x})/|M| = \arg \max_{Y \in \{0,1\}} c(Y|\mathbf{x}) \approx \arg \max_{Y \in \{0,1\}} P(Y|\mathbf{x}) \tag{5}$$

It is worth noting that for classification purposes only the relative frequency or concentration of the molecular labels are important.

4 Molecular Computing for Pattern Learning

In the previous section we assumed 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. Our update procedure is motivated from in vitro evolution [15, 12]. In vitro evolution starts with a library of molecules and evaluates their goodness. Then, the fitter ones are selected as the basis for generating mutants that build the next generation of library. The iteration of the selection-amplification cycle can come up with the identification of molecules that best fits to the target function. In vitro evolution has been used to identify active compounds from composite mixtures [7]. In recent years, a number of methods have been developed to isolate molecules with desired functions from libraries of small organic molecules, nucleic acids, proteins, peptides, antibodies or single-chain antibody fragments, or other polymers [5]. In vitro evolution has also been used to design genetic circuits [16], finite state machines [10], and game programs [14].

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, we extract from the library the patterns matching $\mathbf{x}$. 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 duplicated: $L \leftarrow L + \{(\mathbf{u}, v)\}$. Optionally, if the label v is incorrect, i.e. $v \neq y$, the matching library pattern is removed from the library: $L \leftarrow L - \{(\mathbf{u}, v)\}$. The update of library in this way more or less like evolutionary computation [1, 17] 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 procedure since the library improves its classification performance as new examples are presented.

The molecular algorithm for the whole evolutionary learning procedure is summarized as follows.

- 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.

In Step 4, $\Delta c(\mathbf{u}, v)$ denotes the number of copies of $(\mathbf{u}, v)$. It should be noted that here we make use of the fact that the addition or removal operation can be performed in parallel in DNA computing. 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. For effective implementation of the learning procedure, experimental protocols should consider advanced techniques for improving extraction efficiencies, such as the refinery or super-extract model. Note also that the learning rule has a parameter Δc that reflects the strength of learning for each training example. How to set this parameter will be addressed in the next section.

5 Derivation of Bayesian Update Rule

What is the theoretical basis of the molecular evolutionary learning algorithm described in the previous section? We consider the evolution of the probability

distribution in the library L. Let L_0 denote the initial library and let its probability distribution be $P_0(X, Y)$. As the nth training example $(\mathbf{x}, y)$ is observed, L_{n-1} is updated to L_n. Thus, the general form of the learning rule is written as $L_n \leftarrow L_{n-1} + \{\Delta c(\mathbf{x}, y)\}$, where all the class-dependent updates are integrated into one term $\{\Delta c(\mathbf{x}, y)\}$. This update entails the revision of the distribution $P_{n-1}(X, Y|\mathbf{x}, y)$ of L_{n-1} into $P_n(X, Y|\mathbf{x}, y)$ of L_n. Thus, in terms of probability the learning rule is rewritten as

$$P_n(X, Y|\mathbf{x}, y) = (1 + \delta)P_{n-1}(X, Y|\mathbf{x}, y), \tag{6}$$

where δ is a learning parameter determining the strength of update.

Using the Bayes rule we can write $P(X, Y|\mathbf{x}, y) = \frac{P(\mathbf{x},y|X,Y)P(X,Y)}{P(\mathbf{x},y)}$. Note that $P(X, Y|\mathbf{x}, y) = P_n(X, Y|\mathbf{x}, y)$ and $P(X, Y) = P_{n-1}(X, Y|\mathbf{x}, y)$. Thus, inserting Eqn. (6) into the Bayes rule we get

$$(1 + \delta)P_{n-1}(X, Y|\mathbf{x}, y) = \frac{P(\mathbf{x}, y|X, Y)P_{n-1}(X, Y|\mathbf{x}, y)}{P(\mathbf{x}, y)}. \tag{7}$$

Therefore,

$$\delta = \frac{P(\mathbf{x}, y|X, Y) - P(\mathbf{x}, y)}{P(\mathbf{x}, y)}. \tag{8}$$

This indicates that the molecular algorithm follows the Bayesian evolutionary update rule [17].

