A Parallel Hybrid Genetic Algorithm for Multiple Protein Sequence Alignment

Hung Dinh NGUYEN [1], Ikuo YOSHIHARA [2], Kunihito YAMAMORI [2], Moritoshi YASUNAGA [3]

[1] Graduate School of Engineering, Miyazaki University,
[2] Faculty of Engineering, Miyazaki University,
[1,2] 1-1, Gakuen-Kibanadai-Nishi, Miyazaki, 889-2192, Japan
[3] Institute of Information Sciences and Electronics, University of Tsukuba,
1-1-1, Tennoudai, Tsukuba, 305-8573, Japan

Abstract - This paper presents a parallel hybrid genetic algorithm (GA) for solving the sum-of-pairs multiple protein sequence alignment. The method is based on a multiple population GENITOR-type GA and involves local search heuristics. It is then extended to parallel to exploit the benefit of multiprocessor system. Benchmarks from the BAliBASE library are used to validate the method.

I. INTRODUCTION

Sequence alignment plays an important role in molecular sequence analysis. It can help to build a phylogenetic tree of related DNA sequences or to predict the function/structure of unknown protein sequences by aligning with other sequences whose function/structure is already known. There are three domains of biological sequences, namely, DNA, RNA, and protein. Sequence alignment makes sense only if all involved sequences are defined on the same domain. This paper mainly deals with the alignment of protein sequences. However, the method can easily be extended to apply to other kinds of sequences.

Sequence alignment aims to construct an alignment of two or more sequences so as to maximize similarities of these sequences. When there are two sequences involved, it is called pairwise alignment. Otherwise, it is called multiple sequence alignment. Fig. 1 shows a fragment of a multiple alignment of four protein sequences.

```
KDSNAPKRAMTSFMFF----SEDFRSKHS
----KPKRPRSAYNIYV---SEDFQEAKD
--ADKPKRPLSAYMLWLNSAREDIKRENP
-DPNKPKRAPSAFFVFMGEFREDFKQKNP
```

Fig. 1: A fragment of an alignment of four protein sequences.

Here, the gap characters '–' are inserted into sequences so as to find similar regions of sequences (marked in boxes). In order to solve the sequence alignment problem, we need a definition to determine how good an alignment is. The most commonly used definition is the sum-of-pairs multiple sequence alignment.

It is well known that the sum-of-pairs multiple sequence alignment can be exactly solved by dynamic programming algorithm [1], which converts the original problem to that of searching for the shortest path in a weighted directed acyclic k-dimensional graph. However, due to the fact that the converted problem is NP-complete, algorithms that guarantee to find the true optimal solution have running times growing exponentially with the size of problem. Therefore, most of practical alignment algorithms are based on heuristics producing quasi-optimal alignments.

There have been some surveys on algorithms for the multiple sequence alignment [2]-[4]. The majority of multiple sequence alignment heuristics is now carried out using a progressive approach (e.g. CLUSTAL, MULTAL) [5]-[7]. This approach has the advantages of speed and simplicity. However, its main disadvantage is the local minimum problem, which stems from the greedy nature of the approach.

Another approach is to use the extension of dynamic programming for simultaneously aligning multiple sequences, e.g. MSA [8], OMA [9], Kobayashi and Imai's A* algorithm [10], and ComAlign [11] etc. In general, algorithms of this approach often have higher quality solutions than those of progressive approach. However, they have drawbacks of complexity, running time and memory requirement, so they can only be applied to problems with limited number of sequences (about 10).

There is also iterative approach for the multiple sequence alignment. This approach includes the iterative algorithm, the simulated annealing, and genetic algorithm (GA). GA is distinguished from the others because it searches for the solution from a population of potential solutions [12],[13]. Therefore, it can evade being trapped in a local minimum. When used alone, however, it has drawbacks of relatively poor quality and more requirement of computing time.

There have been some attempts to solve the multiple sequence alignment by GA, e.g. the works of Notredame and Higgins (SAGA) [14], Hanada *et al.* [15], and Horng *et al.* [16], with the most successful method seems to be SAGA. The SAGA program uses a total of 22 operators, including semi-hill climbing operators, and applies a dynamic scheduling for selecting these operators. The quality of solutions found by SAGA can compete with those of MSA. However, its main disadvantage is its very slow speed.

