Evolving Regular Expression-based Sequence Classifiers for Protein Nuclear Localisation

Amine Heddad, Markus Brameier and Robert M. MacCallum
Stockholm Bioinformatics Center, Stockholm University, 106 91 Stockholm, Sweden
{heddad,brameier,maccallr}@sbc.su.se

January 2004

Abstract

A number of bioinformatics tools use regular expression (RE) matching to locate protein or DNA sequence motifs that have been discovered by researchers in the laboratory. For example, patterns representing nuclear localisation signals (NLSs) are used to predict nuclear localisation. NLSs are not yet well understood, and so the set of currently known NLSs may be incomplete. Here we use genetic programming (GP) to generate RE-based classifiers for nuclear localisation. While the approach is a supervised one (with respect to protein location), it is unsupervised with respect to already-known NLSs. It therefore has the potential to discover new NLS motifs. We apply both tree-based and linear GP to the problem. The inclusion of predicted secondary structure in the input does not improve performance. Benchmarking shows that our majority classifiers are competitive with existing tools. The evolved REs are usually "NLS-like" and work is underway to analyse these for novelty.

1 Introduction

Many bioinformatics tools aim to increase our knowledge about the growing number of proteins for which little experimental information is available. One important approach is to transfer annotations from characterised proteins to less well studied proteins. This is usually achieved through similarity searches against databases of whole sequences (e.g. with BLAST[1] or FASTA[22]) or domain libraries (e.g. INTERPRO[19]), or with more localised motif or subsequence searching (e.g. PROSITE[3] and PRATT[12]). On the other hand, there are many tools which directly predict aspects of protein function without reference to previously known proteins. For example, protein functional categories can be predicted to some extent with neural networks trained on sequence-derived properties[11]. A number of groups have developed methods to predict the subcellular compartment(s) in which a protein does its job[21, 7]. Subcellular location is important low-resolution functional information which can guide experimental work and speed progress.

In eukaryotic cells, the nucleus is one such compartment of great importance. Most control mechanisms converge on the nucleus, where the paradigm states that genes (which usually encode proteins) are turned on and off in response to changing conditions. Proteins are manufactured on ribosomes outside the nucleus, and must get back to the nucleus if they are needed there. Most are actively transported in an energy-consuming process through the so-called *nuclear pore complex*[16]. Import through the nuclear pore is controlled by proteins called importins which bind to short recognition sequences on the cargo proteins called nuclear localisation signals (NLS). A number of NLS subtypes have been identified experimentally[5], and regular expressions (REs, basically the same thing as PROSITE patterns) have been developed for them[6].

Clearly, one way to determine nuclear localisation for a protein is to scan its sequence against a library of known NLS motifs[6]. This will not be sufficient, of course, for proteins which do not contain these particular motifs but perhaps contain some other kind of NLS which has not yet been discovered. Here we need a more generalised approach like, for example, neural network-based predictors which take amino acid frequencies as input[23, 20]. Predictors which use only this global information perform less well than pattern-based methods, however. Hybrids methods using both global and pattern information, such as PSORT II[21], may perform better. PSORT II may be disadvantaged, however, because it uses only a small set of NLS REs.

Here, we make hybrid predictors that are not restricted to a predefined set of NLS patterns. Our approach is to use genetic programming[14] (GP) to evolve classifiers which can, if required, simultaneously consider global information (e.g. amino acid frequencies) and local sequence motifs. These motifs are not predefined, but are evolved at the same time as the other classification rules. Koza *et al* presented the first GP system for evolving sequence classifiers for subcellular location[15]. They used a low-level automaton-like implementation, and because of this, perhaps, they did not present, analyse or distribute the evolved classifiers. By choosing higher-level implementations for pattern matching in GP we make the eventual post-analysis more straightforward.

