Using Evolutionary Algorithms in the Design of Protein Fingerprints

Björn Olsson
Deptartment of Computer Science, University of Skövde
Box 408, 541 28 Skövde, Sweden
Phone: +46-500-464716
Fax: +46-500-464725
Email: bjorne@ida.his.se

Abstract

This paper shows how Evolutionary Algorithms (EAs) are used as components in a system for design of protein fingerprints. The system is used for automated mining of data from protein sequence databases, with the purpose of deriving protein family fingerprints. The fingerprints are expressed as patterns, which can be used for recognition of sequences belonging to specific protein families. The system constructs candidate patterns by analyzing multiple sequence alignments, and selecting pattern elements corresponding to evolutionary conserved positions. Since most candidate patterns are too specific, we use stochastic search algorithms for generalization of the candidate patterns. In a previous version of the system a hill-climbing algorithm was used. In this paper we show how results can be substantially improved by using EAs for this task. We also compare a "standard" EA with a host-parasite EA, and show that it can significantly reduce the number of evaluations.

1 Protein Sequence Patterns

Much of the work in Bioinformatics [5] is focused on building systems for analysis of bio-sequence data [7], such as DNA, RNA, or protein sequences. In this context, a *pattern* is a representation of a regular expression which should match all members of some class of sequences. A protein sequence pattern, therefore, represents a regular expression which matches, for example, all sequences of a protein family, or all sequences containing a particular motif or set of motifs. As an example, figure 1 shows a pattern taken from the PROSITE Dictionary of Protein Sites and Patterns [4], together with a short subsequence of human insulin which matches the pattern (the complete insulin sequence consists of 110 amino acids). When this example pattern is used to scan the SWISS-PROT database [3] - which contains approximately $7*10^5$ protein sequences - it will match 157 of the 162 sequences which are known to belong to the insulin family. In addition, it will match one non-insulin sequence.

C-C-{P}-x(2)-C-[STDBEKPI]-x(3)-[LIVMFS]-x(3)-C

C C T S I C S L Y Q L E N Y C

Figure 1: PROSITE pattern for the insulin family, and a short subsequence of human insulin which matches the pattern.

Proteins are represented in SWISS-PROT [3] by their amino acid sequence as strings of characters, using a 23-letter alphabet, where most one-letter codes correspond to a given amino acid[1]. This public domain database, administered by the Swiss Institute of Bioinformatics (SIB), contains 76,803 protein sequences with an average length of 363 characters (SWISS-PROT release 36 and updates until 18 December 1998). These proteins can be grouped into families, based on evolutionary and functional relatedness, and the sequences belonging to a family can usually be characterized by the presence of a small number of amino acids which are important for the protein's function, and therefore evolutionary conserved in the family. This pattern of conserved amino acids can be called a "fingerprint" [1], since it can be used for identi-

[1] Apart from 20 characters for the amino acids, there are also three characters corresponding to two or more amino acids each. For further details, see the SWISS-PROT manual [2].

fying sequences belonging to the family. A fingerprint usually covers one or more of the functionally important motifs found in the sequences of the family.

PROSITE [4], which is also administered by SIB, is a collection of 1,359 patterns (release 15 and updates until 18 December 1998). PROSITE is cross-referenced to SWISS-PROT, so that each pattern is accompanied by the results of scanning SWISS-PROT with the pattern, showing all of the true and false hits obtained. An important usage of PROSITE is for analysis of newly sequenced proteins. A new sequence can be scanned for occurrences of the patterns stored in PROSITE, and thereby identified as belonging to a particular family. Of course, if PROSITE is to be reliable for this form of usage, the patterns should ideally be diagnostic, i.e. each pattern should match every sequence belonging to the family, and reject all non-family sequences.

The PROSITE patterns are built manually [4] from sequence alignments, which is obviously an inefficient method - especially since new sequences are added to SWISS-PROT at a rapid rate, so that patterns must be updated in order to maintain their discriminatory power. On the other hand, there are also advantages of building patterns manually. Detailed biological knowledge can be used in the process - both general biochemistry and more specific information about important sites in the protein family. In order to achieve an automated system for pattern construction, such knowledge must either be accounted for in the algorithms, or compensated for by efficient search algorithms. As will be seen in the following, we are using both approaches, i.e. we are building biological knowledge into the algorithm, as well as using efficient search algorithms for parts of the pattern construction process.

