A novel method for reducing computational complexity of whole genome sequence alignment

Ryuichiro Nakato[1]

E-mail: rnakato@genome.ist.i.kyoto-u.ac.jp

Osamu Gotoh[1,2]

E-mail: o.gotoh@i.kyoto-u.ac.jp

[1] *Department of intelligence Science and Technology,*
Graduate School of informatics, Kyoto University,
Yoshida-Honmachi, Sakyo-ku, Kyoto-shi, Kyoto 606-8501, Japan

[2] *National Institute of Advanced Industrial Science and Technology,*
Computational Biology Research Center,
2-42 Aomi, Koto-ku, Tokyo 135-0064, Japan

Genomic sequence alignment is a powerful tool for finding common subsequence patterns shared by the input sequences and identifying evolutionary relationships between the species. However, the running time and space requirement of genome alignment have often been very extensive. In this research, we propose a novel algorithm called Coarse-Grained AlignmenT (CGAT) algorithm, for reducing computational complexity necessary for cross-species whole genome sequence alignment. The CGAT first divides the input sequences into "blocks" with a fixed length and aligns these blocks to each other. The generated block-level alignment is then refined at the nucleotide level. This two-step procedure can drastically reduce the overall computational time and space necessary for an alignment. In this paper, we show the effectiveness of the proposed algorithm by applying it to whole genome sequences of several bacteria.

Keywords: Genome Alignment; Multiple Alignment; Sequence Analysis; Comparative Genomics.

1. Introduction

With the rapid increase in genomic sequence data available in recent years, there is a great demand for alignment programs that can allow direct comparison of the DNA sequences of entire genomes. However, whole genome sequence alignment is a difficult problem in the points of time and space complexity. Optimal pairwise alignment using Dynamic Programming (DP) requires $O(L^2)$ time and $O(L)$ space, where L is the length of an input sequence.[1] As the length of an entire bacterial genome usually exceeds 1Mb, application of full-blown DP is impractical, therefore, it is necessary to devise more efficient methods.

There are several existing algorithms for pairwise genomic sequence alignment. These algorithms generally apply fast word-search algorithms, such as suffix tree, suffix array, and look-up table, to extract high scoring pairs (HSPs) of subsequences from the input genome

sequences. The HSPs are then chained to conform to coherent alignment.[2–6] If necessary, the chained HSPs may serve as anchor points to the subsequences between which are aligned by a standard DP algorithm. In this report, we propose a novel algorithm for pairwise alignment named Coarse-Grained AlignmenT (CGAT) algorithm. We developed a preliminary version of computer program, $Cgaln$, that implements the proposed algorithm. Comparison of the results of $Cgaln$ with those of $Blastz$[3] indicated that $Cgaln$ is as sensitive as $Blastz$ while considerably more specific than $Blastz$, when appropriate parameters are given. The block-level local alignments are generated in a very short period of time, and the overall computation speed was an order of magnitude faster than that of $Blastz$ with the default setting.

2. Method

2.1. *Outline*

Figure 1 shows the flow of CGAT. CGAT divides the input sequences into "blocks" with a fixed length. These blocks are taken as "elements" to be aligned. The similarity between two blocks, each from the two input sequences, is evaluated by frequency of words (k-mers) commonly found in the blocks. Similar methods based on word counts have been used for rapid estimation of the degree of similarity between two protein sequences.[7,8] For block-level alignment, we apply the Smith-Waterman local alignment algorithm[9] modified so that sub-optimal similarities are also reported.[10] The nucleotide-level alignment is conducted upon the restricted regions included in the block-level alignment found in the first stage. For the nucleotide-level alignment, we adopt a seed-extension strategy widely used in homology search programs such as $Blast$[2,3] and $PatternHunter$.[4]