In this paper, we have benchmarked a number of implementation variations: including two different genetic programming systems, and have experimented with predicted secondary structure as an extra input. Simple combinations of multiple predictors are shown to improve performance. Our results are competitive with both PSORT II and PredictNLS[6]. A web interface is provided for the community and work is continuing to propose biological explanations and possibly novel NLSs for proteins that our methods confidently and uniquely predict as nuclear. `http://www.sbc.su.se/∼maccallr/nucpred`

2 Methods

2.1 Data for Supervised Learning

Swiss-Prot[2] is an expert-maintained protein "knowledgebase" and is the source of our training data. We extracted two sets of proteins (and their sequences) from Swiss-Prot release 40 (8 Jan 2003) according to the following definitions:

nuclear – subcellular location annotation matches 'nuclear' (4135 proteins)

non-nuclear – eukaryotic proteins where the subcellular location annotation contains 'cytoplasmic', 'mitochondrial', 'chloroplast', 'peroxisomal', 'extracellular', 'endoplasmic reticulum' but not 'nuclear' (6055 proteins)

In both sets above, proteins are excluded where the subcellular location is annotated as 'by similarity', 'probable', 'possible', 'potential', or 'predicted'. Because we do not discard proteins with multiple locations (as some others have done), our two sets should strictly be described as "proteins which have a role in nucleus" and "proteins which have no role in the nucleus". Two separate non-redundant benchmark sets, A and B, were generated from these proteins, as detailed in the supplementary information (`http://www.sbc.su.se/∼maccallr/supplementary/heddad2004`). These benchmark sets are both divided into a 10-fold *cross-validation* set (A has 1368 nuclear and 1231 non-nuclear proteins; B has 760 nuclear and 1147 non-nuclear proteins) and a *final validation set* (A 456 nuclear, 412 non-nuclear; B 178 nuclear, 293 non-nuclear).

2.2 Fitness evaluation

We use a correlation coefficient called MCC (first used by Matthews in the area of computational biology[18]) as the fitness measure for our classifiers evolved by genetic programming. MCC conveniently takes into account both over-prediction and under-prediction and imbalanced data sets. It is defined as:

$$MCC = \frac{tp \times tn - fn \times fp}{\sqrt{(tn + fn)(tn + fp)(tp + fn)(tp + fp)}}$$

True positives (tp) are correctly predicted nuclear proteins, true negatives (tn) are correctly predicted non-nuclear proteins, and so on.

2.3 PerlGP Implementation

In PerlGP[17], evolved code is expanded from a tree-based genotype into a string before being evaluated with Perl's `eval()` function. The trees of each individual are built (and later, mutated) according to a grammar and are strongly typed. In this application, we want the evolved code to look like the example given in Figure 3; that is to say, the solution should be some arithmetic expression containing constants and RE matches against a protein sequence. The `matches()` function feeds the number of separate RE matches into the arithmetic expression. If the result of the expression for a given sequence is greater than zero, it is predicted/classified as nuclear, otherwise it is non-nuclear.

The grammar for the arithmetic is fairly standard, the operators are `+ - * pdiv(x,y) plog(x)`, where `pdiv` and `plog` are protected functions. The RE grammar is given in Figure 1 in PerlGP syntax, and includes character classes, nesting/grouping and quantifiers.

2.3.1 Secondary structure predictions and "dual REs".

PSIPRED[13] was used with default parameters to predict the secondary structural state for each amino acid in the protein: helix (H), strand (E for extended) and coil (C). We provided GP with a function to perform parallel matching with two REs against two sequences of the same length but with different alphabets (amino acid and predicted secondary structure). The first RE is matched against the first sequence and the positions of matching fragments are noted. The second RE is then matched against fragments of the second sequence corresponding to those found in the first, and the number of matches are returned. This can be applied in either order: sequence→structure or structure→sequence.

2.4 Linear GP Implementation

We have modified a linear GP (LGP) system[4] to the needs of this study. In LGP the program representation basically consists of variable-length sequences of instructions from an imperative programming language. Operations manipulate variables (*registers*) and constants and assign the result to a destination register, e.g., $r_0 := r_1 + 1$. In this modification, each GP individual has two parts (see Fig. 5 for example code). In the first, a set of REs is defined, and registers are loaded with the results of matching them against the protein sequence input string. In the second, the registers are manipulated with arithmetic operations. For each data instance, the final value of `r[0]` is used for classification as described in Section 2.3. Regular expression support is provided by the PCRE library [9].