This paper presents a new GA-based method for more efficient multiple protein sequence alignment. First, we propose a suitable chromosome representation and its corresponding genetic operators. Next, we investigate local search heuristics and incorporate them into our algorithm. Finally, we parallelize our method to exploit the benefit of multiprocessor system.

Section 2 gives a formal definition of the sum-of-pairs multiple sequence alignment. Section 3 describes our parallel hybrid GA. Experiment results and validations are presented in section 4. Section 5 gives conclusions for the paper and a brief description for future research.

II. THE SUM-OF-PAIRS SEQUENCE ALIGNMENT

Suppose that a family $S = (s_1,\ldots,s_k)$ of k sequences of various length n_1 to n_k is given. Each sequence element represents a character from a given alphabet A. (For protein sequences, the alphabet A consists of 20 characters of amino acids.) Suppose that A does not contain the gap character '–'. An alignment of the sequences S is a $k \times N$ matrix $M = (m_{ij})$, where the following conditions are satisfied:

0-7803-7282-4/02/$10.00 ©2002 IEEE

- $m_{ij} \in A' = A \cup \{'-'\}$,
- When removing all gap characters from the row m_i, we receive exactly the sequence s_i, and
- M has no column that contains only gap characters.

Here, N is the length of the alignment and must be greater than or equal to the length of the longest sequence in S and less than or equal to the sum of all sequence lengths.

Next, let assume that there is a cost matrix C between two any letters in A′ that has the following characteristics:

- $C(a, b) = C(b, a), \forall a, b \in A'$,
- $C(a, '-') = G, \forall a \in A$, and
- $C('-', '-') = 0$.

Here, G is a constant and is called the gap cost (or gap penalty).

The cost between two rows m_i and m_j of alignment M is defined as follows:

$$\text{Cost}(m_i, m_j) = \sum_{1 \le p \le N} C(m_{ip}, m_{jp}). \quad (1)$$

And the cost of alignment M is defined as the sum of all pairwise costs in M:

$$\text{Cost}(M) = \sum_{1 \le i < j \le k} \text{Cost}(m_i, m_j). \quad (2)$$

Let call this value the sum-of-pairs cost. Then the sum-of-pairs multiple sequence alignment is defined as the problem to find the alignment that has the minimum sum-of-pairs cost.

In addition, if the cost for each pairwise alignment is multiplied by a weight before summing up to calculate the cost of the alignment as follows:

$$\text{Cost}(M) = \sum_{1 \le i < j \le k} W_{ij} \times \text{Cost}(m_i, m_j). \quad (3)$$

Then the problem is called the weighted sum-of-pairs multiple sequence alignment. The weights are introduced to diminish bias caused by having more sequences from some species and fewer sequences from others [17].

If we define a gap is a continuous segment of gap characters, then a gap of length *l* has a cost of *l*×G in the above cost model. This is called the linear gap cost. There is another type of gap cost in which the open gap character is charged differently with follow gap characters, i.e. a gap of length *l* has a cost of the form: O + *l*×G. This is called the affine gap cost and O is called the open gap cost. The affine gap cost is more appropriate than the linear gap cost from the biological point of view. However, Altschul pointed out that this gap cost has very high complexity so it becomes impractical for even a modest number of sequences [18]. He compromised by proposing a simpler type of gap cost, called the "quasi-natural" gap cost. Furthermore, there is another option in which the terminal gaps are not charged.

There have been many cost matrices introduced so far for the multiple protein sequence alignment, e.g. the PAM and BLUSOM series of matrix [19],[20]. For different cost model (i.e. cost matrix, weight, and type of gap cost etc.), the optimal alignment may be different. Therefore, besides the task of finding algorithm to search for the optimal or quasi-optimal alignment with a given cost model, there is also a task of finding a cost model so that the true mathematical optimal alignment is coincided with (or closed to) the true biological optimal alignment. In this paper, we limit our efforts to only the former task.

III. THE PARALLEL HYBRID GA (PHGA)

A. Chromosome representation

Unlike other GA-based method proposed for the multiple sequence alignment, our method does not solve the problem directly, but instead solve the converted problem of finding the shortest path in a weighted directed acyclic k-dimension graph (where k is the number of sequences). This leads us to a different chromosome representation.

For easy understanding, the ideas of chromosome representation, crossover and mutation operators will be presented in the two-dimensional case, i.e. for pairwise alignment. However, they can be easily extended to a multi-dimensional case.

