The Evaluation of a Stochastic Regular Motif Language for Protein Sequences

Brian J. Ross
Department of Computer Science
Brock University
St. Catharines, Ontario
Canada L2S 3A1
bross@cosc.brocku.ca

Abstract

A probabilistic regular motif language for protein sequences is evaluated. SRE-DNA is a stochastic regular expression language that combines characteristics of regular expressions and stochastic representations such as Hidden Markov Models. To evaluate its expressive merits, genetic programming is used to evolve SRE-DNA motifs for aligned sets of protein sequences. Different constrained grammatical forms of SRE-DNA expressions are applied to aligned protein sequences from the PROSITE database. Some sequences patterns were precisely determined, while others resulted in good solutions having considerably different features from the PROSITE equivalents. This research establishes the viability of SRE-DNA as a new representation language for protein sequence identification. The practicality of using grammatical genetic programming in stochastic biosequence expression classification is also demonstrated.

1 INTRODUCTION

The rate of biological sequence acquisition is accelerating considerably, and this data is freely accessible from biosequence databases such as PROSITE (Hofmann *et al.* 1999). Research in bioinformatics is investigating more effective technology for classifying and analysing this wealth of new data. One important problem in this regard is the automated discovery of sequence patterns (Brazma *et al.* 1998*a*). A sequence pattern, also known as a *motif* or *consensus pattern*, encodes the common characteristics of a set of biosequences. From one point of view, a sequence pattern is a signature identifying a set of related biosequences, and hence can be used as a means of database query. Alternatively, and perhaps more importantly, a motif can also characterizes the salient biological and evolutionary characteristics common to a family of sequences. The use of computational tools which automatically determine biologically meaningful patterns from sets of sequences is of obvious practical importance to the field.

The contributions of this research are two-fold. Firstly, the viability of SRE-DNA, a new motif language, is investigated. SRE-DNA shares characteristics of deterministic regular expressions and stochastic representations such as Hidden Markov Models (Krogh *et al.* 1994). Since full SRE-DNA is likely too unwieldy to be practical, this research investigates what restrictions to the language are practical for biosequence classification. To do this, genetic programming (GP) is used to evolve SRE-DNA motifs for aligned sequences. SRE-DNA's probabilistic basis can be exploited during fitness evaluation in GP evolution.

A second goal of this research is to test the practicality of logic grammar-based genetic programming in an application of bioinformatics. The system used is DCTG-GP, a logic grammar-based GP system based on definite clause translation grammars (DCTG) (Ross 2001*a*). With DCTG-GP, a variety of constrained grammatical variations of SRE-DNA are straight-forwardly defined and applied towards motif discovery.

Generally speaking, motif discovery for aligned sequences is a simpler problem than for unaligned sequences. With aligned sequences, the basic problem of determining the common subsequences amongst a set of sequences has been already determined. Nevertheless, a number of fundamental issues regarding the viability of SRE-DNA are more clearly addressable if aligned data is studied initially. In the course of these experiments, it was discovered that motif discovery for some families of aligned data is very challenging. This justifies studying aligned sequences before commencing on unaligned data.

Section 2 gives an overview of biosequence identification, stochastic regular expressions and DCTG-GP. Section 3 discusses experiment design and preparation. Results are reported in Section 4. A discussion concludes the paper in section 5.

2 BACKGROUND

2.1 Biosequence Identification

DNA molecules are double-stranded sequences of the four base nucleic acids adenine (A), thymine (T), cytosine (C) and guanine (G) (Alberts *et al.* 1994). The A and T bases bond together, as do the C and G. Other molecular forces will cause the strand to bend and convolute, creating a 3-dimensional double-bonded structure essentially unique to the molecule, and critical to various organic functions. In terms of sequence characterization, one of the strands of bases is adequate for identification purposes, since the other strand of bonded base pairs is complementary. A complete molecule, or a portion of it denoting a particular structure of interest, is denoted by a sequence of A, T, C and G bases. A higher level of representation is often used, in which the 20 unique amino acids created from triples of nucleic acids are represented. This results in smaller sequences using a larger alphabet.

The representation and automatic identification of subsequences in organic molecules has attracted much research effort over the years, and has resulted in a number of practical applications. New sequences can be searched for instances of known subsequences ("aligned"), which can indicate organic properties of interest, and hence identify their genetic functionality. Families of sequences can be classified by their distinguishing common sequence patterns. Sequence patterns are natural interfaces for biosequence database access. Sequences are also conducive to mathematical and computational analyses, which makes them natural candidates for automated synthesis and search algorithms.