2.5 GP Parameters

Both implementations were run with non-migrating populations of 2000 individuals and a tournament selection scheme. In LGP, the number of evolved REs was 10, and the maximum program length (second part) was 50 instructions. Full details of parameters are available in the supplementary web material (URL given in Section 2.1).

```
$F{REGEXP} = [ '{REGEXP}{REGEXP}', '({REGEXP}){QUANT}'
               '{AAS}{REPEAT}', '[{HAT}{AAS}]{REPEAT}' ];
$F{AAS}    = [ '{AA}', '{AAS}{AAS}' ];

$T{REGEXP} = [ '.' ]; $T{HAT} = [ '', '^' ]; $T{REPEAT} = [ '', '+' ];
$T{QUANT}  = [ '{1,1}', '{1,3}', ... '{3,8}' ];
$T{AAS}    = $T{AA} = [ 'A', 'C', 'D', 'E', 'F', ... 'V', 'W', 'Y' ];
```

Figure 1: Grammar definition (simplified and abbreviated) for the generation of regular expressions in the PerlGP implementation. The `F` and `T` hash tables contains function (branching) and terminal (non-branching) nodes respectively. Note that this is very close to the Backus-Naur format for grammars, the main difference being that ⟨*type*⟩ is written here as `{TYPE}` and ::= becomes =.

3 Results and Discussion

3.1 PerlGP

3.1.1 Strategies.

Three approaches for generating GP individuals were tested:

- *Strategy 1* evolves expressions containing only trivial REs which count the occurrences of single amino acids.
- *Strategy 2* evolves expressions containing full REs for amino acid sequence.
- *Strategy 3* as Strategy 2 plus "dual REs" using an additional input of predicted secondary structure.

Because the size of the search space differs between strategies, we tried to allow sufficient run-time in each case so that we could characterise the learning (and over-training) dynamics. Figure 2 shows the final fitnesses for a set of 10-fold cross-validation runs with different run-times using data from Set *A*. Relying only on amino-acid composition, strategy 1 performs poorly compared to the approaches able to detect local sequence motifs. Strategy 1 also exhibited over-training after around 40 hours. Strategies 2 and 3 did show a gap between training and testing performance but we conclude from Figure 2 that "pathological" over-training (where test set performance is no longer showing the same trends as the training fitness) has not yet occurred.

Since proteins are three-dimensional objects which usually interact with each other and the cellular machinery at their surfaces, we expect signal sequences to be present on the surface of proteins. Although the actual 3D structure is only available for some proteins in our data sets, the predicted secondary structure can be obtained for all (see Section 2.3). Geometric considerations lead us to hypothesise that signal motifs should be more common in coil regions than in helix and strand, where they would be less accessible (strand in particular). Strategy 3 was introduced to see if GP could take advantage of this possible source of extra information. Based on test set performance (see Fig. 2) we conclude that it does not, but many of the evolved expressions (not shown) do behave as expected and restrict the sequence pattern search to predicted coil regions. Others have recently shown surface/non-surface predictions to be of use in composition-based predictions[20].

Many of the evolved expressions from strategy 2 contain REs which broadly agree with the patterns derived for actual NLSs, in that they contain a lot of arginine (R) and lysine (K). One of the more compact solutions is shown in Figure 3. We are currently working to gain a deeper understanding of the relationship between our evolved REs and already-known NLS motifs.

