Multiple molecular sequence alignment by island parallel genetic algorithm

L. A. Anbarasu*, P. Narayanasamy† and V. Sundararajan‡,**

*Department of Mathematics, †Ramanujan Computing Centre, Anna University, Chennai 600 020, India
‡Centre for Development of Advanced Computing, Pune University Campus, Pune 411 007, India

This paper presents an evolution-based approach for solving multiple molecular sequence alignment. The approach is based on the island parallel genetic algorithm that relies on the fitness distribution over the population of alignments. The algorithm searches for an alignment among the independent isolated evolving populations by optimizing *weighted sum of pairs* objective function which measures the alignment quality. The parallel approach is implemented on PARAM 10000, a parallel machine developed at the Center for Development of Advanced Computing, Pune, and is shown to consistently perform better than the sequential genetic algorithm. The algorithm yields alignments that are qualitatively better than an alternative method, ClustalW.

1. Introduction

Multiple molecular sequence alignments are among the most important tools for analysing biological sequences. Multiple alignments are used to study molecular evolution, to help predict the secondary or tertiary structure of new sequences, RNA folding, gene regulation and polymerase chain reaction primer design. The fact that the multiple molecular sequence alignment problem is of high complexity has led to the development of different algorithms. These algorithms roughly fall into two categories: the greedy ones that rely on pairwise alignment and those that attempt to align all the sequences simultaneously.

It is a standard practice to use the dynamic programming method to align a pair of sequences[1]. To find an optimal alignment for a pair of sequences of length m and n, the dynamic programming method requires $O(mn)$ time and $O(mn)$ space. The complexity of this method grows to $O(m^n)$ when applied to m sequences of length n. Therefore, all of the methods capable of handling larger problems in practical time scales make use of progressive alignment of Feng and Doolittle[2]. In progressive alignment, sequences are aligned in an order imposed by some estimated phylogenetic tree. It first aligns the most closely related sequences, gradually adding the more distant ones. Some of the most widely used multiple molecular sequence alignment packages like ClustalW[3], Mutal and Pileup are based on this algorithm. They have the advantage of being fast and simple as well as reasonably sensitive. Their main drawback is the 'local minimum' problem that stems from the greedy nature of the algorithm. This means that alignments formed early in the process are constructed without the knowledge of most of the available data, and therefore, may easily freeze in a mistake that cannot be corrected later. It is to avoid this pitfall that the second type of method has been designed.

The second type of method uses information from all the input sequences at the same time. Finding the best alignment from the number of possible alignments by this method is very hard and for the 'sum of pairs' this problem has been shown to be NP-Complete[4]. However, using Carrillo and Lipman algorithm[5], the multiple sequence alignment program aligns up to ten sequences by reducing the solution space to a relatively small region[6]. Other global alignment techniques using the *weighted sum of pairs* cost function involve the use of stochastic heuristics such as simulated annealing[7], Gibbs sampling and genetic algorithms[8].

There are two main advantages of using these stochastic optimization methods. First of all they do not have any strong restrictions on the number of sequences to align or the length of those sequences. Secondly, they are very flexible in optimizing any objective function. Genetic algorithm is one of the well-known stochastic search methods that is capable of finding near optimal alignment from totally unaligned sequences[9]. *Implicit parallelism* is an added advantage of genetic algorithms and could be exploited to get both speed-up and better quality in convergence.

Two types of models are used to exploit the *implicit parallelism* of the genetic algorithms: data parallel model and the algorithmic model (island or fine-grained). The data parallel model results only in a speed-up of the algorithm without any qualitative improvement to the solutions. To gain better solution, we designed an island parallel genetic algorithm (iPGA) inspired by the natural process of migration. Recent publications also indicate that parallel genetic algorithm with isolated subpopulations that exchange individuals from time to time may offer an advantage over the sequential approaches[10,11].