A variety of representation languages have been used for biosequence identification, including regular languages (Arikawa *et al.* 1993, Brazma *et al.* 1998*b*), context–free and other languages (Searls 1993, Searls 1995), and probabilistic representations (Krogh *et al.* 1994, Sakakibara *et al.* 1994, Karplus *et al.* 1997). Although languages higher in the Chomsky hierarchy are more discriminating than lower-level representations, they may be less efficiently parsed or synthesized than than lower–level languages. In many cases, simple languages such as regular languages are the most practical representation for biosequence identification and database access. The PROSITE database, for example, uses a constrained regular expression language.

Much work has been done on machine learning techniques for families of biosequences using regular languages as a representation language (Brazma *et al.* 1998*a*, Baldi and Brunak 1998). GP has been used successfully to evolve regular motifs for unaligned sequences (Hu 1998, Koza *et al.* 1999).

2.2 Stochastic Regular Expressions

Stochastic Regular Expressions (SRE) is a probabilistic regular expression language (Ross 2000). It is essentially a conventional regular expression language (Hopcroft and Ullman 1979), embellished with probability fields. It is similar to a stochastic regular language proposed by (Garg *et al.* 1996), where a number of mathematical properties of the language are proven.

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:

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

The terms denote atomic actions, concatenation, iteration (Kleene closure and '+' iteration), and choice. Plus iteration, E^{+p}, is equivalent to $E : E^{*p}$. The probability fields work as follows. With choice, each term $E_i(n_i)$ is chosen with a probability equivalent to $n_i/\Sigma_j(n_j)$. With Kleene closure, each iteration of E occurs with a probability p, and the termination of E occurs with 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. With choice, the aforementioned probability of a selected term is multiplied by the probability of its chosen expression E_i. Each iteration of Kleene iteration also includes the probability of the iterated expression E. The overall effect of this probability scheme is the definition of a probability distribution of the regular language denoted by an expression. Each string $s \in L(E)$ has an associated probability, while any $s \notin L(E)$ has a probability of 0. It can be shown that SRE defines a well-formed probability function (the sum of all the probabilities for all $s \in L(E)$ is 1).

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 $Pr = \frac{3}{2+3} = 0.6$; 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. 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.

As with conventional regular expressions (Hopcroft and Ullman 1979), string recognition for SRE expressions is of polynomial time complexity. Note, however, that the interpretation of regular expressions can be exponentially complex with respect to overall expression size. For example, in $((a+b)^*)^*$, even though the expression's language is equivalent to that for $(a+b)^*$, there is a combinatorial explosion in the number of ways the nested iterations can be interpreted with respect to one other: a string of size k can be interpreted 2^k different ways.

SRE-DNA, a variant of SRE, is used in this paper. A number of embellishments and constraints are used, which are practical for biosequence identification. Details are given in Section 3.1.

2.3 DCTG-GP

```
expr ::= guardedexpr^^A, expr^^B
<:>
(construct( E:F ) ::- A^^construct(E),
                      B^^construct(F)),
(recognize(S, S2, PrSoFar, Pr) ::-
          check_prob(PrSoFar),
          A^^recognize(S, S3, PrSoFar, Pr1),
          check_prob(Pr1),
          B^^recognize(S3, S2, Pr1, Pr)).
```

Figure 1: DCTG rule for SRE-DNA concatenation

DCTG-GP is a grammatical genetic programming system (Ross 2001*a*). It is inspired by other work in grammatical GP (Whigham 1995, Geyer-Shulz 1997, Ryan *et al.* 1998), and in particular, the LOGENPRO system (Wong and Leung 1997). Like LOGENPRO, DCTG-GP uses logical grammars for defining the target language for evolved programs. The logic grammar formalism used is the definite clause translation grammar (DCTG) (Abramson and Dahl 1989). A DCTG is a logical version of a context-free attribute grammar, and it permits the complete syntax and semantics of a language to be defined in one unified framework. DCTG-GP is implemented in Sicstus Prolog 3.8.5 (SICS 1995).

