Evolving the Structure of Hidden Markov Models

Kyoung-Jae Won, Adam Prügel-Bennett, and Anders Krogh

***Abstract*—A genetic algorithm (GA) is proposed for finding the structure of hidden Markov Models (HMMs) used for biological sequence analysis. The GA is designed to preserve biologically meaningful building blocks. The search through the space of HMM structures is combined with optimization of the emission and transition probabilities using the classic Baum–Welch algorithm. The system is tested on the problem of finding the promoter and coding region of *C. jejuni*. The resulting HMM has a superior discrimination ability to a handcrafted model that has been published in the literature.**

***Index Terms*—Biological sequence analysis, genetic algorithm (GA), hidden Markov model (HMM), hybrid algorithm, machine learning.**

I. Introduction

HIDDEN MARKOV models (HMMs) are probabilistic finite-state machines used to find structures in sequential data. An HMM is defined by the set of states, the transition probabilities between states, and a table of emission probabilities associated with each state for all possible symbols that occur in the sequence. They were developed for use in speech applications, where they remain the dominant machine learning technique [1]. Over the past decade, they have also become an important tool in bioinformatics [2]. Their great attraction is that they allow domain information to be built into their structure while allowing fine details to be learned from the data through adjusting the transition and emission probabilities. The design of HMM architectures has predominately been the province of the domain expert.

Automatic discovery of the HMM structure has some significant attractions. By eliminating the expert, it allows the data to "speak for itself." This opens the possibility of finding completely novel structures, unfettered by theoretical prejudice. In addition, automation of the design process allows many more structures to be tested than is possible if every structure has to be designed by hand. However, automation also comes at a cost. Imposing a structure on the HMM limits the possible outcome (probability distribution for sequences) that can be learned. In the language of machine learning, imposing a structure reduces the learning capacity of the HMM. This has the benefit of reducing the estimation errors for the learned parameters given the limited amount of training data. Provided that the structure being imposed faithfully captures some biological constraints, it should not significantly increase the approximation error. As the generalization error is the sum of the estimation and approximation error, imposing a meaningful structure on the HMM should give good generalization performance. By automatically optimizing the HMM architecture, we are potentially throwing away the advantage over other machine learning techniques, namely their amenability to incorporate biological information in their structure. That is, we risk finding a model that "overfits" the data.

The aim of the research presented in this paper is to utilize the flexibility provided by genetic algorithms (GAs) to gain the advantage of automatic structure discovery, while retaining some of the benefits of a hand-designed architecture. That is, by choosing the representation and genetic operators, we attempt to bias the search toward biologically plausible HMM architectures. In addition, we can incorporate the Baum–Welch algorithm which is traditionally used to optimize the emission and transition probabilities as part of the GA. GAs appear to be very well suited to this application. The optimization of HMM architectures is a discrete optimization problem which is easy to implement in a GA. We can simultaneously optimize the continuous probabilities by hybridizing the GA with the Baum–Welch. Furthermore, GAs allow us to tailor the search operators to bias the search toward biologically plausible structures. This would be far harder to accomplish in a technique such as simulated annealing, where the freedom to choose the move set is often constrained by the wish to maintain detailed balance or some similar condition.

HMMs have received little attention from the evolutionary computing community. We are aware of only two other groups that have used a GA to optimize HMMs, and neither has published in evolutionary computing journals [3], [4] (we discuss the relation of this work to our own in the next section). This is, perhaps, surprising considering the large amount of work on using GAs to optimize the structure of neural networks (see, for example, the review [5]) and more recent work on graphical models [6], [7]. We believe that GAs may be an important tool for evolving HMM architectures. Furthermore, there may be lessons about guiding search GAs to be learned from this application.

The rest of this paper is organized as follows. In the next section, we give a brief review of HMMs. We include this section to make the paper self-contained for readers who are not familiar with HMMs. It also serves to define the notation we use later on in this paper. More detailed pedagogical reviews are given in [1] and [2]. In Section III, we describe our GA for optimizing HMMs. Section IV describes the experiments to test our GA. We conclude in Section V.

Manuscript received July 19, 2004; revised November 23, 2004.

K.-J. Won and A. Prügel-Bennett are with the School of Electronic and Computer Science, University of Southampton, Southampton S017 1BJ, U.K. (e-mail: j.won@ecs.soton.ac.uk).

A. Krogh is with the Bioinformatics Center, University of Copenhagen, Copenhagen 2100, Denmark.

Digital Object Identifier 10.1109/TEVC.2005.851271

1089-778X/$20.00 © 2006 IEEE

II. Background

A. HMMs

An HMM is a learning machine that assigns a probability to an observed sequence of symbols. A sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$ consist of symbols x_t belonging to some alphabet $\mathcal{S}$. In biological sequence analysis, the alphabet might be, for example, the set of four possible nucleotides in DNA, "A," "C," "G," and "T," or the set of 20 amino acids that are the building blocks of proteins. We denote the set of parameters that define an HMM by Θ. Given a sequence $\boldsymbol{x}$, an HMM returns a "probability" $\mathbb{P}(\boldsymbol{x}|\Theta)$, where

$$\sum_{\boldsymbol{x}\in\mathcal{S}^T} \mathbb{P}(\boldsymbol{x}|\Theta) = 1$$

so it is a probability distribution over sequences of length T (we use $\mathcal{S}^T$ to denote the set of all sequences of length T). To understand the meaning of this probability, we can imagine some process of interest $\mathcal{P}$ (e.g., molecular evolution) that generates a set of sequences of length T with probability $\mathbb{P}(\boldsymbol{x}|\mathcal{P})$. Our aim is to find an HMM Θ such that $\mathbb{P}(\boldsymbol{x}|\Theta)$ is as close as possible to $\mathbb{P}(\boldsymbol{x}|\mathcal{P})$. Of course, we usually do not know $\mathcal{P}$. Rather, we have some training examples consisting of a set of sequences. We can then use the maximum likelihood principle to estimate the HMM, which corresponds to maximizing $\mathbb{P}(\boldsymbol{x}|\Theta)$ with respect to Θ (the probability $\mathbb{P}(\boldsymbol{x}|\Theta)$ is known as the *likelihood*, when considered a function of Θ).

Often, we are interested in how well a sequence fits the model. To do this, we consider the log-odds of a sequence

$$\log - \text{odd for sequence} = \log\left(\frac{\mathbb{P}(\boldsymbol{x}|\Theta)}{|\mathcal{S}|^{-T}}\right) \tag{1}$$

where $|\mathcal{S}|$ denotes the cardinality of the set of symbols $\mathcal{S}$. The log-odds are positive if the sequence is more likely than a random sequence. To use an HMM for classification, we can set a threshold for the log-odds of a new sequence to belong to the same class as the training data.

The HMM is a probabilistic finite-state machine which can be represented as a directed graph in which the nodes correspond to states and the edges correspond to possible transitions between states. A transition probability is associated with each edge with the constraint that the sum of the transition probabilities for edges exiting a node must sum to one. A node may have a transition to itself. In addition, there is an emission probability table associated with each state which encodes the probability of each symbol $a \in \mathcal{S}$ being "emitted," given that the machine is in that state. We define one state as the start state, which does not emit a symbol but has transitions to the other states. To compute probabilities from our HMM we consider an "event" to be a *path* through the graph where we emit a symbol every time we enter a state. The probability of the event is equal to the probability of the path times the probability of emitting that sequence of observed symbols, given that we have taken that path through the finite-state machine. The probability of a sequence is then found by averaging over all possible paths.