3.1.2 Majority classifiers.

Perhaps one disadvantage of evolutionary computation compared to more deterministic opti-

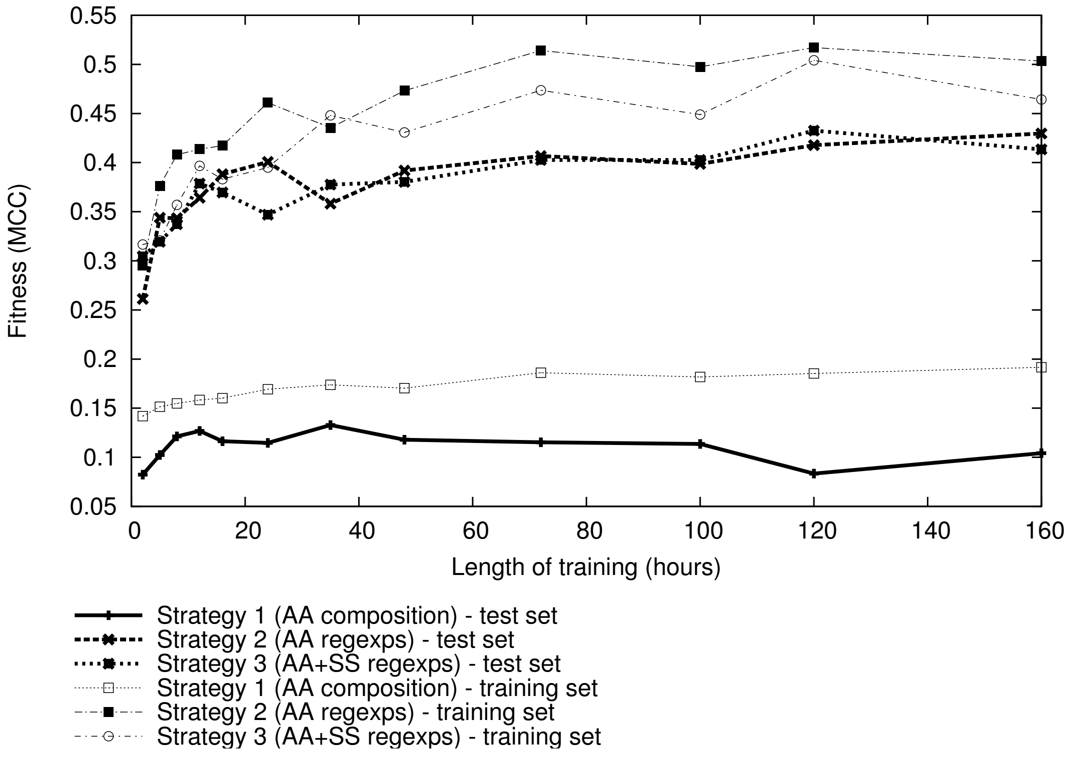

Figure 2: Best-of-tournament training fitness and testing performance obtained at the end of separate 10-fold cross-validation experiments (dataset A) with different run-times. Test set performance is calculated from the pooled "left-out" data, and individuals were selected on the basis of training fitness only. Training fitnesses are calculated as the mean training fitness over the 10 cross-validation runs.

misation strategies is that runs often fall into local minima and explore mutually exclusive regions of solution space. This can often be turned into an advantage however by combining the "knowledge" of separately evolved classifiers into one "meta-classifier".

A new set of 10×10-fold cross-validation runs using strategy 2 were performed. Dataset B is now used, after we found that dataset A is liable to over-estimate performance (see Section 2.1 and supplementary information). The smaller dataset and a few other optimisations mean that training for 36 hours is now sufficient. The mean MCC on the test set proteins is now just 0.29. From past experience we suspect that majority voting by different classifiers will produce better results. For each test-set partition of the cross-validation set, we can do a majority vote using the 10 different evolved predictors which were not trained on that data. This means that for each protein, 5 or more of the 10 classifiers must predict "nuclear" in order to classify a protein as such. Using this approach, the MCC for goes up to 0.38 from a mean of 0.29 for the individual predictors.

3.1.3 Final validation set.

If we want to combine all 100 evolved predictors then we must use the final validation set (see Section 2.1) because none of these sequences (or their close relatives) have been used in the cross-validation training runs. We restarted each of the 100 populations and performed a single tournament using the entire cross-validation set as the "training set" and chose the best-of-tournament individuals (on training data) for the final validation set performance calculations. The mean MCC for individual predictors on this set is 0.33. When 50 or more predictors out of 100 vote nuclear then the MCC again rises to 0.47. Of course, thresholds other than

```
$nuclear = (((((matches($seq, qr/[K][R]/)
          + matches($seq, qr/([KR])([KR]+)[RKHM]/))
          + matches($seq, qr/([KRH])([RQ]+)[RKYM]+/))
          + matches($seq, qr/(([RKHD]){1,4}([R])[FKMT]+)[KRH]/))
          - matches($seq, qr/AKV/))
          - matches($seq, qr/[^W]+/));
```

Figure 3: Example of an evolved nuclear localisation predictor. If the result is greater than zero, the protein with sequence `$seq` is predicted to be nuclear.