2.2. *Block-level alignment*

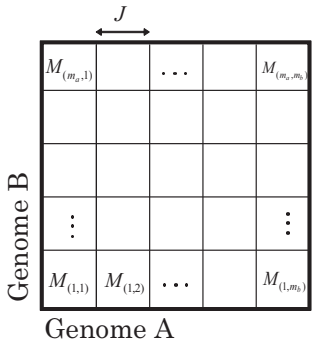
Let's denote the given input genome sequences G_a and G_b. Let L_a and L_b be the lengths of G_a, G_b, respectively, and m_a and m_b be the numbers of blocks in G_a and G_b, respectively. Thus $m_a = \lceil L_a/J \rceil$ and $m_b = \lceil L_a/J \rceil$, where J is the length of a block. Let b_x^a be the x-th block of G_a and b_y^b be the y-th block of G_b. The measure of similarity between b_x^a and b_y^b is denoted by $M_{x,y}$. We evaluate $M_{x,y}$ by the frequency of words commonly found in both b_x^a and b_y^b, where a word is a contiguous or discrete series of nucleotides of length k (k-mer). (In the discrete case, the value for k refers to the "weight", i.e. the number of positions where nucleotide match is examined.[4]) Thus,

$$M_{x,y} = \sum_i (c - \log p_i^a p_i^b) \delta_a(k_i^x) \delta_b(k_i^y), \tag{1}$$

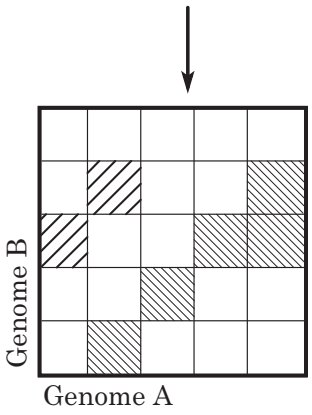
where the summation is taken for all k-mers, and $\delta_a(k_i^x) = 1$ if k_i is present in b_x^a, otherwise $\delta_a(k_i^x) = 0$. The same notation applies to $\delta_b(k_i^y)$ as well. p_i^a and p_i^b are the probabilities that the word k_i appears in b_x^a and b_y^b, respectively, assuming its random distribution along the entire genome. Thus,

$$p_i^a = n_{k_i}^a J/L_a \simeq n_{k_i}^a/m_a, \tag{2}$$

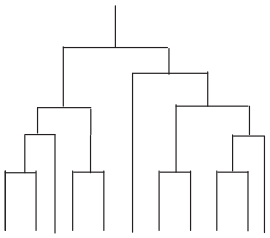
$$p_i^b = n_{k_i}^b J/L_b \simeq n_{k_i}^b/m_b, \tag{3}$$



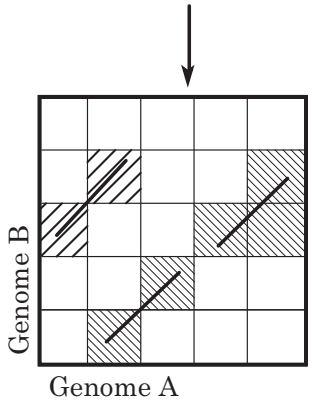
(1) Input sequences are divided into "blocks" with a fixed length, J. Each cell of the mesh-like structure is associated with the block-to-block similarity score $M_{(x,y)}$.



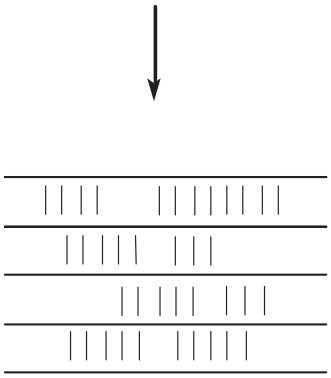
(2) Block-level local alignments are obtained based on the $M_{(x,y)}$ score and a gap penalty with a DP algorithm.



(2') A multiple block-level alignment is obtained with a progressive algorithm.



(3) Nucleotide-level alignment is conducted within the aligned block-level cells.



(3') Multiple nucleotide-level alignment is conducted within the aligned block-level hyper-cubes.

Fig. 1. The flow of CGAT algorithm. (1)-(2)-(3) is a pairwise alignment flow, and (1)-(2)-(2')-(3') is a multiple alignment flow (future work).

where $n^a_{k_i}$ and $n^b_{k_i}$ are the total numbers of k_i in G_a and G_a, respectively. The term c is a constant that may be estimated with some evolutionary model. At this moment, however, we treated c as an adjustable parameter.