To formalize the HMM, we denote the set of states by $\mathcal{Q}$, the transition probabilities from state i to j by a_{ij}, and the probability of emitting a symbol $a \in S$, given that we are in a state i by $e_i(a)$. Let $\boldsymbol{q} = (q_1, q_2, \ldots, q_T)$ be a sequence of states, then the likelihood of a sequence $\boldsymbol{x}$ is given by

$$\mathbb{P}(\boldsymbol{x}|\Theta) = \sum_{\boldsymbol{q}\in\mathcal{Q}^T} \mathbb{P}(\boldsymbol{x}, \boldsymbol{q}|\Theta) \tag{2}$$

and

$$\mathbb{P}(\boldsymbol{x}, \boldsymbol{q}|\Theta) = \prod_{t=1}^{T} a_{q_{t-1}q_t}\, e_{q_t}(x_t). \tag{3}$$

Here, q_0 denotes the initial state. Naively, the computation of the likelihood seems to grow exponentially with the length of the sequence. However, all the computations we need can be computed efficiently using dynamic programming techniques. We can compute the likelihood using the "forward" algorithm. The forward variable $\alpha_t(i)$ is defined as

$$\alpha_t(i) = \mathbb{P}(x_1, \cdots, x_t, q_t = i|\Theta). \tag{4}$$

Starting from $\alpha_1(i) = a_{0i}e_i(x_1)$, we can find $\alpha_t(i)$ for all states $i \in \mathcal{Q}$ for successive times $1 \le t \le T$ using the recursion

$$\alpha_t(i) = e_i(x_t) \sum_{j\in\mathcal{Q}} a_{ji}\alpha_{t-1}(j). \tag{5}$$

This follows from the Markovian nature of the model. When we have found $\alpha_T(i)$ we can compute the likelihood by marginalizing out the final state

$$\mathbb{P}(\boldsymbol{x}|\Theta) = \sum_{i\in\mathcal{Q}} \alpha_T(i). \tag{6}$$

Having an absorbing state (with no outgoing transitions) in the model would result in a probability distribution over sequences of all possible lengths, but we use the first formulation in the remainder of the theoretical part of the paper. There exists an analogous backward algorithm that can also be used to compute the likelihood. We define the backward variable $\beta_t(i)$ to be the probability of matching the sequence $x_{t+1}, \cdots, x_T$, given that we are in state i at time t

$$\beta_t(i) = \mathbb{P}(x_{t+1}, \cdots, x_T|q_t = i, \Theta). \tag{7}$$

Again, this can be obtained recursively using

$$\beta_t(i) = \sum_{j\in\mathcal{Q}} a_{ij}e_j(x_{t+1})\beta_{t+1}(j) \tag{8}$$

with initial condition $\beta_T(i) = 1$ for all $i \in \mathcal{Q}$. The likelihood is given by

$$\mathbb{P}(\boldsymbol{x}|\Theta) = \sum_{i\in\mathcal{Q}} a_{0,i}\, e_i(x_1)\beta_1(i).$$

More importantly, the backward variable can be used in combination with the forward variable to compute important quantities needed for parameter estimation.

B. HMM Parameter Estimation

The continuous parameters of the HMM can be estimated to maximize the model likelihood $P(x|\Theta)$ on the given sequences. This is achieved using the Baum–Welch algorithm which is an example of an expectation maximization (EM) algorithm [8]. Because the state sequences are not directly observable, statistical information is used to adjust the parameters. The model parameters that maximize the likelihood value are calculated iteratively. The update rule of the transition probabilities are

$$a_{ij} = \frac{n_{ij}}{\sum_{j' \in \mathcal{Q}} n_{ij'}} \tag{9}$$

where n_{ij} is the number of transitions from state i to state j summed over the sequence, that is

$$n_{ij} = \sum_{t=1}^{T} n_{ij}(t). \tag{10}$$

The update rule of the emission probabilities can be obtained as

$$e_i(a) = \frac{n_i(a)}{\sum_{a' \in \mathcal{S}} n_i(a')} \tag{11}$$

where $n_i(a)$ is the number of times the symbol a is emitted when in state i. Equations (9) and (11) are self-consistent equations which are satisfied when the likelihood for the training data is locally maximal. We satisfy the equation by iteratively updating a_{ij} and $e_i(a)$ according to the expected values for n_{ij} and $n_i(a)$, which are computed using the forward and backward algorithm. If we train on a limited amount of data, this estimation of the emission probabilities is liable to overfit the data. For example, we may not have seen some symbol a emitted at time t in our training data. If we build an HMM with a zero probability of emitting a symbol at this time we would perfectly fit our training data, but we may be reading more into a finite sample of data than we should be. To avoid excessive overfitting we can add some "pseudocounts," α_{ij} to our estimate for n_{ij} and $\alpha_i(a)$ to our estimate for $n_i(a)$. Our estimated transition and emission probabilities are given by

$$a_{ij} = \frac{n_{ij} + \alpha_{ij}}{\sum_{j' \in \mathcal{Q}} (n_{ij'} + \alpha_{ij'})} \tag{12}$$

and

$$e_i(a) = \frac{n_i(a) + \alpha_a}{\sum_{a' \in \mathcal{S}} (n_i(a') + \alpha_{a'})}. \tag{13}$$

Although, this appears to be an *ad hoc* fix, it can be motivated from a Bayesian perspective. We can consider the training set to be a sample from a multinomial distribution and assume a Dirichlet distribution for the prior probability, then the pseudocounts drop out as the coefficients of the prior [9].

In the Baum–Welch algorithm, unknown transition and emission frequencies are replaced by their expected values and the parameters re-estimated using the forward and the backward variables. The parameters are updated according to

$$\begin{aligned} n_{ij}(t) &= \mathbb{P}(q_{t-1} = i, q_t = j | \boldsymbol{x}, \Theta) \\ &= \frac{\alpha_{t-1}(i) a_{ij} e_j(x_t) \beta_t(j)}{\mathbb{P}(\boldsymbol{x}|\Theta)} \end{aligned} \tag{14}$$

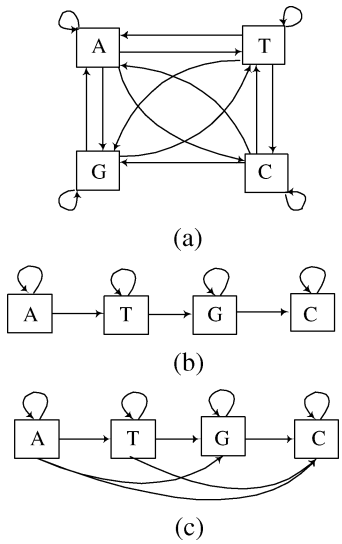


Fig. 1. Several types of HMM topologies: (a) ergodic model, (b) self-loop model, and (c) left–right model.

$$\begin{aligned} n_i(t) &= \mathbb{P}(q_t = i | \boldsymbol{x}, \Theta) \\ &= \frac{\alpha_t(i) \beta_t(i)}{\mathbb{P}(\boldsymbol{x}|\Theta)}. \end{aligned} \tag{15}$$

The Baum–Welch algorithm acts as a local search algorithm and is liable to become trapped at a local optimum. For more details on the training of HMMs, see, e.g., [1], [2], and [10].