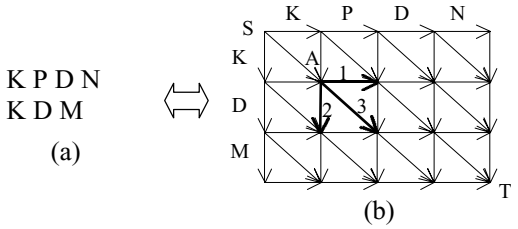

Fig. 2: Pairwise alignment and the converted graph

Consider the example in Fig. 2. The problem of aligning two sequences (Fig. 2(a)) can be converted to the problem of finding the shortest path from the source S to the target T in the directed acyclic graph in Fig. 2(b). Each alignment of the two sequences corresponds to a path from the source S to the target T in the graph. Each column of the alignment corresponds to a vector in the path and the length of the alignment equals to the number of vectors in the path.

In our chromosome representation, each locus represents a vector in a path. Since for a problem of k sequences, there are 2^k - 1 possible vectors that come out from a point (e.g. A), we need at least k bits to encode all the possible vectors into values. Let assume that the values for encoding horizontal, vertical, and diagonal vectors that come out from point A in Fig. 2(b) are 1, 2, and 3, respectively.

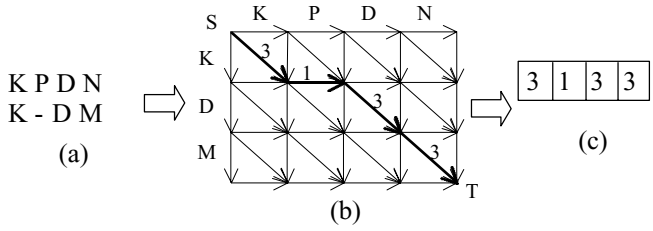

Fig. 3: Chromosome representation

The alignment in Fig. 3(a) corresponds to the bold line path in Fig. 3(b). Its chromosome representation is "3 1 3 3" as illustrated in Fig. 3(c). Since the number of loci of a chromosome is equal to the number of vectors in the path (and the length of the corresponding alignment), the length of chromosome is variable, not fixed as with conventional representation.

Because all of the benchmarks used for validating our method have less than 32 sequences and for the sake of simplifying the codes, we currently use an *unsigned int* (*uint*) variable to represent a locus. The *uint* type has 32 bits in length, therefore limiting the maximal number of sequences in one problem to 32. We will modify our codes later when bigger problems are tested.

0-7803-7282-4/02/$10.00 ©2002 IEEE

B. Crossover

Traditional crossovers such as one point crossover cannot be applied to the chromosome representation described above, because they often create invalid solutions. Besides, the crossover operator should transfer genetic information from both parents to the offspring. Therefore, we propose a new crossover operator (Fig. 4).

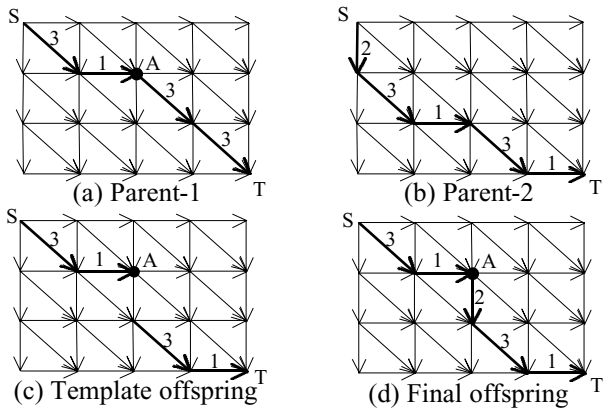

Fig. 4: Crossover

First, two parents, parent-1 and parent-2 (Fig. 4(a), (b)), are randomly selected. Their chromosomes representations are "3 1 3 3" and "2 3 1 3 1". Next, a random crossover point A, which lies in the path of the parent-1, is chosen and the segment from the source S to the crossover point A of parent-1 is directly copied to the offspring. This segment is "3 1". After that, loci are copied from parent-2 backward from the target T until a valid chromosome can be guaranteed (Fig. 4(c)) and the template offspring is "3 1 – 3 1". Finally, a random segment is generated to complete the offspring as "3 1 2 3 1" (Fig. 4(d)). The vectors of the random segment in the offspring are inherited from neither parent. However, since the length of this segment is often relatively short compared to the whole length of the offspring, it does not have much effect on the quality of the offspring.

