Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB 2007)

Evolutionary-Progressive Method for Multiple Sequence Alignment

Paweł Kupis and Jacek Mańdziuk
Faculty of Mathematics and Information Science
Warsaw University of Technology
Plac Politechniki 1, 00-661 Warsaw
POLAND
Emails: {p.kupis, mandziuk}@mini.pw.edu.pl

***Abstract*— In this paper a new evolutionary-progressive method for Multiple Sequence Alignment (MSA) is proposed. The method efficiently combines flexibility of evolutionary approach with speed and accuracy of progressive technique. Both stages of introduced hybrid method are described in detail. The results of comparison with several well-known methods show that proposed evolutionary-progressive method is an interesting alternative for purely genetic and purely progressive approaches.**

I. Introduction

Multiple sequence alignment (MSA) is one of the most important tasks in nowadays 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], [7], [8]) the progressive method, due to its high speed and reasonable accuracy, still remains one of the most popular tools for solving MSA problem.

Other possible approaches rely on heuristical, local block alignment [9], [10], linear programming [11], Hidden Markov Models [12], ant colony optimization [13], simulated annealing [14], [15], *genetic algorithms (GA) / evolutionary programming (EP)* [19], [20], [21], [22], [23], [24] or hybrid methods [25], [26] and [16]-[18].

In this paper, following [16], we present a hybrid evolutionary-progressive (E–P) method for simultaneous aligning of several amino acid sequences. Introductory notions concerning MSA as well as foundations of GA/EP are omitted in the paper.

II. Evolutionary-progressive method

In the straightforward EP-based approach to MSA each individual in a population represents an entire alignment. Despite its simplicity, such representation suffers from a very large search space and enforces large population size. In consequence the execution time is dramatically increased. Another consequence of such representation are complicated genetic operators, which also has an impact on method's efficiency. An 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 approaches was presented by Zhang and Wong in [16]. 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 [16] it is not possible to exactly follow the idea. In particular the question on 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 defined in [16]. Our claim is that mutation operator of the form presented in [16] is inefficient. Additionally we propose some refinements in the definition of a crossover operator. Finally, in the second phase of the algorithm it is suggested to use the progressive method (ClustalW-like in our implementation) instead of simple greedy optimization.

In general, the two-phase construction of both methods (Zhang and Wong's and our) is atypical with respect to a search space definition in GA/EP. Instead of most widely used space of matrices describing the entire alignment the pre-alignment's space is used, which makes the major difference between these two methods and the ones listed in Section I.

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

A. The evolutionary stage

Primarily the notion of the pre-alignment has to be precisely defined. A definition uses the notions of *identical column* and *columns block*, which will be clarified first. 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 an alignment.

Consider for example the three following sequences:

```
MAAFCP
MACFMCP
MACMFCP
```

All possible identical columns that can be defined for these sequences are presented below (numbers in columns are indices of respective sequences):

1	1	1	1	2	3	4	5	5	5	5	6
1	1	5	5	2	2	4	3	3	6	6	7
1	4	1	4	2	2	5	3	6	3	6	7

Finally all possible formed blocks of identical columns can be determined:

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 the pre-alignment (all columns defined in pre-alignment can be concurrently fixed in the alignment). The examples of correct pre-alignment of the above three sequences are shown below.

1	2	5	6
1	2	3	7
1	2	6	7

1	2	4	5	6
1	2	4	6	7
1	2	5	6	7

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 [16][1]. Intuitively a harmful block can be described as the one connecting two too distant parts of sequences (see Fig. 1).

Fig. 1. Schematic example of a harmful block.