The block-level local alignment uses two tables, the "word table" and the "index table." The word table stores the number of occurrences of each word in a genome, $n^a_{k_i}$ and $n^b_{k_i}$, whereas the index table stores the list of positions where a particular k-mer resides. These tables are made only once for each genomic sequence. Using these tables, the similarity measure matrix, $M_{x,y}$ ($x = 1..m_a, y = 1..m_b$), is obtained in $O(L_a L_b/4^k)$.

The block-level alignment is conducted using DP as follows:

$$F_{x,y} = \max \begin{pmatrix} F_{x-1,y-1} + M_{x,y} \\ F_{x-1,y} + M_{x,y} - d \\ F_{x,y-1} + M_{x,y} - d \\ 0 \end{pmatrix}, \tag{4}$$

where d is the gap penalty. Equation (4) is based on Smith-Waterman algorithm.[9] For obtaining the optimal and suboptimal locally best matched alignments, we use the algorithm presented by Gotoh.[10] This method can greatly reduce the storage requirement, while the computational time remains $O(m_a m_b)$.

2.3. *Nucleotide-level alignment*

$Cgaln$ applies the nucleotide-level alignment within the restricted areas that were composed of cells included in the block-level local alignments. For the nucleotide-level alignment, we use the seed finding approach like $Blast$[2,3] or $PatternHunter$.[4] In each cell, the

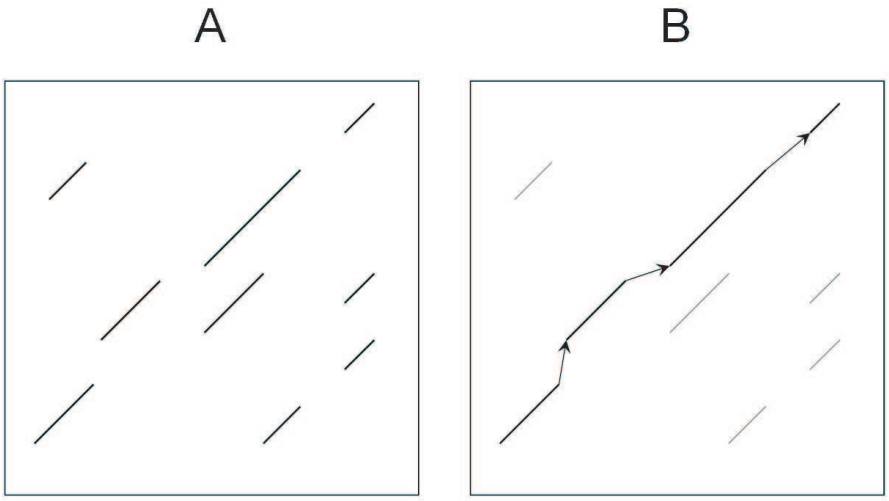


Fig. 2. The nucleotide-level alignment in a cell. (A) Finding seed matches and integrating into HSPs. (B) Computing a maximal-scoring ordered subset of HSPs, and chaining them.

Table 1. The computational complexity of CGAT algorithm

		Time		Space
Block-level alignment	making two tables	$O(\min(L_a, L_b))$	the word table	$O(4^k)$
	matrix scoring	$O(L_a L_b/4^k)$	the index table	$O(\min(m_a, m_b))$ $(O(\min(L_a, L_b)))$
	the DP alignment	$O(m_a m_b)$	the DP matrix	$O(\min(m_a, m_b))$
	total	$O(m_a m_b + \min(L_a, L_b))$	total	$O(4^k + \min(m_a, m_b))$ $(O(4^k + \min(L_a, L_b)))$
Nucleotide-level alignment	generating HSPs	$O(JN_c)$	the two tables	$O(\min(L_a, L_b))$
	chaining HSPs	$O(\overline{n_h}^2 N_l)$	the HSPs	$O(\overline{n_h}^2)$
	total	$O(JN_c + \overline{n_h}^2 N_l)$	total	$O(\min(L_a, L_b) + \overline{n_h}^2)$

seed matches (hits) are searched for by using the word table and the index table once again. Figure 2 shows the nucleotide-level alignment within a cell. A group of hits are integrated into one larger matching segment if the hits are closer to each other than a threshold with no gap (laid on the same diagonal in the dot matrix). We define such a gap-less matching segment as a high scoring pair (HSP). Next, $Cgaln$ computes a maximal-scoring ordered subset of HSPs, and the HSPs are chained to one global alignment within each block-level local alignment. This step can eliminate the noise such as repeats.

