The Evolution of Stochastic Regular Motifs for Protein Sequences

Brian J. ROSS

Brock University, Dept. of Computer Science
St. Catharines, Ontario, Canada L2S 3A1
bross@cosc.brocku.ca

Abstract Stochastic regular motifs are evolved for protein sequences using genetic programming. The motif language, SRE-DNA, is a stochastic regular expression language suitable for denoting biosequences. Three restricted versions of SRE-DNA are used as target languages for evolved motifs. The genetic programming experiments are implemented in DCTG-GP, which is a genetic programming system that uses logic–based attribute grammars to define the target language for evolved programs. Earlier preliminary work tested SRE-DNA's viablility as a representation language for aligned protein sequences. This work establishes that SRE-DNA is also suitable for evolving motifs for unaligned sets of sequences.[*1]

Keywords Protein Motifs, Stochastic Regular Expressions, Grammatical Genetic Programming, Evolutionary Computation.

§1 Introduction

A motif is a representation modeling some shared characteristic of a family of proteins.[5] There are a number of ways in which motifs can be used. The most common use of a motif, as considered in this paper, is to act as a device for characterizing the sequence pattern common to a particular protein family, and therefore distinguishes them from unrelated sequences. In other words, a motif is a signature for a protein family. Once an effective motif is established for a set of protein sequences, it can be used to query a database of proteins in order to extract the sequences belonging to the family. This permits the discovery of structural similarities between new sequences added to the database

[*1] *New Generation Computing*, vol.20, n.2, Feb. 2002, pp. 187-213.

and others whose functionalities have been established earlier. Hence, motifs are means for determining potential protein functionalities for new sequences, which is obviously an important practical tool for biologists.

An active research area is automated motif discovery. Given the growing rate at which biosequences are being cultivated and added to databases, the manual construction of useful motifs becomes increasingly impractical, given the sheer volume of data to be analyzed. Furthermore, manual design of motifs is error prone, since a human being may not recognize the subtleties that determine protein family membership. The use of machine learning techniques for automatic motif synthesis has been studied by many researchers.[3, 6)] Given the technical complexities of predicting protein functionality from raw biosequences, there does not yet exist a machine learning technology which can determine the most biologically pertinent motif for a given family of sequences. In the meantime, however, different machine learning algorithms and sequence representations can serve as tools that geneticists can use to study new sequence patterns, and therefore aid them in deriving the most biologically sound representation for new families of interest.

This paper describes research in the automatic synthesis of biosequence motifs using evolutionary computation. A contribution of this work is the introduction of a new motif language, stochastic regular expressions (SRE-DNA). SRE-DNA has characteristics of other, more established motif representations. Its regular expression bases is similar to commonly used regular expression motif languages, such as that used by the PROSITE protein database.[12)] On the other hand, its probabilistic nature is similar to stochastic biosequence representations such as HMM's.[22)] Another goal of this paper is to show that SRE-DNA motifs are readily synthesizable by genetic programming. SRE-DNA's stochastic modeling of biosequences can be exploited directly by the evolution process, both during fitness evaluation, and during the evolutionary search itself. It is not a goal of this paper to suggest that this approach to motif discovery is necessarily superior to other established techniques, but rather, to illustrate that the automatic discovery of stochastic motifs is feasible using SRE-DNA and genetic programming.

Section 2 reviews the problem of motif identification. The SRE-DNA motif language is outlined in Section 3. Genetic programming is reviewed in Section 4. Section 5 overviews the design of the experiments. Results are presented in Section 6, and evaluated in Section 7. Comparisons to related work are given in Section 8. Conclusions and future directions conclude the paper in Section 9.

§2 Biosequence Identification

2.1 Motif Representations

Genetic similarities due to common evolutionary origins amongst different species can often be identified by the similar protein functionalities observed at the bio-molecular level. The precise functionality of a transcribed protein is fundamentally determined by its 3D structure. Although of primary importance, such 3D structures are difficult and impractical to ascertain directly from biosequences themselves, and thus remains a critically important open problem in bioinformatics. Consequently, more rudimentary characterizations are used for protein classification and prediction. Motifs that model protein families via the shared similarities in their biosequence composition are widely used. Even though such motifs are crude, indirect denotations of the *real* factor of importance (3D structure), often they are currently the most practical means for characterizing protein families, due to their parsimony, efficiency of interpretation, and amenability to automatic acquisition from raw sequences.

Many factors are pertinent to biosequence representation, which different motif representations may incorporate in varying degrees.[5, 8] Of central importance is the ability to represent *conserved regions* of amino acid sequences. These are the sequences common within a protein family, where differences are usually minor, and inserts and deletes are rare. Typically the conserved region is surrounded by similar sequential patterns unique to an identified protein, but which vary in composition, size, and location according to the particular species of interest. Naturally, when larger sequence lengths in the vicinity are considered, wider variability to motif patterns are introduced; a point will be reached when the length is too large to be representationally beneficial. Consensus patterns can vary with respect to their location within supersequences. They can also repeat within sequences, and possibly overlap. What can greatly complicate the use of motifs is the phenomena of convergent evolution, in which two separate evolutionary paths have evolved completely different sequences, but which have similar (converged) protein functionalities. In other words, the 3D structure of these different sequences yield similar functionalities. Such cases prohibit straight-forward use of motifs, and they will not be considered further in this paper.

Different motif languages have different linguistic strengths and weaknesses with respect to biosequence representation. The simplest motif is a consensus sequence, which defines the exact sequence of amino acid codons common to a family of proteins. Since such common sequences may be too small to be practical, longer consensus patterns are often used. Pattern representations permit variability, which reflect species diversity due to evolution. Consensus patterns may be symbolic, for example, regular languages [2, 6, 7] and higher-level grammars.[27, 28] Symbolic representations are called such because the structure of the motif is directly mappable to the symbolic representation within the motif expression. This lends symbolic representations such as regular expressions their greatest advantage: the user-level interface to a family of sequences is an algebraic abstraction of the consensus pattern. Such motifs are akin to query languages used in conventional databases. A disadvantage of symbolic representations is that more complex protein families result in correspondingly complex symbolic expressions. Numeric and probabilistic representations are also commonly used in motif representations.[8, 22, 16] Their main advantages are their inherent ability to account for structural variation, and their natural encoding of "scores" with which to heuristically judge the relative similarity of candidate sequences to a family profile. An instance of a motif language sharing both symbolic and numeric features is the stochastic context-free grammar of Sakakibara *et al.*,[26] as well as the stochastic regular expression language used in this paper.

The comparative linguistic power of different motif representations can be better understood when formal language theory is considered.[14] An important insight obtained from such a viewpoint is that motif languages can be formally categorized with respect to their expressive power, according to their position in the Chomsky hierarchy. For example, all regular motif languages are expressively equivalent; regular expressions are no more or less powerful than regular automata such as HMM's. Furthermore, context-free grammar motifs are representationally more powerful than regular language motifs. Formal language theory also lends insight into the complexity of recognizing sequences with respect to motif representations. For example, strings can be recognized by regular languages in polynomial time. This means that efficient database access is possible for regular motif languages; such advantages will be lost if higher-level denotations such as context-sensitive grammars are used. Finally, a language-theoretic view of motif representation can indicate if the automatic induction of motifs is feasible.

Not withstanding the advantages of characterizing motifs as formal languages, this purely formal perspective can be somewhat myopic. At a fundamental level, biosequences are indeed programs: they are used as instructions during protein translation. However, this does not imply that a family of biosequences is most naturally or effectively denoted by a particular formal language in the Chomsky hierarchy. Although symbolic or numeric representations can be useful classification and prediction tools, they are not designed as models of the pertinent structural effects that determine protein functionality. For example, a regular expression motif may adequately segregate a family of protein sequences from non-family sequences. However, this sequence-level classification does not account for the deeper structural principles defining the family. A more intelligent motif representation would use 3D stuctural information and other domain-specific knowledge. Such a motif representation would likely be too complex for user-specified protein classification and database access, at least compared to regular motifs. Despite shortcomings, there are often benefits to simplicity.