In this paper, we describe an iPGA strategy that runs on a distributed network of workstations. We show that our

**For correspondence. (e-mail: sundar@cdac.ernet.in)

approach performs better than the sequential genetic algorithm and an alternative method, ClustalW. The rest of the paper is organized as follows: the second section describes the multiple molecular sequence alignment problem. The third section briefly describes genetic algorithms. The fourth section discusses the iPGA. The fifth section gives the implementation details and results. The last section draws the conclusions and summarizes the present work.

2. Multiple molecular sequence alignment

Let $S = \{S_1, S_2, \ldots, S_n\}$ be the input sequences and assume that n is at least 2. Let Σ be the input alphabet; we assume that Σ does not contain the character '–', so that a dash can be used to denote a gap in the alignment. A set $S' = \{S'_1, S'_2, \ldots, S'_n\}$ of strings over the alphabet $\Sigma' = \Sigma \cup \{-\}$, is called an *alignment* of S if the following two properties hold:

1. The strings in S' have the same length.
2. Ignoring dashes, string S'_i is identical with string S_i.

An alignment can be interpreted as an array with n rows, one row for each S'_i. Two letters of distinct strings are called aligned under S' if they are placed into the same column. Figure 1 shows an example of multiple alignment. It has fifteen rows, one per sequence. In this case, each input sequence has different lengths, and the output alignment has 60 columns.

Algorithms that construct multiple molecular sequence alignment require a cost model as a criterion for constructing optimal alignment. In the simplest cost model there is a cost function $sub: \Sigma' \times \Sigma' \rightarrow N$. It can be defined such that $sub(a, b)$ is the cost of substituting a b in the second sequence for an a in the first sequence; also, $sub(-, b)$ is the cost for columns where the first sequence has a gap and the second has a b, and $sub(a, -)$ is the cost for columns where the first sequence has an a and the second has a –. Each multiple alignment induces a pairwise alignment on the pair of sequences S_i, S_j. The cost of pairwise alignment $S'_{i,j}$ induced in a multiple alignment S' of width w is

$$c(S'_{i,j}) = \sum_{1<k<w} sub(S'[i][k], S'[j][k]).$$

With this, the basic *weighted sum of pairs* multiple sequence alignment problem is to minimize the pairwise sum

$$c(S') = \sum_{i<j} W_{i,j}\, c(S'_{i,j}).$$

3. Genetic algorithms

Genetic algorithms are efficient stochastic search methods based on the principles of natural selection and genetics. Genetic algorithm maintains a population of potential solutions that evolves over time and ultimately converges to a unique solution. Each solution is evaluated to give some measure of its *fitness*. Then a new population is formed by a *selection* mechanism that identifies the fittest individuals of the current population. Selection always ensures that the best individual has a higher probability to reproduce and breed to form a new generation. Some members of this new population undergo alterations by means of two operators based on natural genetics: cross-