The general pattern syntax can be described as

$$E_1 - x(i_1, j_1) - E_2 - x(i_2, j_2) - ... - E_n \qquad (1)$$

where E_k is an *element*, and $x(i_k, j_k)$ is a *wild-card region*. An element specifies either a single amino acid (e.g. C) or a set of alternative amino acids (such as $[LIVMFS]$), which the sequence must contain. In addition, an element enclosed by curly brackets, e.g. $\{P\}$, specifies amino acids which the sequence must not have at the given position. A wild-card specifies an arbitrary stretch of amino acids, the length of which must be at least i_k and at most j_k. For $i_k = j_k$ the shorter notation is $x(i_k)$, and $x(1)$ can be written x. For convenience, all zero-length wild-cards $x(0)$ are omitted from the pattern.

The example pattern which is shown in figure 1 specifies that in order to match this pattern the sequence must contain a subsequence beginning with C, followed by another C, followed by any character except P, followed by two arbitrary characters, followed by C, followed by either S, T, D, B, E, K, P, or I, followed by three arbitrary characters, followed by either L, I, V, M, F, or S, followed by three arbitrary characters, followed by C.

One advantage of patterns is that they are very simple objects. In an implementation, the pattern can easily be converted into the implementation language's (Perl, in our case) syntax of regular expressions, and a database can very rapidly be scanned for sequences matching the pattern. Of course, there is also a price to pay for this simplicity. Since patterns have limited descriptive power [6], it is difficult to find diagnostic patterns for families where the degree of sequence similarity is very low. For this reason, PROSITE uses profiles [4] as a complement to patterns for some of the most difficult families.

2 An Automated System for Pattern Construction

In previous work [12] we presented a system for automated pattern construction and evaluated its performance on 439 protein families. The system uses sequence alignments as input, analyses them to identify columns that seem to correspond to evolutionary conserved positions, and generates candidate patterns from these columns. The result is usually a pattern which is specific for the family, but also over-specific in the sense that it rejects some of the family members. As a final stage the pattern must therefore be generalized, so that its sensitivity is improved. The generalization phase can be formulated as a search problem: given a specific pattern, there is a space of possible generalizations which can be searched more or less efficiently, depending on the choice of search algorithm.

Very briefly described, the system constructs a candidate pattern for a particular protein family using the following four steps (see also figure 2):

i) For each column of the multiple sequence alignment, calculate the entropy of the column, based on an estimation of the probability of observing each of the amino acids in the column. In the estimation of amino acid probabilities we combine the amino acid frequencies observed in the column with Dirichlet mixture densities [18], which encode prior information about typical amino acid distributions. Using this form of prior has been shown to improve the generalization capacity of statistical models such as hidden Markov models [9] and phylogenetic trees [17]. We adapted its use for design of patterns in [10] and [12].

ii) Choose the column with lowest entropy, and make an initial pattern consisting of a single element which includes the symbols from this column.

iii) In order of increasing entropy, incrementally add elements corresponding to additional columns, and complement these with wild-card elements which reflect the relative positions of the columns. Measure the specificity of each candidate pattern by scanning the database and counting the number of false matches. The process of adding elements to the pattern continues until specificity is 1.0, i.e. until the pattern does not match any non-family sequences in the database.

iv) Measure the sensitivity of the pattern by scanning the set of family sequences. If any family sequence is not matched: generalize the pattern.

