Discovering Biological Motifs With Genetic Programming

Comparing linear and tree-based representations for unaligned protein sequences

Rolv Seehuus*
Dept. of Computer Science
Norwegian University of
Science and Technology
Sem selands v 7-9
7000 Trondheim, NORWAY
rolv.seehuus@idi.ntnu.no

Amund Tveit
Dept. of Computer Science
Norwegian University of
Science and Technology
Sem selands v 7-9
7000 Trondheim, NORWAY
amund.tveit@idi.ntnu.no

Ole Edsberg
Dept. of Computer Science
Norwegian University of
Science and Technology
Sem selands v 7-9
7000 Trondheim, NORWAY
ole.edsberg@idi.ntnu.no

ABSTRACT

Choosing the right representation for a problem is important. In this article we introduce a linear genetic programming approach for motif discovery in protein families, and we also present a thorough comparison between our approach and Koza-style genetic programming using ADFs.

In a study of 45 protein families, we demonstrate that our algorithm, given equal processing resources and no prior knowledge in shaping of datasets, consistently generates motifs that are of significantly better quality than those we found by using trees as representation. For several of the studied protein families we evolve motifs comparable to those found in Prosite, a manually curated database of protein motifs.

Our linear genome gave better results than Koza-style genetic programming for 37 of 45 families. The difference is statistically significant for 24 of the families at the 99% confidence level.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Heuristic methods

General Terms

Algorithms, Performance

Keywords

Genetic Programming, protein motifs, representations

1. INTRODUCTION

Understanding protein function is one of the keys to understanding life. Knowledge discovery about proteins to predict function from protein sequences is one of the grand challenges in biology today — as it has been for the last two decades.

Swiss-Prot is an annotated database of protein sequences from many different organisms, and includes descriptions of known protein function and domain structure as well as publication lists on each protein [2]. As of release 45.5 (released January 4th, 2005), the Swiss-Prot database contains 167089 distinct protein sequence entries, consisting of approximately 60 million amino acids. In contrast TrEMBL, the computer annotated cousin of Swiss-Prot, contains 1.5 million protein sequences in release 28.5 (also released January 4th 2005). From release 45 to 45.5 Swiss-Prot grew with 2%. TrEMBL grew with 6%. In other words, there is an information gap between the number of known protein sequences and the number of studied protein sequences. This gap is growing faster with every release of the SwissProt and TrEMBL databases.[1] Computer methods to aid understanding and to help pose meaningful hypothesises about protein function to help close this gap is thus one of several important research problems in bioinformatics.

Motifs, short regular expression-like structures, are a way to describe homologous relationships between proteins. Often such a motif can highlight important characteristic regions of proteins with similar function and common ancestral background. Also, motifs are easy to interpret — a biologist may immediately see the important motif or conserved region of a protein family.

Prosite is a manually created and curated database that describes homologous relationships with local motifs and patterns [9]. A drawback of Prosite is of course that the patterns are manually created, requiring a significant amount of work by skilled biologists — both creating the motif and at keeping the database synchronized with new data. Therefore, the automatic discovery of conserved motifs have received quite a bit of attention during the last two decades. Brazma and Jonassen give a good survey on methods based on enumeration and other search methods, and also give a nice view on the different problems in the domain [4]. Rigoutsos et al. also give a survey of motif discovery algorithms in relation with their own works [14]. Although new motifs in Prosite are now often presented as stochastic profiles with much resemblance to Hidden Markov Models (HMMs), the automatic generation of biological motifs re-

*Author to whom communications should be addressed

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

[1]Release data acquired from the Swiss-Prot web-site at http://us.expasy.org/sprot/

mains an active research area. A new motif that can replace the need of the statistical profile is always welcome.

The patterns in Prosite are given as simplified regular expressions as in the following example:

`A-N-[EXAMPL]-E-X(1,3)-P-A-T-{TERN}`

Each letter (residue) is picked from a twenty-letter alphabet representing different amino acids. Residues in brackets represent "one of" a group and the `X` represents a wildcard region (in this case it represents a region with at least one and at most three residues). The final residues enclosed in the curly braces represents "not one of" a group. It is also possible to constrain the motif to be at one of the ends of the protein string, using `<` and `>` (meaning beginning and end, respectively.)

As will become clear during the discussion of previous and related work, this is the first work we are aware of that uses GP and the entire Swiss-Prot database as examples to evolve protein motifs.