The example below shows that situation in case of the following set of sequences:

```
MARAFCPMWAAAFCPTMAAFCP
MAARFTCPMAAFCPMAAFCP
MRAAFCPMWAAFCPMAAFTCP
```

If identical column for symbol **T** is fixed, one of the sub-optimal alignments of the above sequences is:

```
MARAFCPMWAAAFCP-----T--MAAFCP------
---------------MAARFTCPMAAFCPMAAFCP
MRAAFCPMW-AAFCPMAA-FTCP------------
```

When the constraint previously applied to symbol **T** is removed more accurate alignment can be constructed:

```
MARAF-CPMWAAAFCPTMAAF-CP
MAARFTCPM--AAFCP-MAAF-CP
MRAAF-CPMW-AAFCP-MAAFTCP
```

The example above shows a harmful block of length 1, but generally there is no restriction on the length of such a block. The method of identifying possible identical columns should not prefer such harmful columns, because usually such columns make efficient aligning difficult. On the other hand the method ought to utilize all symbols in sequences in order to build a representative subset of identical columns. Finally, upper limit of the columns found and execution time should be restricted to reasonable limits. Thus, the method must provide mechanisms for having these parameters under control. After several preliminary trials the following method, depicted in Fig. 2, has been developed.

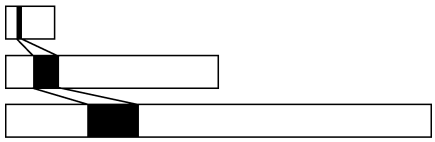

Fig. 2. The idea of search windows.

The method is characterized by two parameters - c_{max} is the upper limit of the columns that compose the pre-alignments (technically c_{max} is an approximate limit of columns number

[1]Please note, that in [16] 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. Rationales and consequences of that decision are discussed later in this section.

that could be identified by the method) and $w_{\%}$ defines the size of the 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 determined. Next, for each sequence other than S a window of width $w_{\%}$ of the length of the sequence is defined and its midpoint is positioned at rp. Then, in each sequence one symbol within window's range is randomly selected[2]. 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 $\lceil \frac{c_{max}}{m} \rceil$ times for each symbol $s \in S$, where m is the length of 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 $< 100, 400 >$. In case the initial population size is smaller than c_p it will be enlarged to c_p by the first selection operation.

Once the initial 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 - simple elitism is used to ensure that the best individual at each generation is incorporated in a subsequent generation. Originally, the mutation operator was planned to be implemented in a way described in [16], but preliminary tests revealed that it was very hard to set the threshold for automatic elimination of harmful blocks. 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 which, according to tests carried out, is beneficial for method's performance. The new proposed fitness 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 (col function is the original fitness function proposed in [16]) 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:

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

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

$$\begin{aligned} len_{min}(p) &= \max_{1 \le j \le q}(b_{1_{1,j}} - 1) + \max_{1 \le j \le q}(s_j - b_{m_{w_m,j}} - 1) \\ &+ \sum_{k=2}^{m} \max_{1 \le j \le q}(b_{k_{1,j}} - b_{k-1_{w_{k-1},j}} - 1) + \sum_{k=1}^{m} w_k = \\ &= \max_{1 \le j \le q}(b_{1_{1,j}}) + \max_{1 \le j \le q}(s_j - b_{m_{1,j}}) \\ &+ \sum_{k=2}^{m} \max_{1 \le j \le q}(b_{k_{1,j}} - b_{k-1_{1,j}}) \end{aligned}$$

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

Please note, that neither in Zhang–Wong method [16] nor in our approach the mutation operator does extend the search space. In both algorithms the sub-space truly searched in evolutionary stage is limited to the combinations of the blocks included in individuals that compose the initial population. In other words, due to the lack of mutation operator, no additional information (data) can be created in the evolutionary phase besides this already included in the initial population. This fact asserts significance of the quality of the method used for the initial population generation.

[2]For extreme symbols (at the beginning and at the end of the sequence) effective window's width is less than $w_{\%}$ of the sequence length because distance from rp to sequence's edge is less than $\frac{w_{\%} \times m}{2}$, where m is the length of the sequence. Therefore complete sequences scan is performed for $w_{\%} = 2.0$ not for $w_{\%} = 1.0$ (however, in that case many harmful columns are included).