2.4. *Computational complexity*

Here we describe the computational complexity of CGAT (see Table 1.) The block-level alignment phase takes space for the word table, the index table, and the DP matrix as major components. The word table requires $O(4^k)$ space, where k is the size of a word. Both the index table and the DP algorithm require $O(\min(m_a, m_b))$ space. However, we also use the index table at the nucleotide-level alignment, and hence the space requirement is formally $O(\min(L_a, L_b))$. Then, the space requirement for the block-level alignment is $O(4^k + \min(L_a, L_b))$.

The time complexity is $O(\min(L_a, L_b))$ for making the two tables, $O(L_a L_b/4^k)$ for preparing the similarity matrix, and $O(m_a m_b)$ for DP alignment. As we choose k such that $\min(L_a, L_b)/4^k$ is not much greater than 1, the computational complexity is $O(m_a m_b + \min(L_a, L_b))$.

The nucleotide-level alignment phase takes $O(JN_c)$ time for generating HSPs, and $O(\overline{n_h}^2 N_l)$ time for chaining, where N_c is the number of cells included in the block-level local alignments, N_l is the number of the block-level local alignments, and $\overline{n_h}$ is the average number of HSPs included in each block-level local alignment. Then, the time requirement for the nucleotide-level alignment is $O(JN_c + \overline{n_h}^2 N_l)$. The space requirement is $O(\min(L_a, L_b))$ for the two tables and $O(\overline{n_h}^2)$ for chaining HSPs, and the total is $O(\min(L_a, L_b) + \overline{n_h}^2)$.

2.5. *Preparation of data*

Whole genome sequences of $E.\ coli$ CFT073 (5,231,428 bp), $E.\ coli$ O157 (5,498,450 bp), and $S.\ dysenteriae$ (4,369,232 bp) were obtained from DDBJ.[a] Before applying the genomic sequences to alignment programs, we masked repetitive sequences by $WindowMasker$[16] with default parameters. As $WindowMasker$ can mask genome sequences without a library of known repetitive elements, it is suitable for a comparative genome analysis.

All experiments were performed on a 2.0 GHz $(\times 2)$ Xeon dual core PC with 4 Gbyte memory.

3. Results

3.1. *Comparison of accuracy by dotplots*

We compare the accuracy and computational time of $Cgaln$ with $Blastz$.[3] $Blastz$ is a pairwise alignment tool for long genomic DNA sequences, and it is used as an internal engine of several multiple genomic sequence alignment tools such as $MultiPipMaker$,[13] TBA,[14] $MultiZ$,[14] and Choi's algorithm.[15]

We obtained the global view of the results of $Cgaln$ and $Blastz$ by the dotplot outputs (Figure 3) for two kinds of pairwise alignments: (A) $E.\ coli$ CFT073 vs. $E.\ coli$ O157, and (B) $E.\ coli$ CFT073 vs. $S.\ dysenteriae$. The $Cgaln$ results were generated by gnuplot[b] whereas the $Blastz$ results were generated by $PipMaker$.[c] We examined $Blastz$ with two sets of parameter values; with the default parameter set and with a tuned parameter set (T=2, C=2). The option "T=2" disallows transitions, which speeds up computation but slightly reduces sensitivity. The option "C=2" directs "chain and extend", which contributes to reduction in noise.

We did not consider segmental inversion in comparison of the two $E.\ coli$ strains, because the tight evolutionary relationship between the two sequences precludes such a possibility. In the case of cross-species comparison, we did consider the possibility of inversions. We adjusted the value for parameter c for each case of comparison, but the other parameters were unchanged.