In a DCTG-GP application, the syntax and semantics of a target language are defined together. Each DCTG rule contains a syntax field and one or more semantic fields. The syntax field is the grammatical definition of a language component, while the semantic fields encode interpretation code, tests, and other language and problem specific constraints. The general form of a rule is:

$$\begin{array}{l} H \ ::= \ B_1,\ B_2,\ ...,\ B_j \\ <:> \\ S \ ::-\ G_1,\ G_2,\ ...,\ G_k. \end{array}$$

The rule labeled with nonterminal H is a grammar rule. Each term B_i is a reference to a terminal or nonterminal of the grammar. Embedded Prolog goals may also be listed among the B_i's. These grammar rules are used to denote programs in the population, which are in turn implemented as derivation trees. Hence DCTG-GP is a tree-based GP system. The rule labeled S is a semantic rule associated with nonterminal H. Its goals G_i may refer to semantic rules associated with the nonterminal references B_i, or calls to Prolog predicates.

Figure 1 shows the DCTG-GP rule for SRE-DNA's concatenation operator. The grammatical rule states that concatenation consists of a guarded expression followed by an expression. The `A` and `B` variables are used for referencing parts of the grammar tree for these nonterminals within the semantic rules. The first semantic rule `construct` builds a text form for the rule, for printing purposes. The ":" operator denotes concatenation. The second semantic rule `recognize` is used during SRE-DNA expression interpretation. The argument `S` is a string to be consumed, and `S2` is the remainder of the string after consumption. The value `PrSoFar` is the overall probability thus far in the interpretation, and `Pr` is the probability after this expression's interpretation is completed. The references to `recognize` in the semantic rule are recursive calls which permit the two terms in the concatenation to recognize portions of the string. Finally, `check_prob` determines if the current running probability is larger than the minimal required for interpretation to continue.

3 EXPERIMENT DETAILS

3.1 SRE-DNA Variations

1. $$\begin{aligned} \mathit{expr} &::= \mathit{guard} \mid \mathit{choice} \mid \mathit{guard}\!:\!\mathit{expr} \\ &\quad \mid \mathit{expr}^{*p} \mid \mathit{expr}^{+p} \\ \mathit{choice} &::= \mathit{guard}(n) + \mathit{guard}(n) \\ &\quad \mid \mathit{guard}(n) + \mathit{choice} \\ \mathit{guard} &::= \mathit{mask} \mid \mathit{mask}\!:\!\mathit{skip} \\ \mathit{skip} &::= x^{*p} \mid x^{+p} \end{aligned}$$

2. $$\begin{aligned} \mathit{expr} &::= \mathit{guard} \mid \mathit{guard}\!:\!\mathit{expr} \mid \mathit{expr}^{+p} \\ \mathit{guard} &::= \mathit{mask} \mid \mathit{mask}\!:\!\mathit{skip} \\ \mathit{skip} &::= x^{+p} \end{aligned}$$

3. $$\begin{aligned} \mathit{expr} &::= \mathit{guard} \mid \mathit{guard}\!:\!\mathit{expr} \mid \mathit{expr}\!:\!\mathit{guard} \\ &\quad \mid \mathit{expr}^{*p} \mid \mathit{expr}^{+p} \\ \mathit{guard} &::= \mathit{mask} \mid \mathit{mask}\!:\!\mathit{skip} \\ \mathit{skip} &::= x^{*p} \mid x^{+p} \end{aligned}$$

4. $$\begin{aligned} \mathit{expr} &::= \mathit{guard} \mid \mathit{choice} \mid \mathit{expr}\!:\!\mathit{expr} \\ &\quad \mid \mathit{expr}^{*p} \mid \mathit{expr}^{+p} \\ \mathit{choice} &::= \mathit{guard}(n) + \mathit{guard}(n) \\ &\quad \mid \mathit{guard}(n) + \mathit{choice} \\ \mathit{guard} &::= \mathit{mask} \mid \mathit{mask}\!:\!\mathit{skip} \\ \mathit{skip} &::= x^{*p} \mid x^{+p} \end{aligned}$$

Figure 2: SRE-DNA Variations

A goal of this research is to explore how language constraints affect the quality of motif solutions. To this end, four different grammatical variations of SRE-DNA are defined in Figure 2. SRE-DNA embellishes SRE as follows. Firstly, masks are introduced. The mask $[\alpha_1 ... \alpha_k]$ denotes a choice of atoms α_i each with a probability $1/k$. This is equivalent to $\alpha_1(1)+...+\alpha_k(1)$ in SRE. Secondly, skip terms are defined. A skip term x^{*p} is a Kleene closure over the wild-card element x, which substitutes for any atom. The skip expression x^{+p} is equivalent to $x\!:\!x^{*p}$.