C. Mutation

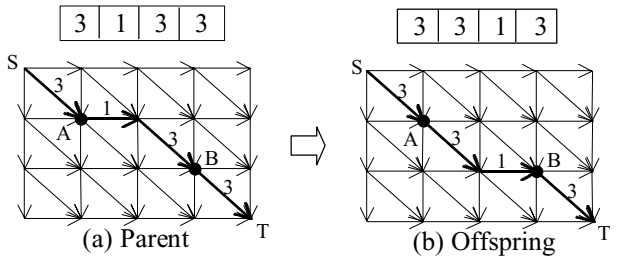

Fig. 5: Mutation

The mutation operator should slightly alter the parent to introduce new genetic information. Fig. 5 illustrates our proposed mutation. The parent is "3 1 3 3" as shown in Fig. 5(a). First, a short segment of the alignment with random length is selected as the mutation segment (segment "1 3" between AB). Next a new segment (segment "3 1") is randomly generated that shares the same two end points A, B with the mutation segment. Finally, connect this new segment with two terminal segments of the parent to form the offspring "3 3 1 3" (Fig. 5(b)).

D. System Prototype

We use a multiple population GENITOR-type GA. GENITOR is a steady-state GA which was originally developed by Whitley and available on the Internet [21],[22]. In our algorithm, only one offspring is created (by either crossover or mutation) in each generation and is immediately inserted into the population to replace the worst parent if the offspring is fitter (GENITOR replaces the worst individual in the population). This replacement scheme causes slower convergence but it can maintain more diversity population. The algorithm does not allow duplicate chromosomes in each sub-population. The selection of the parent is based on linear ranking selection. After a predefined number *migration_interval* of generations, some of the best individuals are exchanged between sub-populations. The algorithm terminates either after a predefined number of generations or after a predefined number of generations that the best individual could not be improved. The following pseudo-code illustrates our algorithm.

```
Procedure PHGA
{
  For each sub-population do
    Initialize sub-population;
  Repeat {
    For each sub-population do {
     If ( rand() < mutation_rate ) {
      Select one parent p using linear ranking;
      Mutation(p, c)
      Replace child c to parent p if fitter;
     } Else {
      Select two parents p1, p2 using linear ranking;
      Recombination(p1, p2, c);
      Replace child c to the worst parent if fitter;
     }
    }
    At predefined migration_interval do
     Migrate between sub-populations
  }
  Until converged;
}
```

E. Hybridization of GA and Heuristics

Since GA has global search ability and heuristics have local search ability, we believe that their hybridization will possibly form a more powerful search. In our hybrid GA, we use two kinds of heuristic; one is for creating the individuals of the first generation and the other is used as mutation for improving individuals during the search.

1) Heuristic for creating individuals

We use the progressive heuristic to create individuals of the first generation of GA. Conventionally, it works by aligning the closest sequences first and gradually adding the more distant ones. The order of adding sequences is often based on a guide tree constructed by some methods, e.g. the Neighbor-Joining method [23]. However, our heuristic uses a random order instead of the order based on the guide tree. By allowing this, a variety of individuals for the first generation can be created.

2) Heuristic for improving individuals

The iterative alignment heuristic is used as mutation for improving individuals. However, instead of applying this heuristic for the whole alignment, we only apply it for a short segment of the alignment. By doing this, the computational

0-7803-7282-4/02/$10.00 ©2002 IEEE

time spent by this heuristic is bound by a constant, not depending on the length of sequences of the problem. Our heuristic works as follows. First, a short segment of the parent is chosen at random. Next, the segment is randomly divided into 2 or 3 groups and columns with only gap characters are removed from each group. Then the dynamic programming algorithm is used to combine these groups into a new segment of alignment. Finally, the new segment is connected with two terminal segments of the parent to form the offspring.

F. Parallelization

Since our algorithm uses a multiple population strategy, we apply the so-called coarse-grained parallel model [24]. In this model, each process is assigned to handle a sub-population and some of the best individuals are exchanged between sub-populations at a predefined interval. The processes communicate between sub-populations via socket channels. However, the current version can only run on a single machine so it can exploit the benefit of multiprocessor system. For a single processor machine this implementation is a little slower than the serial version due to extra time spent on communication. In general, by using this parallel model the computing time of our algorithm is reduced almost linearly with the increase of the number of processors (up to the number of sub-populations).

IV. EXPERIMENTS AND VALIDATIONS

A. Experimental method

The OMA and MSA methods were chosen to compare with our method PHGA since they follow the approach of simultaneously aligning multiple sequences and can find high quality solutions. They are the best appropriate methods that are available to us.