3.2. *Comparison of computational time and memory*

Table 2 summarizes the actual computation time and memory used in our experiments. $Blastz$ with the default parameters took nearly 200 s for either intra- or inter-species comparison. The computation time was considerably reduced with the tuned parameter set (T=2 and C=2). However, $Cgaln$ runs faster than $Blastz$ even with this tuned parameter set, spending only 40 s and 14 s (including inversions) for the intra- and inter-species comparisons, respectively. Of these total computation times, 7 s and 11 s were consumed

[a] http://www.ddbj.nig.ac.jp/
[b] http://www.gnuplot.info/
[c] http://pipmaker.bx.psu.edu/pipmaker/

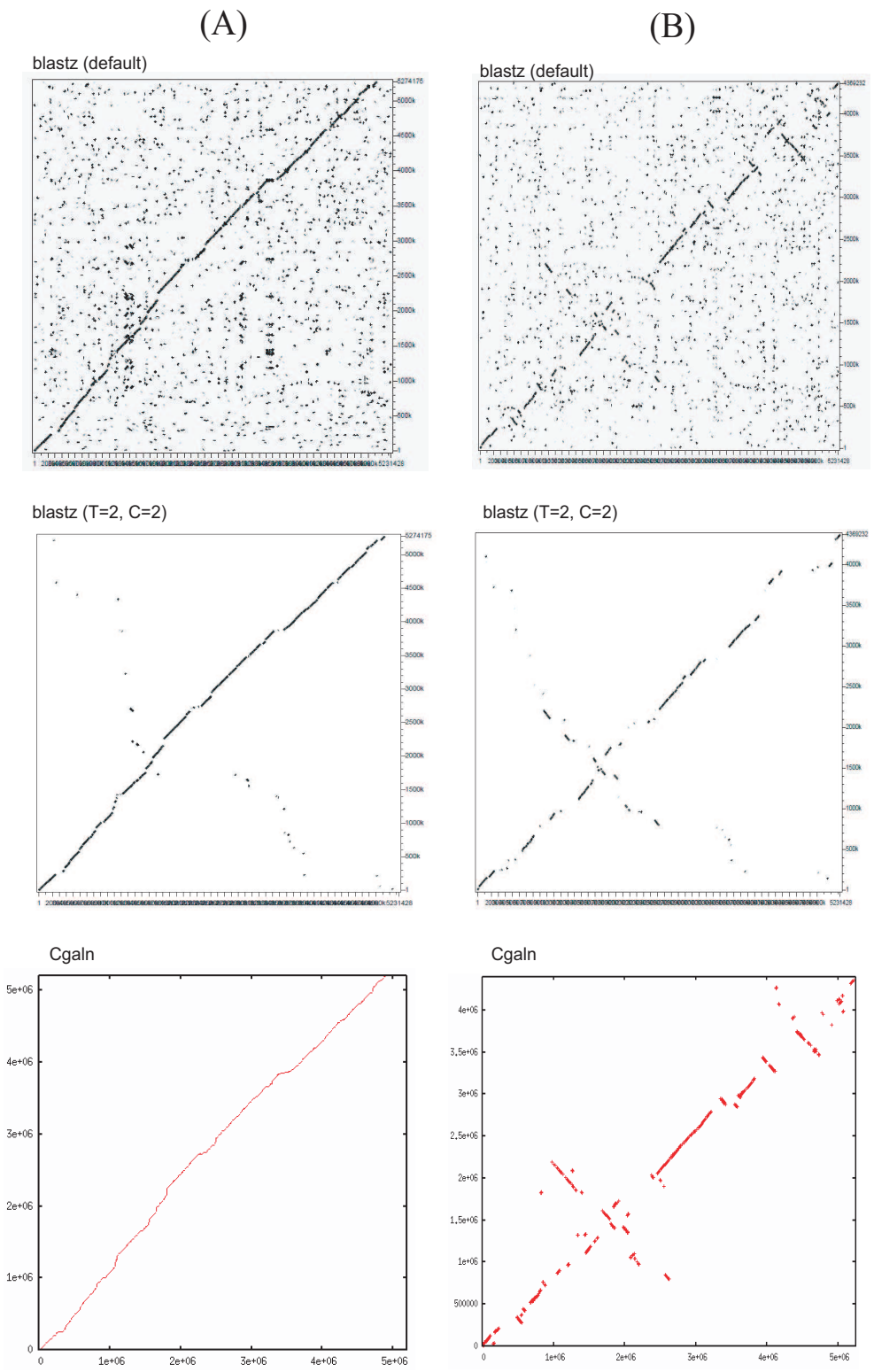


Fig. 3. (A) The dotplot outputs of the alignment between $E.\ coli$ CFT073 and $E.\ coli$ O157. (B) The dotplot outputs of the alignment between $E.\ coli$ CFT073 and $S.\ dysenteriae$. Top: alignment by $Blastz$ with the default parameter set. Middle: alignment by $Blastz$ with a tuned parameter set (T=2, C=2). Bottom: alignment by $Cgaln$. In each alignment, the abscissa indicates $E.\ coli$ CFT073 and the ordinate indicates the counterpart. $Cgaln$ did not consider the inversions in (A), but considered the inversions in (B).

Table 2. Comparison of computational time and memory used by $Blastz$ and $Cgaln$.

	time (s)	memory (MB)
($E.\ coli$ CFT073 - $E.\ coli$ O157)		
$Blastz$ (default)	224	222
$Blastz$ (T=2, C=2)	51	197
$Cgaln$	40	155
($E.\ coli$ CFT073 - $S.\ dysenteriae$)		
$Blastz$ (default)	192	201
$Blastz$ (T=2, C=2)	36	179
$Cgaln$	14	143

by the block-level alignment. When the inversions were omitted, $Cgaln$ took only 7 s for overall alignment and 5 s for the block-level alignment between $E.\ coli$ CFT073 and $S.\ dysenteriae$. $Cgaln$ requires a slightly smaller memory size than $Blastz$. This is reasonable because $Cgaln$ uses 11 mer (11 match positions out of 18 word width) while $Blastz$ uses 12 mer (12 match positions out of 19 word width) to index discrete words in their default settings, respectively. A larger k-mer generally increases both speed and memory consumption.

4. Discussion