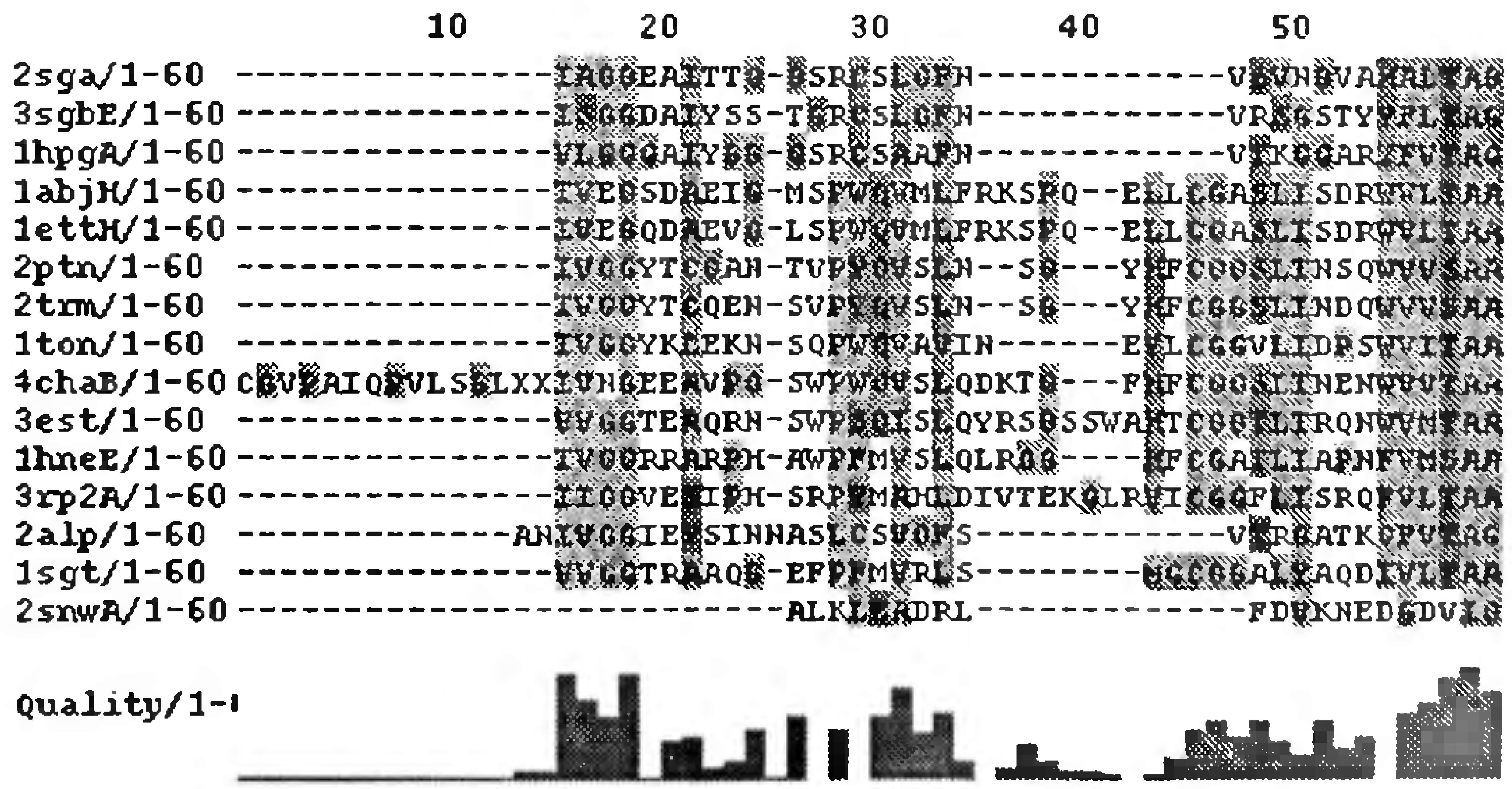

Figure 1. Alignment of prefixes of fifteen S protease sequences.

over and mutation. *Crossover* combines the features of randomly chosen individuals (parents) to form two similar offspring by swapping corresponding segments of the parents. *Mutation* arbitrarily alters some values within the individual, by a random change with a probability equal to the mutation rate. A simple genetic algorithm is shown in Figure 2. It is important to note that these algorithms depart from the traditional algorithms in the following ways: Firstly, it searches from a population of points rather than starting from a single point. Secondly, it manipulates the representation of the variables rather than the variables themselves. Moreover, importance is given to a set of operators that help in generating the new set of solutions, called individuals, from the old set. Hence, in a single run more than one solution is possible in this approach.

Genetic algorithms are generally able to find good solutions in reasonable amounts of time, but as they are applied to larger and complex problems like multiple sequence alignment there is an increase in the time required to find good alignments. Their effectiveness is determined largely by the size of their population. As the population size increases, the genetic algorithm has a better chance of finding the global solutions, but the computation cost also increases as a function of the population size. Many variations on traditional genetic algorithms have been devised to address this limitation and one of the promising choices is to use parallel implementation.