A summary of the SRE-DNA variants in Figure 2 is as follows. Grammar 1 uses constrained concatenation and choice expressions, in which *guards* are used. A guard is a term borrowed from concurrent programming, and specifies a constrained action. Guards promote efficient interpretation, because expressions are forced to consume string elements whenever a guard is encountered. It also reduces the appearance of iteration and choice in concatenation expressions, which helps reduce the scope of the target expressions. An intention for doing this is to try to make SRE-DNA have similar characteristics to conventional motif languages such as PROSITE's. In addition, the grammar prohibits nested iteration. This prevents some of the efficiency problems discussed in Section 2.2. Three minor variations of grammar 1 are used, each having different maximum iteration ranges ("i"): 1a (i=0.5); 1b (i=0.1); and 1c (i=0.2).

Grammar 2 is the closest to the PROSITE language. Choice is not used, and all skip and iterations use "+" iteration. It is also the only grammar that permits nested iteration. Grammar 3 is a minor relaxation of grammar 1, in which guards can be the first or second term in a concatenation. Nested iteration is prohibited. Finally, Grammar 4 is the least restrictive grammar, where concatenation uses general SRE-DNA expressions in both terms. Choice expressions still use guards, however, and nested iteration is prohibited.

It should be mentioned that a full version of SRE-DNA without guards or nested iteration constraints was initially attempted. Expression interpretation was very inefficient in that language, due to the preponderance of nested "*"–iterations, as well as iterations within choice and concatenation terms. The above constrained grammars are more efficient to interpret, and do not suffer any practical loss of expressiveness, at least with respect to the problem of motif recognition tackled here.

3.2 Fitness Evaluation

Fitness evaluation tests an expression's ability to recognize positive training examples, and to reject negative examples. Positive examples comprise a set of N aligned protein sequences. Negative examples are N randomly generated sequences, each having approximately the same length as the positive sequences.

Consider the formula:

$$Fitness = N + NegFit - PosFit$$

where *NegFit* and *PosFit* are the negative and positive training scores respectively. A fitness of 0 is the ideal "perfect" score. It is not attainable in practice, because the probabilities incorporated into *PosFit* are typically small.

Positive example scoring is calculated as:

$$PosFit = \sum_{e_i \in Pos} maximum(Fit(e'_i))$$

where Pos is the set of positive training examples, and e'_i is a suffix of example e_i (ie. $e_i = se'_i, |s| \geq 0$). For each example in Pos, a positive test fitness Fit is found for all its suffixes, and the maximum of these values is used for the entire example. Fitness evaluation incorporates two distinct measurements: the probability of recognizing an example, and the amount of the example recognized in terms of its length:

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

Here, s_{max} is the longest recognized prefix of e, $|s_{max}|$ is its length, and $Pr(s_{max})$ is its probability of recognition. The first term accounts for the probability obtained when recognizing substring s_{max}, and the second term scores the size of the covered substring relative to the entire example. The fitness pressure obtained with *Fit* is to recognize an entire example string with a high probability. In early generations, the sequence cover term dominates the score, which forces fitness to favour expressions that recognize large portions of examples. The probability field comes into consideration as well, however, and is especially pertinent in later generations when expressions recognize a large proportion of the example set. At that time, the probability fitness measure favours expressions that yield high probabilities.

Negative fitness scoring is calculated as:

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

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.

3.3 GP Parameters

Table 1 lists parameters used for GP runs. Although most parameters are self–explanatory, some require explanation. The initial population is oversampled, and culled at the beginning of a run. Reproduction may fail, for example, due to tree size limitations, and so a maximum of 3 reproduction attempts are undertaken before the reproduction is discarded. The terminals are a subset of amino acid codons, determined by the alphabet used in the positive training examples.

Crossover and mutation use the methods commonly applied by grammatical GP systems that denote programs with derivation trees. For example, when a subtree node of nonterminal type t is selected in one parent, then a similar node of type t will be selected in the other parent, and the two selected subtrees are swapped. Some SRE specific crossover and mutation operators are used. SRE crossover permits mask elements in two parents to be merged together. SRE mutation implements a number of numeric and mask mutations. The SRE mutation range parameter specifies that a numeric field is perturbed ±10% of its original value. Mask mutations include adding, removing, or changing a single item from a mask.