1.1 Our experiments

In this study we investigated and compared the relative capabilities of tree-based Genetic Programming (TreeGP) with that of linear-genome Genetic Programming (ListGP) as tools to discover interesting motifs in unaligned proteomic data. We demonstrate that ListGP gives good classifiers for several protein families without the use of domain specific knowledge. We also show that our linear genome algorithm consistently performs better than TreeGP, and that it tends to have more stable performance over a wide range of protein families — an important property of algorithms for knowledge acquisition.

Recent approaches have applied very general constrained genetic programming methods, where the search has been done in valid parse-trees for a given grammar describing the target language (e.g., the Prosite language). Such an approach is very general in terms of what kind of problems one can formulate and solve, but it might not be tailored to the problem of motif discovery.

Due to the sequential nature of protein sequences, a natural representation for conserved motifs without gaps or wildcards would be sequences. Even though the Prosite language has support for flexible gaps, wildcards and groups, the emphasis is on conserved positions in a set of strings. Therefore, even if we expanded the language to include all the features found in the Prosite language, the focus should be on the conserved parts.

Authors reporting similar research have concentrated on a very small number of protein families. In this report we give results for 45 different protein families. Also, researchers tend to use few positive examples for training - possibly due to runtime considerations. To our knowledge, no researchers in the GP-community have reported experiments on this many protein families before. Our experiments support our main claim that the linear genome genetic programming for motif discovery is more efficient than tree-based methods.

We also employed a specialized hardware called the "Pattern Matching Chip" (PMC) to evaluate the individuals [1]. This yields nearly a *fifty-fold speedup* in evaluation speed in our experiments compared with a standard "off the shelf" computer using regexp matching software like "grep."

We would like to use boosting la AdaBoost in future work, and therefore it is of importance to us that our base learner is as efficient as possible — for boosting to work, we must be sure that the algorithm performs better than a random walk in the solution space [13].

In this work we have implemented a very simple linear genetic programming algorithm evolving motifs in a small subset of the Prosite language including only residues and groups, and compare these results with a tree-based method previously described for motif discovery for the same language [10]. Extending these studies to approaches containing larger portions of the Prosite language is planned in future work.

In section 2 we discuss related works and contrast it to our approach. The algorithms we use are described in section 3, whereas our experiments and methodology is described in section 4. We give results from our experiments in section 5, and finally a short discussion and some concluding remarks in section 6.

2. RELATED WORK

Evolutionary Algorithms (EA) have been applied to the problem of motif discovery at a few occasions [10, 8, 17, 12]. The experiments have usually been small, presenting results for only a limited number of protein families (usually around two to six.) In addition, researchers have usually performed some kind of pre-shaping of training sets. Some such strategies are pre calculated alignments of positive examples [17, 12], particularly difficult sequences have been used as negative examples [10] and relatively small numbers of randomly generated sequences have been used as negative samples [18]. Also, most of the studies have used the default "Koza-style" tree-structure to evolve candidates, sometimes in conjunction with a grammar.

2.1 Careful selection of negative examples

Koza and Andre have evolved motifs for the DEAD-box family and Super-oxide Manganese families [10, 11]. They used a population size of 256000 individuals, running for a maximum of 201 generations. They elaborated on two motifs found by their algorithm after 42 and 64 generations. Assuming that these were "early sightings" — motifs with good qualities found early in runs — this means that they performed somewhere above ten million fitness evaluations before finding a quality motif.

They evolved motifs with the same subset of the Prosite pattern format as we use in this work, allowing only an exact match with one of the twenty amino acid residues and character sets matching one or more residues. This limits patterns to the following format:

`[IVPTQ]-[IVPTQ]-I-E-M-[NR]-[WP]-[FCNR]`

In their work these patterns are represented with the standard GP-tree structure, using automatically defined functions (ADFs) for the residues enclosed in the optional brackets.

Koza and Andre did some preprocessing of their training data to lessen the computational burden of the search. They preprocessed the negative data and constructed a dataset with 210 examples with 30 residues, which all contained a partial match with the dead-box and superdioxide manganese motifs without being in the family. One might say that they performed a kind of manual boosting of the training data (AdaBoost works in this manner, by allowing the algorithm to gradually place increased emphasis on difficult data-points [6]). This might cause the algorithm to falsely find motifs that seem to be of significance (having high fit-

ness due to no negative hits), but exist in other parts of the Swiss-Prot data — a kind of over-fitting. Some of the results reported in their articles actually show some behavior of this kind.

Their approach is, in other words, not able to pose a completely new hypothesis about a set of related protein strings with certainty. To create motifs of any quality, they require that the motifs found "hit" the string in the same locations as the target motifs. Still their experiments are an early demonstration that genetic programming might be amenable to such studies.