There are several ways of finding a state sequence in a given observation sequence. The posterior decoding method is to choose the states which are individually most likely

$$q_t^* = \arg\max \mathbb{P}(q_t = i | \boldsymbol{x}, \Theta). \tag{16}$$

From (4) and (7), we get

$$\arg\max \mathbb{P}(q_t = i | \boldsymbol{x}, \Theta) = \frac{\mathbb{P}(\boldsymbol{x}, q_t = i | \Theta)}{\mathbb{P}(x|\Theta)}. \tag{17}$$

This can be evaluated using the forward and backward algorithms.

C. Topologies of HMMs

The effectiveness of an HMM depends on the number of states and the structure of the graph connecting the states. Traditionally, prior knowledge about a problem is encoded in this structure. In choosing a structure, there is a tradeoff between simplicity and complexity. A too simple model is unlikely to be able to generate the data set with a high likelihood, while a too complex model will easily learn the data set, but is unlikely to generalize well. The complexity depends on the number of states and the structure. The most complex structures are completely connected graphs, while the simplest are linear chains. Fig. 1 shows several types of HMMs. In these examples, we only allow one symbol to be emitted from each state (in general, one state could emit any symbol from the alphabet, $\mathcal{S}$, with a probability given by the emission probabilities). The fully connected model [Fig. 1(a)] can generate any sequence. The self-loop model of Fig. 1(b) can generate sequences such as AAATTTGCCC, while the left-right model can generate the

same sequence as the self-loop but can also jump states to produces sequences such as AATTCCC.

A common method to decrease the model complexity is to "tie" states so they have the same emission and/or transition probabilities [11]. States are tied if we believe that they model similar parts of a sequence. This reduces the number of free parameters that need to be learned and thus reduces the problem of overfitting. Biological sequences often have approximately the same biological functionality even though they are slightly different in their characters. A way to incorporate the variant feature of the biological string is to tie parameters. Parameter tying is used in many HMM applications such as TMHMM [12].

D. Previous Work

There has been relatively little work in estimating the HMM structure without any prior knowledge. In speech recognition, state merging methods have been used to find a compact HMM structure [13]. In this paper, an HMM architecture is built from a large initial model, which is shrunk by merging states. In the bioinformatics field, there has been an attempt to find HMM structures using a local search that inserts or deletes states [14]. They used statistical information about the states to decide the point to insert new states. These attempts have not been very successful for designing the architecture for complex problems. The use of GAs to find an HMM structure has been tried previously in speech recognition [4]. They considered a five-state model and evolved strings which represented the transitions and emissions of an HMM. We are aware of one other group that has used GAs for optimizing HMM structures for biological sequence analysis [3]. An earlier attempt to hybridize GAs with HMMs is described here [15]. In both cases, GAs were used to insert and delete states and swap transitions. They could deal with complex biological sequences, but the model produced is hard to interpret because they allowed unlimited transitions between states. In a previous paper, we have attempted to build a humanly interpretable HMM structure using a GA [16]. We used a similar block structure to that described here, but the linking between blocks was very limited. The structures we found were too limited for more general applications.

HMMs are, in fact, a special subclass of graphical models [17]. GAs have been used to find the structure for graphical models (see, for example, [6], [7]). However, the structures being learned in graphical models is the dependency graph. The dependency graph for an HMM is shown in Fig. 2. The structure of the HMM determines the probability model between the states rather than the dependency graph. Nevertheless, both HMMs and graphical models aim to find a graph topology that accurately represents a probabilistic process. Consequently, there may be a crossover of ideas between these two domains.

E. Biological Background

The block models described in this paper are motivated by applications of HMMs in biological sequence analysis. Biological sequences (DNA or protein) often contain "motifs," which are more or less conserved words with more or less homogeneous intervening sequence, which is characterized by the composition of letters (amino acids or nucleotides). The motifs might, for instance, correspond to binding sites for other molecules.

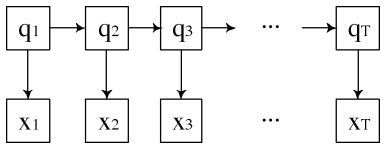


Fig. 2. HMM represented as dynamical Bayesian network.

Such a sequence can be modeled by an HMM containing submodels for the motifs (linear chains of states) and models for the intervening sequences, each of which can be a single state or multiple emmission tied states, if a length distribution is modeled. Other types of sequences are changing between various types of homogeneous sequences. An example is membrane proteins that contain membrane helices 20–30 amino acids long, which are dominated by hydrophobic amino acids and an intervening sequence that is typically more hydrophillic [12]. Such sequences can be modeled with a block of tied states, one block for each type of sequence. Sometimes sequences contain periodic patterns. The region of a gene that codes for a protein is made up of codons, which are nucleotide triplets, each of which codes for one amino acid. The first codon is a special start codon (often the three bases ATG) and the last codon (which actually does not code for an amino acid) is a stop codon (TAA, TGA, or TAG). This gives rise to a three-periodic pattern, which can be modeled with a model like the one shown in Fig. 3.

The block HMM has been tested on genomic DNA sequences from *Campylobacter jejuni* (hereafter *C. jejuni*). *C. jejuni* is an important human intestinal pathogen. Despite intensive study, much is still not known about how to control and intervene in the disease [18]. A better understanding of gene organization, function, and regulation in *C. jejuni* is desirable to provide possible control strategies.

One test is on modeling the coding region of genes as described above and the other in in modeling the promoter. A promoter is a region of DNA located upstream from the transcription start site of a gene. It contains binding sites for RNA polymerase and regions that regulate the rate of transcription. Predicting the promoter region has been one of the main challenges in bioinformatics because the promoter region regulates the expression of the gene. There is still no general computing algorithm for finding promoters with high precision.

In many bacteria, there is a conserved sequence of TTGACA at around 35 base pairs (bp) before the transcription start site (position "-35") and TAtAAT (where "t" indicates either T or A) at around the -10 region, called the TATA box [19]. A ribosome binding site is located between the transcription start site and the start codon of the gene, and it is typically AAGGA. In the promoter region of *C. jejuni*, the TTGACA region is weakly conserved, but a T-rich domain is a common upstream of the TATA box [20]. The *C. jejuni* genome contains more Ts and As than Gs and Cs. Although the ribosomal binding sites can often be spotted, the sequence is not always in the same position in relation to the coding region. In some cases, the sequence AAGGA is highly mutated. It is difficult for nonexperts to figure out which part of the sequence is the TATA box because most part of the promoter region is composed of Ts and As.

Petersen *et al.* suggested an HMM architecture for this promoter region [21]. Their model includes the TATA box

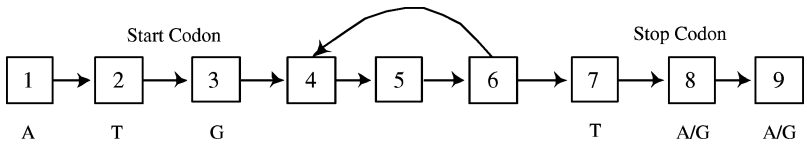


Fig. 3. Simple HMM coding region of a gene.

