Multiple sequence alignment with evolutionary-progressive method

Paweł Kupis and Jacek Mańdziuk

Faculty of Mathematics and Information Science,
Warsaw University of Technology,
Plac Politechniki 1, 00-661, Warsaw, POLAND
kupisp@student.mini.pw.edu.pl, mandziuk@mini.pw.edu.pl

Abstract. A new evolutionary-progressive method for Multiple Sequence Alignment problem is proposed. The method efficiently combines flexibility of evolutionary approach with speed and accuracy of progressive technique. The results show that the hybrid method is an interesting alternative for purely genetic or purely progressive approaches.

1 Introduction

Multiple sequence alignment (MSA) is one of the most important tasks in computational biology. The problem is NP-Hard [1] and consequently high computational complexity and memory requirements make it hard to be approached by the exact, dynamic programming methods (e.g. [2]). In practice dynamic programming methods could be accepted as an effective tool only for pairwise sequence alignment (PSA). In the true MSA case (i.e. for $n \gg 2$, where n denotes the number of sequences to be aligned) their computational load is prohibitive and instead alternative approaches are usually exploited at the cost, however, of loosing the guarantee of finding the optimal solution.

The main alternative to dynamic programming methods is *progressive method* [3, 4], which relies on a series of pairwise alignments in order to build up a final alignment. Closely related sequences are aligned first and subsequently more distant ones. Progressive methods differ in the way the pairwise sequence distance matrix is calculated which has immediate impact on the order according to which sequences are added to the partial solution maintained by the method. In the most renown progressive approach - Clustal W [3] (and its various refinements e.g. [5]) the alignment order is determined by the *phylogenetic tree*, which defines evolutionary distance between sequences. Despite the greedy nature (which can be partly alleviated [6]) the method, due to its high speed and reasonable accuracy, remains one of the most popular tools for solving MSA problem.

In this paper, following [7], we present a hybrid evolutionary-progressive (E–P) method for simultaneous aligning of several amino acid sequences. Due to the space limit several introductory notions concerning MSA as well as foundations of *genetic algorithms* (GA) / *evolutionary programming* (EP) are omitted in the paper. For examples of GA/EP-based approaches to MSA please refer e.g. to [8–11].

2 Evolutionary-progressive method

In the straightforward EP-based approach to MSA each individual in a population represents an alignment. Despite its simplicity, such representation suffers from a very large search space and consequently dramatically increases the execution time. Alternative idea is to apply EP to obtain initial, partial alignment in the restricted search space and subsequently use another method to achieve the final solution.

One of the promising examples of such hybrid approach was presented by Zhang and Wong in [7]. In the first step evolutionary algorithm is used to find the first approximation of the final alignment (called pre-alignment) and then the aligned columns are fixed. In the next step the elements between the pre-aligned columns are aligned by a greedy algorithm using pairwise alignment. The method looks promising, but since several relevant implementation details are missing in [7] it is not possible to exactly follow the idea. In particular the question about how to build the initial population in the pre-alignment space (which is crucial for the quality of obtained result) is not addressed. Another question that can be raised concerns the usefulness of genetic operators proposed in [7]. Our claim is that mutation operator of the form presented in [7] is inefficient. Additionally some refinements are proposed to the definition of a crossover operator. Finally, in the second phase of the algorithm we propose to use the progressive method.

In the next two subsections our modified hybrid method is presented in more detail followed by experimental results (Section 3) and conclusions (Section 4).

2.1 The evolutionary stage

A definition of pre-alignment uses the notions of *identical column* and *columns block*. Column of alignment is called identical if its elements (symbols in each row) are all the same. Identical columns form a block if they are neighbors in alignment. An example presented below shows three sequences, all possible identical columns that can be defined and possible formed blocks. Numbers in columns are indices of respective sequences.