A total of 85 problems taken from the BAliBASE library [25] are used to validate our method. In which 82 problems from *reference1* are used to compare with OMA and MSA methods because they are small enough for both methods to handle and their results to these problems are already available [26]. Three larger problems, namely 1cpt, 1ajsA, and 1lvl from *reference2* are used to test how aptly our algorithm scales with the size of problems.

Although results of MSA method are available, these results are not complete because MSA was limited to run within 12 hours and 2 GB of memory. Therefore, we decided to recompile and rerun the MSA program in our machine, which was a Sun Ultra 80 with four 450-MHz processors, 1 GB of physical memory. The results of OMA were reported by using other type of machine (Sun Ultra Enterprise 450 with 400 MHz processors, 2 GB of physical memory). We estimate that this machine is a little slower than our machine. The GNU gcc software was used to compile the MSA program and our own program.

Since the results of OMA method were reported by using a cost version of the PAM250 matrix with quasi-nature gap cost, we also use it for calculating our results. Although PHGA supports sequence weighting and an option in which terminal gaps are not charged, these features were turned off.

For all benchmarks, the population size was set to 400, which was equally divided into 8 sub-populations. The selection bias was set to 1.25 and the mutation rate to 0.1. The algorithm stops after either 30,000 generations or 3,000 generations of successive un-improvement of the best individual. The migration interval was 500 generations and 5 best individuals of each sub-population were exchanged at migration intervals.

B. Validations

TABLE 1 shows a part of the results. Problems that all three methods have the same cost results are not shown. In this table, *Data* is the problem name. *K* shows the number of sequences and *Max len* shows the maximum length of sequences. *Avg id* denotes average sequence identity, which indicates how close sequences are related. *OMA cost*, *MSA cost*, and *PHGA cost* are the costs of alignments found by OMA, MSA, and PHGA, respectively. The bold numbers indicate that they are the lowest cost among three methods. Column *(best / worst)* shows the best and the worst solutions found by PHGA within 3 runs. Note that for the column *PHGA cost*, we reported the cost of the first run, not the mean of three runs. *OMA time*, *MSA time*, and *PHGA time* express the running times (clock time until completion) for one run in seconds.

1) Quality of solutions

In a total of 82 problems, PHGA wins 29, losses 3 and draws 50 cases compared to the OMA method. The number of wins, losses and equals of PHGA compared to MSA are 30, 10, and 42, respectively. There are 14 cases that PHGA wins both other methods. Notice that there are two cases that the MSA could not find the solution (the message: “Not found” was reported).

2) CPU times

First, let compare PHGA with OMA. PHGA is slower for normal problems but it is significantly faster for some hard problems. For example, the OMA method needs 10,051.20 seconds to solve the 1lvl problem, while PHGA needs only 178.06 seconds. In this problem, PHGA also finds better solution than the OMA method (44,111 vs. 44,136).

Second, let compare PHGA with MSA. Like OMA, the running time of MSA for normal problems is quite fast. For hard problems, however, its running time is worse than that of OMA. For example, the CPU time of MSA for the problem 1pamA is 164,460.12 seconds (about 2 days). On the contrary, PHGA needs only 324.78 seconds to solve this problem, with a little worse result (86,004 vs. 86,144). For the 1havA problem, PHGA is much faster than MSA (119.75 vs. 125,656.20 seconds) and its result is also slightly better than that of MSA (31,705 vs. 31,709).

3) Scalability

The results of three problems from *reference2* are shown in the lower part of TABLE 1. The 1cpt problem has 15 sequences with a maximum length of 434. The 1asjA problem has 20 sequences with a maximum length of 389. The last problem has 24 sequences with a maximum length of 473. The CPU times for solving them were 1,399.85, 2,781.07, and 6,100.92 seconds, respectively. We conclude that our method can scale quite well with problem size.

4) Others

Memory required for PHGA is very small compared to both the OMA and MSA methods. The PHGA method never needed more than 100 MB of memory for solving all tested problems. On the contrary, MSA on some hard cases used nearly all of the physical memory (1 GB) of our machine. The OMA method needed about 1.5 GB of memory to solve the 1lvl problem.