2.2 Regular Expression Motifs

Regular expressions are widely used as symbolic motif languages, and are used in protein databases such as PROSITE and its variants.[12] Their popularity as motif languages arises from their simplicity, which makes them straightforward to learn; their expressive adequacy for representing relatively small sequences; and their computational tractability with respect to interpretation and automatic acquisition. A hierarchy of regular expression motif languages has been proposed.[6] This hierarchy identifies the various degrees of expressiveness with which nondeterministic choice of codons and gap expressions can be articulated. For example, the most deterministic category (class A) is one that denotes simple sequences of required codons ("t-c-t-t-g-a"). Class B extends class A with a wildcard "x" character, which substitutes for any codon ("t-c-x-x-g-a"). Further classes introduce additional devices. At the most expressive extreme, class I languages are those which additionally permit nondeterministic skip expressions, indeterminate gaps, and alternate choices (masks):

d-t-x(2,4)-v-*-a-x-[nq]-g

Here, "x(2,4)" denotes a skip of length 2 to 4, "*" is an indeterminate gap, and "[nq]" denotes a choice of either n or q.

Note that formal language theory states that *all* regular languages, and hence all the motif classes discussed above, are equivalent with respect to the languages denotable. However, the introduction of new language constructs enables motif expressions to be more convenient and parsimonious. Higher classes of regular expressions can characterize particular protein patterns more concisely than more rudimentary classes. Hence the concept of expressiveness as used in the classification scheme in [6] is one of user-oriented convenience, rather than a language-theoretic one. Nevertheless, this classification scheme can have ramifications on the tractability of automatic expression synthesis performed via machine learning algorithms.

2.3 Automatic Motif Acquisition

Much work has been done on machine learning techniques for biosequences identification.[6, 3] The survey in [6] lists 26 different algorithms, published between the years 1983 and 1996. Most work has focussed on regular language representations, since they are efficient to learn, and both adequate and practical for the relatively low complexity of sequences being analyzed. Motif discovery is an instance of a classical machine learning problem: formal language induction.[17] A successful motif should accurately identify a sequence belonging to a given family, while at the same time, reject sequences that are not members. This implies the existence of two sets of examples – a positive set consisting of protein sequences, and negative examples which are disjoint from the positive set. Motif discovery algorithms that use this criteria are *classification* algorithms, as opposed to *conservation* algorithms which only use a set of positive example sequences.[6] Successful classification requires a balance between recognizing member sequences and rejecting non-member sequences in the training set, while at the same time being general enough to recognize and reject sequences not seen in the original training sets.

There are other dimensions by which motif learning algorithms can be classified. The *solution space* of an algorithm is the type of representation used to denote hypotheses, and may include regular consensus patterns, weight matrices, Bayesian networks, or Hidden Markov Models (HMM). The alphabet used by a representation is also pertinent, as are the generalizations used by the motif language, for example, wildcard characters that denote sequence gaps. Learning algorithms are also distinguishable by such factors as whether sequences must be prealigned or not, whether they guarantee solutions, their computational efficiency, and their overall effectiveness in biological applications. The effectiveness of a learning algorithm is difficult to define, since different algorithms exhibit particular advantages and disadvantages in their natural domains. Consequently, the various approaches to motif learning have not yet been empirically compared with one another.

2.4 Motifs and Genetic Programming

Genetic programming (GP) is a machine learning paradigm, and it is reviewed in Section 4. GP has been successfully applied towards various problems in biosequence analysis.[19, 10, 11, 20] Most of these applications involve the evolution of programs which identify various properties of biosequences, for example, intracellular or extracellular portions of sequences.

There are a couple of examples of the use of GP to evolve biosequence motifs. Hu uses GP to evolve motifs for unaligned example sequences, using a regular language equivalent to that used by PROSITE.[15] Expressions are evolved from sets of unaligned example sequences. Hu's evolution algorithm uses a local optimization step, in which expression terms denoting gaps are refined. A number of protein families were studied, and the evolved solutions were often very similar to the source PROSITE motifs used to extract the example sequences.

Koza, Bennett, Andre and Keane use GP with ADF's (automatically defined functions) to evolve a motif for a few unaligned sequence examples.[21] An ADF is a type of module or subroutine. The regular motif language used is simpler than the full PROSITE language as used by Hu. The protein families studied contain sequence repetitions, which makes the use of ADF's advantageous, since the ADF's modularize the repeated structures. Koza did not specify any motif parameterization requirements (eg. window size), other than overall expression depth limits. The motif evolved for one case was found to be more accurate than the accepted one on file for that protein family.

§3 Stochastic Regular Expressions

3.1 SRE

Stochastic Regular Expressions (SRE) is a probabilistic regular expression language, in which regular expressions[14] are embellished with probability fields.[23] A similar language was previously proposed by Garg, Kumar and Marcus, who prove a number of mathematical properties of the language.[9]

Unrestricted SRE has the following form. Let E range over SRE, α range over atomic actions, n range over integers ($n \geq 1$), and p range over probabilities ($0 < p < 1$). SRE syntax is:

$$E \ ::= \ \alpha \ | \ E : E \ | \ E^{*p} \ | \ E^{+p} \ | \ E_1(n_1) + ... + E_k(n_k)$$

The terms denote atomic actions, concatenation, iteration (Kleene closure or * iteration, and + iteration), and choice. The + iteration operator, E^{+p}, is equivalent to $E : E^{*p}$.

The semantics of the language are briefly outlined as follows. With choice, each term $E_i(n_i)$ is chosen with a probability equivalent to $n_i/\Sigma_j(n_j)$. For example, $a(1) + b(2) + c(3)$ means that the terms are selected with probabilities of 1/6, 1/3 and 1/2 respectively. With the Kleene closure term E^{*p}, each iteration of E occurs with a probability p, and the termination of iteration has a probability $1 - p$. Probabilities between terms propagate in an intuitive way. For example, with concatenation, the probability of $E : F$ is the probability of E multiplied by the probability of F. Therefore, the probability of E^{+p} is the same as $E : E^{*p}$, which is the probability of E multiplied by the probability of the Kleene iteration.

The overall effect of this probability scheme is that an SRE expression defines a sound, well-formed model of probability: each expression defines a probability function. The sum of all the probabilities for all $s \in L(E)$ is 1.[23] Furthermore, each string $s \in L(E)$ has an associated probability, while any $s \notin L(E)$ has a probability of 0. This property is important for this research, since biosequences will be associated with probabilities when given to particular SRE-DNA motifs. A task of motif synthesis will therefore be to evolve motifs that yield high probabilities for candidate strings.

An example SRE expression is:

$$(a : b^{*0.7})(2) + c^{*0.1}(3)$$

It recognizes string c with $Pr = 0.054$ (the term with c can be chosen with $Pr = 3/(2 + 3) = 0.6$; then that term iterates once with $Pr = 0.1$; finally the iteration terminates with $Pr = 1 - 0.1 = 0.9$, giving an overall probability of $0.6 \times 0.1 \times 0.9 = 0.054$). The string bb is not recognized; its probability is 0.

An SRE interpreter is implemented and available for GP fitness functions. Like the case with conventional regular expressions,[14] string recognition for SRE expressions is of polynomial time complexity. To test whether a string s is a member of an SRE expression E, the interpreter attempts to consume s

with E. If successful, a probability $p > 0$ is produced. Unsuccessful matches will result in probabilities of 0. The SRE-DNA interpreter only succeeds if an entire SRE-DNA expression is successfully interpreted. For example, in $E_1 : E_2$, if E_1 consumes part of a string, but E_2 does not, then the interpretation fails and yields a probability of 0.

3.2 SRE-DNA

The protein sequences studied here are fairly restricted in form. This is evident if one examines the source PROSITE motifs used to access them from the database. This implies that unrestricted SRE may be too descriptively rich for the motifs required here. This would not normally pose a problem, since a linguistically richer language may enjoy benefits over weaker ones. Unfortunately, unrestricted SRE does indeed pose problems during interpretation of expressions, because of the nature of the iteration and choice operators. Although regular expression and SRE expression interpretation is of polynomial complexity, expression interpretation is of combinatorial complexity with respect to expression size. For example, the expression $((a)^*)^*$ can generate the string $a...a$ of length k a total of 2^k different ways, due to the combinatorial number of ways the two nested iteration operators can interact.

Early work on this project found that inefficient SRE expressions with nested iteration and choice were commonly constructed, and their slow interpretation made processing impossible.[25] As a consequence, a restricted version of SRE, SRE-DNA, is more practical. The details of the grammatical constraints in SRE-DNA are described in Section 5.2, where three variations of SRE-DNA are introduced. All the variants ignore the choice operator, and restrict iteration so that terms with iteration are guaranteed to be constructive.