Table 1: GP Parameters

Parameter	Value
GA type	generational
Functions	SRE-DNA variants
Terminals	amino acid codons, integers, probabilities
Population size (initial)	2000
Population size (culled)	1000
Unique population	yes
Maximum generations	150
Maximum runs	10
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
Max. depth initial popn.	12
Max. depth offspring	24
Min. grammar prob.	10^{-12}
Max. mask size	5

The minimum grammar probability value specifies the minimal probability used by the SRE evaluator before an expression interpretation is preempted. This improves the efficiency of expression evaluation by pruning interpretation paths with negligibly small probabilities.

4 RESULTS

The initial test case is the amino acid oxidase family of sequences. It is completely defined by a relatively small example set (8 unique sequences in the PROSITE database as of November, 2000). Table 2 shows the training results for the SRE-DNA grammars in Figure 2. (Having only 8 examples precluded the ability to perform testing on the results). ΣPr is the sum of recognized probabilities for all the positive examples. The best fitness and ΣPr fields are given for the top solution in the 10 runs for each case, while the average ΣPr is an average of all the solutions from the 10 runs. In the 60 solutions obtained in all these runs, only one expression was unable to recognize the entire

PROSITE ⇒	$[ilmv](2) : h : [ahn] : y : g : x : [ags](2) : x : g : x(5) : g : x : a$
Grammar	
1a	$[iglv] : x^{+.12} : h : x^{+.12} : y : (g : x^{+.45} : g : x^{+.47} : [ghqs] : x^{+.47} :$ $(g : x^{+.47}(947) + [fghqs] : x^{+.14}(101) + [chmvy] : x^{+.14}(842)))^{+.12}$
1b	$[ilv] : x^{+.1} : h : x^{+.1} : y : g : x^{+.1} : g : x^{+.1} : [gq] : x^{+.1} : [ghis] : x^{+.1}$ $: g : x^{+.1} : ([aqst](325) + [afsw] : x^{*.1}(210) + [fhnqs](223))^{*.1}$
1c	$[ilv] : x^{+.1} : h : x^{+.1} : y : x^{*.19} : g : x^{+.19}$ $: (g : x^{+.19} : [gtq] : x^{+.19} : [ghs] : x^{+.1} : g : x+^{.1} : a)^{+.1}$
2	$[ilv] : x^{+.1} : h : x^{+.1} : y : x^{+.1} : [afh] : x^{+.1} : [gs] : x^{+.1} : g : x^{+.19}$ $: [smqt] : x^{+.19} : [wy] : g : x^{+.1} : (a^{+.11})^{+.1}$
3	$[ilv] : x^{+.11} : h : x^{+.1} : y : g : x^{+.19} : [sg] : x^{+.19} : g : x^{+.1} : [aqst]$ $: x^{+.1} : [ghs] : x^{+.13} : g : x^{+.1} : a$
4	$([ligv] : x^{+.14} : h : x^{+.11} : y : g : x^{+.17} : [gs] : x^{+.18} : g : x^{+.17})^{+.11}$ $: [astqi] : x^{+.15} : ([hgs] : x^{+.11} : g : x^{+.19}(567) + ([ihswl] : x^{+.15}$ $: ([hi] : x^{+.11})^{+.15} : ([ligv] : x^{+.11} : h : x^{+.11} : (y : g : x^{+.19})^{+.12}$ $: [gs] : x^{+.17}(567) + (h : x^{+.14})^{+.15}(4)) : g : x^{+.17})^{+.11}$ $: (((y : g : x^{+.19})^{+.12} : [gs] : x^{+.18} : g : x^{+.19})^{+.12} : [gs] : x^{+.17}(567)$ $+g : x^{+.1})^{+.15}(4)) : g : x^{+.17} : g : x^{+.17}(4))$

Figure 3: Best solutions for various grammars: amino acid oxidase

Table 3: Solution statistics for other families (grammar 2)

	Training			Testing (best soln)		
Family	Set size	Seq size	100% solns	Set size	True pos (%)	False neg (%)
a) Aspartic acid	44	12	10	452	100	0.2
b) Zinc finger, C2H2 type	29	23	9	678	93	1
c) Zinc finger, C3HC4 type	21	10	10	168	100	0
d) Sugar transport 1	18	18	0	190	88	1
e) Sugar transport 2	18	26	2	178	100	12
f) Snake toxin	18	21	10	127	51	0
g) Kazal inhibitor	20	23	10	125	93	0

Table 2: Solution statistics (training) for SRE-DNA variations: amino acid oxidase. Grammars 1a, 1b, and 1c use maximum iteration limits of 0.5, 0.1, and 0.2 respectively.