`MAAFCP`	1	1	1	1	2	3	4	5	5	5	5	6
`MACFMCP`	1	1	5	5	2	2	4	3	3	6	6	7
`MACMFCP`	1	4	1	4	2	2	5	3	6	3	6	7

1 2	5 6
1 2	6 7
1 2	6 7

Pre–alignment is defined as a series of identical column blocks which fulfils the following conditions:

– in each row, each number (index) can appear only once,

– in each row numbers are in ascending order.

Each single column is treated as a block of length one. The above conditions guarantee that the final alignment can be build on the basis of pre-alignment.

In our approach, similarly to Zhang and Wong method, an individual in evolutionary algorithm represents a pre-alignment as defined above. The search space is restricted to all correct pre-alignments of a given set of sequences. The

first problem to solve is generating the first population of individuals (i.e. the initial set of pre-alignments). Clearly, identification of all possible identical columns is inefficient, since the complexity of this task equals the complexity of the whole MSA problem. Moreover, a number of columns found would be too big to generate efficient population. For generating the first population and for the whole evolutionary process the notion of *harmful block* is considered. A formal definition of a harmful block and measure of its harmfulness can be found in [7][1]. Intuitively a harmful block can be described as the one connecting two too distant parts of sequences (see Fig. 1). The method of identifying possible identical

Fig. 1. Schematic example of harmful a block.

columns should not prefer such harmful columns, but on the other hand it ought to utilize all symbols in sequences to build a representative subset of identical columns. At last upper limit of the columns found and execution time should be restricted to reasonable limits. After several preliminary trials the following method, depicted in Fig. 2, has been developed. The method is characterized

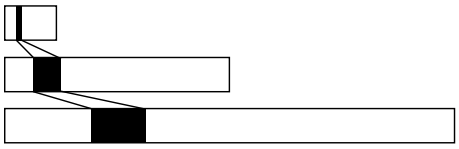

Fig. 2. The idea of search windows.

by two parameters - c_{max} is the upper limit of the columns that compose the pre-alignments and $w_{\%}$ defines the size of a search window (which equals $w_{\%}$ of the sequence length).

At first, one of the sequences, denoted by S, is chosen (in our implementation it is always the shortest one) and then the following procedure is repeated some predefined number of times: a symbol $s \in S$ is selected and its relative position (denoted by rp) with respect to the beginning and the length of S is

[1] Please note, that in [7] a harmful block is used for finding the sub-optimal mutation points during the EP, and not for generating the initial population. Here we suggest to include its notion already in the process of defining the initial population, and to skip the mutation operator.

determined. Next, for each sequence other than S a window of width $w_{\%}$ of the length of sequence is defined and its midpoint is positioned at rp. Then, in each sequence one symbol within window's range is randomly selected. If all selected symbols are pairwise equal to s, then their indices create a new identical column. Assuming the uniform distribution of identical columns in S, the above procedure is repeated $\lceil \frac{c_{max}}{|S|} \rceil$ times for each symbol $s \in S$. Symbols in S are selected in ascending order of their indices. The created identical columns are stored in order of creation. Finally, the initial population of pre-alignments is generated using the following procedure (c_p is the population size, A is an ordered set of identical columns found with the above described method and P is a set of pre-alignments; initially P is empty)

```
foreach(a in A) {
    foreach(p in P) {
        if(a could be added at the end of P) {
            join a to P;
            if possible
                join together a and the last block in P;
            goto next a;
        }
    }
    create new p using a;
    join p to P;
}
sort P by fitness function value;
choose no more than c_p best individuals;
```

The above procedure gathers information about identical columns in restricted number of individuals. It does not guarantee optimal usage of the columns found, but if the order of columns is preserved from the search operation, results are acceptable against execution time. The default values used in our method are $c_{max} = 4\ 000$, $w_{\%} = 0.04$ (4%) and $c_p = \frac{m_a \times n}{10}$, where m_a is the mean sequence length and n is the number of sequences. c_p is additionally restricted to the interval $< 100, 400 >$. Please note, that even if the initial population size is smaller than c_p it will be enlarged to c_p by the first selection operation.