Setting the parameter δ is important to balance the adaptability and stability of the molecular library as a probabilistic model for the data. The larger the value of δ is, the larger gets the change of the distribution. An alternative way to control δ is via the number of copies Δc which dictates how many copies of the current example should be amplified. To see the influence of δ-value on learning effect in terms of Δc, we use the frequentist interpretation to express the probability:

$$P_{n-1}(X, Y|\mathbf{x}, y) \cong c_{n-1}(\mathbf{x}, y)/|L| \qquad P_n(X, Y|\mathbf{x}, y) \cong c_n(\mathbf{x}, y)/|L| \tag{9}$$

where $c_{n-1}(\mathbf{x}, y)$ and $c_n(\mathbf{x}, y)$ are the number of copies of $\mathbf{x}$ before and after the example $(\mathbf{x}, y)$ is presented, respectively. The size $|L|$ of the library is assumed to remain constant. The difference in probability is expressed as

$$\delta P_{n-1}(X, Y|\mathbf{x}, y) = P_n(X, Y|\mathbf{x}, y) - P_{n-1}(X, Y|\mathbf{x}, y) = \frac{c_n(\mathbf{x}, y) - c_{n-1}(\mathbf{x}, y)}{|L|} \tag{10}$$

and from this we have

$$\delta = \frac{\Delta c(\mathbf{x}, y)}{c_{n-1}(\mathbf{x}, y)}. \tag{11}$$

The above equation shows the relationship of the probability amplification factor δ to the number $\Delta c(\mathbf{x}, y)$ of additional copies of molecules. It is interesting that δ is expressed as the amplification ratio of current copies $c_{n-1}(\mathbf{x}, y)$ since this is equivalent to the number of PCR cycles for signal amplification. Thus, the learning parameter δ can be set indirectly by controlling the number of PCR cycles or its fraction.

6 Simulation Results and Discussion

We performed simulations to study the properties of the molecular algorithms before we realize them with wet DNA. The questions we are interested in are:

- Does the evolutionary learning process converge to the best solution available by the training data?
- If yes, how fast is the convergence? And how to control the learning rate?
- To what extent is the molecular algorithm robust against external perturbation?

We use the majority function of input size $n = 13$. This binary function returns 1 if the input pattern contains 6 or more 1s, otherwise it returns 0. The size of the DNA library was 2^{14}. Theoretically, this covers the entire space of examples for this problem (13 inputs plus 1 output). However we generated the library randomly, so the example space is covered only statistically. We simulate this setting considering the case where DNA computation can not cover the whole problem space, which is true in practice.

Initially, an average of K copies of each instance are generated in the library. We experimented on various K values, ranging from 10 to 10^4. Then, learning proceeds by presenting training patterns. We use a training set of N examples and an independent set of N examples for testing the performance of the molecular pattern classifiers. Typically, N ranged from 500 to 1500. It is important to note that though the training set of size N is given, the learning process is on-line, i.e., the probability distribution of the DNA library is updated before the next example is presented.

To answer the questions mentioned above, we ran the experiments by changing the learning-rate parameter δ. We also changed the example presentation sequences. In one set of runs we presented the positive training example and the negative example alternatingly. This is more reasonable procedure since there is the same number of positive examples and negative examples in the majority function. In another set of runs we presented two positive examples and then one negative example alternatingly. This is to test for the robustness of the molecular learning process against some external, statistical perturbation. These sets of experiments were combined with the varying values of δ.

Figure 1 shows the learning behavior of the algorithm. For this simulation experiment, the learning rate was $\delta = 0.1$ and the DNA library maintained 1000 copies of molecules for each training instance. The examples were presented randomly, alternating a positive and a negative example. The training set size was 800, thus this graph shows the learning curve for the first two (random) sweeps through the training set. We observe a monotonic increase in classification rate which was evaluated on a test set of the same size but independent of the training set.

To see the effect of the learning rate on the convergence we ran the same experiment by changing $\delta = 1.0$. The result is shown in Figure 2. It is observed that the classification performance steadily increases until the completion of the

first sweep of the training set, after which the performance degrades. This indicates that the learning rate was set too big. Since we maintained 1000 copies of DNA molecules for each example pattern in this experiment, this means making 1000 copies to learn a single training example may lead to overfitting.