SRE-DNA also introduces mask terms, which are sets of amino acid codons. A mask within an SRE-DNA expression means that any of the listed codons is permissible at that portion of the sequence. For example, the mask in the expression

$$a : [b, c, d] : e$$

denotes a choice of b, c, or d, each with an equal probability of 1/3. Hence the strings *abe*, *ace*, and *ade* are valid strings of the expression, and each have a probability of 1/3. SRE's choice (+) operator is more general than a mask, and can denote the same probabilistic languages. For example, SRE would denoted the above as:

$$a : (b(1) + c(1) + d(1)) : e$$

Choice expressions also permit the modeling of more informative codon probability distributions that might exist in real familes. For example,

$$a : (b(120) + c(12) + d(3)) : e$$

says that codon b has a probability of 120/135 or 88.9%. This expressiveness is not available in a mask set, which treats all codons with the same probability.

Despite the weaker stochastic expressiveness of masks compared to choice expressions, masks are more concise, and consequently lend efficiency to evolution during motif synthesis. Earlier work found that the choice operator is detrimental to motif evolution for the proteins studied here, because it promotes GP intron material (expression bloat).[25] Another reason to adopt masks is that PROSITE motifs use them, and so it might be possible to evolve SRE expressions similar to the non-probabilistic regular motifs used by PROSITE. If a more precise denotation of codon distributions is required, choice expressions should be investigated further.

Core regions of conserved motifs are ungapped, in the sense that insertions or deletions can change the activity dramatically. Nevertheless, as a linguistic convenience, the use of *skip* (gap) expressions is convenient in the context of regular motif languages such as SRE-DNA and PROSITE. This is because the purpose of such regular motif expressions is to concisely denote the common amino acids within a protein sequence. Since there will areas of pattern variability within family sequences, they are best denoted by gaps. Ideally, the resulting codons resident in a motif will be the commonly shared ones in the family. The alternative is to represent all possible codons at these positions explicitly with masks or choice expressions, as is done in [21]. This would result in large motifs whose compactness and utility is lost.

Skip expressions are implemented via the code x, which is a wildcard that replaces any codon in a sequence. SRE-DNA will combined this with iteration operators (for example, "$c^{*.10}$"), which permits variable-length gaps to be represented (details in Section 5.2).

§4 Genetic Programming

Genetic programming [4, 18, 21] is a method of automatic programming in which programs are evolved using a genetic algorithm.[13]. Both GP and GA are characterized by their use of the following (see Fig. 1): (i) an initial popula-

```
1. Initialize: Generate initial randomized population.
2. Evolution:
   GenCount := 0
   Loop while GenCount < maximum generations
      and fitness of best individual not considered a solution {
          Loop until New population size = max. population size {
              Select a genetic operation probabilistically:
              → Crossover:
                     Select two individuals based on fitness.
                     Perform crossover.
              → Mutation:
                     Select one individual based on fitness.
                     Perform mutation.
              Add offspring to new population.
          }
      GenCount := GenCount+1
   }
3. Output: Print best solution obtained.
```

Fig. 1 Genetic Algorithm

tion of random individuals, which in the case of GP are randomly–constructed programs; (ii) a finite number of generations, each of which results in a new or replenished population of individuals; (iii) a problem–dependent fitness function, which takes an individual and gives it a numeric score indicative of that individual's ability to solve a problem at hand; (iv) a fitness–proportional selection scheme, in which programs are selected for reproduction in proportion to their fitness; (v) reproduction operations, usually the crossover and mutation operations, which take selected programs and generate offspring for the next generation.

The essential difference between GP and GA is the denotation of individuals in the population. A pure GA uses genotypes that are fixed–length bit strings, and which must be decoded into a phenotype for the problem being solved. A GP uses a variable–length tree data structure genotype, which is directly interpretable as a computer program by some interpreter. The use of programming code as genotype is a powerful and practical representation for solving a wide variety of problems.

The two main reproduction operators used in GP are crossover and mutation. Crossover permits the genetic combination of program code from programs into their offspring, and hence acts as the means for inheritance of desirable traits during evolution. Crossover takes two selected programs, finds a random *crossover point* in each program's internal representation (normally a parse tree), and swaps the subtrees at those crossover points. Mutation finds a random node in a selected program, and the subtree at that node is replaced with a new, randomly generated tree. This is the means by which new genetic traits can be introduced into the population during evolution. Although crossover and mutation preserve the grammatical integrity of programs, the user must ensure *closure* – that the resulting programs are always executable. So long as closure is maintained, all programs derivable by the GP system will be executable by the fitness function, and hence their fitness will be derivable.

The GP implementation used in this paper is DCTG-GP. DCTG-GP is a grammatical genetic programming system.[24)] It uses logical grammars for defining the target language for evolved programs. The logic grammar formalism used is definite clause translation grammars (DCTG).[1)] A DCTG is a logical version of a context-free attribute grammar, which allows the complete syntax and semantics of a language to be defined in a unified framework. DCTG-GP is useful for these experiments because of the ease with which variations of motif languages can be designed and implemented (see Section 5.2). The use of grammars within GP also helps enhance search efficiency, by pruning the search space into more sensible structures.

§5 Experiment

5.1 Protein Sequences

As discussed in Section 2, motif classification algorithms require the discovery of an expression that recognizes positive protein sequence members, while rejecting negative sequences. The positive examples used here comprise a set of unaligned protein sequences. To generate this set, aligned protein sequences were initially obtained from the SWISS-PROT and TrEMBL sequence databases for the protein families listed in Table 1.[12)][*2]. For example, on August 9 2000, seaching the expression "snake toxin" resulted in 176 instances in the SWISS-

[*2] Accessed between July and November 2000, at http://expasy.cbr.nrc.ca/sprot/

Table 1 Protein families and related parameters

Protein (accession #)	**lab**	**m**	**p**	**w**	**s1**	**s2**
Amino acid oxidase (PS00677)	AAO	5	1e-12	19	8	–
Scorpion toxin (PS001138)	ScT	5	1e-12	24	20	–
Zinc finger, C2H2 type (PS00028)	ZF1	10	1e-13	25	29	678
Zinc finger, C3HC4 type (PS00518)	ZF2	8	1e-13	10	21	168
Snake toxin (PS00272)	SnT	5	1e-12	22	18	127
Kazal inhibitor (PS00282)	KI	5	1e-12	24	24	125

PROT database, and an additional 98 examples from TrEMBL. Each example in the database is a contribution by biologists, and contains a protein pattern from various species and genome locations. Note that many sequences are duplicated in the databases. In addition, the entire family of snake toxin patterns has a reference PROSITE motif expression. This pattern is useful for comparison with SRE-DNA motifs, as there are enough similarities in the PROSITE regular expression language and SRE-DNA that similar patterns can often be detected between them in many experiments.

The following is done to create a set of unaligned sequences from the above aligned data. First, duplicate sequences are removed. Then the aligned sequences are padded evenly on each side with randomly generated sequences of amino acid codons. The length of unaligned sequences for all experiments is 150 codons. When possible, a subset of these unaligned sequences is used as training data, and the remainder are testing data. Otherwise, in cases when there are few examples, the entire set of sequences is used for training. Using artificial sequences such as these has advantages and disadvantages. A possible disadvantage is that, unlike real unaligned data, the synthesized unaligned data's random padding is unlikely to share subsequences outside of the functional domain of the protein in question. This suggests that the unaligned sequences we are using may be easier to process than what might be seen in a more realistic production environment. On the other hand, an advantage in using them is that distracting noise is being avoided. This permits the experiments to focus on the basic problem of SRE-DNA motif synthesis, without becoming encumbered by noisy instances of shared artifacts that arise in real unaligned sequences. In the future, the problem of evolving SRE-DNA motifs in a noisy, more realistic environment should be addressed.

Negative examples are randomly-generated amino acid sequences, each having a length approximately the size of the original aligned PROSITE data (the *window size*). Again, this is not as realistic as is possible. A more challenging negative set would consist of a variety of real sequences, many of which might be closely related to the family belonging to the positive training set. For the purposes of this research, however, a randomly-generated negative set is adequate for ensuring that evolved motif expressions are sufficiently discriminating.

Table 1 summarizes characteristics of the example protein sequences used for the GP experiments. The amino acid oxidase and scorpion toxins were chosen because the PROSITE source motifs were of intermediate complexity. The zinc fingers, snake toxin, and kazal inhibitors were chosen in order to compare the results to that of Hu.[15)] The protein names and PROSITE accession numbers are in column 1. Column 2 (*lab*) contains a shorthand label used in subsequent figures. The m column gives the maximum mask size permitted in SRE-DNA expressions for that protein family. It is based on the maximum mask size used in the PROSITE motif for that family, and is typically a little larger than used by PROSITE in order to give motif evolution some extra freedom. The minimum probability value (p) is the minimum computed probability required by an SRE-DNA expression to continue processing, before terminating expression interpretation. The w value is the size of the window used for that protein set, and is configured to be large enough to cover the largest aligned sequence for the family as defined by its source PROSITE motif. Finally, *s1* and *s2* are the sizes of the disjoint training and testing sets respectively.

