An Evolutionary Computation Approach for Detecting Repetitions in Biosequences

Adam Adamopoulos, [1] **Katerina Perdikuri,** [2]

Keywords: biological sequences, repetitions, evolutionary programming, genetic algorithms

1 Introduction.

One of the most important goals in computational molecular biology is allocating repeated patterns in nucleic or protein sequences, and identifying structural or functional motifs that are common to a set of such sequences. In this paper we introduce a new approach to detect the repetitions of fixed length in Biosequences using an Evolutionaty Computation approach. Our approach involves evolving a population of patterns in an evolutionary manner and gradually improving the fitness of the population as measured by an objective function, which measures the approximate repetitions of the patterns in the given sequence. The general attraction of the approach is the ability to detect repeated schemas, thus inferring motifs of fixed length from biosequences. Genetic Algorithms and Evolutionary Computation have been succesfully applied so far in the Multiple Molecular Sequence Alignment problem in order to identify similarities among sequences [1].

2 Methodology.

Biosequences, such as DNA and Protein sequences, can be seen as long texts over specific alphabets, encoding the genetic code of living beings. Searching for repeated sub-sequences of any length over those texts could be modeled as searching for a set of given patterns in a "text".

In our approach we consider the population of a Genetic Algorithm as the population of p words of length l. For each particular run, the population size p (i.e. the number of individuals) of each generation, as well as the length l of each one word of the population are kept constant. After establishing a population of words the population is randomly initialized. When the initialization procedure is completed all words of the population are random strings drawn from the Σ_{DNA} alphabet. The fitness f of each particular word is evaluated considering as fitness (or evaluation) function the number of approximate occurrences (repetitions) of the word in the input sequence.

The overall structure of the method is shown in Figure 1. To go from one generation to the next, children are derived from parents that are chosen by some kind of natural selection. To create a child, an operator is selected that can be a crossover (mixing the contents of the two parents) or a mutation (modifying a single parent). Each operator has a probability of being chosen. Thus the algorithm is divided in two stages. The first one is the evolutionary phase where the new population of individuals/words is generated and the searching phase where each individual is evaluated by counting its number and exact positions of occurrences.

Compared to other techniques ([2], [3]) our algorithm is linear to the length of the input sequence and has the advantage of allowing the user to specify the exact length of

[1]Laboratory of Medical Physics, Department of Medicine, Democritus University of Thrace, Alexandroupolis, Greece. E-mail: `adam@med.duth.gr`

[2]Research Academic Computer Technology Institute, 61 Riga Feraiou Street, 26221 Patras, Greece. E-mail: `perdikur@ceid.upatras.gr`

the repetitions the biologist looks for. Taking into consideration the easy parallelisation of Genetic Algorithms we believe that our method can be used in many practical applications. Moreover Genetic Algorithms could be successfully used as a practical way to solve many computationally difficult problems in the areas of Sequence Search and Alignment. They are intellectually satisfying in their simplicity and the way they attempt to mimic biological evolution.

```
FIND REPETITIONS($X$,$l$, $p$, $n$, $p_m$, $p_c$, $elitism$)
Initialize population of words
WHILE $n \geq 1$, DO
    Evaluate-Fitness: compute the repetitions of each word of the population;
    Produce Next Generation: compute the next generation;
        If $elitism \neq 0$, perform elitism;
        If $p_m \leq const$, perform mutation;
        If $p_c \leq const$, perform uniform crossover;
    Report individuals in descending order
END FIND REPETITIONS
```

Figure 1: Schematic View of the Genetic Algorithm Methodology

3 Conclusions and Future Work.

Our method efficiently computes the repetitions inside a biosequence by evolving a population of repeated patterns in an evolutionary manner (mutation and crossover) and finally reporting those with high fitness function. Our future work is three fold. The first one concerns the modification of the algorithm by assigning a credit to the operators of mutation and crossover. Thus, each time a new individual is generated, if it yields some improvement over its parents, the operator that was directly responsible for its creation gets the largest part of the credit and so in the new generation we can dynamically change the probability of the mutation or crossover operator. This can reduce the time complexity needed to compute the mutation and crossover operation for the population in each generation. The second research direction concerns the addition of one operator responsible for inserting gaps inside repeated patterns thus giving the possibility of inferring structured patterns from the input biosequence. Finally an interesting problem arises from having "don't care symbols" in the input sequence [4]. A "don't care" symbol has the property of matching any symbol of a given alphabet. We believe that our approach can efficiently compute the repetitions even in biosequences with "don't cares".

References

[1] Zhang, C., Wong, A.K. 1997. A genetic algorithm for multiple molecular sequence alignment. *Comput. Appl. Biosci.*, Vol. 13. (1997) pp. 565-581.

[2] Kurtz, S., Schleiermacher, C. 1999. REPuter: fast computation of maximal repeats in complete genomes. *Bioinformatics*, Vol. 15, (1999) pp. 426-427.

[3] Tsunoda, T., Fukagawa, M. Takagi, M.T. 1999. Time and memory efficient algorithm for extracting palindromic and repetitive subsequences in nucleic acid sequences. In *Proceedings of the Pacific Symposium on Biocomputing*, Vol. 4, (1999) pp. 202-213.

[4] Iliopoulos, C., Mohamed, M., Mouchard, L., Perdikuri, K., Smyth, W.F., Tsakalidis, A. 2003. String Regularities with Don't Cares. *Nordic Journal of Computing*, Vol. 10 (2003) pp. 40-51.