0-7803-7282-4/02/$10.00 ©2002 IEEE

TABLE 1
COMPARATIVE RESULTS OF OMA, MSA, AND PHGA METHODS

Data	K	Max len	Avg id	OMA cost	OMA time	MSA cost	MSA time	PHGA cost	(best / worst)	PHGA time
1aboA	5	80	15	**10,674**	973.64	10,721	11.04	**10,674**	(10,674 / 10,674)	47.59
1ac5	4	483	29	43,341	34.51	**43,325**	258.87	**43,325**	(43,325 / 43,325)	104.14
1ad2	4	213	30	**19,714**	14.68	19,726	0.44	**19,714**	(19,714 / 19,714)	63.51
1ad3	4	447	47	39,218	15.80	**39,209**	1.68	**39,209**	(39,209 / 39,209)	63.99
1aho	5	67	44	**9,807**	6.06	9,813	0.10	**9,807**	(9,807 / 9,807)	44.55
1ajsA	4	387	15	38,468	4,242.99	**38,435**	707.75	38,437	(38,437 / 38,437)	172.42
1ar5A	4	203	42	**17,722**	9.03	17,730	0.34	**17,722**	(17,722 / 17,722)	59.93
1bbt3	5	192	13	30,740	570.38	**30,691**	4,079.57	30,730	(30,709 / 30,730)	134.05
1bgl	4	993	31	90,863	43.05	**90,839**	280.44	**90,839**	(90,839 / 90,843)	162.34
1cpt	4	434	20	39,908	22.86	39,834	2.59	**39,823**	(39,820 / 39,823)	93.25
1csy	5	104	30	**16,361**	6.32	16,366	0.19	**16,361**	(16,361 / 16,361)	102.44
1fjlA	6	70	28	15,973	4.02	15,993	0.12	**15,965**	(15,965 / 15,965)	64.15
1fkj	5	110	44	**15,809**	4.30	15,815	0.17	**15,809**	(15,809 / 15,809)	50.94
1gowA	4	481	31	45,328	28.30	**45,321**	86.21	45,328	(45,328 / 45,328)	104.57
1gpb	5	828	47	117,934	46.29	117,959	9.81	**117,933**	(117,933 / 117,933)	101.89
1gtr	5	436	42	64,202	21.53	**64,190**	59.38	**64,190**	(64,190 / 64,190)	73.90
1havA	5	199	15	31,867	3,003.78	31,709	125,656.20	**31,705**	(31,700 / 31,718)	119.75
1hpi	4	81	33	**6,870**	2.86	6,879	0.13	**6,870**	(6,870 / 6,870)	53.71
1idy	5	58	14	9,542	3.97	9,543	2.83	**9,510**	(9,508 / 9,510)	61.75
1lcf	6	691	49	149,508	1,534.27	**149,485**	60,026.37	**149,485**	(149,485 / 149,490)	146.50
1lvl	4	449	19	44,136	10,051.20	44,128	34.32	**44,111**	(44,111 / 44,112)	178.06
1mrj	4	266	33	24,166	12.34	**24,157**	2.15	**24,157**	(24,154 / 24,157)	51.14
1pamA	5	572	18	86,357	457.83	**86,004**	164,460.12	86,144	(86,064 / 86,203)	324.78
1pfc	5	117	28	**17,708**	19.96	17,710	0.64	**17,708**	(17,708 / 17,708)	63.26
1pkm	4	449	34	42,100	18.36	**42,081**	24.18	**42,081**	(42,081 / 42,081)	98.27
1plc	5	99	46	14,205	4.96	**14,195**	0.16	**14,195**	(14,195 / 14,195)	67.48
1ppn	5	220	46	**31,731**	11.10	31,733	0.66	**31,731**	(31,731 / 31,731)	69.28
1r69	4	78	13	**7,294**	21.82	7,318	1.57	**7,294**	(7,294 / 7,294)	60.61
1rthA	5	541	42	80,352	36.29	80,358	4.36	**80,350**	(80,350 / 80,351)	115.55
1sbp	5	263	19	43,115	3,773.48	**43,040**	13,647.36	43,053	(43,048 / 43,054)	143.17
1taq	5	928	40	133,913	1,901.03	**133,846**	52,573.30	133,854	(133,854 / 133,869)	164.03
1thm	4	279	49	**24,978**	11.37	24,986	0.59	**24,978**	(24,978 / 24,978)	66.71
1ubi	4	94	18	8,631	37.85	8,639	0.36	**8,626**	(8,626 / 8,631)	92.03
1uky	4	220	15	21,116	422.99	Not found	---	**21,089**	(21,089 / 21,094)	98.91
1wit	5	106	17	**16,517**	120.91	16,533	1.52	**16,517**	(16,517 / 16,517)	62.75
1ycc	4	116	29	**10,538**	6.03	10,548	0.21	**10,538**	(10,538 / 10,538)	42.30
2ack	5	482	28	77,139	1,491.15	**77,028**	14,785.62	77,069	(77,030 / 77,069)	150.96
2hsdA	4	262	19	25,284	323.90	25,354	14.02	**25,274**	(25,274 / 25,274)	104.60
2myr	4	474	16	43,541	125.94	**43,501**	1,247.02	43,556	(43,514 / 43,557)	247.65
2pia	4	287	20	**26,606**	353.81	**26,606**	329.74	26,612	(26,606 / 26,612)	70.35
3grs	4	237	14	**23,478**	21.97	23,489	41.86	23,486	(23,483 / 23,486)	110.51
451c	5	87	27	**13,364**	200.28	13,380	1.57	**13,364**	(13,364 / 13,364)	58.03
4enl	3	421	20	18,923	31.14	18,912	1.29	**18,909**	(18,909 / 18,909)	47.01
5ptp	5	245	43	**34,752**	11.51	34,762	0.73	**34,752**	(34,752 / 34,752)	61.07
actin	5	395	45	56,940	20.51	56,942	2.13	**56,938**	(56,938 / 56,938)	72.52
gal4	5	395	14	62,901	31.43	Not found	---	**62,558**	(62,558 / 62,620)	354.50
glg	5	486	31	74,397	175.66	74,384	843.70	**74,381**	(74,381 / 74,382)	118.24
kinase	5	276	20	46,104	38.67	**46,019**	8,800.60	46,039	(46,019 / 46,039)	121.55
1ajsA (ref2)	20	389	35	---	---	---	---	**1,066,434**		2,781.07
1cpt (ref2)	15	434	29	---	---	---	---	**645,403**		1,399.85
1lvl (ref2)	24	473	30	---	---	---	---	**2,023,737**		6,100.92