As input to the pattern construction process, our system uses pre-derived multiple sequence alignments from Pfam [19], which contains alignments for 1,390 families (release 3.3). For each family, Pfam contains a "seed alignment", which is an accurate hand edited alignment of a subset of the members of the sequences of the family. In addition, Pfam contains "full alignments", which are made automatically from hidden Markov models, which are built from the seed alignments. Since the full alignments often contain errors, we use the seed alignments as input to our system. The drawback is that there is no guarantee that a pattern derived from an alignment of a subset of the family will match all family members, and the majority of patterns consequently need generalization in order to become diagnostic.

```
FLAV_ANASP   KIGLFYGTQTGKTESVAEII
FLAV_SYNP7   KIGLFYGTQTGVTQTIAESI
FLAV_SYNY3   KIGLFYGTQTGNTETIAELI
FLAV_SYNP2   KIGLFFGTQTGNTEELAQAI
FLAV_ECOLI   ITGIFFGSDTGNTENIAKMI
FLAW_ECOLI   NMGLFYGSSTCYTEMAAEKI
FLAV_AZOCH   KIGLFFGSNTGKTRKVAKSI
FLAV_ENTAG   TIGIFFGSDTGQTRKVAKLI
FLAV_KLEPN   NIGIFFGTDTGKTRKIAKMI
FLAV_CHOCR   KIGIFFSTSTGNTTEVADFI
FLAV_CLOBE   .MKIVYWSGTGNTEKMAELI
FLAV_MEGEL   MVEIVYWSGTGNTEAMANEI
FLAV_DESDE   KVLIVFGSSTGNTESIAQKL
FLAV_DESGI   KALIVYGSTTGNTEGVAEAI
FLAW_DESGI   KALVVFGSTTGNTETVAEVV
FLAV_DESSA   KSLIVYGSTTGNTETAAEYV
FLAV_DESVH   KALIVYGSTTGNTEYTAETI
FLAV_CLOAB   KISILYSSKTGKTERVAKLI
FLAW_RHOCA   .MLIFYVSAYAATAHVAQAI
             12345678901234567890
```

Column	Entropy	Pattern
13	0.16	T
17	0.18	T-x(3)-A
10	0.63	[TY]-x(2)-T-x(3)-A
11	0.96	[TY]-[ACG]-x(2)-T-x(3)-A
20	1.14	[TY]-[ACG]-x(2)-T-x(3)-A-x(2)-[ILV]

Figure 2: Upper: Small portion of multiple alignment. Lower: Subset of corresponding incrementally built candidate patterns, and for each pattern, the entropy of the alignment column corresponding to the most recently added pattern element.

3 An Evolutionary Algorithm for Pattern Generalization

Given the pattern syntax in (1) we can define pattern generalization as a process which may use the following three generalization operators:

Op 1: Generalize an individual element by adding a character.

Op 2: Generalize the pattern by deleting an element.

Op 3: Generalize a wild-card region by decreasing the lower bound on the length (or increasing the upper bound).

In previous work [13] [12] we used a hill-climbing algorithm which randomly applied these three operators. In each step, the hill-climber chooses randomly one of the generalization operators and applies it at a random position in the pattern. It then replaces the current pattern P with the generalized pattern P' if and only if P' has improved sensitivity (without any deterioration in specificity). Sensitivity and specificity are checked by scanning SWISS-PROT and counting the number of family and non-family sequences (according to the PROSITE listing of known family members) which match the pattern.

Sensitivity and specificity are here defined as

$$Sens = \frac{True_{pos}}{True_{pos} + False_{neg}} \quad (2)$$

$$Spec = \frac{True_{pos}}{True_{pos} + False_{pos}} \quad (3)$$

where $True_{pos}$ is the number of family sequences matched by the pattern, $False_{neg}$ the number of family sequences not matched by the pattern, and $False_{pos}$ is the number of non-family sequences matched by the pattern. This sensitivity measure is equal to the fraction of the known family sequences which are matched by the pattern, whereas specificity equals the probability that a sequence which is

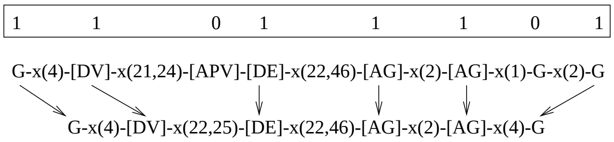

Figure 3: A bit-string chromosome represents a generalization of a given pattern, and this pattern is fitness evaluated against the sequence database.

matched by the pattern really belongs to the family.

In order to see how we can apply a Genetic Algorithm to the generalization problem, it is useful to consider a pattern of N elements. In order to achieve a bit-string representation for a GA, we restrict the generalization to a search for subsets of N, so that bit-strings represent which elements of N to include in the subset - as illustrated in figure 3. This way, the search is restricted to generalization operator *Op 2* only, i.e. to deletion of pattern elements. The advantage of such an approach is its simplicity, allowing a "standard" bit-string GA to be used. The disadvantage is the restriction it places on the search, when only element deletions are used.