5.2 SRE-DNA Definition

DCTG-GP was used to derive 3 restricted variations of SRE-DNA. Grammars for these variants are shown in Figure 2. All the grammars offer various levels of grammatical constraints on full SRE, while at the same time use important SRE-DNA operations such as probabilistic skip and/or iteration. The grammars also encode general characteristics similar to what are found in established PROSITE motifs for protein families. In particular, the use of alternating skip and mask expressions will be favoured in evolved solutions.

One motivation for using constrained SRE-DNA is the fact that unrestricted SRE-DNA will generally result in inefficient expressions. For example, previous work[25)] established that SRE-DNA's choice operator is distinctly un-

$$
\begin{array}{lrcl}
G_1 & \mathit{expr} & ::= & \mathit{guard} \mid \mathit{guard}\!:\!\mathit{expr} \mid \mathit{expr}^{+p} \\
 & \mathit{guard} & ::= & \mathit{mask} \mid \mathit{mask}\!:\!\mathit{skip} \\
 & \mathit{skip} & ::= & x^{+p} \\
 & & & \\
G_2 & \mathit{expr} & ::= & \mathit{guard} \mid \mathit{guard}\!:\!\mathit{expr} \\
 & \mathit{guard} & ::= & \mathit{mask} \mid \mathit{mask}\!:\!\mathit{skip} \\
 & \mathit{skip} & ::= & x^{*p} \mid x^{+p} \\
 & & & \\
G_3 & \mathit{expr} & ::= & \mathit{gexpr} \mid \mathit{gexpr}^{*p} \mid \mathit{gexpr}^{+p} \\
 & \mathit{gexpr} & ::= & \mathit{guard} \mid \mathit{guard}\!:\!\mathit{expr} \mid \mathit{expr}\!:\!\mathit{guard} \\
 & \mathit{guard} & ::= & \mathit{mask} \mid \mathit{mask}\!:\!\mathit{skip} \\
 & \mathit{skip} & ::= & x^{*p} \mid x^{+p}
\end{array}
$$

Fig. 2 SRE-DNA Variations

desirable for denoting the types of sequences being analyzed here. The choice operator promotes intron material in expressions, which is program code which does not contribute meaningfully to computations. It was found that this operator could be removed without any loss in expressiveness, at least with the sequences studied here (its value as a nondeterministic operator is replaced by mask terms). Likewise, totally unrestricted iteration tends to result in intron material, as well as very inefficient expressions. Restricting iteration is therefore practical.

Grammar 1 permits nested + iteration and + skip. Iteration is restricted by the use of guard terms, which forces iteration to be applied to suffixes of concatenated expressions. In other words, guards create a bias towards prefix consumption, which promotes efficient sequence interpretation. Grammar 1 also permits directly nested iteration. Nesting should not be common, however, because nested iterative expressions usually have low probabilities, and hence low fitness values. They therefore become extinct during evolution.

Grammar 2 is the only grammar without the iteration operator, and is the language most similar to PROSITE's motif language. Expressions in grammar 2 will take the form of alternating masks and skips. Both * and + skip are used.

Grammar 3 is the least-constrained grammar. Besides using both + and * iteration, it permits a more symmetric distribution of iteration expressions on either side of expressions (unlike grammar 1's bias towards iteration on the right-side of expressions). Like grammar 2, both * and + skip are used. However, unlike grammar 1, nested iteration is not allowed. This is done via the use of the *gexpr* nonterminal.

The entire syntax and semantics of all the languages denoted in Figure 2 are implemented together in DCTG-GP. The semantic rules of the DCTG permit various interpretation and analytical processing to be programmed within the grammatical definitions. An advantage of this is that many grammatical devices are concisely denoted with a DCTG. For example, the actual implementation of grammar 3 does not use the distinct nonterminals *expr* and *gexpr*. Rather, the semantics for *expr* are used to directly test whether expressions are iterative or not, in order to ensure that iteration is not nested. Using these semantic devices greatly reduces the complexity of the grammar, which in turn promotes efficiency of the GP implementation.

Note that, although not encountered in the sequences studied here, SRE-DNA's iteration operators permit the denotation of protein repeats (eg. Huntington's polyCAG pattern).

5.3 Fitness Evaluation

Practically speaking, regular motif languages such as PROSITE's and SRE-DNA are most suitable for relatively small sequences. Large sequences require impractically large expressions, which in SRE-DNA's case, yield very small probabilities. The aligned segment of sequences shared by a family is but a fraction of the overall unaligned sequence. Therefore, to make processing more efficient, the GP system requires that the user supply a window size parameter, which defines the length of the longest window or aligned sequence length of interest. In addition, the SRE-DNA interpreter onlys succeeds in evaluating an expression if the entire SRE-DNA expression is used. For example, in *E:F* (where *F* does not recognize an empty string), if an input string is fully consumed by *E*, then interpretation of the entire expression fails because *F* has not been used. Taken together with the window length, the result is that successful SRE-DNA expressions are ones that are used in their entirety in recognizing a complete window from the unaligned sequence.

A candidate motif must recognize members of the protein family of interest, while at the same time reject non-member sequences. Likewise, a fitness measure must balance the acceptance and rejection characteristics of expressions. Consider the formula,

$$Fitness = N + NegFit - PosFit$$

NegTot and *PosFit* are the negative and positive training scores respectively, and N is the number of positive (and negative) training sequences. A hypothetical solution can yield a total *PosFit* score of N (N positive examples multiplied by a score of 1 each, which is illustrated below). The *NegFit* score will be 0 for a correct solution. Therefore, the value of *Fitness* in the above forula yields 0 for a perfect solution. This is impossible to obtain in practice, because the values represented by the *PosTot* term are usually small, due to the way probability scores are used within them.

Positive scoring is performed on each positive example sequence. A sequence is processed by extracting every consecutive window from it, and computing a fitness *Fit* on each window. If a sequence e_i has length $|e_i|$, and the window size is w (where $w < |e_i|$), then there are $|e_i| - w + 1$ windows to process. For example, the sequence *abcdef* has 4 windows of length 3: *abc*, *bcd*, *cde*, and *def*. The positive fitness formula is then:

$$PosFit = \sum_{e_i \in Pos} maximum(Fit(win_j(e_i)))$$

where e_i is a positive example sequence being processed, and win_j is one of its j windows. Here, for each positive sequence e_i, each window win_j is extracted from it, and a positive score is obtained. The maximum of all the window scores for a sequence is used for that sequence. Finally, all these maximal scores for all the positive sequences in set *Pos* are summed together, giving an overall positive score.

Positive fitness evaluation of a window incorporates two measurements: the probability of recognizing the window, and the proportional amount of the window recognized in terms of its length. Consider the formula,

$$Fit(win) = \frac{1}{2}\left(Pr(s_{max}) + \frac{|s_{max}|}{|e|}\right)$$

Here, s_{max} is the longest prefix recognized from some window win, and $|s_{max}|$ is its length. The term $Pr(s_{max})$ is the probability of recognizing s_{max}. The second term measures the proportion of the window recognized. The fitness pressure introduced by this formula is to favour expressions that recognize entire windows with high probabilities. In early generations, the window-length term dominates the score, which forces fitness to favour expressions that recognize large portions of windows. The probability term still comes into consideration, however, especially when the population converges to expressions that recognize windows from a significant number of sequences. At that time, the probability fitness measure favours expressions that yield high probabilities.

Negative fitness scoring is calculated as:

$$NegTot = maximum(Fit(n_i)) * N$$

where $n_i \in Neg$ (negative examples). The highest obtained fitness value for any recognized negative example suffix is used for the score. A discriminating expression will not normally recognize negative examples, however, and so $Fit(n_i) = 0$ for most n_i.

