EURASIP Journal on Applied Signal Processing 2004:1, 138–145

A Genetic Programming Method for the Identification of Signal Peptides and Prediction of Their Cleavage Sites

David Lennartsson
Saida Medical AB, Stena Center 1A, SE-412 92 Göteborg, Sweden
Email: david.lennartsson@saida-med.com

Peter Nordin
Department of Physical Resource Theory, Chalmers University of Technology, SE-412 96 Göteborg, Sweden
Email: peter.nordin@mc2.chalmers.se

Received 28 February 2003; Revised 31 July 2003

A novel approach to signal peptide identification is presented. We use an evolutionary algorithm for automatic evolution of classification programs, so-called programmatic motifs. The variant of evolutionary algorithm used is called genetic programming where a population of solution candidates in the form of full computer programs is evolved, based on training examples consisting of signal peptide sequences. The method is compared with a previous work using artificial neural network (ANN) approaches. Some advantages compared to ANNs are noted. The programmatic motif can perform computational tasks beyond that of feed-forward neural networks and has also other advantages such as readability. The best motif evolved was analyzed and shown to detect the h-region of the signal peptide. A powerful parallel computer cluster was used for the experiment.

Keywords and phrases: signal peptides, genetic programming, bioinformatics, programmatic motif, artificial neural networks, cleavage site.

1. INTRODUCTION

The huge and growing amount of unanalyzed data present in genetic research creates a demand for automatic methods for classification of proteins and protein properties. Automatic mechanical means for property screening of interesting proteins would accelerate the process of finding new drug candidates.

Classification rules for the processing of amino acid sequences can be obtained either by human design or by a mechanical process, the latter often through the use of machine-learning algorithms.

A signal peptide is a short region of amino acid residues situated at the N-terminal part of some peptide chains. Commonly, signal peptides are referred to as the address tags within the cell since they control the transport of proteins through the *secretory pathway*, the mechanism that moves proteins through cell membranes. These proteins are produced by ribosomes in the cytoplasm but the produced peptide does not fold to become a protein at this stage. Instead, the first part of the peptide, the signal peptide, attaches itself to a *translocon* in the membrane. This binding opens a channel and the peptide starts to transport itself through the translocon channel. After transportation through the membrane, the signal peptide cleaves from the protein's peptide and the channel is closed. The protein's peptide is now free and can fold itself to become an active, or *mature*, protein.

The existence of a signaling mechanism in the cell was first postulated by Günther Blobel in 1971. After a series of experiments, he came to the correct conclusion that the signal, or address tag, was coded with amino acids as part of the peptide and the transport went through channels in the membranes. Later, Blobel could verify that the process was universal. The same mechanisms work not only in animal cells but also in bacteria, yeast, and plants. For his work, Blobel received the Nobel prize in medicine in 1999.

The knowledge about signal peptides has been instrumental in understanding some hereditary diseases caused by proteins not reaching their intended destination. It is also believed that signal peptides will help in engineering yeast cells into drug factories. Drugs could then be delivered from the cells through secretion.

2. PREVIOUS RESEARCH

An early approach to signal peptide classification is the matrix method used by von Heijne in [1]. The matrix was

constructed out of the known signal peptides at the time and gave results of a sequence level performance of 78% correct classification for eukaryotic sequences.

Nielsen et al. [2] improved on the weight matrix method and carried out an experiment where they used feed-forward artificial neural networks trained with backpropagation to predict if a peptide had a signal peptide attached or not.

To compare this method with the more traditional weight matrix method, they started with a recalculation of the matrix weights using the sequences already known. In 1996, the number of known signal peptides was 5–10 times greater than in 1986. However, the results were considerably worse than the results obtained by von Heijne in 1986, and only 66% of the eukaryotic sequences were classified successfully. Nielsen et al. attributes the failure either to larger variation in the signal peptides found since 1986 or to more frequent errors in the dataset. The 1986 dataset was hand-compiled while Nielsen et al. used an automatic method.