However, allowing only element deletions may not be such a serious restriction as appears at first sight. To see this, consider an initial pattern containing every single element derived from the alignment, so that N contains as many elements as there are columns in the alignment. It should be clear that from this starting point, even a search procedure which uses only pattern deletions can still generate all patterns which are consistent with the multiple alignment. Naturally, we will not be using this starting point in our search, since the search space would be far too large even for a stochastic/heuristic search algorithm such as a GA. However, we will be using as starting point a pattern with "extra" elements added. Recall that in the incremental addition of pattern elements, we stopped the process as soon as the candidate pattern was specific enough to reject all non-family sequences in the database. Instead of using this candidate as starting point for GA-generalization, we will continue adding elements to create an over-specific pattern as starting point for GA-generalization.

In our initial experiment, we used the algorithm to evolve generalizations of an over-specific pattern for the *2_Hacid_DH* family. We selected a pattern with 16 elements, which gave six false negatives and no false positives. For this family, the original system finds - using hill-climbing generalization - a 10-element

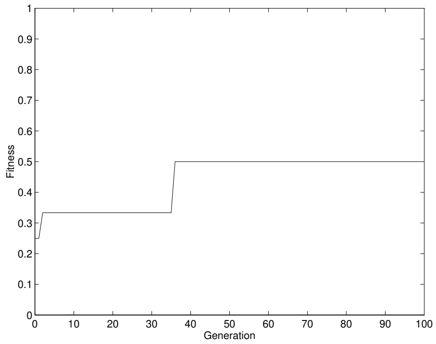

Figure 4: The population's best fitness during 100 generations. The discrete "jumps" in fitness correspond to discoveries of patterns with fewer false positives. For example, the "jump" to fitness $\simeq 0.5$ at generation 37 corresponds to discovery of a pattern with only two false negatives.

generalized pattern which only gives five false negatives. We used the GA to search for generalizations to the 16-element pattern, using a population of 20 bit-string chromosomes, and evaluating each chromosome by searching the whole database and assigning fitness according to

$$\frac{1}{fp(P') * 100 + fn(P') + \epsilon} \tag{4}$$

where $fp(P')$ and $fn(P')$ are the number of false positives and false negatives for the generalized pattern P' specified by the chromosome (and ϵ is a small constant to protect from division by zero). The algorithm was run for 100 generations. Figure 4 plots the population's best fitness for every generation. At generation 37, the GA discovers a pattern with only two false negatives. This pattern has a sensitivity of 0.93, to be compared with the sensitivity of 0.83 for patterns produced by the hill-climber. Both patterns have specificity 1.0.

To evaluate the GA-approach in more detail, we tested it on 20 protein families. We selected at random 20 families for which the hill-climber had found a pattern P with $fp(P) = 0$ and $fn(P) > 0$. For each family, we made a single GA-run with the same parameters as in the previous experiment on *2_Hacid_DH*. Detailed results from these 20 runs are shown in table 1. We only show the sensitivity, since all resulting patterns retained a specificity of 1.0.

As can be seen from these results, a GA may be a powerful tool for finding improved generalizations. The average sensitivity is considerably higher than that found

Family	Sensitivity			
	PRO	HC	EA	HP
14-3-3	1.000	0.960	1.000	1.000
2_Hacid_DH	0.767	0.833	0.933	0.867
6PGD	0.968	0.806	0.935	0.935
7_tm2	0.925	0.863	0.941	0.941
ADP_glu_Plase	1.000	0.889	0.926	0.963
ALAD	1.000	0.962	0.962	0.962
ATP-gua_Ptrans	1.000	0.839	1.000	1.000
ATP-synt	0.814	0.930	1.000	1.000
ATP-synt_A	0.980	0.928	0.979	0.948
Acyl-CoA_dh	0.848	0.758	0.848	0.879
Bac_DNA_binding	0.972	0.972	1.000	1.000
Bcl-2	1.000	0.619	0.667	0.667
Bet_v_I	0.914	0.829	1.000	1.000
Bowman-Birk_leg	1.000	0.917	1.000	1.000
COX1	0.971	0.921	0.978	0.989
COX2	0.971	0.930	0.971	0.965
Calc_CGRP_IAPP	1.000	0.964	1.000	1.000
Cu-oxidase	0.956	0.822	0.889	0.822
DAG_PE-bind	0.952	0.964	0.952	0.964
DNA_pol_A	0.926	0.889	1.000	1.000
Average	0.948	0.880	0.948	0.945

Table 1: Sensitivity results for 20 protein families. Comparison of the original PROSITE pattern (PRO), the best pattern found by the hillclimber (HC), the evolutionary algorithm (EA), and the host-parasite algorithm (HP).

by the hill-climber, and close to that of PROSITE's patterns. The main problem the GA faces is the fact that every individual is evaluated by testing the pattern against the whole database, which is a very costly operation. Using a population size of 20 means that only 2,000 candidate patterns are evaluated during a run, but given that the database contains over $7 * 10^5$ sequences, this corresponds to evaluation of approximately $1.4 * 10^8$ pattern-sequence pairs.