Grammar	Best Fitness	Best Σ Pr	Avg Σ Pr
1a	3.999611	0.00078	0.000140
1b	3.999977	0.00005	0.000009
1c	3.999044	0.00191	0.000356
2	3.998157	0.00369	0.000588
3	3.992940	0.01412	0.002502
4	3.999396	0.00121	0.000272

training set. Clearly, version 3 of SRE-DNA (unrestricted, but no choice operator) yielded the strongest solutions.

Figure 3 shows the best solutions obtained for the runs in Table 2, along with the PROSITE expression used to obtain the training set. Note that PROSITE motifs are typically made manually by scientists, and are error-prone. While similarities are often seen between the GP solutions and PROSITE expression, there are also differences in the way consensus patterns are handled between them. Note how E^{+p}, S^{*p}, and x^{*p} are nonexistent in the best overall solution (grammar 3). It seems to contradict conventional GP wisdom that this richer grammar containing these superfluous operators performs better than grammar 2, which omits these operators in the first place. One hypothesis for this is that the iterative terms in grammar 3 help conserve and transport useful genetic material from early generations, but disappears later.

The solution motif that least matches the others is the one from grammar 4 (unrestricted with choice). This expression suffers from bloat, in which intron material is attached to low-probability choice terms. Even though such intron material may not contribute to language membership, it definitely has a negative impact

a) Aspartic	$P : c : x : [dn] : x(4) : [fy] : x : c : x : c$
	$S : c : x^{+.19} : [dn] : x^{+.19} : [fy] : x^{+.1} : c : x^{+.1} : c$
b) Zinc C2H2	$P : c : x(2,4) : c : x(3) : [cfilmvwy] : x(8) : h : x(3,5) : h$
	$S : c : x^{+.19} : c : x^{+.19} : [afkr] : x^{+.19} : [fhqrs] : x^{+.19} : [ahlrs] : x^{+.19} : [hlnt] : x^{+.19} : [hikrv] : x^{+.19}$
c) Zinc C3HC4	$P : c : x : h : x : [filmvy] : c : x(2) : c : [ailmvy]$
	$S : c : x^{+.1} : h : x^{+.19} : c : x^{+.19} : c : x^{+.1}$
d) Sugar 1	$P : [agilmstv] : [afgilmsv] : x(2) : [ailmsv] : [de] : x : [afilmvwy] : g : r : [kr] : x(4,6) : [agst]$
	$S : [agilm] : x^{+.32} : [dilr] : x^{+.32} : g : r : x^{+.32} : [gilmv] : x^{+.32}$
e) Sugar 2	$P : [filmv] : x : g : [afilmv] : x(2) : g : x(8) : [fily] : x(2) : [eq] : x(6) : [kr]$
	$S : [filmv] : x^{+.19} : (g : x^{+.48} : g : x^{+.48} : [fgily] : x^{+.48} : [ailtv] : (x)^{+.48})^{+.21}$
f) Snake	$P : g : c : x(1,3) : c : p : x(8,10) : c : c : x(2) : [denp]$
	$S : g : c : x^{+.12} : c : x^{+.49} : [gkrv] : x^{+.48} : [gl] : x^{+.48} : c : c : x^{+.12} : [kt] : x^{+.1}$
g) Kazal	$P : c : x(7) : c : x(6) : y : x(3) : c : x(2,3) : c$
	$S : c : x^{+.39} : [cp] : x^{+.39} : [acdgs] : x^{+.39} : y : x^{+.11} : [nsy] : x^{+.1} : c : x^{+.38} : c^{+.11}$

Figure 4: PROSITE (P) and best solutions (S) for other families (grammar 2)

on the overall probability distribution of a motif.

The solutions generated from a single experiment can often vary considerably. Consider this alternate solution from the grammar 1c runs ($\Sigma Pr = 0.00007$):

$$[ilv] : [iv] : h : x^{+.1} : y : x^{+.19} : [ghs] : x^{+.19} : g \\ : x^{+.19} : [ghst] : x^{+.19} : g : x^{+.19}$$

Comparing it with the solution for 1c in Figure 3, it more precisely discriminates the beginning of the sequence.