0-7803-7282-4/02/$10.00 ©2002 IEEE

Another advantage of the PHGA method is that it can produce different results for different runs since it is based on randomness. As a result, we can improve the quality of solutions by running the method several times and take the best of these runs. For example, the numbers of wins, losses and equals of the best of 3 runs of PHGA over OMA are 30, 1, and 51, respectively. These numbers when comparing with MSA method are 31, 8, and 43. However, the running time in this case is 3 times longer than the single run case. We can also notice that the variation between runs is quite small. The results of three runs are the same for most of the tested problems.

Since the quality of solutions from the biological point of view depends not only on our algorithm but also on the cost model, we did not perform experiments to validate our algorithm from this point of view. It needs more time and efforts to determine which cost model is the most suitable for real world applications and we leave them for future research.

V. CONCLUSIONS

This paper presents a GA-based method for solving the sum-of-pairs multiple protein sequence alignment. First, a new chromosome representation and its corresponding genetic operators are proposed. Next, local search heuristics are combined with GA to compensate for its lack of local search ability. Two kinds of heuristics are employed; the first one is for creating individuals in the initial generation and the second one is used as mutation for improving individuals during the search. Finally, the method is extended to parallel processing.

The method has been validated with many benchmarks taken from the BAliBASE library. Experimental results show that the method is in general superior to the OMA and MSA methods. It often finds better solutions and has better scalability compared to OMA and MSA.

The proposed method is also required much less memory than OMA and MSA. Therefore, it can handle bigger problems. Our current implementation can be applied to problems with up to 32 protein sequences while OMA and MSA can only be applied to problems with about a dozen protein sequences. If we consider that the current limit (32) is because of the limit of data structure we use, not because of the limit of the memory, this limit can be further expanded. For example, if we use an array of 4 integers for each locus, then the maximum number of sequences increases to 128. However, this complicates the codes of our algorithm.

There are several issues for future works. First, we want to modify the codes so that it can solve problems with more than 32 sequences. Second, we want to extend the method to run in parallel on a network of computers (e.g. a cluster system) instead of a single multiprocessor system as currently. Third, we want to validate which cost model is the most suitable for real world applications.

Acknowledgements

This research is partly supported by Grant-in-Aid number 13208001 for scientific research, Japan Society for the Promotion of Science.

References

[1] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins”, Journal of Molecular Biology, 48, 1970, pp. 443-453.