The neural network method combined the results of two individually trained networks that were trained on different tasks. The first network tried to predict if a specific position in the sequence was part of the signal peptide or not while the second network tried to predict if the position was the cleavage site. The combined output from the two networks was based on changes in the output from the first network close to peaks in the output from the second network. Together, the two networks managed to predict 70% of the eukaryotic sequences correctly and 68% of the sequences from the human dataset. Their method and signal peptide identification service is known as *signalP*.

The use of genetic programming (GP) for protein classification tasks has been pioneered by Koza. In [3], he uses it to find protein *motifs* and in [4] he coined the term *programmatic motif* and used the method for evolving a rule that predicted the cellular location of a given protein. Both experiments produced results better than any other method at the time, including hand-crafted motifs.

3. DATA

In our experiments, we used the data Nielsen et al. made public on their ftp-server [5]. It is the same data they used in their own experiments and the data originates from SWISS-PROT version 29 [6]. Nielsen et al. started with selecting sequences marked with SIGNAL. From the SIGNAL group, they removed all proteins where they could suspect that they had been tagged as SIGNAL in a nonverified way, that is, by the use of prediction algorithms or guessing. As a background, they chose different known cytoplasmic and nuclear proteins. Here they also removed all entries that seemed to be nonverified.

Furthermore, they also compared the data and excluded sequences that were too similar to others. In this way redundancy in the dataset was reduced. For a more detailed description of the extraction and preparation of the dataset, see [2, 7].

Nielsen et al. performed their experiment on several different groups of proteins including human, *E. coli*, eukaryotes, and gram+ and gram− bacteria, with similar results for all groups. For experiments described in this paper, we chose to work only with the human dataset.

In our experiments, the data was split into two sets: one *training set* consisting of 176 background proteins and 291 signal peptides and one *validation set* consisting of 75 background proteins and 125 signal peptides. For every position in the peptide sequence, the dataset included information telling whether it was part of a mature protein or part of a signal peptide. An excerpt from the dataset is shown in Figure 1.

The peptide sequences were truncated after 70 amino acids for background proteins. In the case of signal peptides, the signal part and the first 30 positions of the mature protein were kept. This makes sense since the process of translocation starts before the whole peptide is produced by the ribosome.

4. METHOD

We have used the machine-learning technique GP. GP is a branch of evolutionary algorithms where computer programs are evolved from first principles to solve a problem specified by a fitness function. Although GP has many features in common with other branches of evolutionary computation, such as genetic algorithms (where often fixed-length binary genomes are evolved), the solutions evolved by a GP system are more complex and can solve harder problems; they are often complete *programs* or *algorithms*.

In GP, a population of *solution candidates*, individual programs, is kept and these individuals compete for the right to reproduce. During mating, variations are introduced in the offspring's genome by the use of *genetic operators*. Two common simulated operators are mutation and sexual recombination. The undirected mechanisms of random variation combined with selection through survival of the fittest leads to evolution. The competing individuals in the population will usually improve over time at the task by which they are graded, and the more *fit* individuals survive and proliferate.

The solution candidates, or the individuals, have two appearances, the *genotype* and the *phenotype*. The genotype is the genome, the recipe that builds the phenotype, and the behavior of the program. In GP, the phenotype is a program being executed on a real or simulated *machine*. Depending on the phenotype's performance, the genotype may reproduce. Since the selection criterion is defined as an external property, the algorithm might be seen as more similar to breeding than to actual evolution.

Three different types of genomes are common in GP: tree-like, linear, and graph-like. In this experiment, a linear representation of the genome was used. For more background on GP and discussions about genome, representation, theory, and different selection mechanisms, see [8, 9, 10, 11].

The individuals in the population had variable-length genomes that could contain up to 300 instructions. Evolution started with a population with genomes of random length and random content (genes).

```
 0
70 RPB2_HUMAN    DNA-DIRECTED RNA POLYMERASE II 140 KD POLYPEPTIDE
MYDADEDMQYDEDDDEITPDLWQEACWIVISSYFDEKGLVRQQLDSFDEFIQMSVQRIVEDAPPIDLQAE
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM


 1
51 1OKS_HUMAN     21 CLARA CELLS 10 KD SECRETORY PROTEIN PRECURSOR (CC10).
MKLAVTLTLVTLALCCSSASAEICPSFQRVIETLLMDTPSSYEAAMELFSP
SSSSSSSSSSSSSSSSSSSSCMMMMMMMMMMMMMMMMMMMMMMMMMMMM
```

Figure 1: All the sequences have a class, a name, and a specification of which kind of peptide the acid is part of. Here, S means that the amino acid is part of the signal peptide while C and M are parts of the mature protein; C marks the cleavage site.