Once the first population is generated the evolutionary process begins. The method uses traditional selection operator - fitness proportionate selection, also known as roulette-wheel selection with one modification - the best individual is extra promoted. Originally mutation operator was planned to be implement as described in [7], but preliminary tests revealed that it was very hard to set the threshold in order to eliminate harmful blocks automatically. Since the major function of mutation is to prevent formation of too long alignments it was decided to control this by appropriate fitness function instead. Consequently mutation operator is not used in the proposed method. The proposed function is defined as follows (p is an individual):

$$fitness(p) = 100 \times \frac{col(p)}{(len_{min}(p))^{\alpha}} \tag{1}$$

where $col(p)$ returns the number of columns in p and $len_{min}(p)$ returns the minimal possible length of alignment constructed on the basis of pre-alignment

represented by individual p. More precisely, let the i-th block of pre-alignment p be denoted by b_i:

$$\begin{matrix} b_{i_{1,1}} & b_{i_{2,1}} & \cdots & b_{i_{w_i,1}} \\ b_{i_{1,2}} & b_{i_{2,2}} & \cdots & b_{i_{w_i,2}} \\ \vdots & \vdots & \ddots & \vdots \\ b_{i_{1,n}} & b_{i_{2,n}} & \ldots & b_{i_{w_i,n}} \end{matrix}$$

where w_i is the width of b_i and let m denotes the number of blocks in pre-alignment p and s_r the length of the r-th sequence, then:

$$len_{min}(p) = \max_{1 \le j \le n}(b_{1_{1,j}}) + \sum_{k=2}^{m} \max_{1 \le j \le n}(b_{k_{1,j}} - b_{k-1_{1,j}}) + \max_{1 \le j \le n}(s_j - b_{m_{1,j}}) \quad (2)$$

Exponent α in (1) specifies the significance of *length penalty* ($\alpha = 20$ by default).

In the crossover operation a random cutting point for each of the two chosen pre-alignments is independently selected and subsequently individuals exchange information. A cutting point never splits existing blocks. Additionally, crossover operator merges blocks in the offsprings if possible (it is not a costly operation, since only blocks neighboring the cutting points have to be checked). Another modification is adding a condition that preserves the best individual in the population, i.e. at least one of the children has to be better than both of the parents in order to allow children replace their parents. Also incorrect pre-alignment never replaces its parents. Cutting point before the first or after the last block causes empty pre-alignment, that alignment also never replaces its parents. Crossover probability was set to 0.4. Evolutionary process can be stopped due to one of the following reasons:

– fitness of the best individual did not change in the last 40 generations,
– the limit of 1 000 generations was exceeded.

After termination of the evolutionary algorithm the best individual is selected and the evolutionary method is recurrently called for substrings located between its blocks. Recursion is stopped if at least one of the following conditions is fulfilled:

– the maximum distance between neighboring blocks in less than 20,
– the algorithm found no identical columns between neighboring blocks.

In that case the progressive method (Section 2.2) is called for the remaining substrings.

2.2 The progressive stage

In the second stage a typical progressive algorithm is used. In our implementation it is Clustal W [3] like method. A phylogenetic tree is built with the use of neighbor-joining and mid-point rooting methods. For pairwise alignment Myers-Miller method [12] is applied enhanced to use position-specific gap penalties and other improvements described in [3], in particular:

– sequence weighting,
– gap opening penalty (GOP) modification depending on existing gaps,

– gap extension penalty (GEP) modification depending on existing gaps,
– GEP modification depending on difference in the lengths of the sequences.

Please note that implementation of the progressive part of the whole MSA method must also be very efficient because in some cases only a few or even no identical columns could be found.

3 Results

We compared our implementation of evolutionary-progressive method with ones of the most popular programs in both evolutionary and progressive category. We chose SAGA 0.95 [13] and Clustal W 1.83 [14] as representatives of evolutionary method and progressive one, resp.