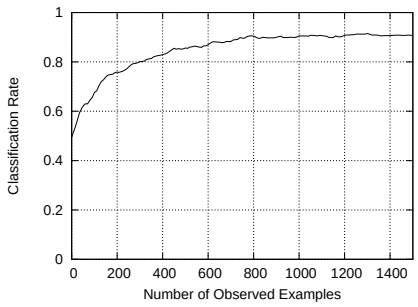

Fig. 1. Learning curve of the simulated DNA-computing classifier: $\delta = 0.1$

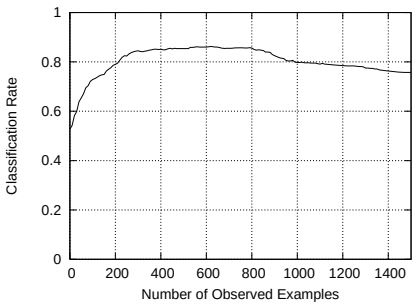

Fig. 2. Learning curve of the simulated DNA-computing classifier: $\delta = 1.0$

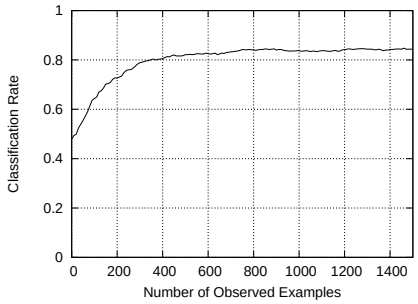

Fig. 3. Learning curve of the simulated DNA-computing classifier: $\delta = 0.1$ with alternating presentations of two positive examples and one negative example.

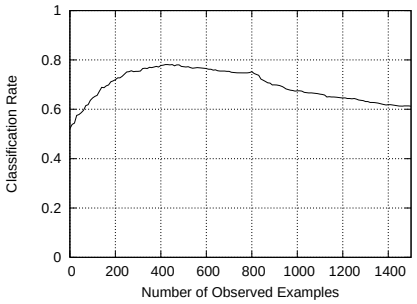

Fig. 4. Learning curve of the simulated DNA-computing classifier: $\delta = 1.0$ with alternating presentations of two positive examples and one negative example.

We varied the example presentation order to see the effect of statistically perturbed training sequence. Figures 3 and 4 depict the results which replace the experiments for Figures 1 and 2 by presenting two positive examples and then one negative example alternatingly (and randomly). We observe lower classification performance for these cases than the cases for presenting one positive and one negative example alternatingly. However, when the learning rate is small, the performance still increases steadily, showing robustness against external statistical perturbation. We see some overfitting or instability when the learning rate

is too big. For instance, in this run the performance starts to get lower earlier than when the training set correctly reflects the probabilistic distribution of the data space.

To summarize, the simulation results show the learning behavior is relatively stable against the learning rate parameter δ in the range of $0 \leq \delta \leq 1.0$ for the experimental settings we studied. Generally, when δ is set too big, the performance may degrade after a sweep through the training set or later. The system was also relatively robust against the statistical perturbation of example presentation sequences. This seems due to the fact that our learning algorithm has an error-correction component (matched wrong-answering molecules are not copied). Finally, it should also be noted that removing incorrect-answering examples from the library can speed up the learning process, but may lead to instability of the convergence behavior (results not shown). Thus, it should be used with care and only if necessary.

7 Conclusion

We presented a DNA computing algorithm that evolves probabilistic pattern classifiers from training data. Based on Bayesian theory we derived the rule for determining the learning-rate parameter and showed this is related to the number of copies of matched molecules in the DNA library.

We performed simulations to evaluate the performance and stability of the molecular Bayesian evolutionary algorithm. The convergence of the algorithm was quite stable, considering the statistical bias coming from the small number of training examples in our experimental setting. This seems attributed to the probabilistic nature of our molecular computing based on the huge number of molecules to represent the statistical distributions of data. It was also observed that the convergence was relatively stable against external perturbations such as the presentation sequence of training examples.