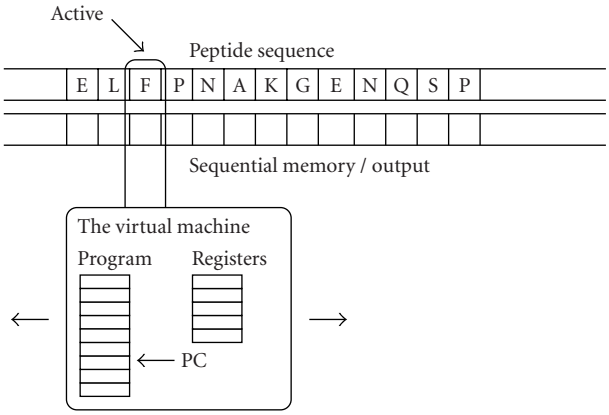

Figure 2: The evolved program instructs the virtual machine to move along the sequence and to perform calculations on registers and writing to memory.

4.1. The virtual machine

The linear genomes of the individuals are interpreted as a computer program by a *virtual machine*. The virtual machine used was implemented as a register machine. The machine had the ability to analyze the peptide sequence, perform arithmetics with five registers, and use a sequential memory. A schematic of the machine is shown in Figure 2.

Each position in the individual's genome represents a complete instruction and is encoded as a 32-bit integer. The first eight bits encodes the operation while the following three bytes are passed as arguments. The most common argument is a pointer to a register, but depending on the operation, it could also be interpreted as a real-valued constant or a relative program address. Regardless of how a gene is coded, it is always reinterpreted as a valid instruction with valid arguments.

The following operations were supported by the machine:

(i) Boolean operators: and, or, xor, not;
(ii) register setting operators: one, clear, set;
(iii) arithmetic operators: add, sub, mul, div, sigmoid;
(iv) branching operators: ifgtz, jmp, jmpgtz;
(v) head-moving operators: for, rev, home;
(vi) memory-altering operators: read, write;
(vii) amino acid residue detecting operators: ala, arg, asn, asp, cys, glu, gln, gly, his, ile, leu, lys, met, phe, pro, ser, thr, trp, tyr, val, aliphatic, aromatic, charged, hydrophobic, negative, polar, positive, small, tiny.

The application-specific operators in this virtual machine are the amino acid residue detecting operators. These instructions return positive if the machine is positioned over the respective target. Otherwise, a negative result is returned. There are also instructions to determine if a target has a specific chemical property.

The genome of an individual contains up to 300 instructions forming a program. The program is the individual and from this point that is what we refer to when using the word program. The virtual machine and the computational methods around it, such as fitness measurement, are referred to as the system.

The evaluation of an individual program was executed once for every peptide in the training set of fitness cases. Before every run, both registers and sequential memory were being reset to zero and the program counter was initiated to zero. The head of the virtual machine was moved to the first position in the sequence of the peptide to examine.

When the program was executed, it could instruct the virtual machine to move along the peptide chain and check for amino acid residues or properties of the residues. In between those operations, it could perform calculations on its registers and/or write to sequential memory. The sequential memory would also be treated as the output of the program. If a memory cell in the sequential memory held a value greater than zero at program termination, that cell's position was considered to be a prediction of a cleavage site. The value zero or less was considered as no prediction.

Programs terminated when reaching the end of the program or when a jump instruction instructed the machine to jump outside the program. If a program used all of its allowed executions, all branching operators were treated as NOPs (no operation) and the program terminated when the end of the program was reached. The execution limit was set to 800 instructions per run. The program would also terminate if the head was moved outside the peptide sequence.

For a more thorough description of register machine GP, see [8].