2.2 Pre-aligning the positive data

Björn Olsson did some work on evolution of motifs using a genetic algorithm to evolve patterns not unlike the Prosite motifs [12, 3]. In these works, a genetic algorithm (GA) is used to discover conserved regions in multiple alignments of the proteins in the family.

They restrict themselves to using alignments that are precalculated and "trusted" in the biological community (e.g., hand tailored). In this respect they rely on only a small fraction of the positive samples, and the results of their experiments depend on the skills of the biologists who picked the samples — thus still requiring a considerable amount of manual labor.

The key differences between their patterns and the Prosite motifs, is that they search for global motifs over the entire alignment — as opposed to the short conserved motifs used in the PROSITE databases.

Ross has published several papers on the application of GP to evolving what he call Stochastic Regular Motifs (SRMs) [15, 16, 17]. He has reported some experiments on results using prealigned positive examples.

2.3 Local search optimization

Hu did some work where he first evolved motif candidates, followed by a local search refinement [8]. He also used an expanded terminal set including frequently occurring amino acid groups provided by a domain expert, instead of having the algorithm evolve the ambiguities. He reported good results, with motifs very similar to the target Prosite motifs for a few families.

2.4 Unbiased selection of training points

Ross has also used GP and stochastic regular expressions for training on unaligned protein sequences [18]. In that study he reports some results for six different protein families. Notably he has, like Olsson et al., a small number of positive training samples and he uses the majority of his data points for testing. As negative training data he uses randomly generated sequences instead of sequences or sequence segments from the Swiss-Prot database.

He also employs a fitness measure that differs from what is used by others. A comparison of his results with our work and the work of others might have been easier if he had calculated and presented the correlation coefficients as well. Nonetheless, he argues that GP is able to learn SREs and express meaningful motifs for proteins.

3. ALGORITHMS

Both the linear GP runs (ListGP) and the tree-based GP runs (TreeGP) were done with populations of 1500 induhviduals for 100 generations. This yields a total of 150000 fitness evaluations in every run - a fraction of the number of evaluations reportedly used by Koza et al in their study [10]. Every time a new best individual was found, it was evaluated on the test-data and results recorded. The selection mechanism is standard tournament selection for both algorithms.

3.1 Tree-based GP with ADFs

The experiments on tree-based genetic programming were done with the LilGP genetic programming system [20]. The architecture was constructed in the same way as done by Koza et al. [10]. We used two ADFs for the groups, and one for the main motif. The evolutionary operators used were subtree-swapping crossover and standard mutation.

Because of the dynamics of tree-based GP we expected that most of the evolved trees would have a "dendritic" look [5]. This made it difficult to ensure that the "expected length" of the motifs evolved by the two different algorithms would be equal, as well as assuring that the search-space of the different algorithms were comparative. Therefore we chose settings such that our TreeGP runs would be able to express motifs at least at similar lengths as those expressed with our ListGP algorithm. We allowed the maximum height to be 15 for the result-producing branch. As we did not expect the groups to include more than five residues, we constrained the ADFs to be no taller than five.

Before evaluation, an individual is transformed to a query string by concatenating the residues in prefix order traversal. The ADFs were transformed to groups in brackets, and any duplicate residues in an ADF were removed, as groups are sets of optional objects.

3.2 Linear GP for protein motifs

In ListGP the genome is represented by a list of nodes. Each node represents either an amino acid residue, or a group of residues allowing matches of two or more different amino acids.

The crossover operator in our experiments is a two-point crossover operator, where the crossover points are selected uniformly over the list in both genomes. This allows for the genomes to be of variable length, and to grow as new interesting positions are recognized. This form of crossover does also seem to be meaningful for the problem of motif discovery, placing an emphasis on the conserved residues - either in the form of single residues, or as groups.

Mutations reflects standard string-edit operations. When an individual is selected for mutation and the position for the mutation is selected uniformly over the genome, one of insertion, deletion and replacement operations were chosen with equal probability.

When it is necessary to create a new node (both under initialization of the genome and when an insertion or edit mutation occurred) a choice is made to either create a group or a single residue — 60% being residues, 40% groups. When creating a group, the length is first chosen from a normal distribution with expected length four and standard deviation one, and thereafter it is filled with different random amino acid residues selected uniformly.

The initial population was created by, for each individual, first picking a length, where the length was sampled from a normal distribution with expected length 8 and standard deviation 4. Afterward, each list-element was filled. The length of the motifs found with ListGP was limited to maximum 64 residues (also counting the residues in groups).