4 A Host-Parasite Algorithm for Pattern Generalization

Host-parasite algorithms were introduced in [8], and have since then been investigated by several authors, for example in [16], [14], and [11]. In the relatively small body of literature on host-parasite algorithms almost exclusively artificial problems are used, such as game-playing and sorting networks design, or problems designed specifically to test properties of the algorithms. The only example of a real-world application seems to be the drug design problem mentioned in [15], where a host-parasite algorithm was used to design antiviral drugs to overcome viral resistance.

Algorithm: Host-Parasite Algorithm
{
  $t := 0$
  exec initialize(H_t)
  exec initialize(P_t)
  while (termination_criterion == 0) **do** {
    exec evaluate(H_t,P_t)
    exec evaluate(P_t,H_t)
    H_{t+1} := reproduce(H_t)
    P_{t+1} := reproduce(P_t)
    $t = t + 1$
  }
}

Figure 5: Pseudo code of a host-parasite algorithm.

The central idea behind host-parasite algorithms is to coevolve a population of candidate solutions and a population of fitness cases. The classical example in [8] was to coevolve sorting networks and input sequences, letting fitness of a sorting network depend on the number of correctly sorted input sequences, and letting the fitness of a set of input sequences depend on the number of networks which fail to sort these sequences.

An illustration of the host-parasite approach is given in the pseudo-code of figure 5. The population of hosts H contains candidate solutions, while the population of parasites P contains problem instances. Evaluation of a host is done by testing it on all (or a sample) of the parasites, and setting its fitness to the number of parasites which it solves correctly. Conversely, the fitness of each parasite is set to the number of hosts which fails to solve the problem instance(s) which the parasite represents.

In the current application we use a host-parasite algorithm by letting hosts represent candidate pattern generalizations (as in the previous chapter), and letting parasites represent sets of protein sequences. Figure 6 shows an overview of how host-parasite coevolution is applied to the pattern generalisation problem. The chromosome representation for hosts is exactly the same as that used in the non-coevolutionary GA. In the parasite population, each chromosome is a vector of 500 integers, representing database entry numbers (which identify sequences in the database).

Figure 6 shows how fitness evaluation is done: *i)* A

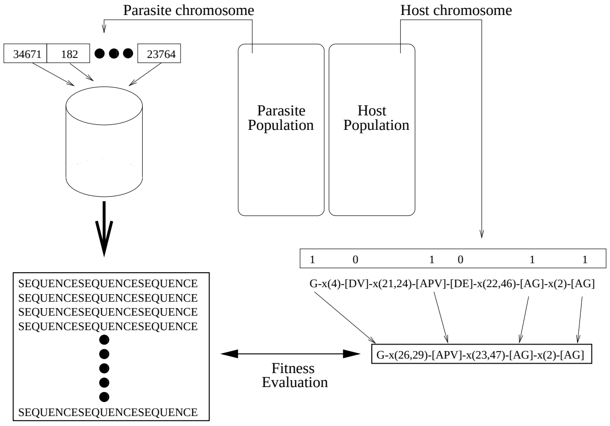

Figure 6: Overview of host-parasite evolution of patterns and sets of test sequences. See the text for details.

chromosome from the parasite population is replaced by the set of sequences that its database indices (genes) identify, and sequences belonging to the family are removed from this set. *ii)* A host chromosome is replaced by the generalised pattern that its bitstring represents. *iii)* The pattern is tested on the sequences, and a record is made of the number of sequences which match the pattern.

Every host-parasite pair is tested against each other, and each parasite's fitness is set to the total number of times that any of its sequences is matched by any pattern (excluding multiple hits by the same pattern on the same sequence). Hosts' fitnesse values are set according to (4). In order to determine the number of false positives, each pattern is tested on the set of family sequences, in addition to the parasite-evolved sequence sets. To monitor the real progress made, we take the fittest pattern from each generation, and test it against the whole database.