4. Island parallel genetic algorithm for sequence alignment

Methods that are used to parallelize genetic algorithm result only in speed-up without any qualitative improvements to the solutions. Parallel genetic algorithm with isolated subpopulations or *the island model* is used to gain better solutions[10]. In this approach, independent subpopulations of individuals with their own fitness functions evolve in isolation, except for an exchange of some individuals (*migration*). A set of n individuals (problem solutions) is assigned to each of the N processors, for a total population size of $n \times N$. The set assigned to each processor is its subpopulation. The processors are connected by an interconnection network with a ring topology.

```
$t \leftarrow 0$
initialize $P(t)$
evaluate $P(t)$
**foreach** generation
        $t \leftarrow t + 1$
        select $P(t + 1)$ from $P(t)$
        recombine $P(t + 1)$
        evaluate $P(t + 1)$
**endfor**
```

Figure 2. Simple genetic algorithm.

Initial subpopulations consist of randomly constructed alignments created at each processor. Each processor, disjointly and in parallel, executes the sequential genetic algorithm on its subpopulation for a certain number of generations. Afterwards, each subpopulation exchanges a specific number of individuals (*migrants*) with its neighbours. We exchange the individuals themselves, i.e. the migrants are removed from one subpopulation and added to another. Hence the size of the population remains the same after migration. The process continues with the separate evolution of each subpopulation for a certain number of generations. At the end of the process the best individual that exists constitutes the final alignment.

4.1 *Characteristics of island parallel genetic algorithm*

4.1.1 *Initial subpopulation*: Initial subpopulation at each processor is created randomly. It consists of a set of alignments containing only terminal gaps. Alignments are created by choosing a random offset for all the sequences and then moving each sequence to the right according to its offset. To have a same length, sequences are padded with null signs.

4.1.2 *Fitness function*: The fitness of each individual in a subpopulation is calculated by scoring each alignment according to *weighted sums of pairs* objective function. The overall alignment cost is calculated by adding a substitution cost and gap cost to each pair of aligned residues in each column of the alignment with their weights. The cost function includes gap opening and extension penalties. We use pam250 (ref. 12) substitution matrix and natural affine gap penalties for calculating cost function[13,14]. Since our purpose is to minimize the objective function, the alignment scores are inverted for the fitness calculation.

4.1.3 *Selection:* All the individuals are ranked according to their fitness function, and the new children replace the weakest individuals in the old population. An overlapping generation technique is used where half of the population will survive unchanged, the other half will be replaced by the children during each generation[15]. The expected offspring value of an individual is derived from its fitness and used as the probability for each individual to be chosen as a parent. Parents are selected for breeding according to their expected offspring value in a spinning wheel.

4.1.4 *Genetic operators*: *Crossover*: Crossovers are responsible for combining two different alignments into a new one. Two different types of crossover, one-point and uniform crossovers are implemented. Two parent alignments are combined through a single exchange in one-point crossover. The first parent is cut straight at a randomly chosen position. The second one is tailored so that the right piece can be joined to the left piece of the first parent and vice versa. Any vacant space that appears at the junction point for alignment is filled with null signs. This filling of null signs at the junction point forces to design an operator that combines the properties of traditional crossover and those of mutation. The best one out of the two children produced in this way is retained in the population. We designed the uniform crossover to promote multiple exchanges between areas of homology. Consistent positions in an alignment are identified first in both the parents. Two positions are said to be consistent if each column contains the same residue or a null coming from the same gap. Blocks between consistent positions are swapped to create a new alignment.

Gap insertion: While the crossovers combine patterns, there is still a need to generate these patterns. Gap insertion operator extends alignment by inserting gaps. The sequences are split into two groups based on an estimated phylogenetic tree. A gap of randomly chosen length is inserted in each of the sequences of one group at a randomly chosen position. A gap of same length is also inserted into all of the sequences of the second group at a position that has maximum distance from the first gap insertion.

Block shuffling: Generating an optimal arrangement after a gap insertion can often be a matter of shifting a gap to the left or to the right. Thus, the block shuffling operator is used to move blocks of gaps or residues within an alignment. A set of overlapping stretches of residues from one or more sequences is called a block of residues. Each subsequence can be of different length but all subsequences must overlap. A block is chosen first by selecting one residue or gap position from the alignment and moved to a specified position.