3.3 Evaluation of motifs

To evaluate each population, we used a PCI-card with 16 PMC from Interagon [1]. This hardware implements a subset of regular expressions, special tailored to meet typical search requirements and patterns often used by the standard UNIX tool "grep".

The evaluation of one population of 1500 individuals parallelizing the search on 10 PMCs took on the average 80 seconds, and one run took approximately about three hours to complete. To compare the speed with respect to a typical search, we timed some queries with the same size and complexity as the ones found in Table 6 using the UNIX program "grep" on a computer equipped with an AMD Athlon 2700+ CPU and 256Mb of RAM. Searching the raw sequence data from the database took around two and a half seconds for a single motif, not including the (small) overhead incurring while registering and handling matches in the database. The hardware gave us a nearly fifty-fold speed-increase, allowing us to measure experiment times in hours instead of weeks.

To calculate the classification quality, we used the Matthews correlation coefficient, $-1 \leq C \leq 1$, as given in equation 1. In this formula P is a hit in the positive dataset, N a hit in the negative set and T and F is true and false respectively. -1 indicates perfect negative correlation and 1 indicates a perfect positive correlation. If the data have no correlation at all, C equals zero.[2] The Matthews correlation coefficient is a well established measure in biology and it's use as a fitness measure was inspired by other works [10, 12].

$$C = \frac{TP * TN - FN * FP}{\sqrt{(TN + FN)(TN + FP)(TP + FN)(TP + FP)}} \quad (1)$$

During a run a query might produce results that gave the numbers in the denominator the value zero. Examples of such cases would be when a query placed all samples in the same class - making for instance the sum of true positives and false negatives equal to zero. On such occurrences, the correlation was set to zero.

4. DATA AND EXPERIMENTS

The protein families we investigated were selected according to two criteria: selected families contained at least 200 different proteins, and the motifs or profiles describing the family should have more than zero false negatives or false positives. The first criterion was selected to ensure that we had enough positive examples in our test sets, to give our results some statistical validity. The second criterion ensured that we could possibly improve on the Prosite motifs. This process left us with some duplicates, as for example the Egf proteins and Zinc Finger proteins have two entries in the database. Removing these left us with 45 different protein families for our study, giving a broad view of the two algorithms' performance on different protein families.

For each family, we labeled all proteins in the family as positive, and all other proteins in the Swiss-Prot database as negative. For most families there existed proteins that were labeled as possible or partial members of the families. These proteins were excluded from each study.

Comparing classifiers is not easy. According to Salzberg, the recommended approach is to perform ten-fold cross validation, count the number of successes on a test set in each run, and perform a pooled Z-test for statistical significance. Due to time limitations we instead extracted 20% of the proteins for each family, and used these as a test set [19].

[2]Correlation of zero does not, on the other hand, indicate that the data are independent.

Table 1: Parameter settings for the different TreeGP runs

Run ID	P_{rep}	P_{xo}	P_{mut}	$P_{internal}$	$P_{external}$
Aggressive	0.0	0.8	0.2	0.2	0.8
Koza	0.1	0.9	0.0	0.0	0.0
Common	0.1	0.8	0.1	0.9	0.1

Table 2: Parameters for ListGP classifiers. Parameters enclosed in double lines must sum to one.

Parameter	setting
Tournament size	7
Expected motif length	8
Crossover	0.7
Mutation	0.3
Residue rate	0.6
Group rate	0.4

Also, still following Salzberg, it is adviced to use half of the training set as a tuning set to optimize parameter settings for both learners [19]. Performing the required tuning runs for 180 different experiments — tuning on half of our training-set for all 45 families for both the linear and tree-based representations — was again not feasible due to time-limitations. Instead we decided to compromise on this issue: we quite arbitrarily chose some parameters for our ListGP algorithm and allowed these to be fixed over all experiments, see Table 2. Then we chose three different parameter settings for the TreeGP algorithm. The parameter-settings used are described in Table 1. The first set of parameters in Table 1 is an "odd-ball" setting using a more aggressive mutation scheme, focusing on the terminals instead of internal nodes (labeled *Aggressive* in tables.) The second parameter set is similar to those used by Koza et al. in his early experiments. The last parameter set is labeled *Common*, and reflect "common practice" in other works.

This approach did not give us statistically significant data for all families, but it did so for enough families to allow us to make a confident judgment on the relative capabilities of the different representations.