4.2. Fitness measurement

After the evaluation of the peptide sequences, the result had to be analyzed in order to assign a fitness to the individual. This process may be the most important in GP due to the principle "what you train is what you get."

The main part of the fitness was made up of errors associated with the distance between the real and the predicted cleavage site. For every predicted position, the error d^2 was added to the fitness. If the program tagged several positions, it would receive multiple penalties and thus such behavior would result in poor fitness. If no position was tagged on a signal peptide, the program would get a penalty that corresponds to a distance d of 17. The same was true for nonsignal peptides that were falsely classified to have a cleavage site.

To further guide the evolution, the fitness assigning function was made more smooth by adding a small error for every position in the memory. The system expected the program to return one for cleavage sites and minus one for every other position. Deviations from these values and an extra penalty $p = 0.15$ for falsely classified positions were added to the fitness.

Later when the system activated *parsimony pressure*, it also added a small cost associated with execution of instructions to the fitness. This cost was small enough not to affect the results of the comparison other than when the system had to choose between two equally performing individuals with different sizes. Finally, there were some penalties needed to avoid cheating and control the behavior of the program. These penalties were large. First, if a program used recursion and did not terminate before using its available 800 instructions, it would be punished for loop violation. Second, if a program produced constant output for different peptides in the set, the program would get punished.

The last punishment was received if the program tried to move the head of the virtual machine outside the peptide sequence. This was needed to avoid cheating where the program otherwise could locate the end of the sequence and count a certain number of steps back from that point. Such "cheating" solutions were often evolved by the system if no penalty was given. The total fitness function is

$$f = \frac{1}{\text{peptides}} \sum_{\text{Peptides}} \left(d^2 + \text{parsimony} + \frac{1}{\text{length}} \sum_{\text{Positions}} (e^2 + p) \right) + \text{loop\_violation} + \text{constant\_output} + \text{illegal\_move}. \quad (1)$$

The fitness was balanced in such a way that individuals first prioritize minimizing d, then e, and lastly the size of solution (parsimony pressure). The penalties for illegal behavior dominate over all of the above.

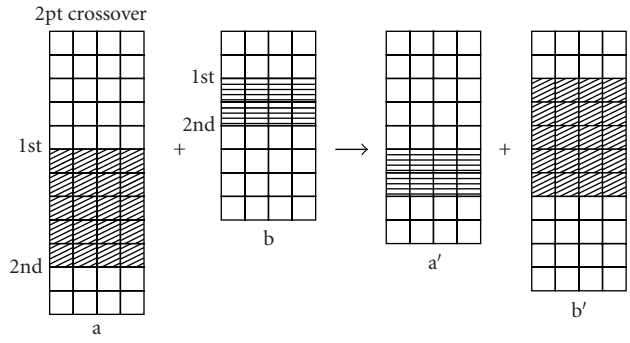

Figure 3: If sexual recombination takes place, the children (a′) and (b′) will be a combination of the parents (a) and (b) genomes. Recombination works by letting the crossover operator exchange two random parts of the genomes.

4.3. Selection and genetic operators

We used steady-state tournament selection. For every evolutionary step, four arbitrary individuals are selected. They compete against each other in two pairs and the best two individuals from the two (semifinal) games mate.

Mating produces two offspring. It can be either two perfect copies of the parents or recombinations of the parents genomes. Two-point crossover was used for recombination, shown in Figure 3. There is also a small chance that the genome of a child will be mutated at a single position.

The two less-performing individuals who were defeated in the tournament are removed while the parents and the offspring stay in the population. The process of tournaments is iterated over many generations.

4.4. Parallelization

To speed execution up, six workstations were clustered together using demes. Equal-sized subpopulations were kept in each deme and one percent of the population migrated to another deme every generation. The demes were connected with a ring-like topology.

The clustering gave a full linear speedup and there was no performance degradation due to clustering. Indications of superlinear speedups [10] were found but the experiment did not run sufficient number of times to statistically support such claims. A comparison of the evolutionary progress for a single population and a population spread over demes can be seen in Figure 4. When the system utilizes demes, the population evolves faster. It can be noted that the effort in Figure 4 is measured in computer time and that the system taking advantage of clustering was more than six times faster in real time than the system utilizing a single workstation.