50 can be used, and this is described below.

3.1.4 Comparison with other methods.

We give the name "NucPred" to the strategy 2 approach, and define the *NucPred score* as the fraction of predictors (usually 100) which vote nuclear. In Figure 4 we show the performance of NucPred, in terms of sensitivity and specificity, at different score thresholds and compare this with the performance of the two most widely used existing methods: PSORT II and PredictNLS. On this data (validation set B), the NucPred approach does a little better than all other at all sensitivities. However, previous results trained and validated on dataset A showed NucPred to be equal to PSORT II and slightly worse than PredictNLS, so the exact ranking of methods is difficult. The best performance at high specificity on validation set B is obtained using combinations of NucPred and PredictNLS (see Figure 4 for details).

3.2 Linear GP

Our work evolving Perl-based classifiers has shown that string matching code can be produced at a high level (using REs), providing an alternative to low level approaches where the matching mechanism is evolved as part of the solution[15, 10]. This should reduce search space complexity and solutions will also be more human-readable. However, the PerlGP solutions often contain duplicate REs which are wastefully matched more than once against the same sequence. With our linear GP (LGP) system[4], it is possible to perform the RE matches once and use their results many times during calculation of the final output (see Section 2.4 and Fig. 5 for further details). Additionally, no crossover is allowed between REs in the individuals, so duplications of complex REs are highly improbable. The LGP system has a mechanism for removing ineffective code prior to evaluation[4]. This is particularly useful because it automatically identifies which of the evolved REs are not used, and does not perform the (relatively expensive) match for them. We were also interested in measuring the performance of simpler forms of RE than were used with the Perl system.

Cross-validation set B was used to generate LGP classifiers (500 generations). Combinations of two strategies were tested: building REs with and without character sets (a character set, e.g. "`[FWV]`", means match either F,W or V); and loading the registers with the number of RE matches (multiple) or simply zero or one signifying non-match or match (boolean).

Classification performance figures for the final validation set are shown in Tables 1&2. The LGP performance is very close to the PerlGP results for both individual predictors (mean MCC = 0.33) and majority voting (MCC = 0.46). The sensitivity-specificity curves (not shown) for the best performing LGP configurations are also similar to the one in Figure 4. The relative performances of the RE strategies were interesting: individuals using REs without character sets (essentially just short sequences of amino acids) performed almost identically to those with character sets. Boolean matching in general does not perform well although, as expected, the use of character sets helps. Boolean matching may also encourage more "interest-