As in the previous experiment on evolving patterns with a non-coevolutionary GA, both populations contain 20 chromosomes. This means that $20 * 20 * 500 = 2 * 10^5$ pattern-sequence pairs are tested in each generation, which gives a total of $2 * 10^7$ pairs during a run of 100 generations. Recall that the standard GA in the previous chapter tested a total of approximately $1.4 * 10^8$ pattern-sequence pairs in a run. In other words, the standard GA tests seven times as many pairs during a run as the host-parasite algorithm. The results presented in table 1 show that despite this reduction in computation time spent on fitness evaluation, the host-parasite algorithm still acheives almost the same average sensitivity as the standard GA.

5 Conclusions and Future Work

We have shown how EAs can be used for pattern generalization to improve the efficiency of this crucial part of the pattern design process. Our results are quite preliminary, and must be extended to many more protein families before more certain conclusions can be drawn. So far we have restricted the use of the EAs to families where the pattern had perfect specificity before generalization, since the fitness function is designed for these cases. It remains to be seen if a modified fitness function, which is less biased against false positives, will be needed for patterns with sub-optimal specificity.

Since the EA takes such a long time to run, we are looking for ways of reducing the amount of computation. The host-parasite algorithm is an example of this, and it seems to reliably produce good results substantially faster than the "standard" EA. We have not yet explored possibilities of reducing the number of fitness cases even further - either using refined versions of the host-parasite algorithm, or using other alternative approaches.

References

[1] T.K. Attwood, M.E. Beck, A.J. Bleasby, K. Degtyarenko, and D.J. Parry Smith. Progress with the PRINTS protein fingerprint database. *Nucleic Acids Research*, 24(1):182–8, 1996.

[2] A. Bairoch. The SWISS-PROT protein sequence data bank user manual (release 36), November 1998.

[3] A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence data bank and its supplement TREMBL in 1998. *Nucleic Acids Research*, 26:38–42, 1998.

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

[5] A. Baxevanis and B.F.F. Ouellette. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*. John Wiley & Sons, 1998.

[6] A. Brāzma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5:279–305, 1998.

[7] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic*

Models of Proteins and Nucleic Acids. Cambridge University Press, 1998.

[8] D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, editors, *Artificial Life II: Proceedings of the Second Artificial Life Workshop*, pages 313–324. Addison-Wesley, 1992.

[9] R. Hughey and A. Krogh. Hidden markov models for sequence analysis: extension and analysis of the basic method. *Computer Applications in the Biosciences*, 12(2):95–107, 1996.

[10] K. Laurio. Probabilistic modeling of protein families. Master's thesis, University of Skövde, Sweden, 1997.

[11] B. Olsson. Evaluation of a simple host-parasite genetic algorithm. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 53–62. Springer-Verlag, 1998.

[12] B. Olsson and K. Laurio. Discovery of diagnostic patterns from protein sequence databases. In *Proceedings of PKDD98 - The 2nd European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 167–175. Springer-Verlag, 1998.

[13] B. Olsson, K. Laurio, A. Mandal, D. Lundh, and A. Narayanan. Automated discovery and refinement of diagnostic sequence patterns. In *CISM School-Workshop in Computational Biology*, Udine, Italy, June 10-19, 1998.

[14] J. Paredis. Coevolving cellular automata: Be aware of the red queen. In T. Bäck, editor, *Proceedings of the Seventh International Confonference on Genetic Algorithms (ICGA97).* Morgan Kaufmann Publishers, 1997.

[15] C.D. Rosin. *Coevolutionary Search Among Adversaries.* PhD thesis, University of California, San Diego, 1997.

[16] C.D. Rosin and R.K. Belew. Methods for competitive co-evolution: Finding opponents worth beating. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95).* Morgan Kaufmann Publishers, 1995.

[17] K. Sjölander. Bayesian evolutionary tree estimation. *Mathematical Modelling And Scientific Computing (Special Issue: Proceedings of the Eleventh International Conference on Mathematical and Computer Modelling and Scientific Computing)*, 8, 1997.

[18] K. Sjölander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I.S. Mian, and D. Haussler. Dirichlet mixtures: A method for improved detection of weak but significant protein sequence homology. *Computer Applications in the Biosciences*, 12(4):327–345, 1996.

[19] E.L.L. Sonnhammer, S.R. Eddy, E. Birney, A. Bateman, and R. Durbin. Pfam: Multiple sequence alignments and HMM-profiles of protein domains. *Nucleic Acids Research*, 26:320–322, 1998.