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 to 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, which also never replaces its parents. Crossover probability was set to 0.4 by default.

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 is less than 20,
- the algorithm found no identical columns between neighboring blocks.

In that case the progressive method (Section II-B) is called for the remaining substrings.

Parameters for the evolutionary process were selected after some preliminary tests. The chosen values assure stable and reasonable behavior of the algorithm, but they should not be regarded as the optimal ones.

B. 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 [27] and mid-point rooting methods. For pairwise alignment Myers-Miller method [28] 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.

Percentage identity measure for pairwise alignment is used to build sequences' distance matrix for neighbor-joining method (pairwise alignments for distance matrix are constructed with Myers-Miller method without any enhancements). After that phylogenetic tree (guide tree) is build and progressive aligning is performed (previously calculated pairwise alignments are cached and used in progressive part as needed). Besides enhancements in Myers-Miller method another one is used in this stage - substitution matrix used for symbol comparison is automatically selected depending on the distance (measured in the guide tree) between the two sequences or groups of sequences to be compared.

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 can be found during the evolutionary phase.

III. RESULTS

We have compared our implementation of evolutionary-progressive method with ones of the most popular programs used for MSA, i.e. T-Coffee 4.45 [6], MUSCLE 3.6 [7], MAFFT 5.8 [8], DIALIGN 2.2.1 [9], Probcons 1.11 [25], Clustal W 1.83 [29], and SAGA 0.95 [30]. Majority of the above programs represent progressive approach, but evolutionary and local block alignment based methods are also included in comparison.

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 was calculated exactly in the same way as in dynamic programming method and included substitution matrix usage and affine gap penalty. Two SPS were calculated for the following two parameter sets:

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

and finally their mean value was obtained. Since SPS represents the cost of alignment, *the lower the SPS, the higher the quality of alignment*. CS is calculated in a standard way defined in [31]. On the contrary to SPS for the CS measure *the higher its value the better the alignment*. Both measures are defined to have nonnegative values. Quality of alignment produced by any of the above mentioned programs was measured in relation to quality of the reference alignment. Thus, all quality measure values presented in this section are related to measure values for the reference alignments and expressed in percent.

	BAliBASE 2.01	BAliBASE 3.0
publication date	2000	2005
number of test cases	141	218
number of sequences in test case	3 - 28	4 - 142
length of sequences	49 - 993	49 - 7923

TABLE I
COMPARISON OF BALIBASE VER. 2.01 AND 3.0.

Test cases were taken from the most widely used multiple alignment benchmark - BAliBASE, which is the reference database, 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 [32] includes new, more challenging test cases, representing the real problems

encountered when aligning large sets of complex sequences. General comparison of both versions is presented in Table I.

All tests were executed on desktop PC (AMD Athlon XP 2000+ 1.70 GHz [12.5 $\times$ 136 MHz] with 1.00 GB memory [333 MHz]). Clustal W 1.83, MUSCLE 3.6, Probcons 1.11 and our method were run under the control of Microsoft Windows Server 2003 Standard Edition SP1, SAGA 0.95, T-Coffee 4.45, DIALIGN 2.2.1 and MAFFT 5.8 worked under Fedora Core 3 Linux (kernel 2.6.9). E–P method was implemented in C#.NET 2.0. In all programs default parameter settings were used. 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 exceeded 1 GB.

TABLE II
RESULTS OBTAINED FOR BALIBASE VER. 2.01 TEST CASES.

	the average SPS ratio	the average CS ratio	sum of the execution times	% of successfully completed test cases	the average length of alignment ratio
ClustalW 1.83	101.2	89.7	90	100.0	96.6
MUSCLE 3.6	99.8	95.2	65	100.0	99.1
MAFFT 5.8	99.7	99.7	24	100.0	100.4
DIALIGN 2.2.1	94.4	77.0	289	100.0	113.0
Probcons 1.11	99.2	94.0	1796	100.0	104.2
T-Coffee 4.45	99.2	95.0	1732	100.0	101.3
SAGA 0.95	101.9	77.1	51503	90.8	94.6
E-P ($w_{\%} = 0.01$)	104.1	81.5	47	100.0	100.1
E-P ($w_{\%} = 0.02$)	104.7	86.0	43	100.0	101.2
E-P ($w_{\%} = 0.04$)	105.5	92.7	38	100.0	102.0