Then, for each family, we took the best motif from each of the four runs (List, Aggressive, Koza and Common) and performed a one sided pooled Z-test with a 99% significance level to see if it had a significantly higher number of successes than the other motifs [7]. Our null-hypothesis was that the classifiers were equal, and our alternate hypothesis was that the motif with the best results actually was better.

To check the stability of the different algorithms, we calculated the average classification accuracy in terms of failure-rates over all protein families, with 99% confidence intervals for these averages. These data were also compared to the results for a majority vote "classifier" (labeled Majorvote in figures). Because of the large difference with respect to the negative and positive datasets, assigning all samples a negative label gives 99% correct classification results for all families.

Table 3: The number of runs the different configurations produced the best result, according to when they were significantly better than the competing methods with a significance level of 99%

Config	All	Some	None	Total
List	24(53.3%)	10(22.2%)	4(6.67%)	38(84.4%)
Aggressive	2(4.44%)	3(11.1%)	1(2.22%)	6(13.3%)
Koza	0	0	0	0
Common	0	1(2.22%)	0	1(2.22%)

Table 4: The number of wins for the different classifiers, for correlations and successes

Parameter set	By correlation	By count
List	36	37
Aggressive	8	7
Koza	0	0
Common	1	1

5. RESULTS

Table 5 summarizes the results from our hypothesis tests. The data are reported under the columns All, Some and None. Results in the different columns report how many times the motif with the highest score is significantly higher than All, Some or None of the other three motifs, respectively. For the latter column the Z-test indicates that the motifs are equally good classifiers. As can be seen from the table, motifs evolved with ListGP have a higher number of successful classifications for 38 of the 45 families, and they are significantly better than the other three for 24 families. The three times the aggressive settings were significantly better than some of the other settings, they were not significantly better than the motif evolved by ListGP.

The ten times ListGP was significantly better than only Some, it lost against the Common and Aggressive settings six times each, while Koza settings only caused a loss one of the times.

In other words, for 24 of the families, we have reasons to believe that ListGP evolved a classifier that is better than the others, and for 2 runs we have reasons to believe that one of the parameter settings for TreeGP is better. For the remaining twenty families, our hypothesis tests give us no reason to reject the null hypothesis that the evolved motifs are equal with respect to classification quality.

Looking at the data for the ZF_RING_2 family, the majority vote actually has the highest number of correct classifications. Running our hypothesis tests shows that it actually has a significantly better number of correct classifications than all the configurations for TreeGP, but not significantly better than the ListGP run. When we review the correlations on these runs given in Table 5, ListGP comes out with a correlation of 0.34, where the next competitor, Common, has 0.31 in correlation (a majority vote has a correlation of zero, of course.) Correlations for all evolved motifs are given in the table, and the best values are printed in bold font. Note also that for the TUBULIN family, both List and Aggressive have a bold font. For that experiment they did not tie only in value, but evolved the exact same motif (the evolved motif for the TUBULIN family is one of those listed in Table 6.) Other values in the table with equal values comes from rounding errors.

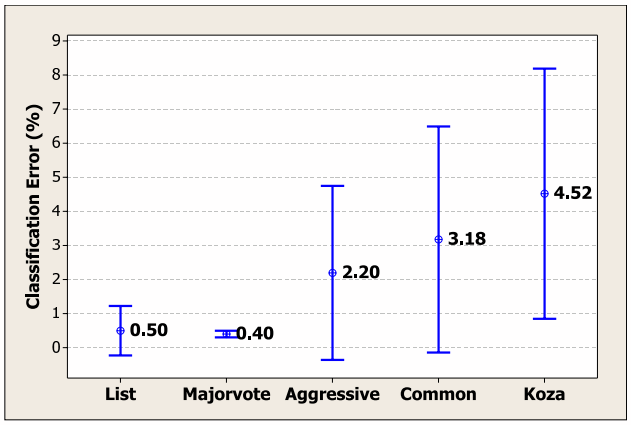

Figure 1: Interval plots for results over all 45 families, with 99% confidence intervals

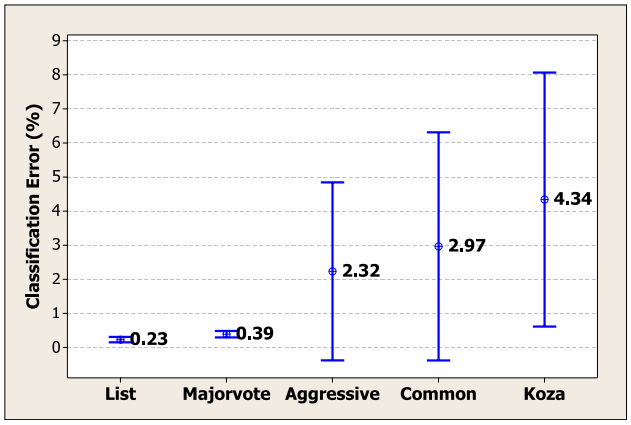

Figure 2: The interval plot for our results, after removing the data from the Cytochrome_C family. Still with 99% confidence intervals.