Experiments using other families of sequences were undertaken using grammar 2. Training and testing results are shown in Table 3. The maximum iteration limit was changed for different families, in an attempt to address the relative range of skipping allowed in the corresponding PROSITE expressions. "100% solns" indicate the number of solutions from the 10 runs that recognize the entire set of training examples, "True pos" is the proportion of true positives (positive examples correctly identified from the testing set), and "False neg" is the proportion of the false negatives (negative examples falsely identified as being member sequences). The positive and negative testing sets are the same size.

The testing results suggest that nearly all of the experiments found acceptable solutions. One exception is the snake toxin case, whose positive testing results are poor. This is probably due to over-training on an inadequately small training set. The sugar transport examples (d and e) were also challenging. Experiment (d) yielded no expressions which completely recognized the entire training set. Considering the results of Table 2, better results might have arisen if grammar 3 had been used instead of grammar 2. Also note that a strong overall probability score does not necessarily directly correlate with a high testing score. This is because a motif might recognize a lower-proportion of true positives, but with high probabilities. A good solution will balance the probability distribution and positive example recognition.

The motif expressions for the best solutions in Table 3 are given in Figure 4. In the aspartic and zinc C3HC4 experiments (a, c), all the runs generated the identical expression. In the aspartic case, the solution is nearly a direct match to the PROSITE expression, except that SRE-DNA's probabilistic skipping is used. In the solution for experiment (c), evolution chose skip expressions instead of the PROSITE $[filmvy]$ and $[ailmvy]$ terms. The preference of skip terms instead of masks was not always the case, as is seen in other solutions in Figure 4.

An interesting characteristic of many of the evolved motifs using grammar 2 is that the + iteration operator usually evolved out of final expressions. In the 80 grammar 2 motifs evolved for all the protein families studied, only 28 motifs used the iteration operator. In three families (aspartic acid, zinc finger 2, and snake toxin), none of the solution motifs used iteration. When iteration arose, it was often highly nested, indicating that it was being used as intron code. Even though iteration is not an important operator for expressing these motifs, it does seem to be beneficial for evolution performance, as was seen earlier in Table 2.

Regular expressions are coarse representations of the 3D structure relevant to a protein's organic functionality. Nevertheless, it is interesting to consider whether any of the evolved motifs have captured the essential biological feature of the given protein. In some cases, the important features were indeed found. For example, in the snake toxin example, the four *c*'s evident in both the PROSITE and SRE-DNA motifs are involved in disulfide bonds. In the aspartic acid motif, the hydroxylation site at the *d* or *n* codon is correctly identified. In the sugar 1 example, part of a strong sub-motif "$g : r : [kr]$" in the PROSITE source is seen in the SRE-DNA motif (the "$g : r$" term was found).

5 CONCLUSION

This research establishes that SRE-DNA is a viable motif language for protein sequences. SRE-DNA expressions were successfully evolved using grammatical GP, as implemented with the DCTG-GP system. A number of families were tested, and acceptable results were usually obtained. Like other regular motif languages, SRE-DNA is most practical for small- to medium-sized sequences, since larger sequences require correspondingly large expressions that generate relatively miniscule probabilities. Variations of SRE-DNA were tried, and preliminary results show that the most successful variation is one with unrestricted non-nested iteration, guards, and no choice operator. The choice operator is definitely detrimental, as it increases the frequency of intron material. Although the iteration operator was not important in final solutions, using it enhances evolution performance. One hypothesis for this is that iteration acts as a transporter of genetic material in early generations. Further testing on more families of sequences should confirm these results.

The style of motifs obtained is highly dependent upon grammatical constraints. Besides the kinds of grammar restrictions tested in the experiments, such factors as minimum and maximum iteration limits and maximum mask sizes are also critical factors in the character of realized motifs. Mask usage can be increased by reducing the maximum skip iteration limit, thereby increasing the likelihood of more guarded terms, and hence masks. Increasing the maximum mask size, however, does not result in better solutions. Larger masks tend to generate less discriminating motifs (higher false negative rates), and also are less efficiently interpreted. If the maximum iteration limit is set too large, evolved expressions tend to take the form:

$$(unique\ prefix) : (x)^{+.9} : (unique\ suffix)$$

In other words, evolution tends to find an expression that has two discriminating components for the beginnings and ends of sequences, while it skips the majority of the sequence in between. By reducing the iteration limit, more interesting motifs are obtained.