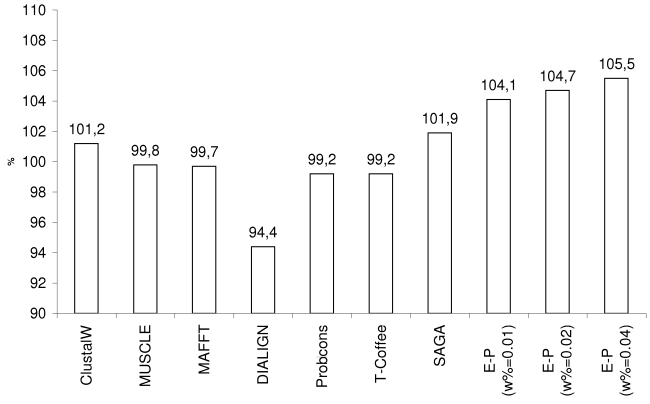

Fig. 3. The average SPS values obtained for BAliBASE 2.01.

First, test cases from the version 2.01 were considered. The results are presented in Table II, where it is shown that only SAGA did not successfully complete all tests (in almost 10% of them at least one of test limits was exceeded).

It can be seen from Table II that all progressive methods obtain similar results (see Fig. 3 for graphical interpretation of this data). The best result in SPS category obtained by

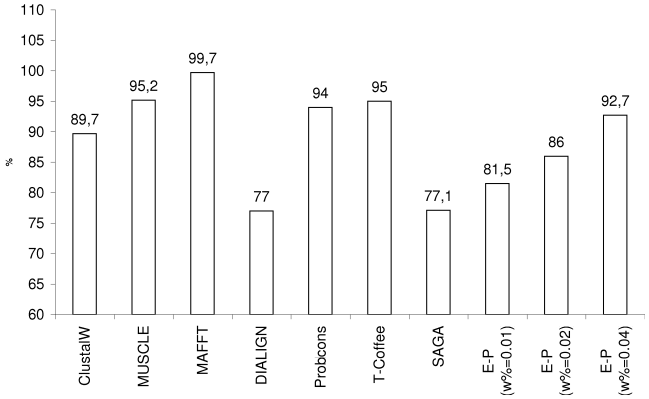

Fig. 4. The average CS values obtained for BAliBASE 2.01.

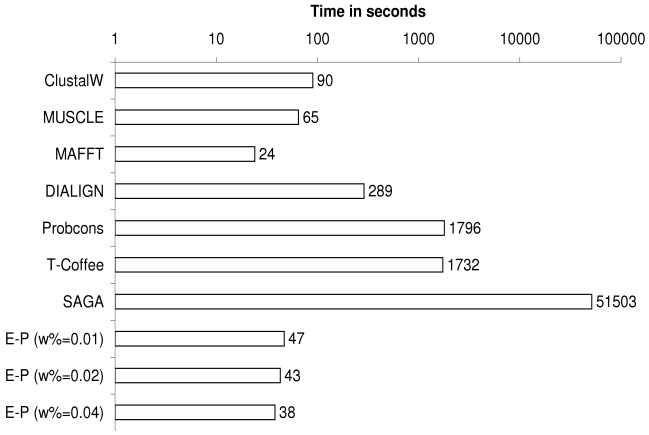

Fig. 5. The sum of execution times of all successfully completed test cases for BAliBASE 2.01.

DIALIGN program has consequences in weak result in CS category (see Fig. 4), big alignments' lengths and high execution time.