5.4 Genetic Programming Parameters

Table 2 GP Parameters

Parameter	**Value**
GA type	generational
Maximum generations	100
Maximum runs/experiment	6
Functions	SRE-DNA variants
Terminals	amino acid codons, probabilities
Population size (initial)	2000
Population size (culled)	1000
Unique population	yes
Max. depth initial popn.	12
Max. depth offspring	24
Tournament size	7
Elite migration size	10
Retries for reproduction	3
Prob. crossover	0.90
Prob. mutation	0.10
Prob. internal crossover	0.90
Prob. terminal mutation	0.75
Prob. SRE crossover	0.25
Prob. SRE mutation	0.30
SRE mutation range	0.1

Table 2 lists genetic programming parameters used for the experiments. Generational evolution is performed, in which distinct populations are evolved during each generation. Each population always consists of syntactically unique expressions. The initial population size is 2000 individuals, which is generated using Koza's ramped half-and-half tree generation scheme.[18] Half the trees generated are *grow trees*, in which a terminal or nonterminal can be randomly selected as the root of each subtree, while the remaining half are *full trees*, in which nonterminals are always selected so long as the tree depth limit is not exceeded. During tree generation, the tree depths are staggered (or *ramped*) from depths 2 through 12. The result is a population of random expressions having a fair distribution of varied tree shapes. This initial population is culled by selecting the 1000 most fit individuals. This oversampling of the initial population helps discard the weak expressions which commonly arise during random tree generation.

Nonterminals and terminals are determined by the SRE-DNA grammar variant used (Section 5.2). All the grammars use amino acid codons in mask terms, as well as numeric fields for probabilities. Reproduction operations on these fields work in a number of ways. Crossover and mutation are selected with probabilities of 90% and 10% respectively. When one of these reproductive operations is selected, either a conventional GP version or a hybrid SRE-DNA version may be applied. For example, the table entry indicating the probability of SRE crossover means that, when crossover is to be performed, there is a 25% chance that SRE-DNA crossover is used. Conventional grammatical crossover selects a subtree having the same nonterminal or terminal type of the grammar in each parent, and swaps these subtrees, yielding the offspring. SRE-DNA crossover works on masks: two mask fields are selected in the parents, and the two offspring created are identical to each respective parent, but with the selected masks a merge of the elements from the parent masks.

Conventional grammatical mutation takes a subtree, and replaces it with a randomly generated subtree of the same nonterminal or terminal type. SRE-DNA mutation can involve a number of special operations: (i) a numeric field is perturbed $\pm 10\%$ of its original value; (ii) a mask has a random element inserted into it; (iii) a mask has a random element removed from it; and (iv) a mask element is randomly replaced.

5.5 Solution Refinement

```
For each mask $mask_i$ in expression $expr$:
    For each element $\alpha_j \in mask_i$:
        - Remove $\alpha_j$ from $mask_i$, and call new expression $expr'$.
        - If fitness $expr' >$ fitness $expr$
            then $expr \leftarrow expr'$.
```

Fig. 3 Mask refinement

Often, an evolved solution motif can be further improved after the run has terminated. Its overall performance as measured by its probability score on recognized sequences can be increased by refining its mask terms. Many experiments result in solution expressions that have unnecessarily large masks, which yield smaller probabilities than optimally-sized ones. Although mask mutation might occassionally delete extraneous mask codons, this occurs too infrequently during evolution to prevent these bloated masks from appearing in solutions.

Mask refinement is performed on solution expressions by the procedure in Figure 3. The procedure performs hill-climbing transformations on an expression, by deleting the mask elements whose removal improves overall fitness. The algorithm is a greedy one, and it is not guaranteed to find the optimal refinement for an expression. In other words, depending on the SRE-DNA grammar used, some combinations of mask refinements might yield higher probabilities than others. The algorithm used is adequate for the majority of expressions evolved in this paper.

§6 Results

Table 3 summarizes the training and testing performance results for the protein families in Table 1. G_i is the grammar used from Figure 2. The best measure of a motif's performance is to compute the average probability obtained when recognizing positive example sequences. We call this value the *probability score*, and denote it by $\overline{pr}$. Then, "avg $\overline{pr}$" is the average probability scores for the best motifs for all six runs. Similarly, "best $\overline{pr}$" is the probability score of the overall best motif. Testing was performed on the best evolved motif for every grammar (the motif used to obtain "best $\overline{pr}$"). "%tp" is the percentage of true positives obtained by the best motif on the testing set. "$\overline{pr}$" is the probability score obtained for the recognized testing sequences. This score is normalized for the true positives recognized. The percentage of false negatives (positive test cases yielding a probability of 0.0) is in column "%fn", and is merely 100 - %tp.

Table 3 Solution statistics for example proteins

		Training		*Testing (best)*			
Family	G_i	avg $\overline{pr}$	best $\overline{pr}$	%tp	$\overline{pr}$	%fn	%fp
AAO	G_1	1.846e-4	8.565e-4				
	G_2	5.306e-5	1.914e-4	–	–	–	–
	G_3	5.414e-5	2.143e-4				
ScT	G_1	1.312e-7	5.170e-7				
	G_2	2.349e-8	5.537e-8	–	–	–	–
	G_3	2.451e-5	1.469e-4				
ZF1	G_1	1.113e-11	2.898e-11	67.4	3.562e-11	32.6	0.0
	G_2	7.330e-11	4.128e-10	81.3	4.645e-10	18.7	0.0
	G_3	6.027e-11	3.374e-10	89.1	4.772e-10	10.9	0.0
ZF2	G_1	9.614e-3	2.019e-2	100.0	1.959e-2	0.0	0.0
	G_2	1.278e-2	1.919e-2	100.0	1.953e-2	0.0	0.0
	G_3	1.415e-2	2.542e-2	100.0	2.410e-2	0.0	0.0
SnT	G_1	1.202e-7	2.398e-7	80.3	1.975e-7	19.7	0.0
	G_2	5.153e-8	1.633e-7	82.7	1.311e-8	17.3	0.0
	G_3	9.786e-8	1.857e-7	64.6	2.967e-7	35.4	0.0
KI	G_1	2.964e-9	5.461e-9	80.0	5.416e-9	20.0	0.0
	G_2	1.340e-8	3.772e-8	68.8	3.337e-8	31.2	0.0
	G_3	2.354e-8	4.546e-8	70.4	5.424e-8	29.6	0.0

Finally, false positives (erroneously identifying a negative example as a member sequence) is in column "%fp". The actual testing results for the Kazal inhibitor (KI) experiments using all three grammars can be viewed on the web[*3].

Table 4 presents some additional testing results on the best solutions tested in Table 3. False negative (FN) and false positive (FP) sequences were obtained from the PROSITE database for four of the protein families. This data represents sequences which either were missed by the original PROSITE motif (FN), or were erroneously classified as belonging to the protein family (FP). Not all the families have FN and FP available (the AAO and ScT proteins had none, and are omitted). In the #FN column in Table 4, the first number represents the number of false negative sequences identified by the SRE-DNA motif. The number in parentheses is the total number of FN sequences extracted from PROSITE for that protein. In terms of matching false negative sequences, higher

[*3] http://www.cosc.brocku.ca/~bross/research/kazal.html

Table 4 Additional testing results

Family	G_i	#FN (tot)	$\overline{pr}$	#FP (tot)	$\overline{pr}$
ZF1	G_1	2 (3)	7.764e-12	12 (12)	3.882e-11
	G_2	1	1.925e-10	6	1.724e-10
	G_3	2	2.163e-11	7	1.829e-10
ZF2	G_1	4 (12)	2.398e-2	5 (5)	1.918e-2
	G_2	4	2.398e-2	5	1.918e-2
	G_3	4	1.942e-2	5	1.831e-2
SnT	G_1	2 (10)	6.662e-8	(0)	–
	G_2	2	3.707e-8		–
	G_3	0	0.0		–
KI	G_1	0 (1)	0.0	0 (2)	0.0
	G_2	0	0.0	1	8.569e-9
	G_3	0	0.0	1	4.371e-9

numbers in the FN column are preferred, as they indicate that the SRE-DNA motif recognized candidate proteins that were missed by the original PROSITE regular motif expression. The $\overline{pr}$ value is the normalized average probability of recognizing the FN sequences. The next two columns are similar to the FN ones, except that false positive sequences are tested. Here, lower numbers in the FP column are preferred, since they mean that SRE-DNA motifs were more discriminatory than the PROSITE motifs, correctly classifying non-member sequences that PROSITE erroneously permitted.

Tables 5 and 6 list the best solutions for the different protein families and SRE-DNA variations, along with the source PROSITE motif used to obtain the example sequences. Note that the PROSITE motif language differs from SRE-DNA (see Section 2.3).