Our work shows that molecular computation provides several interesting properties for probabilistic computation in general and for pattern classification in specific. The most important property the present work is exploiting, and thus our simulated molecular pattern classifiers are to be useful, is the huge number of molecular-scale data items combined with the highly-parallel molecular recognition mechanism which provides the theoretical and technological basis for the probabilistic DNA library.

Acknowledgements

Thanks are due to Bruce Tidor and the members of the Laboratory for Molecular Science at MIT CSAIL for many fruitful discussions. This research was supported by the National Research Laboratory Program of the Korea Ministry of Science and Technology and by the Next Generation Technology Program of the Korea Ministry of Industry and Commerce.

References

1. Baeck, T., Kok, J.N., Rozenberg, G., "Evolutionary computation as a paradigm for DNA-based computing," *Evolution as Computation*, Landweber, L.F. and Winfree, E. (Eds.), Springer-Verlag, pp. 15-40, 2002.
2. Baum, E. B., "Building an associative memory vastly larger than the brain," *Science*, 268:583-585, 1995.
3. Chen, J. Deaton, R. and Wang, Y.-Z., "A DNA-based memory with in vitro learning and associative recall," *Proc. 9th Annual Meeting on DNA-Based Computers*, pp. 127-136, 2003.
4. Duda, R.O., Hart, P.E., and Stork, D.G., *Pattern Classification*, 2nd Ed. Wiley, 2001.
5. Famulok, M. and Verma, S., "In vivo-applied functional RNAs as tools in proteomics and genomics research," *Trends in Biotechnology*, 20(11):462-466, 2002.
6. Garzon, M. Bobba, K. and Neel, A., "Efficiency and reliability of semantic retrieval in DNA-based memories," *Proc. 9th Annual Meeting on DNA-Based Computers*, pp. 137-149, 2003.
7. Landweber, L.F. and Pokrovskaya, I.D., "Emergence of a dual-catalytic RNA with metal-specific cleavage and ligase activities: The spandrels of RNA evolution," *Proc. Natl. Acad. Sci. USA*, 96:173-178, 1999.
8. Lim, H.-W., Yun, J.-E., Jang, H.-M., Chai, Y.-G., Yoo, S.-I., and Zhang, B.-T., "Version space learning with DNA molecules," *Lecture Notes in Computer Science*, 2568:143-155, 2003.
9. Reif, J. and LaBean, T. "Computationally inspired biotechnology: Improved DNA synthesis and associative search using error-correcting codes and vector quantization," *Lecture Notes in Computer Science*, 2054:145-172, 2001.
10. Rose, J. A., Deaton, R. J., Hagiya, M., Suyama, A., "A DNA-based in vitro genetic program", *Journal of Biological Physics*, 28:493-498, 2002.
11. Shin, S.-Y., Lee, I.-H., Kim, D. and Zhang, B.-T. "Multi-objective evolutionary optimization of DNA sequences for reliable DNA computing," *IEEE Transactions on Evolutionary Computation*, 2004 (to appear).
12. Wilson, D.S. and Szostak, J.W., "In vitro selection of functional nucleic acids," *Ann. Rev. Biochem.*, 68:611-647, 1999.
13. Winfree, E. and Bekbolatov, R. "Proofreading tile sets: Logical error correction for algorithmic self-assembly," Talk at *The 9th Annual Meeting on DNA-Based Computers*, Madison, WI, June 2003.
14. Wood, D. "DNA starts to learn Poker," Proc. DNA7, *Lecture Notes in Computer Science*, 2340:92-103, 2003.
15. Wright, M.C. and Joyce, G.F., "Continuous in vitro evolution of catalytic function," *Science*, 276:614-617, 1997.
16. Yokobayashi, R., Weiss, R., and Arnold, F.H., "Directed evolution of a genetic circuit," *Proc. Natl. Acad. Sci. USA*, 99(26):16587-16591, 2002.
17. Zhang, B.-T., "A unified Bayesian framework for evolutionary learning and optimization," *Advances in Evolutionary Computation*, Chapter 15, pp. 393-412, Springer-Verlag, 2003.