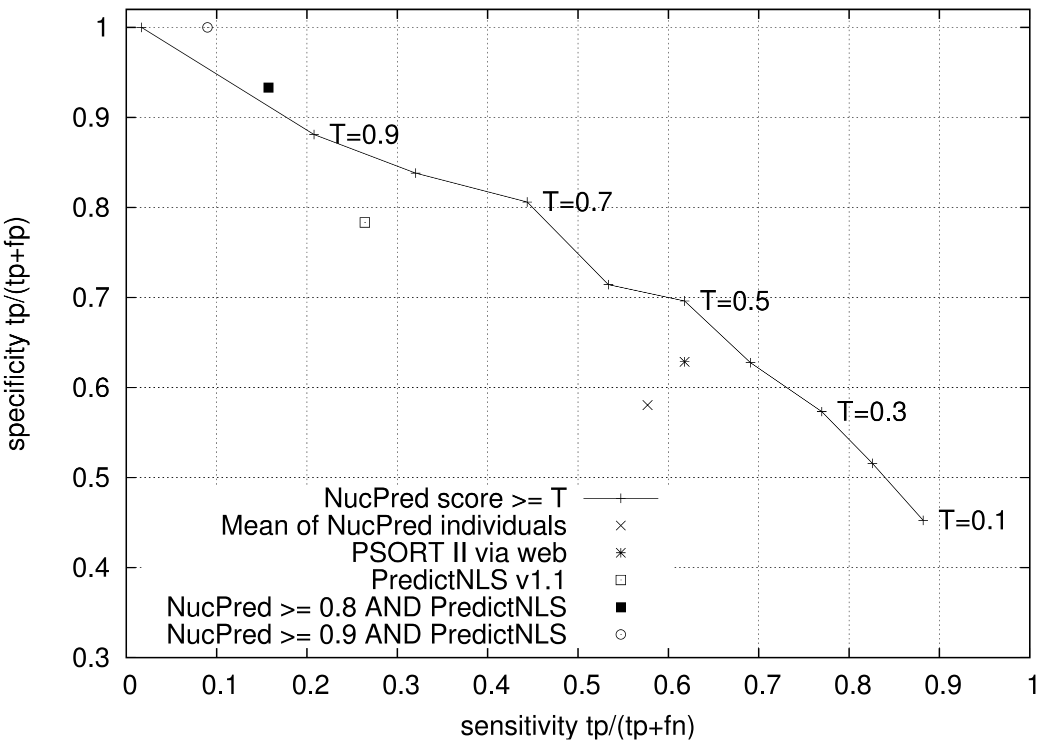

Figure 4: Performance of NucPred, PSORT II and PredictNLS on validation set B (471 sequences). Specificity and sensitivity are shown for different thresholds of NucPred score (fraction of predictors voting “nuclear”).

ing” REs to evolve (data not shown). An example classifier is shown in Figure 5.

4 Final Remarks

We have, with relative ease, evolved classifiers which use global and local sequence information and which are least as good as other published methods. Direct comparison is always difficult because of differences in test set construction. Further improvements can be gained by building a multi-location predictor[21, 7] since, for example, proteins predicted to be integral membrane proteins are unlikely to be nuclear.

In both of the GP implementations we used, the RE matching is performed by efficient C-coded routines. We have not performed runtime profiling, but we suspect that the RE matching is the principal drain on CPU in our two approaches and for the evolution of REs in general. Interestingly, the results from the “no character set” LGP runs suggest that this classification problem may not require RE matching at all – it may be sufficient (and much more efficient) to pre-calculate all the frequencies of all k-tuplets ($1 \leq k \leq 3$) and store them in hash tables. Structural biology, however, suggests that longer sequences are involved in the nuclear import recognition process[8].

4.0.1 Acknowledgements

We thank Andrea Krings for the PredictNLS data, Arne Elofsson and Erik Granseth for helpful discussion, the PDC for cluster administration, the Swedish Foundation for Strategic Research (SBC funding) and the Knut and Alice Wallenberg Foundation (cluster funding).

References

[1] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. H. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and

```
void gp(double *r, char *seq)
{
  r[3] = match("SA", seq);
  r[4] = match("[RK]R", seq);
  r[5] = match("[P]GP", seq);
  r[6] = match("[FWV]", seq);
  r[10] = match("[QS]", seq);

  r[3] = r[3] + r[5];
  r[8] = r[6] + r[3];
  r[0] = r[4] - r[8];
  r[0] = r[10] + r[0];
  r[0] = r[4] * r[0];
}
```

Figure 5: Example best-of-run individual from linear GP using character sets and multiple matching. Only effective REs and register operations are shown (5 REs were not used).

Table 1: Mean *MCC*, sensitivity and specificity for 100 individuals measured on validation set *B*. Sensitivity and specificity are defined in Figure 4.

char. sets	matching	*MCC*	sens.	spec.
no	boolean	0.21	0.49	0.56
yes	boolean	0.25	0.40	0.56
no	multiple	0.31	0.55	0.60
yes	multiple	0.33	0.59	0.57

Table 2: Majority voting by 100 individuals on validation set *B*.

char. sets	matching	*MCC*	sens.	spec.
no	boolean	0.26	0.32	0.63
yes	boolean	0.32	0.48	0.61
no	multiple	0.45	0.59	0.70
yes	multiple	0.46	0.65	0.66

PSI-BLAST: a new generation of protein database search programs. *Nucleic Acid Research*, 25:3389–3402, 1997.