[2] M. A. McClure, T. K. Vasi, and W. M. Fitch, “Comparative analysis of multiple protein-sequence alignment methods”, Molecular Biology and Evolution, 11(4), 1994, pp. 571-592.

[3] P. Briffeuil, G. Baudoux, C. Lambert, X. De Bolle, C. Vinals, E. Feytmans, and E. Depiereux, “Comparative analysis of seven multiple protein sequence alignment servers: clues to enhance reliability of predictions”, Bioinformatics, 14, 1998, pp. 357-366.

[4] J. D. Thompson, F. Plewniak, and O. Poch, “A comprehensive comparison of multiple sequence alignment programs”, Nucleic Acids Research, 27, 1999, pp. 2682-2690.

[5] D. F. Feng and R. F. Doolittle, "Progressive sequence alignment as a prerequisite to correct phylogenetic trees”, Journal of Molecular Evolution, 25, 1987, pp. 351-360.

[6] J. D. Thompson, D. G. Higgins, and T. J. Gibson, “CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice”, Nucleic Acids Research, 22, 1994, pp. 4673-4680.

[7] D. G. Higgins and W. R. Taylor, “Multiple Sequence Alignment”, Protein Structure Prediction – Methods and Protocols, D. M. Webster, ed., Humana Press, 2000, pp. 1-18.

[8] S. K. Gupta, J. D. Kececioglu, and A. A. Schaffer, “Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment”, Journal of Computational Biology, 2(3), 1995, pp. 459-472.

[9] K. Reinert, J. Stoye, and T. Will, “An iterative method for faster sum-of-pairs multiple sequence alignment”, Bioinformatics, 16, 2000, pp. 808-814.

[10] H. Kobayashi and H. Imai, “Improvement of the A* algorithm for multiple sequence alignment”, in Proceedings of Genome Informatics Workshop, 1999, pp. 120-130.

[11] K. B. Lassen, O. Caprani, and J. Hein, “Combining many multiple alignments in one improved alignment”, Bioinformatics, 15, 1999, pp. 122-130.

[12] J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.

[13] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, 1989.

[14] C. Notredame and D. G. Higgins, “SAGA: sequence alignment by genetic algorithm”, Nucleic Acids Research, 24, 1996, pp. 1515-1524.

[15] Y. Harada, T. Yokoyama, and T. Shimizu, “Multiple sequence alignment by genetic algorithm”, Genome Informatics, 11, 2000, pp. 317-318.

[16] J. T. Horng, C. M. Lin, B. J. Liu, and C. Y. Kao, “Using genetic algorithm to solve multiple sequence alignments”, in Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2000), 2000, pp. 883-890.

[17] S. F. Altschul, “Gap costs for multiple sequence alignment”, Journal of Theoretical Biology, 138, 1989, pp. 297-309.

[18] S. F. Altschul, R. J. Carroll, and D. J. Lipman, “Weights for data related by a tree”, Journal of Molecular Biology, 207, 1989, pp. 647-653.

[19] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, “A model of evolutionary change in proteins”, in Atlas of Protein Sequence and Structure, vol. 5, suppl. 3, National Biomedical Research Foundation, Washinton DC, USA, 1978, pp. 345-352.

[20] S. Henikoff and J. G. Henikoff, “Amino acid substitution matrices from protein blocks”, in Proceedings of the National Academy of Sciences USA, 89, 1992, pp. 10915-10919.

[21] D. Whitley, “The GENITOR algorithm and selective pressure: why rank-based allocation of reproductive trials is best”, in Proceedings of the 3rd International Conference on Genetic Algorithms, J. D. Schaeffer, ed., Morgan Kaufmann, 1989.

[22] GENITOR group homepage, http://www.cs.colostate.edu/~genitor/

[23] N. Saitou and M. Nei, “The neighbor-joining method: a new method for reconstructing phylogenetic trees”, Molecular Biology and Evolution, 4, 1987, pp. 406-425

[24] E. Cantu-Paz, “Implementing fast and flexible parallel genetic algorithms”, Practical Handbook of Genetic Algorithms – Volume III, 1999, pp. 65-84.

[25] J. D. Thompson, F. Plewniak, and O. Poch, “BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs”, Bioinformatics, 15, 1999, pp. 87-88.

[26] OMA homepage, http://bibiserv.techfak.uni-bielefeld.de/oma/

0-7803-7282-4/02/$10.00 ©2002 IEEE