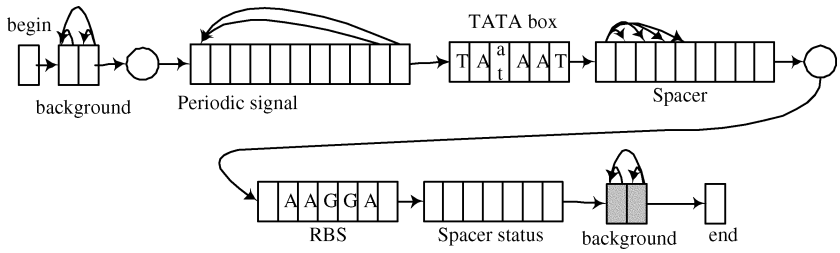


Fig. 4. Model for predicting promoter region of *C. jejuni*, adopted from [21] and modified. They modeled this structure with biological knowledge.

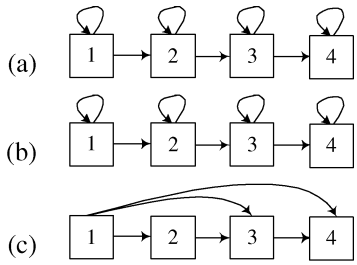


Fig. 5. HMM blocks that compose whole HMM structure: (a) linear block, (b) self-loop model (tying is optional), and (c) forward-jump block (tying is optional).

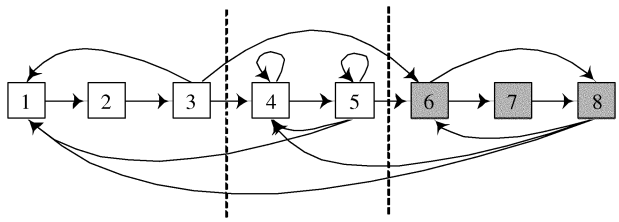


Fig. 6. Example of HMM composed of blocks (block HMM). Three blocks are used in model and all blocks are fully connected to each other.

(TAtAAT) and a ribosomal binding site (AAGGA). During the testing of various models, a periodic pattern was discovered upstream of the TATA box, and this was included in the model. This model is shown in Fig. 4. The forward transitions are used to represent sequences with varying length. The states in the background region are tied together to represent nonspecific sequences with a small number of states.

III. Block HMM

A. Biological Block Model

To constrain the search of HMM topologies to biologically meaningful structures, we represent the HMM structure as a number of blocks. The blocks we consider are one of three basic structures that are frequently used in biological sequence analysis. These are: linear, self-loop, and forward blocks. The self-loop and forward block can be either tied (we follow the convention of shading tied blocks) or untied. That is, all the emission and transition probabilities are set equal. A forward block can have a transition from the first block to the n last blocks in the chain. Examples of these block structures are shown in Fig. 5.

The linear block can model a particular pattern of a sequence, the so-called conserved regions. The self-loop can be used to model a sequence of any length, while the forward jump block can be used to represent varying length subsequences up to some fixed length. Tying can be switched on and off in the self-loop and forward blocks.

The blocks are fully linked together to form the whole HMM architecture. This is illustrated in Fig. 6. The last state of a block has transitions to the first states of all the blocks. The resulting HMM can be thought of as a fully connected graph consisting of "super vertices" made up of blocks whose internal states are not fully connected. We call this structure a block HMM. Special patterns like periodic signals can be generated with a path between blocks.

B. Genetic Operators for Block HMM

Each block is represented by a pair. The first element defines the length of the block, while the second element gives the type (a, b, or c corresponding to linear, self-loop, or forward-jump block, respectively). The type also specifies whether the states are tied or untied (t or u) and, in the case of forward blocks, the number of forward connections. The full HMM is represented as a string of pairs as shown in Fig. 7. For example, the HMM in Fig. 6 would be represented by "((3,a),(2,bu),(3,ct1))." As the blocks are equivalent in their connectivity, there is no information in the ordering of the blocks.

In crossover, two parent strings are chosen at random. Some number of randomly chosen blocks are then swapped to create two children. The children then replace the parents. When we swap blocks, the transition probabilities leaving the block are kept unchanged. Since the position of the blocks does not carry any meaning, we do not impose any constraint on which blocks are swapped. The number of blocks is kept fixed. However, as the blocks can have variable lengths, the number of states is not fixed. We also allow blocks consisting of no states (zero blocks), which effectively allows us to have a variable number of blocks up to some maximum. The evolution of a variable size structure is similar to the situation common in many applications of genetic programming. This scheme in a way emulates natural

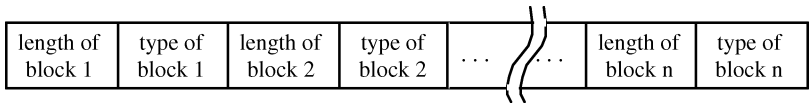


Fig. 7. String representation of block HMM. Information on lengths and types of blocks are stored.

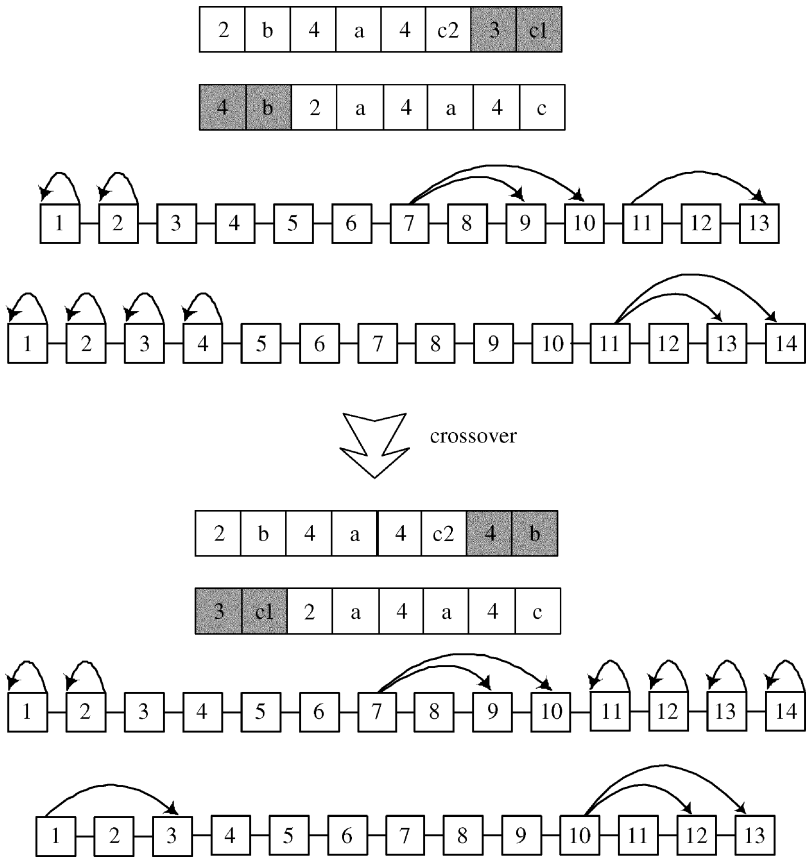


Fig. 8. Crossover in block HMM. Crossover swaps HMM states without breaking property of HMM blocks.

evolution which can cross over DNA sequences with different lengths. We chose a generational GA as a way to present the DNA sequences crossed over as a block.

Fig. 8 shows an example of the crossover scheme. The last block of the first child crosses with the first block of the second child. To simplify the diagram, transitions between blocks are not shown here. Under the crossover scheme, the properties of the interpretable blocks are not broken. This allows us to exchange meaningful blocks without causing too much disruption.