Looking at Figure 1, we see the average failure-rates over all the 45 families with 99% confidence intervals (calculated with MINITAB.) This gives a condensed view of the performance of the different algorithms and settings. The column labeled *Majorvote* is the performance of the "classifier" labeling all samples as negative. Its narrow confidence interval, $(0.3\%, 0.5\%)$, reflects that the sizes of all families, and thus the positive-negative samples ratio, are similar for all our experiments.

The figure shows that the failure-rates for most motifs from most configurations are low. This is because the negative sets are very large compared to the positive sets (typically a hundred times larger), and because a randomly created motif is very unlikely to hit any sequences at all — not to mention the correct ones. Therefore a small change in the number of correct classifications might still be due to a significant change in classifier quality.

While inspecting our data, we discovered that our results from the ListGP motifs had at least one result that differed considerably from the others. We removed the data for the

family in question from all experiments (Cytochrome C), and this gives us the results presented in figure 1. After this removal, the confidence interval for ListGP average error changed from $(-0.23\%, 1.22\%)$ to $(0.15\%, 0.31\%)$. This change put both average error rate and the 99% confidence interval below the confidence interval for the Majorvote average.

5.1 Evolved motifs

To give a picture of the motifs evolved with ListGP, we give some examples of motifs and compare them visually with the motifs in Prosite in table 6. These are, of all the motifs we evolved, the ones we found to have most in common with the Prosite motif based on visual inspections. Typically, we included our motif in this table if we could justify that all hits found by our ListGP motif recognize a subset of the sequences recognized by the Prosite motif.

6. DISCUSSIONS AND CONCLUDING REMARKS

As can be seen in Figures 1 and 2, the settings labeled Koza have on average the highest error rates of all experiments. We suspect that these results were caused by the exclusion of mutations, and that our initial populations are much smaller than those used by Koza and Andre in their experiments [10]. Still, adding mutations did not improve the performance considerably (Koza vs. Common in Figures 1 and 2.) The use of the parameter set labeled "Aggressive", highlighting mutations happening at the leaf nodes, was from the start regarded as an odd-ball choice on our part. Still this setting most of the time performed equally well and at some occasions even better than the other settings for TreeGP.

The average failure-rates for ListGP in figures 1 and 2 compares very well to the TreeGP experiments. The considerably narrower 99% confidence intervals for ListGP also give indications of a more stable behavior and less sensitivity to initial conditions. This is a favorable trait for any heuristic algorithm. Also, all tree-based algorithms have wide confidence intervals, suggesting that the behavior of TreeGP is more chaotic (and thus less controllable), independent of the different chosen parameter settings.

Most of the motifs we found were short compared to the Prosite motifs, and many were very different. The differences between the motifs might be accounted for by noting that we implemented a small subset of the Prosite language. The length on the other hand, might improve by running experiments with more generations and bigger initial populations.

As mentioned in the introductory section, these are initial results - and we hope to expand the pattern language applied by including both wildcards and flexible gaps. We also have some plans to develop some homologous crossover operators to investigate their effects on the algorithm performance.

We can not, based on our experiments and results, claim that a linear representation always would be better than a tree-representation for motif discovery in unaligned protein sequences. To establish such a claim requires further experiments and studies. Nevertheless, we have demonstrated that using trees as representations might not be the obvious choice for motif discovery.

Even though we don't improve on the Prosite motifs, the

Table 5: Correlations of evolved classifiers on the test sets for all families. Highest values in bold font. Collumn letters are abbreviations for the different settings: L(istGP), A(ggressive), K(oza) and C(ommon).