5. RESULTS

The results presented in the following sections show the best performing individual. During the run, a population of twenty thousand programs was evolved for four million tournaments. Approximately eight million different solutions were tried. Parsimony pressure was added after two million

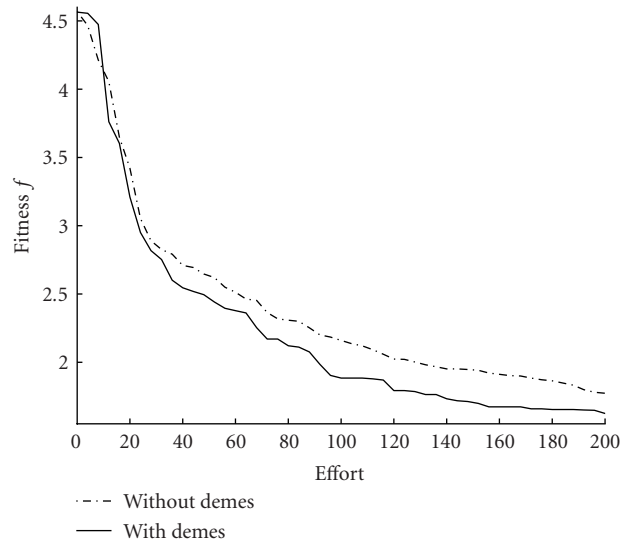

Figure 4: A comparison between a demes population and a non-demes population. The progress of evolution as the function of total computational effort. The mean fitness out of three runs plotted for both having the population spread out over demes or keeping all individuals in a single population.

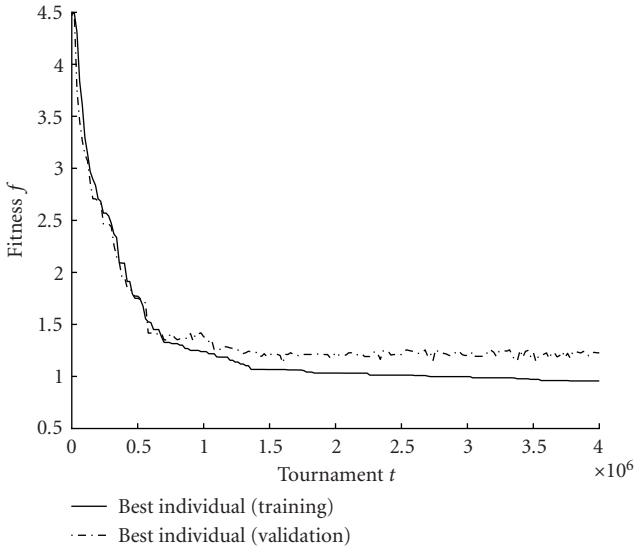

Figure 5: Fitness for population. The fitness of the two best performing individuals on training and validation data.

tournaments. During mating, there were a 98% probability of sexual recombination and 15% probability of mutation.

The best performing individual was 273 instructions long and had formed through 383 genetic operations. The whole run took about three days on standard PC hardware running at 500 MHz.

In Figure 5, we can see how the population becomes more fit over generations. Even though the best individual continues to improve on training, we do not see evidence of any overlearning. The individuals are general solutions to the problem, and fitness on validation data remains similar to that of the training fitness.

Table 1: Performance for the identification of signal peptides (best individual).

	Training set	Validation set	Whole set
Correctly identified (%)	92.5	92.5	92.5
MCC	0.84	0.84	0.84

5.1. Identification of signal peptides

The first quality measurement of the individual is how reliable the program is classifying a sequence as a signal peptide or not. Any sequence that produces an output above zero in any cell of the sequential memory is considered to be a signal peptide, while the sequences where all outputs are at or below zero are considered to be classified as background data.