Comparison of the results presented in Figure 3 indicates that $Cgaln$ is as sensitive as $Blastz$, when appropriate parameters are given. Moreover, the results also indicate that $Cgaln$ is considerably more specific than $Blastz$ as illustrated by the drastic reduction in the level of noise. Although the noise level of $Blastz$ output is appreciably attenuated by application of the "C=2" option, $Cgaln$ appears to generate better outputs with respect to S/N ratios.

Because the performance of $Cgaln$ strongly depends on the outcome of the block-level alignment, a proper choice of parameter values at this level (e.g c, d, and J) is essential for the overall accuracy of $Cgaln$. Although we currently determine these values in an $ad\ hoc$ manner, it would be desired to develop a method for finding a suitable set of the parameter values automatically. More quantitative evaluation of the performance of $Cgaln$ with more examples, in comparison with those of $Blastz$ and other aligners, remains as a future task.

For the nucleotide-level alignment, we adopted a seed-extension strategy used in homology search programs such as $Blast$[2,3] and $PatternHunter$.[4] In view of sensitivity, this scheme can be improved by adding a recursive step which searches for the seed matches with progressively smaller k-mers in the inter-HSPs regions. The overall computational speed and memory requirement of $Cgaln$ was superior to that of $Blastz$. This result suggests that $Cgaln$ may be used for alignment of longer sequences such as entire mammalian chromosomes. In fact, we have already proved that the CGAT algorithm is successfully applied to all-by-all comparison of human and mouse chromosomes. However, it still requires prohibitively large memory to be executed on an ordinary computer. Further improvement

in the algorithm would be necessary to reduce time and memory requirements.

The very short time consumed by the block-level alignment also suggests the capability of CGAT to be extended to the fast multiple genomic sequence alignment. For this purpose, it is necessary to solve the problems of how to adapt the block-level alignment to progressive or iterative algorithms, and how to treat the rearrangement such as inversions. These problems will be tackled in future work.