Our E–P method is a little faster than Clustal W and MUSCLE but significantly faster than its evolutionary competitor SAGA (see Fig. 5 for execution times comparison). The best time result was attained by MAFFT, the worst ones by Probcons and T-Coffee (among programs which completed all test cases). The cost of alignments produced by E–P method was a bit higher than those produced by the progressive methods. As follows from comparison of quality measures the E–P method accomplished competitive results in both progressive and evolutionary categories.

Based on the above results it was decided to use test cases from the newer version of the database only with the fastest programs which additionally successfully completed all tests from version 2.01. The results obtained for BAliBASE 3.0 are presented in Table III (all test cases were successfully completed by all methods). The general conclusion from Table III is that E–P method is comparable in both time and quality to all the other tested methods. Actually, by adjusting the window's width $w_{\%}$ in the evolutionary process one can

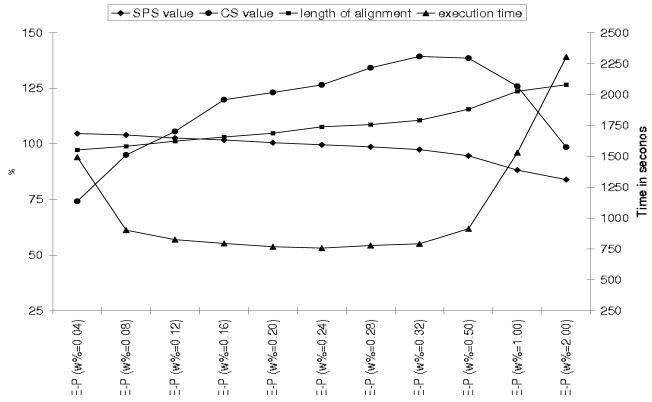

Fig. 6. Results obtained by E–P method for BAliBASE 3.0.

TABLE III
RESULTS OBTAINED FOR BALIBASE VER. 3.0 TEST CASES.

	the average SPS ratio	the average CS ratio	sum of the execution times	the average length of alignment ratio
ClustalW 1.83	103.6	64.9	2902	94.3
MUSCLE 3.6	101.1	84.1	3276	98.9
MAFFT 5.8	100.5	83.0	350	102.8
DIALIGN 2.2.1	91.8	58.9	15689	129.8
E-P ($w_{\%} = 0.04$)	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
E-P ($w_{\%} = 0.32$)	97.4	139.3	792	110.6
E-P ($w_{\%} = 0.50$)	94.6	138.5	913	115.6
E-P ($w_{\%} = 1.00$)	88.2	125.9	1529	123.6
E-P ($w_{\%} = 2.00$)	83.9	98.5	2303	126.6

easily establish the balance between the quality measures (SPS, CS) and the execution time or alignment's length (see Fig. 6). For example E–P excels Clustal W in the SPS category for $w_{\%} \geq 0.12$ (MUSCLE and MAFFT for $w_{\%} \geq 0.20$). Also for almost all tested window's lengths the proposed method outperformed its competitors in the CS measure and the execution time (with the exception of sensational MAFFT's 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 the progressive programs.

An interesting observation from comparison of E–P results for BAliBASE 2.01 and 3.0 is that in the former case the SPS ratio increases with increase of $w_{\%}$, whereas in the latter case - decreases. This observation deserves further investigation and at the moment we are not able to convincingly explain this phenomenon.