§7 Discussion

Firstly, the quality of results is influenced by the level of precision of the experimental parameters. For example, due to time constraints, only 6 runs were undertaken per experiment, which is inadequate for statistically meaningful conclusions to be drawn. Additionally, more precise SRE-DNA interpretation will occur when lower minimal probabilities are used by the interpreter (see Table 1). For example, early experiments suggested that the zinc finger C2H2 (ZF1) family required a smaller minimum probability in order to get meaningful

Table 5 Best evolved motifs for protein families. The motif beside family label is the source PROSITE motif.

AAO	$[ilmv](2) : h : [ahn] : y : g : x : [ags](2) : x : g : x(5) : g : x : a$
G_1	$h : x^{+.1} : y : g : x^{+.17} : [gs] : x^{+.17} : g : x^{+.1} : [iqt] : x^{+.17} : [ghs] : x^{+.13} : g : x^{+.1} : a : x^{+.19}$
G_2	$h : x^{+.1} : y : x^{*.11} : g : x^{+.19} : [gs] : x^{+.16} : g : x^{+.19} : [hst] : x^{+.19} : g : x^{+.16} : a : x^{+.19}$
G_3	$h : x^{+.1} : y : g : x^{+.19} : [gs] : x^{+.19} : g : x^{+.19} : [hst] : x^{+.19} : g : x^{+.17} : a : x^{+.16}$
ScT	$c : x(3) : c : x(6,9) : [ags] : k : c : [imqt] : x(3) : c : x : c$
G_1	$c : x^{+.19} : ((([acg] : x^{+.12} : [cg] : x^{+.19} : k : x^{+.12} : [ct] : x^{+.11} : c : x^{+.19})^{+.19})^{+.19})^{+.19}$
G_2	$c : x^{+.19} : [gkps] : x^{+.19} : [acgk] : x^{+.19} : [agn] : x^{+.19} : c : x^{+.19} : [gn] : x^{+.19} : c : x^{+.1} : c : x^{+.19}$
G_3	$[ck] : x^{+.17} : c : x^{+.18} : k : x^{+.19} : (c : x^{*.15} : k : x^{+.19} : a : x^{+.18} : m : x^{+.15} : f : x^{+.1} : k : c : x^{+.1})^{*.19} : [gn] : x^{+.18} : k : c : x^{+.18}$
ZF1	$c : x(2,4) : c : x(3) : [cfilmvwy] : x(8) : h : x(3,5) : h$
G_1	$[acfkr] : x^{+.19} : [acfsv] : x^{+.19} : [eklr] : x^{+.19} : [fklnst] : x^{+.19} : [adklrt] : x^{+.19} : [hlnrsv] : x^{+.19} : [kr] : x^{+.19} : [hl] : x^{+.19} : [filnrtvw] : x^{+.19}$
G_2	$c : x^{+.19} : c : x^{+.19} : [kqr] : x^{+.18} : [afklr] : x^{+.19} : [lqrs] : x^{+.19} : h : x^{+.18} : [hklrt] : x^{+.19} : [acdhkt] : x^{+.19}$
G_3	$c : x^{+.19} : c : x^{+.19} : [hkr] : x^{+.19} : [afkqrsty] : x^{+.19} : [hlnstv] : x^{+.19} : [ahklnrst] : x^{+.19} : [hkr] : x^{+.19} : [hr] : x^{+.19}$

results. Using a low probability like this enhances precision, but at the expense of computation time. If even smaller probability limits were used, both the training and testing results may be improved, due to the increased likelihood of recognizing some examples.

Other parameters, such as mask size, iteration ranges, window size, and training set size, also affect results. Preliminary runs suggested that larger masks, as used in the zinc finger cases, usually result in lower quality solutions. This implies that small masks are more naturally suitable for SRE-DNA motifs, in comparison to PROSITE's motifs. Similarly, large iteration ranges greater than 0.25 often resulted in the evolution of motifs that would tend to skip large portions of relevant aligned subsequences. The value used in these experiments

Table 6 Best evolved motifs for protein families (cont.).

ZF2	$c : x : h : x : [filmvy] : c : x(2) : c : [ailmvy]$
$G_{1,2}$	$c : x^{+.1} : h : x^{+.19} : c : x^{+.19} : c : x^{+.1}$
G_3	$c : x^{+.1} : h : x^{+.19} : (f : x^{*.11})^{*.19} : c : x^{+.19} : c : x^{+.1}$
SnT	$g : c : x(1,3) : c : p : x(8,10) : c : c : x(2) : [denp]$
G_1	$g : c : x^{+.19} : c : x^{+.19} : [kps] : x^{+.19} : [klv] : x^{+.19} : c : c : x^{+.19} : [dt] : x^{+.19}$
G_2	$g : x^{*.11} : c : x^{+.19} : c : x^{+.19} : [gks] : x^{+.19} : [dklv] : x^{+.19} : c : c : x^{+.19} : [dt] : x^{+.18}$
G_3	$((((g : c : x^{+.19})^{+.1} : c)^{+.12} : p : x^{+.18} : [gkns] : x^{+.19} : [dgls] : x^{+.19} : [iklt] : x^{+.19})^{+.1} : c : x^{+.17})^{+.1} : [dt] : x^{+.19}$
KI	$c : x(7) : c : x(6) : y : x(3) : c : x(2,3) : c$
G_1	$[cv] : x^{+.19} : [elrvy] : x^{+.19} : ([cps] : x^{+.19} : [cgs] : x^{+.19} : [cty] : x^{+.19} : n : x^{+.11} : [ct] : x^{+.19} : [ct] : x^{+.17})^{+.1}$
G_2	$[ct] : x^{+.19} : [celr] : x^{+.19} : [elpqr] : x^{+.19} : [cgk] : x^{+.19} : [dgk] : x^{+.19} : [fty] : x^{+.18} : [dnr] : x^{+.18} : c : x^{+.19} : c : x^{+.18}$
G_3	$c : x^{+.19} : [eir] : x^{+.19} : [epr] : x^{+.19} : c : x^{+.19} : [dgks] : x^{+.19} : y : x^{+.19} : [cn] : x^{+.19} : c : x^{+.19}$

(0.19) resulted in the most interesting motifs.

With respect to the results in Table 3, the results obtained range from acceptable (those with testing percentages less than 70%) to excellent. With respect to the training results, the best SRE-DNA motif for each family was produced twice by grammar 1, once by grammar 2, and three times by grammar 3. Often, the difference in probabilities between best motifs by different grammars was many orders of magnitude. For example, consider the scorpion toxin experiment (ScT), in which the best grammar 3 solution had an average probability over 280 times larger than the next best grammar 1 motif, and over 2600 times larger than the grammar 2 motif. Unfortunately, neither the ScT nor AAO families had enough examples with which to perform testing.

The ZF1 runs were hindered by the small probabilities inherent with the motifs. This seems to be a product of the large mask size (10) combined with the large window size (25). It was observed that many recognized probabilities for examples were in the 1e-12 range. Others may have been smaller than 1e-13, which was beyond the probability limit for those runs, and thus would have been terminated.

In the ZF2 and KI cases, the observed $\overline{pr}$ for testing roughly correlate with the best $\overline{pr}$ for training. The ZF2 motifs were simple, and runs often converged to the identical motif.

The snake toxin testing for G_3's motif suggests that this particular motif may be overtrained. Although it recognized all of the training cases, it could only recognize 64% of the testing set.

The results in Table 4 indicate that SRE-DNA often performed better than the PROSITE equivalents with respect to false negative and false positive sequences. However, this must be considered in balance with the testing results in Table 3.

Cyclic subsequences are not common in the protein sequences studied here. This might imply that the iteration operator is detrimental for GP, and that grammar G_2 would be the most effective variation of SRE-DNA. The results in Table 3, however, suggest otherwise. Iteration can be argued to be beneficial for evolution, given the number of best solutions found with grammars G_1 and G_3 (the grammars with iteration operators). One hypothesis for this is that iteration aids evolution by allowing richer intermediate expressions to evolve, because iteration permits the recognition of greater numbers of positive example subsequences. This occurs because an SRE-DNA expression with iteration recognizes more strings than one without it (ignoring the effects of skip terms, which are controlled by the use of negative examples). Upon inspecting the motifs in Tables 5 and 6, however, it is clear that iteration is not an important factor with respect to the construction of final motifs. In 7 of the 12 experiments using grammars with iteration (G_1 and G_3), the iteration operator did not arise in the best motifs. In the remaining 5 motifs, iteration is incidental, and takes the form of intron material to protect useful terms. An obvious example of this is the G_1 solution for ScT in Table 5. Iteration could be removed from these expressions, resulting in motifs with higher probability performance.