Acknowledgments

The authors would like to thank Drs. T. Yada and N. Ichinose for valuable discussions. This work was partly supported by a Grant-in-Aid for Scientific Research on Priority Areas "Comparative Genomics" from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

1. Myers, E. and Miller, W. Optimal alignments in linear space. *Computer Applications in the Biosciences(CABIOS)*, Vol.4, No.1, pp. 11–17, 1988.
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. Basic local alignment search tool. *Journal of Molecular Biology*, Vol.215, No.3, pp. 403–410, 1990.
3. Schwartz, S., Kent, W.J., Smit, A., Zheng, Z., Baertsch, R., Hardison, R. C., Haussler, D. and Miller, W. Human-mouse alignments with BLASTZ. *Genome Research*, Vol.13, No.1, pp. 103–107, 2003.
4. Ma, B., Tromp, J. and Li, M. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, Vol.18, No.3, pp. 440–445 2002.
5. Delcher, A.L., Kasif, S., Fleischmann, R.D., Peterson, J., White, O. and Salzberg, S.L. Alignment of whole genomes. *Nucleic Acids Research*, Vol.27, No.11, pp. 2369–2376, 1999.
6. Brudno, M., Steinkamp, R. and Morgenstern, B. The CHAOS/DIALIGN WWW server for multiple alignment of genomic sequences. *Nucleic Acids Research*, Vol.32(Web Server issue), pp. w41–w44, 2004.
7. Edgar, R.C. Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Research*, Vol.32, No.1, pp. 380–385, 2004.
8. Jones, D.T., Taylor, W.R. and Thornton, J.M. The rapid generation of mutation data matrices from protein sequences. *Computer Applications in the Biosciences(CABIOS)*, Vol.8, No.3, pp. 275–282, 1992.
9. Smith, T.F. and Waterman, M.S. Identification of common molecular subsequences. *Journal of Molecular Biology*, Vol.147, No.1, pp. 195–197, 1981.
10. Gotoh, O. Pattern matching of biological sequences with limited storage. *Computer Applications in the Biosciences(CABIOS)*, Vol.3, No.1, pp. 17–20, 1987.
11. Brudno, M., Do, C.B., Cooper, G. M., Kim, M. F., Davydov, E., NISC Comparative Sequencing Program, Green, E. D., Sidow, A. and Batzoglou, S. LAGAN and Multi-LAGAN: Efficient Tools for Large-Scale Multiple Alignment of Genomic DNA. *Genome Research*, Vol.13, No.4, pp. 721–731, 2003.
12. Bray N. and Pachter L. MAVID: Constrained Ancestral Alignment of Multiple Sequences. *Genome Research*, Vol.14, No.4, pp. 693–699, 2004.
13. Schwartz, S., Elnitski, L., Li, M., Weirauch, M., Riemer, C., Smit, A., NISC Comparative Sequencing Program, Green, E.D., Hardison, R.C. and Miller, W. MultiPipMaker and supporting tools: alignments and analysis of multiple genomic DNA sequences. *Nucleic Acids Research*, Vol.31, No.13, pp. 3518–3524, 2003.

14. Blanchette, M., Kent, W. J., Riemer, C., Elnitski, L., Smit, A. F. A., Roskin, K.M., Baertsch, R., Rosenbloom, K., Clawson, H., Green, E.D., Haussler, D. and Miller, W. Aligning Multiple Genomic Sequences With the Threaded Blockset Aligner. *Genome Research*, Vol.14, No.4, pp. 708–715, 2004.
15. Choi J., Choi, K., Cho, H. and Kim, S. Multiple Genome Alignment by Clustering Pairwise Matches. *Lecture Notes in Computer Science*, Vol.3388, pp. 30–41, 2005.
16. Morgulis, A., Gertz, E.M., Schaffer, A.A. and Agarwala, R. WindowMasker: window-based masker for sequenced genomes. *Bioinformatics*, Vol.22, No.2, pp. 134–141 (2006).