Mutations can take place in any block of the HMM. There is a variety of different mutations that we allow for. Mutations can change the length of a block. For a forward-jump block, mutations can change the number of transitions. For example, in the case of a four-state forward-jump block, there are six different types of mutations possible. These are illustrated in Fig. 9. The mutation can add [Fig. 9(a)] or delete [Fig. 9(b)] a transition inside a block. To prevent losing the property of the block, deletion of a transition is not allowed when there are only two transitions (to the second state and the last state). In the same way, adding a transition does not take place when the first state of the block has transitions to all the other states. In Fig. 9(c) and (d), a state is deleted from a block. The outcome depends on which block is deleted. In Fig. 9(e) and (f), a state is added to the block. Again, the outcome depends on which block is added. In the cases of linear and self-loop blocks, there is only one way to add and delete a state. These six different types of mutation supply the block HMM with sufficient variation without changing the properties of the block.

In addition to changing the length of the block and the transitions, we also allow another form of mutations, called *type mutations*, that change the type of the block. For self-loop and forward jump blocks, we can mutate between tied and untied versions. We can also mutate the type altogether. Mutations to a block of zero length are also allowed.

C. Fitness Evaluation

Very complex models are likely to fit the data well but to generalize badly. We, therefore, need a way to gauge the generalization ability. To achieve this, we split the training data into two; one half is used as a Baum–Welch training set and the other half as a fitness evaluation set. We use the training set to find the transition and emission probabilities using the Baum–Welch algorithm and the evaluation set to measure the fitnesses used in selecting members of the population. Although we could potentially overfit on the evaluation set, this does not seem to be a problem in practice. One explanation for this is that overly

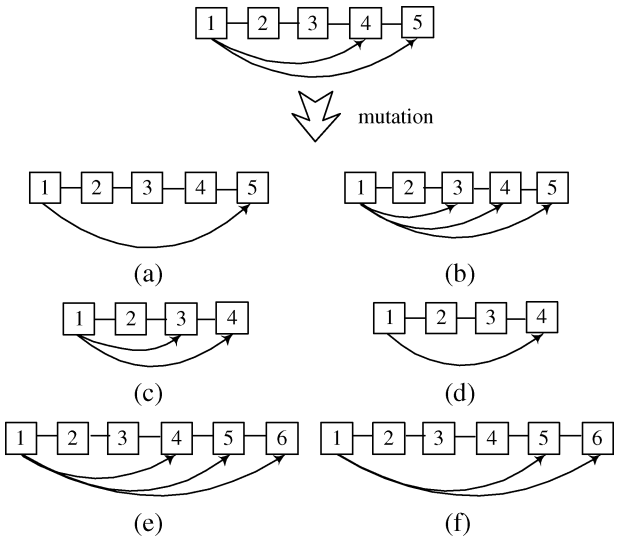


Fig. 9. Six possible types of mutations from four-state jump-forward block.

complex models will overfit the training data when we run the Baum–Welch algorithm. They then perform badly on the fitness evaluation set. Because we use the Baum–Welch at every iteration, overly complex models will always be disadvantaged.

We take as fitness values the reciprocal of the negative log-likelihood

$$E_\mu = \frac{1}{-\log\left(\mathbb{P}(x|\Theta_\mu)\right)} \tag{18}$$

where μ indicates the different members of the population. A member of the population is selected with a Boltzmann probability

$$F_\mu = \frac{w_\mu}{\sum_{\nu=1}^{N} w_\nu}, \quad w_\mu = \mathrm{e}^{s\frac{E_\mu}{\sigma}} \tag{19}$$

where σ is the standard deviation in the distribution of fitnesses in the population. The parameter s controls the selection strength. Stochastic universal sampling is used to reduce genetic drift in selection [22].

D. Training Procedure

To test our HMM, we split our data into a training and test set. The training data was further split into a Baum–Welch training set and an evaluation set, as described above. The number of blocks is chosen at the beginning of the training and kept fixed. The length and type of blocks was randomly chosen so that there were on average the same number of linear, self-loop, forward-jump, and zero blocks.

At each iteration of the block HMM, we applied the Baum–Welch training algorithm to all individuals. We then evaluated the fitness of the population and performed selection (with selection strength $s = 1$), mutation, and crossover. We continued until there was no significant change in the structure of the HMM.

TABLE I
BLOCK HMM PARAMETERS USED IN EXPERIMENT

Parameter	value
Population size	20
Iteration	300
Number of crossover per iteration	1
Number of mutation per iteration	1
Number of type mutation per iteration	1

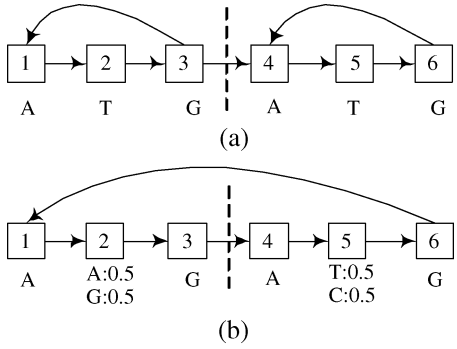


Fig. 10. Result of block HMM with two blocks. (a) $(\mathrm{ATG})^+$. (b) $(\mathrm{AAGATGAGGACG})^+$.

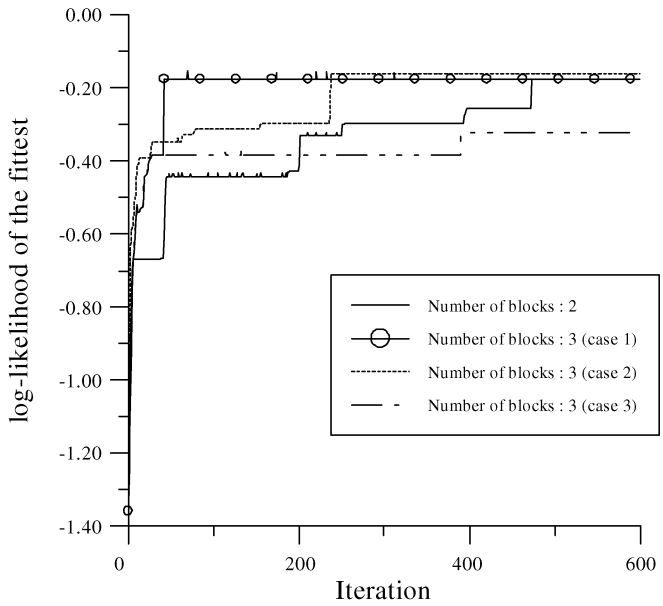


Fig. 11. Behavior of block HMM GA shown as a function of number of iterations for four runs.