Considering the structure of SRE-DNA expressions in Tables 5 and 6, it is clear that majority of evolved solutions are much more complex than the source PROSITE motifs. One reason for this complexity is the fact that expression size was not accounted for in the fitness function. Expression parsimony could be enhanced by incorporating a score for expression complexity, or reducing the GP parameter limiting tree sizes. Furthermore, evolved GP programs are often more complex than manually programmed ones.

§8 Related work

8.1 Other representations: PROSITE, HMMs

SRE-DNA is similar to the regular expressions used by PROSITE and related databases.[6, 12] SRE-DNA is essentially a PROSITE-like language enhanced with a probability model. Not surprisingly, many evolved motifs in Figures 5 and 6 are similar to the PROSITE source motifs. Some of the SRE-DNA motifs, however, vary substantially from their PROSITE equivalents. This is a combined result of the methodology used during their acquisition with GP, such as the fitness evaluation strategy and other GP parameters, and nuances in the SRE-DNA grammar used.

SRE-DNA's main expressive advantage over PROSITE's language is its probabilistic model. A SRE-DNA motif has associated with it a probability distribution, and each member sequence has a corresponding probability computed with it. Non-member sequences have a zero probability. With PROSITE's language, there is no associated probability or scoring scheme; sequences are either members or not members of the motif expression in question. Although the overall structure of the regular expression underlying an SRE-DNA motif is often just as deterministic as with a PROSITE equivalent, the probability distribution helps to numerically justify the strength of membership of sequences within a family. For example, sequences that use longer gaps will have lower probabilities than those with smaller gaps.

SRE-DNA is more expressive at denoting variable-length terms and gaps than PROSITE motifs. In PROSITE, a numeric range such as "X(2, 4)" indicates a variable gap between 2 and 4 codons. In SRE-DNA (as used in this paper), a variable-length gap is expressed as a term like "$x^{+.10}$". Probabilistic iteration can permit gaps of any potential length, but at a probabilistic cost, since long gaps caused by highly-iterated terms have corresponding lower probabilities. A motif that requires a large gap should use an iteration with a higher iterative probability. SRE-DNA's use of probabilistic choice also enhances its expressiveness over PROSITE's motifs. Although choice was not used here, other protein families may benefit by it.

A comparison of SRE-DNA with Hidden Markov Models (HMM's) is also useful.[22] A HMM is a stochastic finite automata. In the HMM model of Krogh *et al.*, a target sequence of length k has associated with it an HMM graph (automata) of approximately $3k$ nodes and $9k$ arcs. Nodes are either match states, insert states, or delete states. Match states arise when sequences match codes of the sequence, delete states cause codes to be skipped, and insert states introduce codes. Every transition between these states has associated with it a probability. Codons themselves have specific probabilities when they are seen when moving between states. All these probabilities can be determined automatically via various training algorithms. Given a candidate sequence, the codons invoke translations through the HMM, until the sequence terminates, resulting in a probability score for that sequence.

HMM's have shown good performance in protein classification, and have performed well in comparison to PROSITE's regular motifs.22) HMM's are not intended as user-level interfaces to protein databases, unlike SRE-DNA and PROSITE motifs. HMM's are much less rigid than regular pattern motifs, as their architecture is designed to permit variations of sequence patterns within a set window size. In comparison, PROSITE and SRE-DNA motifs specify fairly rigid pattern matching criteria (although SRE-DNA motifs are somewhat more relaxed due to probabilistic iteration). The fact that HMM's are more forgiving with sequence variations than regular pattern motifs is a mixed blessing. On one hand, an HMM is able to represent a large assortment of structurally dissimilar sequences, by fitting a probability distribution over them. The HMM architecture is fixed with respect to the maximum sequence size permitted. Should the sequences be very dissimilar, however, the resulting information content encoded in the HMM's probability distribution will become increasingly negligible. PROSITE patterns have a similar ability to permit structure variation through the use of variable gaps and codon choices. It treats sequence identification as a boolean operation, and there is no probability associated with membership. Although SRE-DNA expressions can denote dissimilar sequences with the choice operator, the resulting motif will grow in size and complexity.

Reconciling the expressive differences between HMM's, SRE-DNA, and PROSITE expressions is not straight-forward. There is a 1-to-1 mapping between regular expressions and finite automata: each can be automatically compiled to the other.14) Similarly, there is a mapping between stochastic regular expressions and stochastic finite automata. This implies that the languages denoted by PROSITE, SRE-DNA and HMM motifs are theoretically equivalent. However, even though these models are inter-translatable, it does not imply that given sets of protein sequences are always more conveniently denoted in one model over another. It must be realized that all these motif models are simple structural characterizations of the real factor of importance – the 3D topology of the protein.

8.2 Motifs and Genetic Programming

The first successful application of genetic programming towards evolving PROSITE-style motifs for unaligned biosequences is by Hu.[15] There are similarities and differences between Hu's GP methodology and ours. Similar to our experiments, Hu pre-specifies the maximum window length, maximum gap length, and gap flexibility (ranges for variable gaps of the form "X(i, j)"). Whereas we use amino acid codons, Hu uses a combination of nucleotide and amino acid codons, as well as predefined codon subset codes. The use of predefined subsets is interesting, as it uses domain knowledge to establish useful combinations of possible codons for mask terms. Hu seeds the initial population with substrings from the example set. Although our initial population was randomly generated, in hindsight, seeding might prove advantageous.

One major difference between Hu's approach and ours is his extensive use of greedy local optimization to refine expressions during evolution. Firstly, each pattern sub-term is replaced by a wildcard; if the fitness improves, the wildcard is retained. Then all the legal values for each variable gap are iterated through, to find the combination giving the highest fitness. Although Hu does not specify the performance cost of this optimization procedure, it certainly must be significant. SRE-DNA evolution circumvented the need for gap placement and variable gap refinement, as the skip fields were automatically refined during evolution. We applied greedy optimization to mask terms, but only on the best solution at the end of a run.

Table 7 Comparison of PROSITE, Hu, and SRE-DNA motifs for snake toxin (SnT)

PROSITE:	$g : c : x(1,3) : c : p : x(8,10) : c : c : x(2) : [denp]$
Hu #1:	$g : c : x(1,3) : c : p$
Hu #2:	$c : c : x(1,2) : [denp]$
SRE-DNA G_2:	$g : x^{*.11} : c : x^{+.19} : c : x^{+.19} : [gks] : x^{+.19} : [dklv] : x^{+.19}$ $: c : c : x^{+.19} : [dt] : x^{+.18}$

A qualitative comparison of Hu's results with ours is difficult, because of the difference in motif languages, as well as Hu's lack of any numerical analyses of his results (eg. training and testing measurements). Hu investigated the same ZF1, ZF2, SnT, and KI protein families that we did. Hu's resulting motifs for

ZF1 and KI are impressive, as they are nearly identical to the PROSITE motif expressions. In other results, Hu's motifs were simpler than the PROSITE ones. For example, Table 7 compares the PROSITE, Hu, and SRE-DNA G_2 motifs for SnT. The Hu motifs are clearly too short, as they cover sequences between 4 and 7 codons long, which is far smaller than the maximum window size of 22 codons used in the PROSITE motif. In the G_2 motif, the first 4 terms of the PROSITE motif (up to "p") are nearly identically covered, as is the "c:c" term later in the sequence. The G_2 motif needs to break up the long "x(8,10)" gap with a series of iterations and masks, because the iteration limit of 0.19 prevents a single skip expression from being applied too often (a skip of 0.19 iterated 10 times gives a probability of 5e-8).

The other GP work evolving regular expression motifs for unaligned sequences is by Koza, Bennett, Andre and Keane.[21)] The 2 protein families studied are the D-E-A-D box and manganese superoxide dismutase families. The target language is a subset of the PROSITE motif language, in which gap expressions (fixed or variable) are not used. They also use GP with ADF's, which enable the modularization of common subexpressions in evolved motifs. This is advantageous for the protein families studied, which contain repetitive elements. The two protein families investigated have very small window lengths, of length 8 and 9 respectively. This is considerably smaller than the lengths used in this paper, which ranged up to 25 codons. They reported good results with their work, as one of their evolved motifs improved upon an established PROSITE source motif, due to its replacement of a PROSITE gap with an explicit choice of amino acid codons.

Unlike ours or Hu's work, Koza *et al.* did not need to supply preset window limits. Although fewer user-specified parameters implies increased automation, which is always an ideal in machine learning, a preset window size was used here out of necessity: an indeterminately large window results in long interpretation times, especially when iteration is used. Given the complexity of SRE-DNA and the lengths of the sequences studied here, it is unlikely that an unspecified window would have been feasible.