Block searching: A set of operators including crossovers, gap insertion and block shuffling is able to create any arrangement needed for the correct alignment, but it is also bound to lose a lot of time. Therefore, a crude method of searching for a block is implemented in this operator. Given a substring in one of the sequences, this operator finds the block to which it may belong in an alignment. A substring of random length at a random position in one of the sequences is compared with all substrings of the same length of other sequences. The best match is selected and added to the initial string forming a small profile. Then the best match is located and added to the profile for the remaining sequences. This process continues until a match has been identified in all the sequences.

5. Implementation and results

The algorithm has been implemented on PARAM 10000, a parallel machine with forty nodes of symmetric multiple processors (SUN Ultra450) each having four CPUs running at 300 MHz and a shared memory of 512 MB. The program is written in C using Parallel Virtual Machine (PVM) library for communication across the processes. The results presented here have been achieved with the machines running their normal daily loads in addition to this code.

5.1 *Comparison of island parallel genetic algorithm with other algorithms*

A set of four test cases were chosen from Pascarella structural alignment data bank[16]. We compare the results of iPGA with the best-known results of other algorithms for multiple molecular sequence alignment. The iPGA was executed 45 times per test case with varying parameters. During initialization of the program, all the operators have the same probability of being used. An automatic procedure, *dynamic scheduling* is used for the scheduling of operators. In this model, an operator has a probability of being used that is the function of the efficiency it has recently displayed at improving alignments[17].

Table 1 presents the best-ever-seen results for all algorithms. Here, the alignment score represents the objective function. The scores obtained from iPGA are better than the sequential genetic algorithm (SGA) and ClustalW, an alternative method. All executions of iPGA were based on arbitrary initializations of the random number generator. Due to the stochastic nature of genetic algorithms, the best-ever-seen results of iPGA were not achieved in all executions. However, we should note that solutions equal to the best-ever-seen results were obtained in at least 50% of the individual iPGA executions.

5.2 *Investigation on migration parameters*

The iPGA alternates the maintenance of the subpopulations isolated in different environments with the introduction of individuals to a new environment. Exchanging individuals between subpopulations, i.e. *migration*, will alter the fitness values of the individuals within the

Table 1. Comparison of iPGA with ClustalW and SGA

			ClustalW	SGA	iPGA
Test case	Nseq	Length	Score	Score	Score
Acprot	16	209	18239760	18052396	17905996
Globin A	15	169	10898878	10857787	10832517
Globin B	17	169	14050372	14030690	13930367
Sprot	15	292	21284220	21005754	20915316

subpopulation and introduce new competitors. Migration, of course, is based on various parameters such as migration frequency, number of migrants and migrant selection strategy. To understand the specific effects of these parameters we have performed several experiments. All the results presented in Table 2 are normalized as the percentage exceeding the best score, with the percentage averaged over five runs. For comparison, we also applied a SGA on the total population size. In all the experiments, the iPGA and SGA were executed for the same number of function evaluations, i.e. the product of number of iterations and the total population size was kept constant.

The influence of migration interval for different number of migrants is investigated. The migrants were chosen randomly in a subpopulation and sent to their right neighbour on a uni-directional ring topology. Table 2 shows that the sequential approach is outperformed by all parallel variations when averaged over all test cases considered. Thus, splitting of the total population size into parallel evolving subpopulations increases the probability that at least one of these subpopulations will evolve towards a better result. Table 2 also shows that a limited migration between the subpopulations further enhances the advantages of the iPGA. Two migrants to each neigh-

Table 2. Alignment score with different numbers of migrants and migration interval

		Migration interval								
		25 generations			50 generations			75 generations		
		Migrants			Migrants			Migrants		
Test case	SGA	2	4	6	2	4	6	2	4	6
Acprot	0.620	0.433	0.312	0.338	0.538	0.463	0.385	0.605	0.597	0.522
Globin A	0.068	0.030	0.052	0.058	0.073	0.019	0.068	0.079	0.034	0.096
Globin B	0.464	0.508	0.447	0.411	0.512	0.622	0.332	0.405	0.307	0.491
Sprot	0.959	0.337	0.379	0.446	0.502	0.554	0.552	0.550	0.575	0.411
Average	0.528	0.337	0.298	0.313	0.406	0.415	0.334	0.500	0.378	0.380
% SGA	100	64	56	59	77	79	63	95	72	72

All results are averaged over five runs and normalized as a percentage exceeding the best-known score in Table 1. Thus smaller the value, better the average alignment score.