IV. EXPERIMENT ON BLOCK HMM

A. Experiment With Artificial Data

To test the performance of the block HMM, we conducted three experiments with artificial data. The first two experiments were to find an HMM to represent data generated from the languages $(\mathrm{ATG})^+$ and $(\mathrm{AAGATGAGGACG})^+$ where "+" means any number of repetitions. We used a population composed of HMMs with two blocks. Table I shows parameters used in these toy experiments. That is, at each iteration we randomly choose two individuals and performed crossover to create two children

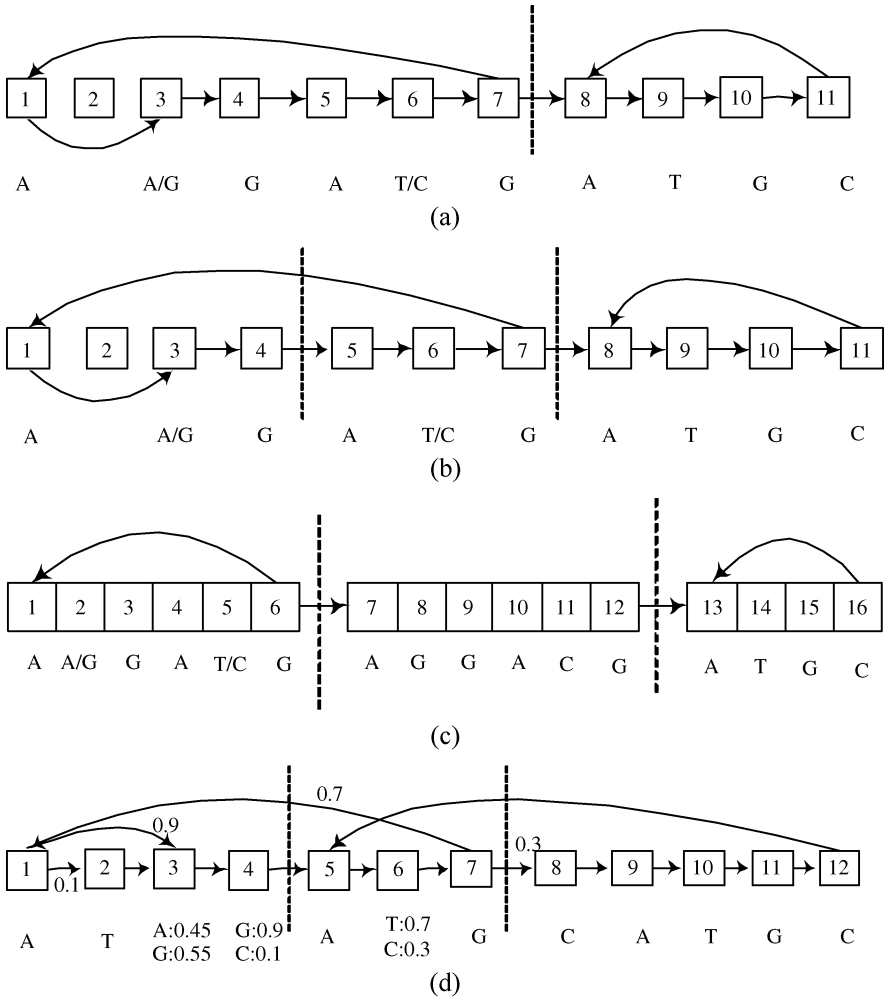


Fig. 12. Result of block HMM for $(\mathrm{AAGATGAGGACG})^{+}(\mathrm{ATGC})^{+}$ (a) with two blocks, (b) with three blocks (case 1), (c) with three blocks (case 2), and (d) with three blocks (case 3).

which replace the two parent strings, one individual where we mutate either the length or the number of transitions in a randomly chosen block, and one individual where we mutate the type of a randomly chosen block (in later experiments we performed more crossovers and mutations each iteration). For this simple problem, we did not allow tying. The solutions to these two problems found by the GA are illustrated in Fig. 10. The resulting HMMs are reasonable solutions to the problem although probably not those that a human would have come up with.

The third test we carried out was to find an HMM to recognize the language $(\mathrm{AAGATGAGGACG})^{+}(\mathrm{ATGC})^{+}$. The first half of the sequence is the same as that used in the previous experiment, while the second half is a repetition of four symbols. The number of iterations used in this experiment is 600. Fig. 11 shows a graph of the maximum log-likelihood value versus the iteration for four different runs of the GA. In case 3 of the three block model, the GA has still not found a particularly good HMM after 600 iterations.

The best solutions found by the GA among the four runs are shown in Fig. 12. Note that we have not shown transitions which the Baum–Welch algorithm has driven to zero. In the first run [Fig. 12(a)], we initiated the GA with two blocks, while in the next three runs [Fig. 12(b)–(d)] we used a three-block model.

These toy examples show that a GA is capable of finding reasonable structures that can be readily interpreted by humans. However, the models are not the simplest that could solve the problem nor do they always have an optimal structure. Nevertheless, they demonstrate that the GA can find reasonably parsimonious solutions which give a good approximation of the true likelihood. To test the performance of the block HMM on more complex sequences, we used biological sequences in the following experiment.

B. Coding Region Model of C. jejuni

To investigate the block HMM's ability to find an HMM structure for biological sequences, we performed an experiment with 200 sequences from the coding regions of *C. jejuni*. The sequence data comprises a start codon (ATG) at the beginning of the sequence, some number of codons, and a stop codon at the end. The length of these sequences ranges from around 200 to 2000. Fig. 3 shows a typical HMM that could be used to represent the coding region [23]. Of the 200 sequences available, 150 sequences are used for training and 50 sequences for evaluation. From looking at the data alone, it is almost impossible for nonspecialists to see that the data consists of codons.

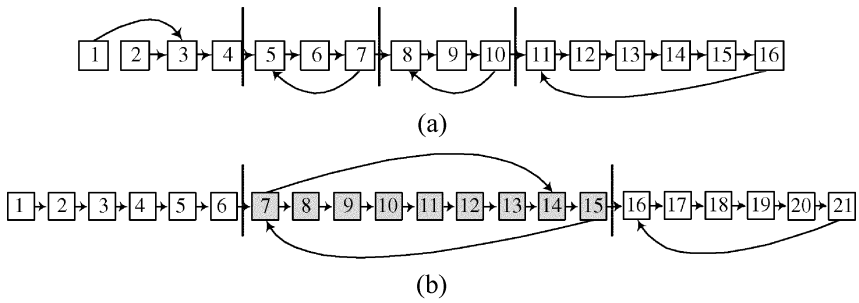


Fig. 13. Result of block HMM. It searched three state loops with GAs.

TABLE II
BLOCK HMM PARAMETERS USED IN EXPERIMENT WITH BIOLOGICAL SEQUENCES

Parameter	value
Population size	30
Iteration	600
Number of crossovers per iteration	4
Number of mutations per iteration	4
Number of type mutations per iteration	4

We conducted the experiment twice, once using four blocks and once with three blocks. The initial block lengths range between three and seven. The GA parameters used in the simulations are shown in Table II.

Fig. 13 shows the resulting structures of the block HMM found using the GA. In Fig. 13(a), the second state of the first block is not used. In Fig. 13(b), the emission probabilities of the state between 7 and 15 are tied. Although these are not the most parsimonious solutions they do seem to have identified the triplet nature of the sequences.

C. Promoter Model of C. jejuni

The block HMM was applied also to the promoter of *C. jejuni* and compared to the results in [21]. We tested if the block HMM could find the biologically meaningful regions without any prior knowledge. The GA parameters are the same as those used in the previous experiment. Of the 175 sequences available, 132 sequences are used for Baum–Welch training and 43 sequences are used for the fitness evaluation. We conducted simulations using nine, eight, and seven blocks.