Multiple runs often find varying solutions that identify different consensus patterns within sequences. It is worth considering whether there is some means by which different solutions might be reconciled or "merged" together. Of course, the best way to judge a consensus pattern is to allow a biologist to examine it, in order to determine whether the identified patterns are biologically meaningful. It is worth remembering that grammatical motifs are crude approximations of the *real* relevant biological factor - the 3D shape of the protein molecule.

One automatic optimization that is easily applied to evolved motifs is to simplify mask terms by removing extraneous elements. This has two effects. First, it increases the probability performance of expressions, because smaller masks have proportionally larger probabilities for selected elements. Secondly, smaller masks make expressions more discriminating. This is easy to see, since a mask of one element is the most discriminating, while a skip term is the least (it is akin to a mask of all elements).

Recently, SRE-DNA has been applied successfully in synthesizing motifs for unaligned sequences (Ross 2001*b*). The results in this paper have been indispensable for this new work, since it is now known which versions of SRE-DNA are apt to be most successful. The knowledge that the choice operator is impractical and should be ignored is very helpful.

This research is similar in spirit to that by Hu, in which PROSITE-style motifs were for unaligned protein sequences (Hu 1998). Hu used demes and local optimization during evolution, unlike this work, which used a single population and no local optimization. Hu also seeds the initial population with terms generated from the example proteins. (Koza *et al.* 1999) have used GP to evolve regular motifs for proteins. One solution performed better than the established motif created by experts. Their use of ADF's was advantageous for the proteins analyzed, given the many instances of repeated patterns.

Acknowledgments

Thanks to Bob McKay for suggesting a means for improving the performance of DCTG-GP, and to anonymous referees for their constructive advice. This work is supported though NSERC Operating Grant 138467-1998.

References

Abramson, H. and V. Dahl (1989). *Logic grammars.* Springer-Verlag.

Alberts, B., D. Bray, J. Lewis, M. Raff, K. Roberts and J.D. Watson (1994). *Molecular Biology of the Cell.* 3 ed.. Garland Publishing.

Arikawa, S., S. Miyano, A. Shinohara, S. Kuhara, Y. Mukouchi and T. Shinohara (1993). A Machine Discovery from Amino Acid Sequences by Decision Trees over Regular Patterns. *New Generation Computing* **11**, 361–375.

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

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

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

Garg, V.K., R. Kumar and S.I Marcus (1996). Probabilistic Language Framework for Stochastic Discrete Event Systems. Technical Report 96-18. Institute for Systems Research, University of Maryland. http://www.isr.umc.edu/.

Geyer-Shulz, A. (1997). The Next 700 Programming Languages for Genetic Programming. In: *Proc. Genetic Programming 1997* (John R. Koza *et al*, Ed.). Morgan Kaufmann. Stanford University, CA, USA. pp. 128–136.

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

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

Hu, Y.-J. (1998). Biopattern Discovery by Genetic Programming. In: *Proceedings Genetic Programming 1998* (J.R. Koza *et al*, Ed.). Morgan Kaufmann. pp. 152–157.

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

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

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

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

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

Ross, B.J. (2001*b*). The Evolution of Stochastic Regular Motifs for Protein Sequences. Submitted for publication.

Ryan, C., J.J. Collins and M. O'Neill (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. In: *Proc. First European Workshop in Genetic Programming (EuroGP-98)* (W. Banzhaf *et al.*, Ed.). Springer-Verlag. pp. 83–96.

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

Searls, D.B. (1993). The Computational Linguistics of Biological Sequences. In: *Artificial Intelligence and Molecular Biology* (L. Hunter, Ed.). pp. 47–120. AAAI Press.

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

SICS (1995). *SICStus Prolog V.3 User's Manual.* http://www.sics.se/isl/sicstus.html.

Whigham, P.A. (1995). Grammatically-based Genetic Programming. In: *Proceedings Workshop on Genetic Programming: From Theory to Real-World Applications* (J.P. Rosca, Ed.). pp. 31–41.

Wong, M.L. and K.S. Leung (1997). Evolutionary Program Induction Directed by Logic Grammars. *Evolutionary Computation* **5**(2), 143–180.