IV. 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 chosen. Traditional, straightforward way of problem representation, in which a chromosome describes the whole alignment, is impractical. Instead, it is proposed to use genetic algorithm to obtain the initial, approximate alignment and subsequently apply efficient heuristic method for its refinement. 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] Robert C. Edgar: MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research **32(5)** (2004) 1792–1797
[8] Kazutaka Katoh, Kei-ichi Kuma, Hiroyuki Toh and Takashi Miyata: MAFFT version 5: improvement in accuracy of multiple sequence alignment. Nucleic Acids Research **33(2)** (2005) 511518
[9] Morgenstern B.: DIALIGN: multiple DNA and protein sequence alignment at BiBiServ. Nucleic Acids Research **32i** (2004) W33W36
[10] Jiang., T., Zhao., P.: A Heuristic Algorithm for Blocked Multiple Sequence Alignment, Proc. IEEE International Symposium on Bio-Informatic and Biomedical Engineering (2000) 176–183
[11] Hunt, F.Y., Kearsley, A.J., O'Gallagher, A.: A Linear Programming Based Algorithm for Multiple Sequence Alignments, Proc. 2003 IEEE Bioinformatics Conference (2003) 532–533
[12] Krogh, A.: An introduction to Hidden Markov Models for biological sequences. In: Salzberg, S.L., Searls, D.B., Kasif, S.: Computational Methods in Molecular Biology. Elsevier (1998) 45–63
[13] Chen, Y., Pan, Y., Chen J., Liu, W., Chen L.: Partitioned optimization algorithm for multiple sequence alignment, Proc. 20th International Conference on Advanced Information Networking and Applications (AINA'06) (2006)
[14] Kim, J., Pramanik, S., Chung, M.G.: Multiple sequence alignment using simulated annealing. Computer Applications in the Biosciences (CABIOS) **10(4)** (1994) 419–426
[15] Zola, J., Trystram, D., Tchernykh, A., Brizuela, C.: Parallel Multiple Sequence Alignment with Local Phylogeny Search by Simulated Annealing, Proc. International Parallel and Distributed Processing Symposium (IPDPS) (2006)

[16] Zhang, C., Wong, A.K.C.: A genetic algorithm for multiple molecular sequence alignment. Computer Application in the Biosciences **13(6)** (1997) 565–581

[17] Zhang, C., Wong, A.K.C.: Toward Efficient Multiple Molecular Sequence Alignment: A System of Genetic Algorithm and Dynamic Programming, IEEE Transactions on Systems, Man, and Cybernetics **27(6)** (1997) 918–932

[18] Zhang, C., Wong, A.K.C.: A technique of genetic algorithm and sequence synthesis for multiple molecular sequence alignment. IEEE International Conference on Systems, Man, and Cybernetics **3** (1998) 2442–2447

[19] Chellapilla, K., Fogel, G.B.: Multiple sequence alignment using evolutionary programming. IEEE Congress on Evolutionary Computation (1999) 445–452

[20] Thomsen, R., Fogel, G.B., Krink, T.: A *Clustal* Alignment Improver Using Evolutionary Algorithms. Congress on Evolutionary Computation (CEC-2002) **1** (2002) 121–126

[21] 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

[22] 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

[23] 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

[24] Abdesslem, L., Soham, M., Mohamed, B.: Multiple Sequence Alignment by Quantum Genetic Algorithm, Proc. International Parallel and Distributed Processing Symposium (IPDPS) (2006)

[25] Do, C.B., Mahabhashyam, M.S.P., Brudno, M., and Batzoglou, S.: PROBCONS: Probabilistic Consistency-based Multiple Sequence Alignment. Genome Research **15** (2005) 330–340

[26] Zhou, H., Zhou Y.: SPEM: Improving multiple-sequence alignment with sequence profiles and predicted secondary structures. Bioinformatics **21(18)** (2005) 3615–3621

[27] Saitou, N., Nei ,M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol. **4(4)** (1987) 406–25

[28] Myers, E.W., Miller, W.: Optimal alignments in linear space. Bioinformatics **4(1)** (1988) 11–17

[29] Chenna, R., et al.: Multiple sequence alignment with the Clustal series of programs. Nucleic Acids Research **31(13)** (2003) 3497-3500

[30] Notredame, C., Higgins, D.G.: SAGA: sequence alignment by genetic algorithm. Nucleic Acids Res. **24(8)** (1996) 1515–1524

[31] Thompson, J.D., Plewniak, F., Poch, O.: A comprehensive comparison of multiple sequence alignment programs. Nucleic. Acids. Res. **27(13)** (1999) 2682–2690

[32] 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