Two different measures were used to compare quality of the produced alignments. The first one is the sum-of-pairs score (SPS) and the second one is the column score (CS). Pairwise alignment score for SPS is calculated exactly the in same way as in dynamic programming method and includes substitution matrix usage and affine gap penalty. Two SPS are calculated for the following two parameter sets and finally the mean value is calculated:

– *Set 1* – GOP: 10, GEP: 0.2, BLOSUM 62 matrix,
– *Set 2* – GOP: 10, GEP: 0.2, 250 PAM matrix.

SPS represents the cost of alignment, which means that *the lower SPS, the higher quality of alignment*. CS is calculated in a standard way defined in [15]. On the contrary to SPS for the CS measure *the higher its value the better the alignment result*. Both measures are defined to have nonnegative values. Quality of alignment produced by any of the programs is measured in relation to quality of the reference alignment. Thus, values truly used in comparisons are related to measures value for reference alignments and expressed in percentage.

Test cases were taken from the reference database - BAliBASE, which is the most widely used multiple alignment benchmark, providing high quality, manually refined, reference alignments. We used version 2.01 of that database and also the latest release 3.0. Version 3.0 of BAliBASE [16] includes new, more challenging test cases, representing the real problems encountered when aligning large sets of complex sequences.

All tests were executed on desktop PC (AMD Athlon XP 2000+ 1.70 GHz with 1.00 GB memory). Clustal W 1.83 and our method were run under the control of MS Windows Server 2003, SAGA 0.95 worked under Fedora Core 3 Linux. E–P method was implemented in C# 2.0. Default parameter settings were used in all programs. SAGA used MSA objective function. A single test case was marked as successfully completed if execution time was no longer than 1 hour and memory consumption did not exceed 1 GB. First, test cases from the version 2.01 were taken. The results are presented in Table 1, where it is shown that only SAGA did not complete successfully all test cases. In almost 10% of test cases at least one of test limits was exceeded. Also, it can be seen that our E–P method is a little faster than Clustal W and significantly faster than its evolutionary competitor SAGA. The SPS values comparison shows that SAGA

	the average SPS	the average CS	summary execution time	the length of alignment	successfully completed test cases
SAGA	101.9	77.1	51503	94.6	90.8
Clustal W	101.2	89.7	90	96.6	100
E–P	105.5	92.7	38	102.0	100

Table 1. Results obtained for BAliBASE ver. 2.01 test cases.

and Clustal W obtained very similar results. The cost of alignments produced by E–P method was a bit higher. As follows from comparison of the CS values the E–P method obtained topmost result in this category. Again the scores of SAGA and Clustal W are comparable.

Based on the above results it was decided to use test cases from the newer version of the database only with programs which successfully completed all tests from version 2.01. The results obtained for BAliBASE 3.0 are presented in Table 2 (all tests were successfully completed). The general conclusion from Table 2 is that E–P method is comparable in both time and quality with Clustal W. Actually, by adjusting the window's width $w_{\%}$ in the evolutionary process one can easily establish the balance between the quality measures (SPS, CS) and the execution time or alignment's length. For example E–P excels Clustal W in the SPS category for $w_{\%} \geq 0.12$. Also for all tested window's lengths the proposed method outperforms its competitor in the CS measure and the execution time. On the other hand, regardless the choice of $w_{\%}$ the length of alignment output by E–P method exceeds the one yielded by Clustal W.

	the average SPS	the average CS	summary execution time	the length of alignment
Clustal W	103.6	64.9	2902	94.3
E–P	104.6	74.2	1492	97.3
E–P ($w_{\%} = 0.08$)	104.0	95.0	902	98.9
E–P ($w_{\%} = 0.12$)	102.6	105.6	825	101.2
E–P ($w_{\%} = 0.16$)	101.8	119.8	795	103.1
E–P ($w_{\%} = 0.20$)	100.5	123.1	768	104.8
E–P ($w_{\%} = 0.24$)	99.6	126.5	757	107.7
E–P ($w_{\%} = 0.28$)	98.7	134.2	777	108.7

Table 2. Results obtained for BAliBASE ver. 3.0 test cases.