We use the Matthew correlation coefficient [12] to determine the performance of a rule in addition to percentage of correctly classified signal peptides. The coefficient is defined as

$$C_{MCC} = \frac{N_{tp}N_{tn} - N_{fp}N_{fn}}{\sqrt{(N_{tn} + N_{fn})(N_{tn} + N_{fp})(N_{tp} + N_{fn})(N_{tp} + N_{fp})}}. \quad (2)$$

The coefficient C_{MCC} equals one for a perfect prediction, minus one for a total opposite prediction, and zero for a completely random prediction. The variables N_{tp}, N_{tn}, N_{fp}, and N_{fn} represent the number of correctly classified positives, correctly classified negatives, falsely classified positives, and falsely classified negatives, respectively.

The performance of the best individual on the task of identifying signal peptides is presented in Table 1. The individual managed equally well on the training and validation cases and actually had a lower fitness on the validation data than on the training set which indicates that there was no overtraining.

5.2. Predicting cleavage site location

After identifying which sequences that include a signal peptide, we would like to know where their cleavage sites are located. The individuals are trained to minimize the distance between predicted and actual cleavage site. This is introduced in the fitness as a sum over d^2.

To verify how well the individuals perform on locating the cleavage site, the percentage of signal peptide sequences with correctly predicted cleavage sites was measured. In this case, a correct prediction is a predicted cleavage site at most two positions away from the real site.

The results of the same best individual as in the previous sections are presented in Table 2. To further know if this result was better than a random guess, the average distance between the predicted cleavage site and the real cleavage site was calculated.

Table 2: Performance for the prediction of cleavage sites (best individual).

	Training set	Validation set	Whole set
Correctly predicted (%)	53.3	61.6	55.8
Mean d^2	12.2	12.7	12.3

To put the measured distance d^2 into perspective, a couple of different test measurements were carried out. First we measured how large the mean value of d^2 would be if the prediction algorithm chose random points distributed uniformly between the two extreme positions for cleavage sites found in the whole dataset. The mean, out of a 100 test runs, yielded a d^2 of 194. This large d^2 is expected since the distribution of cleavage site positions is far from uniform. Next step was to use the discrete frequency distribution in the dataset to transform the randomness to follow the distribution. These runs gave a mean square distance of 55. Thus, no random solutions could compete with the measured distance of the best individual.

Earlier in the studies, the system had produced individuals with constant output which managed to reach quite low fitness and therefore the mean distance for various constant solutions is needed to be measured. The best constant solution was the one stating that the cleavage site was positioned at position 24 in the peptide sequence. This solution had a mean d^2 of 28.

In comparison with the tests above, it is clear that the best individual evolved far from being a random guess or optimal constant solution.

5.3. Analysis of the best individual program

One of the often stated advantages of GP compared, for instance, to artificial neural networks is the ability to produce the result in a human readable form. It is much harder to analyze the weights and get a grip of how an artificial neural network is calculating its results than to analyze program code.

In our case, the task of analysis takes some effort since we let the program evolve without any constraints on its architecture. The individuals could evolve loops and subfunctions with the help of branching instructions. Since the individuals only had one single linear genome, these functions sometimes overlapped. A loop may partially overlap with another loop and some parts of the code will be used differently at different times. Still the function of an individual is not that hard to understand.

Although the mechanism for targeting signal peptides work similar in all organisms, the signal peptides do not share one common sequence. They do however share a common structure. There are some simple rules of thumb to detect a signal peptide. First the sequence should start with a short region, usually of positively charged amino acids, called the *n-region* at the N-terminal of the peptide. It is followed by a somewhat longer region of hydrophobic amino acids called the *h-region*. Between the hydrophobic region and the cleavage site is a short region consisting mainly of polar and uncharged amino acids named the *c-region*. At the positions before the cleavage site, a pattern called the $(-3, -1)$ rule is common. It states that position -1 and -3 relatively to the cleavage site should be occupied by small and neutral residues. The amino acid residue at position -2 can however be an aromatic, charged, or large polar residue.

A quick analysis of the program from the best individual revealed that at most 30% of the instructions contributed to the solution. The others are known in genetic programming as *introns*, genes/instructions that are inactive. Introns are also common in nature and could among other functions be a product of evolution's desire to protect important information in the genome from mutations. In GP, they consist of operations where the results produced will be overwritten by another operator without being used anywhere in between.

