GS-Aligner: A Novel Tool for Aligning Genomic Sequences Using Bit-Level Operations

Arthur Chun-Chieh Shih† and Wen-Hsiung Li†‡*

*Institute of Information Science, Academia Sinica, Taipei, Taiwan; †Department of Ecology and Evolution, University of Chicago; and ‡Genomics Research Center, Academia Sinica, Taipei, Taiwan

A novel algorithm, GS-Aligner, that uses bit-level operations was developed for aligning genomic sequences. GS-Aligner is efficient in terms of both time and space for aligning two very long genomic sequences and for identifying genomic rearrangements such as translocations and inversions. It is suitable for aligning fairly divergent sequences such as human and mouse genomic sequences. It consists of several efficient components: bit-level coding, search for matching segments between the two sequences as alignment anchors, longest increasing subsequence (LIS), and optimal local alignment. Efforts have been made to reduce the execution time of the program to make it truly practical for aligning very long sequences. Empirical tests suggest that for relatively divergent sequences such as sequences from different mammalian orders or from a mammal and a nonmammalian vertebrate GS-Aligner performs better than existing methods. The program and data can be downloaded from http://pondside.uchicago.edu/~lilab/ and http://webcollab.iis.sinica.edu.tw/~biocom.

Introduction

Genomic sequence alignment methods are now much needed for comparing huge amounts of genomic sequence data from different species. Alignments of genomic sequences can help us understand the evolutionary relationship of genomic regions among species (Dubchak et al. 2000), find conserved regions between two genomes such as the human and mouse genomes (Ansari-Lari et al. 1998; Jareborg, Birney, and Durbin 1999; Batzoglou et al. 2000; Lund et al. 2000; Mallon et al. 2000; Göttgens et al. 2001; Wilson et al. 2001), and identify segmental duplications within a genome (Bailey et al. 2001).

However, genomic sequences can be several million bases rather than only hundreds or thousands of bases long, and the traditional methods based on dynamic programming (e.g., Needleman and Wunsch 1970; Smith and Waterman 1981) do not work efficiently because the time and space complexities of these methods are proportional to the product of the lengths of the input sequences. Also, they are not sufficiently sensitive for finding short regions of good alignment flanked by longer regions of poor alignment (Batzoglou et al. 2000). Although popular heuristic similarity search methods such as Fasta (Pearson and Lipman 1988) and Blast (Altschul et al. 1990, 1997) are well suited for finding highly significant matching segment pairs between the query and target sequences, some conserved regions can often be missed because these methods were designed to align only highly conserved regions.

To solve the above problems, several tools for genomic sequence alignment have been proposed. These include MUMmer (Delcher et al. 1999), WABA (Kent and Zahler 2000), Glass (Batzoglou et al. 2000), SSAHA (Ning, Cox, and Mullikin 2001), Dialign (Morgenstern et al. 2002), Avid (Bray, Dubchak, and Pachter 2003), and Longa (Haidong Wang and Wen-Hsiung Li unpublished data). All of these methods adopt a two-step paradigm: the first step is to find alignment anchors and the second step is to use a standard alignment method such as Smith and Waterman's (1981) algorithm to align the region between each pair of two adjacent anchors. The kernel of MUMmer is the suffix tree data structure that is an attractive approach to representing strings and to finding MUMs (Maximum Unique Matches), which are exact matching segment pairs between two sequences and can be used as alignment anchors. Since all MUMs are perfectly matching pairs, not a large number of MUMs may be found when aligning two divergent sequences such as homologous mouse and human sequences. Thus, it may take a considerable amount of time to align those subsequence pairs between two neighboring MUMs by using Smith and Waterman's algorithm. To remove this disadvantage, Wang and Li (unpublished data) proposed a novel seed extension procedure and a two-out-of-three rule to find more significant matching pairs. However, the space required for the construction of a suffix tree in genomic sequence alignment can be so huge that a comparison of two 100 Mb sequences would require about 8 gigabytes of memory (Delcher et al. 1999). Therefore, how to reduce the space requirement is an important issue for this approach. Avid is a newly proposed global alignment program (Bray, Dubchak, and Pachter 2003). The kernel for finding anchors is also the suffix tree approach. With a recursion procedure, Avid can align genomic sequences rapidly. However, this program is an end-to-end global approach, so it cannot detect rearrangements and duplicated regions in sequences.

Glass (Global Alignment System) is a system for aligning hundreds of kilobases of genomic sequences (Batzoglou et al. 2000). There are three fundamental steps in Glass. First, a rough alignment map is constructed by finding long segments that match exactly. Then, the procedure is iterated on the intervening regions searching successively for shorter matching segments. Finally, the remaining regions are aligned using standard alignment techniques. One of the main advantages of Glass is a multilevel (iterative) procedure to find seeds by using different initial matching sizes, so that more long matching

Key words: genomic sequences, sequence alignment, conserved regions, translocations, inversions.