4 Conclusions

The efficient MSA method can be build using evolutionary techniques, but the representation of individuals and definition of the search space have to be suitably mated. Traditional way of representation is impractical. The evolutionary-progressive method described in this paper is a compromise between flexibility of evolutionary approach and advantages of progressive method, which are speed

and quality. The new concept of search space definition makes evolutionary part of the algorithm easier to implement and faster. Progressive part is used for subtasks with reduced problem dimension. Combination of these two ideas creates the method which is an interesting alternative for progressive methods and which is competitive to purely evolutionary approaches.

References

1. Wang, L., Jiang, T.: On the complexity of multiple sequence alignment. Journal of Computational Biology **1(4)** (1994) 337–348
2. Carrillo, H., Lipman, D.J.: The multiple sequence alignment problem in biology. SIAM Journal on Applied Mathematics **48(5)** (1988) 1073–1082
3. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res. **22** (1994) 4673–4680
4. Zhu, M-J., Hu, G-W., Zheng, Q-L., Peng, H.: Multiple sequence alignment using minimum spanning tree, Proc. 4th International Conference on Machine Learning, Guangzhou (2005) 3352–3356
5. Chaichoompu, K., Kittitornkun, S., Tongsima, S.: MT-ClustalW: Multithreading Multiple Sequence Alignment, Proc. International Parallel and Distributed Processing Symposium (IPDPS) (2006)
6. Notredame, C., Higgins, D., Heringa, J.: T-coffee: A novel method for multiple sequence alignment. Journal of Molecular Biology **302** (2000) 205–217
7. Zhang, C., Wong, A.K.C.: A genetic algorithm for multiple molecular sequence alignment. Computer Application in the Biosciences **13(6)** (1997) 565–581
8. Thomsen, R., Fogel, G.B., Krink, T.: Improvement of Clustal-Derived Sequence Alignments with Evolutionary Algorithms, Proc. The 2003 Congress on Evolutionary Algorithms, vol. 1 (2003) 312–319
9. Liu, L., Huo, H., Wang, B.: Aligning multiple sequences by genetic algorithm, Proc. International Conference on Communications, Circuits and Systems (ICCCAS'04), vol. 2 (2004) 994–998
10. Zhang, G-Z., Huang D-S.: Aligning Multiple Protein Sequence by An Improved Genetic Algorithm, Proc. 2004 IEEE International Joint Conference on Neural Networks, vol. 2 (2004) 1179–1183
11. Abdesslem, L., Soham, M., Mohamed, B.: Multiple Sequence Alignment by Quantum Genetic Algorithm, Proc. International Parallel and Distributed Processing Symposium (IPDPS) (2006)
12. Myers, E.W., Miller, W.: Optimal alignments in linear space. Bioinformatics **4(1)** (1988) 11–17
13. Notredame, C., Higgins, D.G.: SAGA: sequence alignment by genetic algorithm. Nucleic Acids Res. **24(8)** (1996) 1515–1524
14. Chenna, R., et al.: Multiple sequence alignment with the Clustal series of programs. Nucleic Acids Research **31(13)** (2003) 3497-3500
15. Thompson, J.D., Plewniak, F., Poch, O.: A comprehensive comparison of multiple sequence alignment programs. Nucleic. Acids. Res. **27(13)** (1999) 2682–2690
16. Thompson, J.D., Koehl, P., Ripp, R., Poch, O.: BAliBASE 3.0: Latest Developments of the Multiple Sequence Alignment Benchmark. PROTEINS: Structure, Function, and Bioinformatics **61** (2005) 127-136