The evolved program consists mainly of two parts where the first part is made up of four nested loops. The program will stay inside these loops and iterate over the peptide sequence until it has come across four aliphatic residues and has not detected any proline or arginine. If encountered, the program will go back and loop some more. When this happens, the program moves around eleven positions forward. There, it performs a simple check and marks the position as a cleavage site if there is no tryptophan there. Tryptophan is a large aromatic residue. Aliphatic residues are also hydrophobic, so it seems that our program has found a simple rule relying on finding the h-region, moving across the most common number of positions and marking the cleavage site if not completely wrong. The code seems very simple but still the program can discriminate between signal peptides and other proteins with good accuracy. It has also successfully predicted cleavage sites as close to the N-terminal as 17 positions and as far away as 37 positions, so the rule spans over signal peptides with quite different characteristics.

6. COMPARISON WITH PREVIOUS METHODS

Nielsen et al. presented their results on the task of the identification of signal peptides with the help of Matthews correlation coefficient and reported it to be $C_{SP} = 0.96$, as the best, for the human dataset. This is a good value but they tried several ways of interpreting the output from the network and also optimized the threshold value used in the interpretation. When they only used their cleavage site predicting network, which is more similar to the approach presented in this paper, and used the highest output to determine if a sequence has a signal peptide or not, they got a $C_{SP} = 0.71$ which is worse than the $C_{SP} = 0.84$ reached in this experiment.

When it comes to predicting the cleavage site, Nielsen et al. reported a 68.0% success rate on the human dataset using the combined output from two different neural networks. The weight matrix method with newly calculated weights scored 66.7%. According to a survey performed by Emanuelsson et al. [13], *TargetP*, the successor to signalP,

correctly predicted 81.1% of the cleavage sites within two positions from the real site. The best individual in our experiment scored 55.8%.

Although this is comparing apples to oranges, it can be interesting to note how much parameters are included in the solutions. The two networks used to classify human signal peptides contained in total 3080 real-valued parameters while the program produced through GP had a length of 273 32-bit instructions. About 30% of these instructions were actually used in the solution. The instruction set is highly redundant and could easily fit into a 16-bit representation. The evolved program can be described using much less information than the neural network.

GP is also generally less sensitive to initial parameter settings than neural networks, making it possibly a more robust search tool.

Another difference between the systems is the ability to learn from the solution derived from the method. The resulting program from the GP system is available in a human-readable form, although it may take some work to sort it out. This way, the GP approach holds promise for the future since it is not only a program that predicts, but also it can produce new human knowledge.

7. DISCUSSION

The evolved programs have a quite complex architecture with the ability to create iterations and conditional loops. The programs evolved by GP can therefore express completely different patterns than practically possible with artificial neural networks. This may also make a hybrid method between neural networks and a candidate for future research.

A great deal of effort was spent to prevent programs from "cheating." Examples of cheating would be to count positions from the end of the peptide in the dataset. Although it is clear that the predictive performance of the neural networks is not affected by this kind of cheating, it is not fully evident from publications if enough effort is spent on preventing the network from building up the kind of function needed for all kinds of possible cheating.

Our results are not verified with cross-validation. Instead, we have relied solely on the use of separate training and validation sets. Since no overlearning has been detected, we judge this method as sufficient. We would however like to use cross-validation in the future but there are questions regarding its accuracy in combination with evolutionary techniques.

The system identified and extracted a rule similar to a hand-discovered rule within signal peptide sequence analysis. On the task of the identification of signal peptides, the evolved rule faired well. The combined score of the neural networks was however significantly better at prediction of the cleavage sites.

The interpretability of solutions enables the GP technique to be used for extraction of new knowledge regarding cleavage sites and signal peptides. The clear text output enables reformulation as human knowledge.

8. CONCLUSION

We have shown that GP can be used to extract features in peptide sequences. The resulting "programmatic motifs" have a high expressiveness and can express other information than practically possible with, for example, neural networks.

Unlike many other methods, the resulting program is available in a human-readable form and is interpretable. An analysis of the program showed that it has evolved a rule that relied heavily on finding the hydrophobic core in the signal peptide.