[2] A. Bairoch and R. Apweller. The SWISS-PROT protein sequence data bank and its supplement TrEMBL. *Nucleic Acid Research*, 25:31–36, 1997.

[3] A. Bairoch, P. Bucher, and K. Hofmann. The PROSITE database, its status in 1997. *Nucleic Acid Research*, 25:217–221, 1997.

[4] M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE-EC*, 5:17–26, February 2001.

[5] D. Christophe, C. Christophe-Hobertus, and B. Pichon. Nuclear targeting of proteins: how many different signals. *CS*, 12(5):337–341, May 2000.

[6] M. Cokol, R. Nair, and B. Rost. Finding nuclear localization signals. *EMBO Rep*, 1(5):411–415, Nov 2000.

[7] O. Emanuelsson, H. Nielsen, S. Brunak, and G. von Heijne. Predicting subcellular localization of proteins based on their N-terminal amino acid sequence. *Journal of Molecular Biology*, 300(4):1005–1016, Jul 2000.

[8] M. R. M. Fontes, T. Teh, D. Jans, R. I. Brinkworth, and B. Kobe. Structural basis for the specificity of bipartite nuclear localization sequence binding by importin-alpha. *Journal of Biological Chemistry*, 278(30):27981–27987, Jul 2003.

[9] P. Hazel. PCRE - Perl Compatible Regular Expressions library. www.pcre.org.

[10] D. Howard and K. Benson. Promoter prediction with a GP-automaton. In *Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM*, volume 2611 of *LNCS*, pages 44–53,

University of Essex, England, UK, 14-16 April 2003. Springer-Verlag.

[11] L. J. Jensen, R. Gupta, H.-H Staerfeldt, and S. Brunak. Prediction of human protein function according to Gene Ontology categories. *Bioinformatics*, 19(5):635–642, Mar 2003.

[12] I. Jonassen, J. F. Collins, and D. G. Higgins. Finding flexible patterns in unaligned protein sequences. *Protein Science*, 4(8):1587–1595, Aug 1995.

[13] D. T. Jones. Protein secondary structure prediction based on position- specific scoring matrices. *Journal of Molecular Biology*, 292:195–202, 1999.

[14] J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT press, Cambridge, MA, 1992.

[15] J. R. Koza, F. H. Bennett III, and D. Andre. Using programmatic motifs and genetic programming to classify protein sequences as to cellular location. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*, pages 437–447, Berlin, 1998. Springer. LNCS 1447.

[16] I. G. Macara. Transport into and out of the nucleus. *Microbiology and Molecular Biology Reviews*, 65(4):570–594, Dec 2001.

[17] R. M. MacCallum. Introducing a Perl Genetic Programming System: and Can Meta-evolution Solve the Bloat Problem? In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 369–378, 2003.

[18] B. W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochem. Biophys. Acta*, 405:442–451, 1975.

[19] N. J. Mulder et al. The InterPro Database, 2003 brings increased coverage and new features. *Nucleic Acid Research*, 31(1):315–318, Jan 2003.

[20] R. Nair and B. Rost. Better prediction of sub-cellular localization by combining evolutionary and structural information. *Proteins: Structure, Function and Genetics*, 53(4):917–930, Dec 2003.

[21] K. Nakai and P. Horton. PSORT: a program for detecting sorting signals in proteins and predicting their subcellular localization. *Trends in Biochemical Science*, 24(1):34–36, Jan 1999.

[22] W. R. Pearson. Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods in Enzymology*, 183:63–98, 1990.

[23] A. Reinhardt and T. Hubbard. Using neural networks for prediction of the subcellular location of proteins. *Nucleic Acid Research*, 26(9):2230–2236, May 1998.