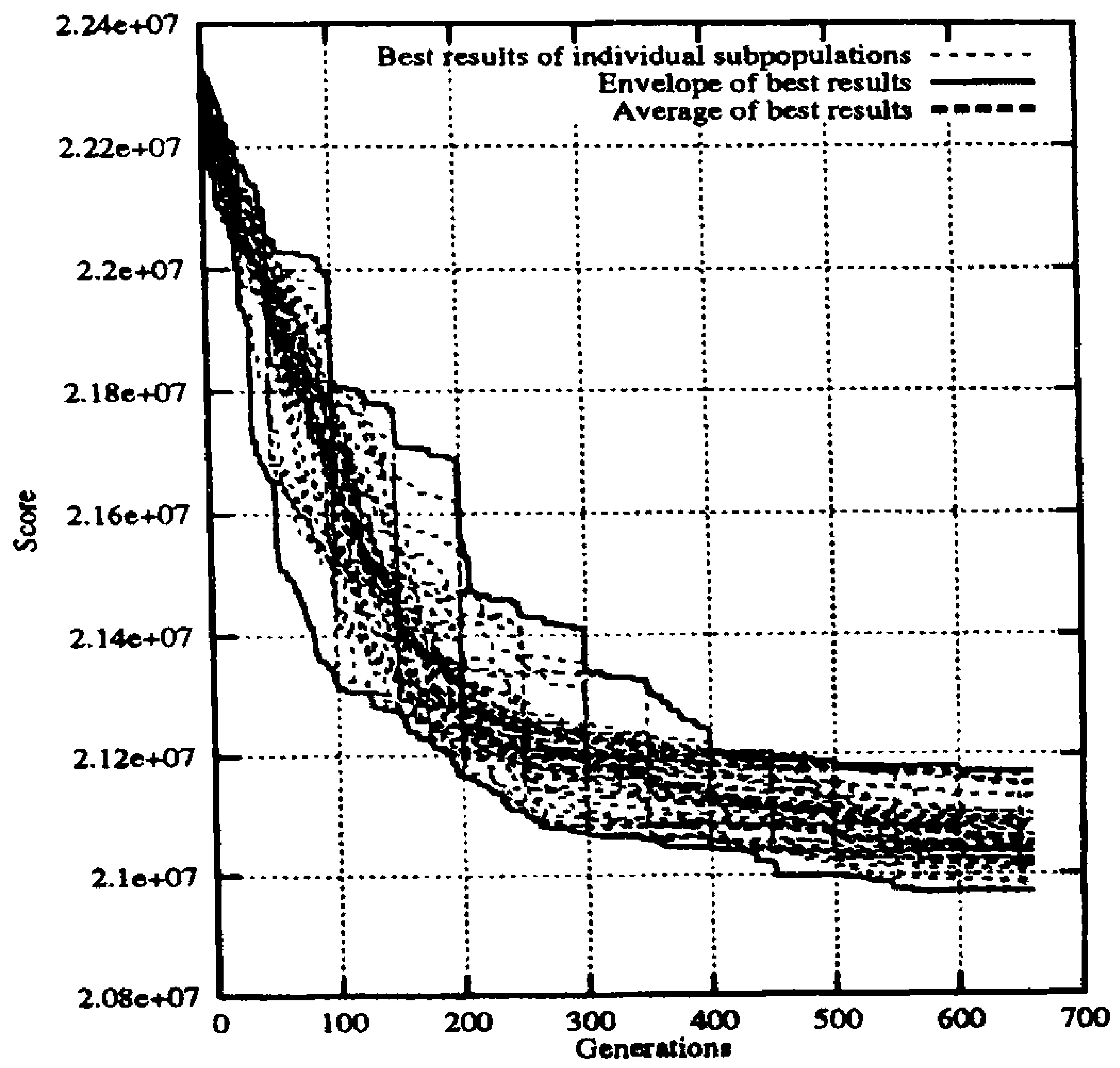

Figure 3. Convergence of best alignment score and the average score in the individual, parallel evolving subpopulations.