GP is still a young research field and this report describes one of the first experiments on peptide classification with this method. Our results points to the feasibility of further use of genetic programming in sequence analysis tasks.

ACKNOWLEDGMENT

Peter Nordin gratefully acknowledges the support from Owe Orwar.

REFERENCES

[1] G. von Heijne, "A new method for predicting signal sequence cleavage sites," *Nucleic Acids Res.*, vol. 14, no. 11, pp. 4683–4690, 1986.

[2] H. Nielsen, J. Engelbrecht, S. Brunak, and G. von Heijne, "A neural network method for identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites," *Int. J. Neural Syst.*, vol. 8, no. 5-6, pp. 581–599, 1997.

[3] J. R. Koza and D. Andre, "Automatic discovery of protein motifs using genetic programming," in *Evolutionary Computation: Theory and Applications*, X. Yao, Ed., World Scientific, Singapore, 1996.

[4] J. R. Koza, F. Bennett, and D. Andre, "Using programmatic motifs and genetic programming to classify protein sequences as to extracellular and membrane cellular location," in *Evolutionary Programming VII: Proceedings of the 7th Annual Conference on Evolutionary Programming*, V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds., vol. 1447, Springer-Verlag, San Diego, Calif, 1998.

[5] H. Nielsen, S. Brunak, J. Engelbrecht, and G. von Heijne, *Data from signalP ftp-site*, http://www.cbs.dtu.dk/ftp/signalp/.

[6] A. Bairoch and B. Boeckmann, "The SWISS-PROT protein sequence data bank: current status," *Nucleic Acids Res.*, vol. 22, no. 17, pp. 3578–3580, 1994.

[7] H. Nielsen, J. Engelbrecht, G. von Heijne, and S. Brunak, "Defining a similarity threshold for a functional protein sequence pattern: the signal peptide cleavage site," *Proteins*, vol. 24, pp. 165–177, 1996.

[8] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction*, Morgan Kaufmann, San Francisco, Calif, 1998.

[9] J. R. Koza, *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass, 1992.

[10] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann, San Francisco, Calif, 1999.

[11] R. Poli and W. B. Langdon, *Foundations of Genetic Programming*, Springer-Verlag, Berlin, 2002.

[12] B. W. Matthews, "Comparison of predicted and observed secondary structure of T4 phage lysozyme," *Biochemica et Biophysica Acta.*, vol. 405, no. 2, pp. 442–451, 1975.
[13] O. Emanuelsson, H. Nielsen, S. Brunak, and G. von Heijne, "Predicting subcellular localization of proteins based on their N-terminal amino acid sequence," *J. Molecular Biology*, vol. 300, no. 4, pp. 1005–1016, 2000.

David Lennartsson has been working as a Consultant in software development for several years. He received his M.S. degree in engineering physics from Chalmers University of Technology, Sweden, in 2003. This paper is originally based on his thesis work. Currently, he is focusing his research efforts on systems for knowledge extraction and decision support using intelligent heuristics such as genetic programming. Mr. Lennartsson is one of the founders of SAIDA Medical which develops methods for automatic statistical inference and modelling.

Peter Nordin received his M.S. degree in computer science and engineering from Chalmers University of Technology, Sweden, in 1989, and his Ph.D. degree in computer science from the University of Dortmund, Germany, in 1997. He has worked for several years as a Researcher and Consultant in the area of knowledge-based systems, artificial intelligence, and evolutionary algorithms at Infologics AB, a subsidiary of Swedish telecom. Dr. Nordin is a Cofounder of Dacapo AB, a Swedish consulting and research company specialised in the state-of-the-art information technology, and an Inventor of the patented AIM-GP genetic programming method, a very efficient approach to GP. He has published 90 papers on genetic programming. He has been Program Cochair of EuroGP'99, Second European Workshop on Genetic Programming, and is in the editorial board of the Journal of Genetic Programming and Evolvable Hardware. Dr. Nordin has been a member of several European research projects. Since 1998, he has been an Associate Professor in the Complex Systems Group at Chalmers University of Technology.