The best structures found by the GA are shown in Fig. 14. The block HMM could find the "AAGGA" and "TAtAAT" regions with nine blocks [Fig. 14(a)] and eight blocks [Fig. 14(b)]. In addition, it found the presence of semi-conserved TGx upstream of TATA box which is characteristic of the *C. jejuni* promoter region [21]. However, when the number of blocks is seven [Fig. 14(c)], the block HMM could find only the AAGGA sequence. The TATA box was buried inside other states. Interestingly, the nine-block model [Fig. 14(a)] also found the ten-base periodicity just before the TATA box, which was discovered in [21]. In Fig. 14, the emission probabilities of the states in the shaded cells are tied.

To perform a quantitative test of the generalization performance of structures found in the previous experiment, we conducted a discrimination test. In order to collect a sufficient amount of data we did a fivefold cross-validation experiment. This replicates a test performed by Petersen *et al.* [21]. As the structures generated in the previous test (shown in Fig. 14) were found using all the training data, we retrained all the emission and transition probabilities starting from random values. However, to retain the main structure, we removed all transition probabilities less than 0.002; to maintain the positions of TATA box and ribosomal binding site, we introduced pseudocounts in these conserved regions. For example, if the emission probabilities of one state is A:0.9 T:0.1 G:0.05 C:0.05, then the pseudocount becomes 90, 10, 5, 5 for the A, T, G, C, respectively. This ensures that these conserved regions occur in the same place when we retrain the model in the cross-validation procedure. We used the Baum–Welch to retrain the weights using 140 out of the 175 sequences as training data. The remaining 35 sequences were used as test data.

To perform a discrimination test, we require a background sequence. To obtain this, we use a 500 000bp sequence generated by a third-order Markov chain that had been trained on the *C. jejuni* genome. To test the discrimination, we set a log-odds threshold so that there were ten or fewer false positives and then measured the number of true and false positives. Table III shows the total number of false positives (FP) and true positives (TP) summed over all of the fivefold cross-validation tests.

We repeated this test on ten different HMM structures found by using a GA. In all but one case, we could get approximately seven to nine false positives and a true positives rate ranging from 70% to 76%. Only in one simulation out of the ten did the GA fail to find a TATA box. These results are superior to those published for a handcrafted HMM [21], although we must be slightly cautious in interpreting our results as the structures were trained using the same data that it was tested on. We were forced to do this because the dataset was small. We did take the precaution of retraining all the weights to reduce the risk of biasing our results, but the pseudocounts were influenced by the whole data set. Although the results are not conclusive, they are suggestive.

V. CONCLUSION

In this paper, we have described a GA which evolves the structure of HMMs in a biologically constrained way. We have performed a number of tests to illustrate that the resulting HMMs are relatively easy to interpret. Furthermore, on the problem of the promoter model of *C. jejuni*, the results are competitive with an expert-designed HMM. This is quite

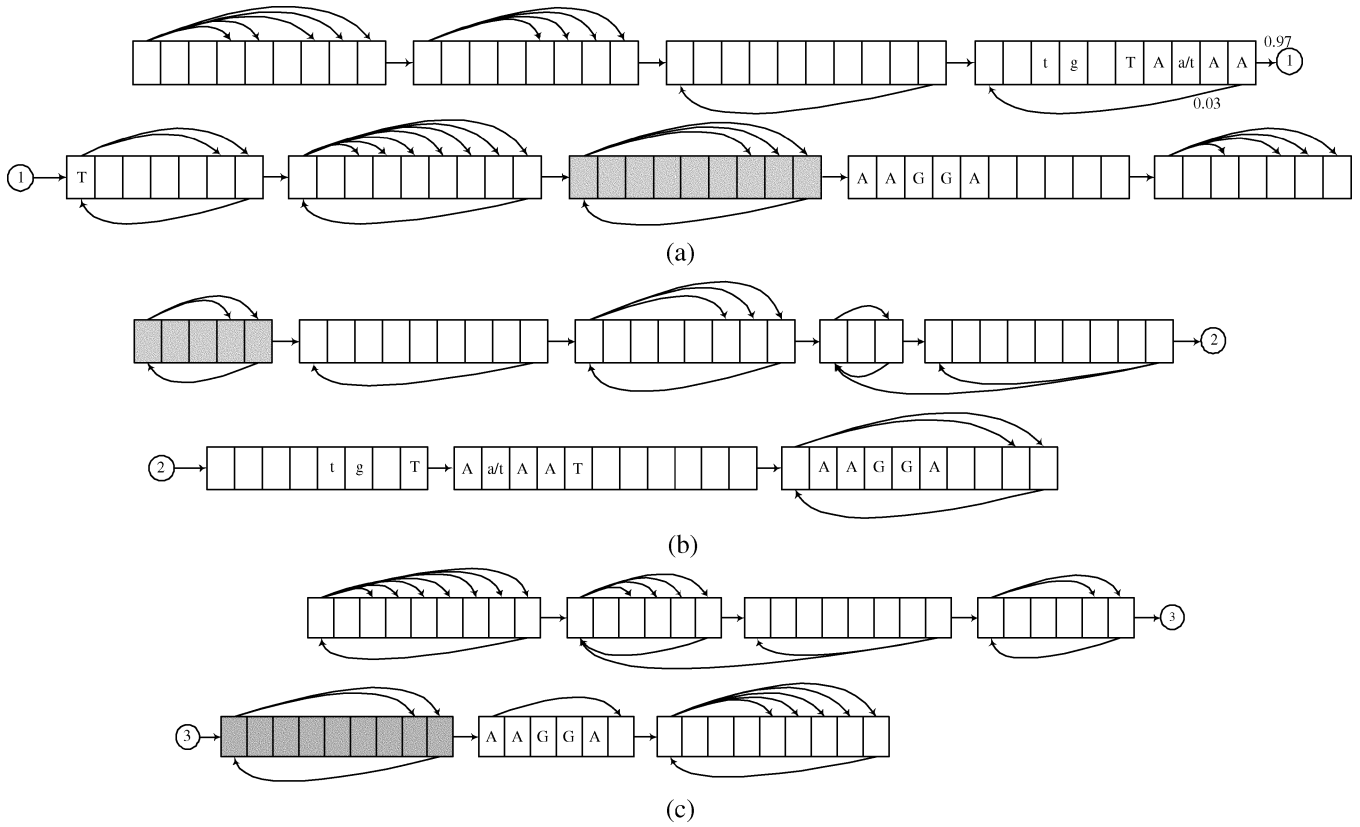


Fig. 14. Best structures found by GA for (a) nine, (b) eight, and (c) seven blocks. "AAGGA" sequence is found on every simulation and "TAtAAT" sequence is found in (a) and (b). Each cell represents a state. Emissions of shaded cells are tied.

TABLE III
RESULT OF BLOCK HMM

Simulation	Number of FP	Number of TP	TP rate
block-HMM (9 blocks)	7	132	75%
block-HMM (8 blocks)	9	126	72%
block-HMM (7 blocks)	10	120	69%
Petersen's HMM	10 (best case)	119	68%

remarkable given that our GA had no prior knowledge of conserved regions such as the TATA box and ribosome binding site as well as other structures which have been acquired by experts over many years. These are promising results from early work. There is a lot more that can be investigated. For example, in order to reduce overfitting we have split our training data into a Baum–Welch training set and an evaluation set used in selecting members of the population. However, we could still overfit the evaluation set. One method to control this is to include a regularization term in the fitness term that punishes more complex models. Such a scheme has been used by Yada *et al.* [3]. However, in early experiments we found that this involved setting a parameter which was difficult to do with the limited amount of data. Nevertheless, this may be worth revisiting, particularly as we try to evolve more complex models. Another method to regulate the HMM model complexity is using *minimal description length* (MDL) [24]. The MDL-based scoring metric can be used to evaluate the fitness of the model balancing the complexity of the HMM with the degree of accuracy. This is a possible future area of study.