bour in the space of 25 generations turned out to be the best parameters when averaged over all the test cases. On the other hand, insufficient migration (migration interval 75) tends towards the isolated parallel approach without any migration.

Figure 3 shows the convergence behaviour of the best individuals in each of the simultaneously evolving subpopulations for Sprot. All the results were obtained with nine subpopulations of size 50. Five runs with nine subpopulations each are plotted, i.e. 45 curves. The plot indicates the importance of migration to avoid premature convergence by inserting new individuals into a stagnating subpopulation.

6. Conclusions

An iPGA has been presented for multiple molecular sequence alignment problem. It has been shown that for all the test cases iPGA outperforms the greedy progressive alignment approach implemented in ClustalW. The results also show that when applied to multiple molecular alignment problem, the iPGA-based on concepts of isolated evolving populations consistently performs better than a SGA. A set of experiments has been performed in order to evaluate the effects of migration parameters on the iPGA. As a result, four migrants in the space of 25 generations lead heuristically to the best results. Asynchronous implementation using multi-form subpopulations for multiple molecular sequence alignment problem is in progress.

1. Needlman, S. B. and Wunch, C. D., *J. Mol. Biol.*, 1970, **48**, 443–453.
2. Feng, D. F. and Doolittle, R. F., *J. Mol. Evol.*, 1987, **25**, 351–360.
3. Thompson, J., Higgins, D. G. and Gibson, T. J., *Nucleic Acids Res.*, 1994, **22**, 4673–4690.
4. Wang, L. and Jiang, T., *J. Comput. Biol.*, 1994, **1**, 337–348.
5. Carrillo, H. and Lipman, D. J., *SIAM J. Appl. Math.*, 1988, **48**, 1073–1082.
6. Lipman, D. J., Altschul, S. F. and Kececioglu, J. D., *Proc. Natl. Acad. Sci. USA*, 1989, **86**, 4412–4415.
7. Ishikawa, M., Toya, T., Hoshida, M., Nitta, K., Ogiwara, A. and Kanehisa, M., *Comput. Appl. Biosci.*, 1993, **9**, 267–273.
8. Zhang, C. and Wong, A. K. C., *Comput. Appl. Biosci.*, 1997, **13**, 565–581.
9. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, New York, 1989.
10. Cohoon, J. P., Hedge, S. U., Martin, W. N. and Richards, D. S., Proc. 2nd Int. Conf. Genetic Algorithms, 1987, pp. 148–154.
11. Sundararajan, V. and Kolaskar, A. S., in *Computer Modelling and Simulations of Complex Biological Systems* (ed. Seetharama Iyengar, S.), CRC Press, 1998, pp. 16–25.
12. Dayhoff, M. O., Schwartz, R. and Orcutt, B. C., *Atlas of Protein Sequence and Structure* (eds Dayhoff, M. O., Pearson, W. A), Suppl. 3, 1978, vol. 5; *Methods Enzymol.*, 1990, **183**, 63–98.
13. Altschul, S. F., *J. Theor. Biol.*, 1989, **138**, 297–309.
14. Altschul, S. F. and Erickson, B. W., *Bull. Math. Biol.*, 1986, **48**, 603–616.
15. Davis, L., *The Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.
16. Pascarella, S. and Argos, P., *Protein Eng.*, 1992, **5**, 121–137.
17. Anbarasu, L. A., Narayanasamy, P. and Sundararajan, V., in *Lecture Notes in Artificial Intelligence* (ed. Bob McKay), Springer-Verlag, 1998, vol. 1585, pp. 130–137.