Family name	L	A	K	C
PROKAR_LIPOPROTEIN	0.15	**0.25**	0.06	0.09
EF_HAND	**0.26**	0.20	0.05	0.04
ZF_RING_2	**0.34**	0.26	0.17	0.31
EGF_1	**0.34**	0.29	0.25	0.30
HOMEOBOX_1	**0.89**	0.18	0.16	0.16
ZINC_FINGER_C2H2_1	**0.81**	0.79	0.72	0.72
HLH_1	**0.20**	0.17	0.13	0.17
RRM_RNP_1	**0.39**	0.14	0.08	0.08
HTH_LYSR_FAMILY	**0.35**	0.16	0.03	0.19
RECA_2	**0.65**	0.59	0.56	0.59
ADH_SHORT	**0.44**	0.28	0.08	0.18
GAPDH	**0.94**	0.69	0.56	0.93
CYTOCHROME_P450	**0.59**	0.43	0.24	0.35
CARBAMOYLTRANSFERASE	0.56	**0.82**	0.13	0.37
PUR_PYR_PR_TRANSFER	**0.54**	0.23	0.23	0.23
GATASE_TYPE_I	**0.73**	0.21	0.22	0.11
PROTEIN_KINASE_ATP	**0.70**	0.12	0.12	0.12
PROTEIN_KINASE_ST	**0.72**	0.28	0.11	0.25
PROTEIN_KINASE_TYR	**0.82**	0.38	0.11	0.35
PROTEIN_KINASE_DOM	**0.73**	0.13	0.13	0.13
PA2_HIS	**0.90**	0.42	0.42	0.42
PA2_ASP	**0.86**	0.41	0.41	0.41
TRYPSIN_HIS	0.93	**0.93**	0.81	0.93
TRYPSIN_SER	**0.97**	0.87	0.74	0.84
ZINC_PROTEASE	**0.65**	0.14	0.14	0.61
ATPASE_ALPHA_BETA	**0.52**	0.26	0.13	0.18
ATPASE_E1_E2	0.87	**0.97**	0.53	0.76
RUBISCO_LARGE	0.95	**0.97**	0.44	0.84
AA_TRNA_LIGASE_I	**0.59**	0.09	0.07	0.07
AA_TRNA_LIGASE_II	**0.23**	0.02	0.04	0.04
CYTOCHROME_C	0.23	**0.63**	0.23	0.23
CYTOCHROME_B_HEME	**0.91**	0.77	0.61	0.76
CYTOCHROME_B_QO	**0.94**	0.75	0.71	0.59
THIOREDOXIN	**0.70**	0.33	0.09	0.07
4FE4S_FERREDOXIN	**0.44**	0.42	0.19	0.42
ABC_TRANSPORTER_1	**0.54**	0.40	0.15	0.30
TUBULIN	**1.00**	**1.00**	0.61	0.92
G_PROTEIN_RECEP_F1_1	**0.51**	0.24	0.15	0.19
CHAPERONINS_CPN60	0.90	0.91	0.89	**0.91**
HSP70_2	**0.85**	0.65	0.65	0.65
HSP70_3	**0.81**	0.62	0.62	0.32
DNAJ_2	0.20	**0.33**	0.19	0.29
IG_LIKE	**0.31**	0.10	0.11	0.11
IG_MHC	**0.48**	0.24	0.18	0.18
WD_REPEATS_1	**0.35**	0.22	0.18	0.22

Table 6: The motifs found by our ListGP algorithm, that contained some resemblance to the Prosite motifs. Matching regions highlighted with bold font in Prosite motif.

Family	ListGP	Prosite
EGF_1	C-[LIQHDE]-C-[VPKH]-[VPKH]-[GNHE]	**C-x-C-x(5)**-G-x(2)-C
GAPDH	A-S-C-T-[FT]-[AVTN]	**[ASV]-S-C-[NT]-T-x**-x-[LIM]
PROTEIN_KINASE_ST	H-[CR]-D-[LIC]-K	[LIVMFYC]-x-**[HY]-x-D-[LIVMFY]-K**-x(2)-N-[LIVMFYCT](3)
TRYPSIN_HIS	[LIVWH]-[STR]-A-[GA]-H-C	**[LIVM]-[ST]-A-[STAG]-H-C**
TRYPSIN_SER	G-D-S-G-G-[APH]	[DNSTAGC]-[GSTAPIMVQH]-x(2)-**G-[DE]-S-G-[GS]-[SAPHV]**-[LIVMFYWH]-[LIVMFYSTANQH]
ZINC_PROTEASE	H-E-[LFV]-[GAFTCD]-H	[GSTALIVN]-x(2)-**H-E-[LIVMFYW]-{DEHRKP}-H**-x-[LIVMFYWGSPQ]
AA_TRNA_LIGASE_I	[LM]-H-[IMVT]-G-H	P-x(0,2)-[GSTAN]-[DENQGAPK]-x-**[LIVMFP]-[HT]-[LIVMYAC]-G-[HNTG]**-[LIVMFYSTAGPC]
CYTOCHROME_C	C-H	C-{CPWHF}-{CPWR}-**C-H**-{CFYW}
TUBULIN	G-G-T-G-[AS]-G	[SAG]-**G-G-T-G-[SA]-G**
HSP70_2	G-G-G-T-[LF]-D	[LIVMF]-[LIVMFY]-[DN]-[LIVMFS]-**G-[GSH]-[GS]-[AST]-x**-x(2)-[ST]-[LIVM]-[LIVMFC]

stability of the performance and some of our "near misses" is encouraging. We hope to present improved results ranging a larger subset of the Prosite language in the near future.