§9 Conclusion

This is a first attempt at evolving SRE-DNA motifs for unaligned sequences, and the results are promising. The expressive characteristics of SRE-DNA motifs are highly dependent upon language restrictions used, such as grammatical restrictions, mask sizes, and iteration ranges. Some of these constraints may be more suitable to particular protein families than others. Further reseach is required to understand in more depth the effect of language constraints on motifs, and which constraints are most practically applicable to the majority of protein families.

One weakness in this paper's application of SRE-DNA is that motifs do not ensure fixed-size sequences for conserved regions. Conserved regions rarely vary in size, since extra or missing amino acids will cause major changes to functionality. GP usually evolves appropriate iteration values that produce good results for the training set. These terms are not fixed in length, however, if iterative gaps are embedded in them. An interesting direction to take in the future is to use an enhanced SRE-DNA grammar that explicitly accounts for conserved regions. For example, the grammar could generate motifs of the form,

$$< var.\ region > \ < conserved\ region > \ < var.\ region >$$

The grammar rules for the variable regions would use iterative probabilities as is done currently, while the conserved region would forgo the use of all iteration operators.

SRE-DNA is a new motif language for protein sequences. As with PROSITE motifs, much of the characterisation of a family of proteins is inherent in the structure of the SRE-DNA motif. Additionally, like HMM's, an SRE-DNA motif defines a probability distribution over the member sequences. In this two-level approach to sequence representation, the more discriminating linguistic level is the regular expression, since it implicitly defines which sequences belong to the modeled family. The probability fields refine this representation by ascribing a probabiity distribution to member sequences. Like PROSITE's motif language, SRE-DNA motifs tend to grow in size in accordance with the complexity and size of the sequences being characterized. GP tends to create larger expressions as well, which can often be simplified afterwards. SRE-DNA's motif size contrasts to HMM motifs, whose structure is fixed for the maximum size of sequences being represented. Unlike HMM's, however, SRE-DNA is a relatively user-friendly motif language, and can be used for interactive database access.

An empirical comparison of SRE-DNA and other representation models would be interesting. Since these different motif representations can vary substantially, however, a direct comparison of them may not be too fruitful. It is worth realizing that the biological functionality of proteins is entirely dependent upon the 3D structure of the protein molecules. Motif representations such as

SRE-DNA, HMM's, regular expressions, and context-free languages, are crude means for modeling characteristic features of proteins. Nevertheless, they are widely used, because they are relatively concise, computationally efficient, and amenable to automatic acquisition.

There are other directions for future investigations. Similar to the mask refinement performed on the final solutions, additional refinement of probability fields can be undertaken, which would improve the probability performance of motifs. Future work will investigate using a fuller variant of SRE-DNA to model sequences with repetitive elements and weighted choice. More sophisticated GP approaches, for example, the use of multiple populations, may result in better evolution performance. Finally, more runs should be undertaken to obtain more meaningful statistical results. A practical limitation to this is the speed of runs, which can be very slow when large masks and low probability limits are used. Porting DCTG-GP to a faster, compiled language such as C would be useful in this regard.

Our fitness scheme uses a strictly structural perspective of motif performance: if a regular language can recognize the positive sequences in a family, and reject sequences outside of it, then it is considered correct. This formal view of motif performance is practical and adequate for many problems, and therefore is adopted in most machine learning applications. On the other hand, such an approach is inherently limited. For example, it cannot account for converged functionality for distantly related proteins. Such real-world phenomena require additional information (domain-specific knowledge about protein structure) beyond what can be modeled by first-order structural principles themselves.

A comparison of different motif learning techniques is worth undertaking in the future. SRE-DNA motifs may be synthesizable by other machine learning algorithms, many of which may yield superior performance over GP.

Acknowledgement: The author thanks two anonymous referees for their helpful comments. This work is supported though NSERC Operating Grant 138467-1998.

References

1) H. Abramson and V. Dahl. *Logic grammars.* Springer-Verlag, 1989.
2) S. Arikawa, S. Miyano, A. Shinohara, S. Kuhara, Y. Mukouchi, and T. Shinohara. A Machine Discovery from Amino Acid Sequences by Decision Trees over Regular Patterns. *New Generation Computing*, 11:361–375, 1993.

3) P. Baldi and S. Brunak. *Bioinformatics: the Machine Learning Approach.* MIT Press, 1998.

4) W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming: An Introduction.* Morgan Kaufmann, 1998.

5) P. Bork and E.V. Koonin. Protein sequence motifs. *Current Opinion in Structural Biology*, 6:366–376, 1996.

6) A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the Automatic Discovery of Patterns in Biosequences. *Journal of Computational Biology*, 5(2):279–305, 1998.

7) A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen. Pattern Discovery in Biosequences. pages 255–270. Springer Verlag, 1998. LNAI 1433.

8) P. Bucher and A. Bairoch. A Generalized Profile Syntax for Biomolecular Sequence Motifs and its Function in Automatic Sequence Interpretation. In R.Altman *et al*, editor, *Proceedings 2nd International Conference on Intelligent Systems for Molecular Biology*, pages 53–61. AAAI Press, 1994.

9) V.K. Garg, R. Kumar, and S.I Marcus. Probabilistic Language Formalism for Stochastic Discrete Event Systems. *IEEE Trans. Automatic Control*, 44:280–293, February 1999.

10) S. Handley. Automated Learning of a Detector for the Cores of α-Helices in Protein Sequences Via Genetic Programming. In *Proceedings 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 474–479. IEEE Press, 1994.

11) S. Handley. Classifying Nucleic Acid Sub-Sequences as Introns or Exons Using Genetic Programming. In *Proceedings 3rd International Conference on Intelligent Systems for Molecular Biology (ISMB-95)*, pages 162–169. AAAI Press, 1995.

12) K. Hofmann, P. Bucher, L. Falquet, and A. Bairoch. The PROSITE database, its status in 1999. *Nucleic Acids Research*, 27(1):215–219, 1999.

13) J.H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1992.

14) J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

15) Y.-J. Hu. Biopattern Discovery by Genetic Programming. In J.R. Koza *et al*, editor, *Proceedings Genetic Programming 1998*, pages 152–157. Morgan Kaufmann, 1998.

16) K. Karplus, K. Sjolander, C. Barrett, M. Cline, D. Haussler, R. Hughey, L. Holm, and C. Sander. Predicting protein structure using hidden Markov models. *Proteins: Structure, Function, and Genetics*, pages 134–139, 1997. supplement 1.

17) M.J. Kearns and U.V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

18) J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

19) J.R. Koza. Automated Discovery of Detectors and Iteration-Performing Calculations to Recognize Patterns in Protein Sequences Using Genetic Programming. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 684–689. IEEE Press, 1994.

20) J.R. Koza, F.H. Bennett, and D. Andre. Classifying Proteins as Extracellular Using Programmatic Motifs and Genetic Programming. In *Proceedings 1998 IEEE World Congress on Computational Intelligence*, pages 212–217. IEEE Press, 1998.

21) J.R. Koza, F.H. Bennett, D. Andre, and M.A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, 1999.

22) A. Krogh, M. Brown, I.S. Mian, K. Sjolander, and D. Haussler. Hidden Markov Models in Computational Biology. *Journal of Molecular Biology*, 235:1501–1531, 1994.

23) B.J. Ross. Probabilistic Pattern Matching and the Evolution of Stochastic Regular Expressions. *International Journal of Applied Intelligence*, 13(3):285–300, November/December 2000.

24) B.J. Ross. Logic-based Genetic Programming with Definite Clause Translation Grammars. *New Generation Computing*, 2001. In press.

25) B.J. Ross. The Evaluation of a Stochastic Regular Motif Language for Protein Sequences. In L. Spector *et al.*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 120–128. Morgan Kaufmann, 2001.

26) Y. Sakakibara, M. Brown, R. Hughey, I.S. Mian, K. Sjolander, R.C. Underwood, and D. Haussler. Stochastic Context-Free Grammars for tRNA Modeling. *Nucleic Acids Research*, 22(23):5112–5120, 1994.

27) D.B. Searls. The Computational Linguistics of Biological Sequences. In L. Hunter, editor, *Artificial Intelligence and Molecular Biology*, pages 47–120. AAAI Press, 1993.

28) D.B. Searls. String Variable Grammar: a Logic Grammar Formalism for the Biological Language of DNA. *Journal of Logic Programming*, 24(1,2), 1995.