The blocks introduced in this paper were adopted from frequently used HMM topology. Beside these three blocks, other topologies have been constructed in bioinformatics depending on the applications [2]. And other possible block models can be addressed. However, the three blocks used in this paper have proven to be sufficiently powerful in this application. We believe that they are likely to be able to model many different types of sequences.

Unrestricted crossover is likely to be very disruptive in this application. To prevent this, we only allow crossover to occur between blocks. Since the blocks are connected to every other block, the structure is much less disrupted. Within a block the structure is evolved using mutation. This is analogous to the evolution of DNA in nature where crossover can exchange genes with different length.

Evolving HMM structures is a time-consuming process since training and evaluating the likelihoods for long sequences takes many operations. At present, this is a weakness for automatic structure finding. However, as computers get faster and the number of applications grow with the exponential rise in the amount of sequencing data, we expect that the automatic HMM structure search will become increasingly important. GAs seem to be ideally placed to play an important part in this development.

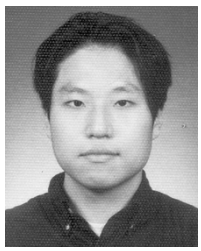
Data Availability: The biological data used in the tests discussed in this paper can be obtained from [25]. In addition, this site contains all the details of the HMM structures found by the GA.

ACKNOWLEDGMENT

The authors would like to thank L. Petersen for her help with the sequence data and the model.

REFERENCES

[1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.
[2] R. M. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 1998.
[3] T. Yada, M. Ishikawa, H. Tanaka, and K. Asai, "DNA sequence analysis using hidden Markov model and genetic algorithm," *Genome Inform.*, vol. 5, pp. 178–179, 1994.
[4] C. W. Chaw, S. Kwong, C. K. Diu, and W. R. Fahrner, "Optimization of HMM by a genetic algorithm," in *Proc. Int. Conf. Aucoustics, Speech, Signal Processing*, 1997, pp. 1727–1730.
[5] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[6] I. Poli and A. Roverato, "A genetic algorithm for graphical model selection," *J. Italian Statist. Soc.*, vol. 2, pp. 197–208, 1998.
[7] I. Inza, P. Larrañaga, and B. Sierra, "Feature subset selection by Bayesian networks: A comparison with genetic and sequential algorithms," *Int. J. Approximate Reasoning*, vol. 27, no. 2, pp. 143–164, 2001.
[8] A. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Roy. Statist. Soc. B*, vol. 39, pp. 1–38, 1977.
[9] K. Sjolander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I. S. Mian, and D. Haussler, "Dirichlet mixture: A method for improved detection of weak but significant protein sequence homology," *Proc. Comput. Appl. Biol. Sci. (CABIOS)*, vol. 12, pp. 327–345, 1996.
[10] A. Krogh and S. K. Riis, "Hidden neural networks," *Neural Computation*, vol. 11, pp. 541–563, 1999.
[11] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI–5, pp. 179–190, 1983.
[12] A. Krogh, B. Larsson, G. von Heijne, and E. Sonnhammer, "Predicting transmembrane protein topology with a hidden Markov model: Application to complete genomes," *J. Molecular Biol.*, vol. 305, no. 3, pp. 567–580, 2003.
[13] A. Stolcke, "Bayesian learning of probabilistic language models," Ph.D. dissertation, Univ. California, Berkeley, CA, 1994.
[14] Y. Fujiwara, M. Asogawa, and A. Konagaya, "Motif extraction using an improved iterative duplication method for HMM topology learning," in *Proc. Pacific Symp. Biocumputing*, 1995, pp. 713–714.
[15] K.-J. Won, A. Prügel-Bennett, and A. Krogh, "Training HMM structure with genetic algorithm for biological sequence analysis," *Bioinformat.*, vol. 20, pp. 3613–3619, 2004.
[16] ——, "The block hidden Markov model for biological sequence ananlysis," *Lecture Notes Artificial Intell.*, vol. 3213, pp. 64–70, 2004.
[17] M. I. Jordan, *Learning in Graphical Models*. Cambridge, MA: MIT Press, 1999.
[18] C. R. Friedman, J. Neimann, H. C. Wegener, and R. V. Tauxe, *Epidemiology of Campylobacter jejuni Infection in the United States and Other Industrialized Nations*. Washington, DC: ASM, 2000, pp. 121–139.
[19] J. M. Berg, J. L. Tymoczko, and L. Stryer, *Biochemistry*, 5th ed. San Frasncisco, CA: Freeman, 2002.
[20] M. M. Wosten, M. Boeve, M. G. Koot, A. C. van Nuene, and B. A. van der Zeijst, "Identification of *Campylobacter jejuni* promoter sequence," *J. Bacteriaol.*, vol. 180, pp. 594–599, 1998.
[21] L. Petersen, T. S. Larsen, D. W. Ussery, S. L. W. On, and A. Krogh, "RpoD promoters in Campylobacter jejuni exhibit a strong periodic signal instead of a −35 box," *J. Molecular Biol.*, vol. 326, no. 5, pp. 1361–1372, 2003.
[22] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proc. 2nd Int. Conf. Genetic Algorithms*, Hillsdale, 1987, pp. 14–21.
[23] A. Krogh, *An Introduction to Hidden Markov Models for Biological Sequences*, ser. Computational Methods Molecular Biol. Amsterdam, The Netherlands: Elsevier, 1998, ch. 4, pp. 45–63.
[24] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*. New York: World Scientific, 1999.
[25] http://www.ecs.soton.ac.uk/~kjw02r/data.htm



Kyoung-Jae Won received the B.S. and M.S. degrees in electronics from Chung-Ang University, Korea, in 1996 and 1998, respectively. Currently, he is working towards the Ph.D. degree at the School of Electronics and Computer Science, Southampton University, Southampton, U.K.

Under the volunteer program of the Korean government, he worked at the College of Science, National University of Hanoi, Vietnam, from 1999 to 2001. His research interests include the areas of bioinformatics, hidden Markov models, and evolutionary algorithms.

Adam Prügel-Bennett received the B.Sc. degree in physics from Southampton University, Southampton, U.K., and the Ph.D. degree in theoretical physics from Edinburgh University, Edinburgh, U.K.

He worked in research jobs in Oxford, U.K., Paris, France, Manchester, U.K., Copenhagen, Denmark, and Dresden, Germany, before finally returning to the School of Electronics and Computer Science, Southampton University, as a Senior Lecturer. His research interests include evolutionary algorithms, machine learning, and computer vision.

Anders Krogh received the Ph.D. degree in theoretical physics from the Niels Bohr Institute, Copenhagen, Denmark, in 1991.

He worked at the University of California, the Sanger Centre, Cambridge, U.K., the Center for Biological Sequence Analysis, Technical University of Denmark, and other places before joining the University of Copenhagen, in 2002, where he heads the Bioinformatics Centre. Originally, he studied neural networks theoretically, but he is now mainly working on probabilistic methods for biological sequence analysis, including the application of hidden Markov models in gene finding and protein modeling.