Acknowledgments

Thanks go to professor Arne Halaas and associate professor Finn Drabloes, for encouragement and insightful discussions. Also thanks to Interagon for giving us access to their pattern matching hardware. Further thanks go to the reviewers of the article, on their insightful comments for improvements to the paper. Finally, thanks to Magnus Lie Hetland for proofreading the paper.The responsibility for any remaining errors lies with the authors.

7. REFERENCES

[1] Halaas A., B Svingen, M Nedland, P Saetrom, Jr. Snove, O., and O.R. Birkeland. A recursive MISD architecture for pattern matching. *IEEE Transactions on Very Large Scale Integraion (VLSI) Systems*, 12(7):727–734, July 2004.

[2] Boeckmann B., BAiroch A., Apweiler R., Blatter M, Estreicher A., Gasteiger E., Martin M, Michoud K., O'Donovan C., Phan I., Pilbout S., and Schneider M. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Research*, 31:365—370, 2003.

[3] B.Olsson, K.Laurio, and L.Gudjonsson. A hybrid method for protein sequence modeling with improved accurasy. *Information Sciences 139 (2001) 113-138*, 2001.

[4] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):277–304, 1998.

[5] Jason M. Daida and Adam M. Hilss. Identifying structural mechanisms in standard genetic programming. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1639–1651, Chicago, 12-16 July 2003. Springer-Verlag.

[6] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[7] Larry Gonick and Woollcott Smith. *Cartoon Guide to Statistics*, chapter 9. HarperPerennial, 1993.

[8] Yuh-Jyh Hu. Biopattern discovery by genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 152–157, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[9] Nicolas Hulo, Christian J. A. Sigrist, Virginie Le Saux, Petra S. Langendijk-Genevaux, Lorenza Bordoli, Alexandre Gattiker, Edouard De Castro, Philipp Bucher, and Amos Bairoch. Recent improvements to the PROSITE database. *Nucl. Acids Res.*, 32(90001):D134–137, 2004.

[10] John R. Koza and David Andre. Automatic discovery using genetic programming of an unknown-sized detector of protein motifs containing repeatedly-used subexpressions. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 89–97, Tahoe City, California, USA, 9 July 1995.

[11] John R. Koza and David Andre. Automatic discovery of protein motifs using genetic programming. In Xin

Yao, editor, *Evolutionary Computation: Theory and Applications*. World Scientific, Singapore, 1996. In Press 1997?

[12] Bjorn Olsson. Using evolutionary algorithms in the design of protein fingerprints. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1636–1642, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[13] Gregory Paris, Denis Robilliard, and Cyril Fonlupt. Applying boosting techniques to genetic programming. In P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001*, volume 2310 of *LNCS*, pages 267–278, Creusot, France, October 29-31 2001. Springer Verlag.

[14] I. Rigoutsos, A. Floratos, L. Parida, Y. Gao, and D. Platt. The emergence of pattern discovery techniques in computational biolog. *Metabolic Engineering*, 2:159–177, 2000.

[15] Brian J. Ross. Probabilistic pattern matching and the evolution of stochastic regular expressions. In Scott Brave and Annie S. Wu, editors, *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pages 229–237, Orlando, Florida, USA, 13 July 1999.

[16] Brian J. Ross. Probabilistic pattern matching and the genetic programming of stochastic regular expressions. *International Journal of Applied Intelligence*, 3(3):285–300, November/December 2000.

[17] Brian J. Ross. The evaluation of a stochastic regular motif language for protein sequences. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 120–128, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[18] Brian J. Ross. The evolution of stochastic regular motifs for protein sequences. *New Generation Computing*, 20(2):187–213, February 2002.

[19] Steven Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328, 1997.

[20] G. C. Wilson, A. McIntyre, and M. I. Heywood. Resource review: Three open source systems for evolving programs–lilgp, ECJ and grammatical evolution. *Genetic Programming and Evolvable Machines*, 5(1):103–105, March 2004.