E-mail: whli@uchicago.edu.

Mol. Biol. Evol. 20(8):1299–1309. 2003
DOI: 10.1093/molbev/msg139
Molecular Biology and Evolution, Vol. 20, No. 8,

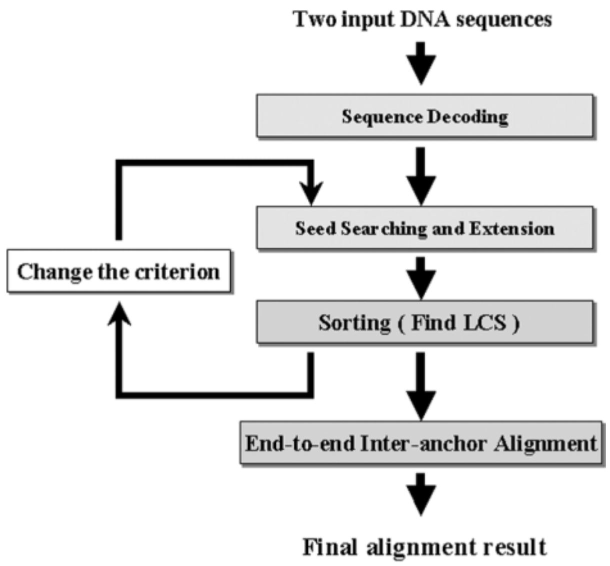


FIG. 1.—The flowchart of GS-Aligner. First, each input DNA sequence is converted into two numeric sequences: one for seed searching and the other for seed extension. Second, alignment anchors, which are segment pairs with a very high sequence similarity between the two sequences and with a length exceeding a threshold, are obtained by extension of all possible seeds. Third, since the obtained anchors are usually unordered and may be inconsistent, the LIS algorithm is used to obtain groups of consistent and ordered anchors. Fourth, to obtain more anchors, we reduce the stringency of the search criteria in the second step and repeat the same procedure; if necessary, the procedure can be repeated further. Finally, the subsequences between two adjacent anchors in the same group are aligned.

segment pairs can be found to serve as alignment anchors. However, like MUMmer, finding exactly matching segment pairs as initial seeds may not lead to a sufficiently large number of anchors when aligning divergent sequences.

Using two binary digits to represent each nucleotide in a DNA sequence is highly efficient for reducing the space requirement (Kent and Zahler 2000; Ning, Cox, and Mullikin 2001). Based on this idea, Ning, Cox, and Mullikin (2001) proposed a hashing table–based approach to search large DNA databases. Since all sequences are constructed to be a hashing table, finding matching pairs between different sequences becomes more efficient and faster than character-based methods. However, the objective of this method is to search for homologous sequences in a huge database. Thus, only those very highly similar regions are found and aligned. Although Blat (Kent 2002) is an improvement in that it allows mismatches in seeds, its objective is also to search for highly homologous sequences in a large database.

In this paper, a novel genomic sequence alignment program called GS-Aligner (Genomic Sequence Aligner) is proposed. GS-Aligner is efficient in terms of both time and space for aligning two very long genomic sequences and for identifying genomic rearrangements such as translocations and inversions. It consists of several ideas from the above mentioned approaches: bit-level coding (Kent and Zahler 2000; Ning, Cox, and Mullikin 2001; Kent 2002), iteratively finding successively shorter matching segments (anchors) (Batzoglou et al. 2000; Morgenstern et al. 2002), longest increasing subsequence (LIS) (Delcher et al. 1999), and optimal local alignment (Smith and Waterman 1981). Moreover, to reduce the execution time, many parts of the program were also implemented in bit-level operations. Basically, GS-Aligner can be divided into four main parts: sequence coding, seed searching and extension, sorting alignment anchors, and closing the gaps between adjacent anchors. These steps are explained in detail below. We shall also compare the performance of our method with those of others.

Method

Figure 1 shows the flowchart of our method. The inputs to the system are two nucleotide sequences. For convenience, the shorter sequence is called sequence A and the other sequence is called sequence B. Without losing generality, all nucleotides in the two sequences are assumed to be only As, Ts, Gs, and Cs. However, some ambiguous or unknown bases (usually denoted as Ns) sometimes appear in a genomic sequence, and we use a list to record the locations of these bases, which are to be avoided as seeds or anchors.

Sequence Decoding

In the first stage of GS-Aligner, two procedures have to be executed: conversion of the input sequences to *x*-mer-packet numeric sequences and creation of a lookup table for seeding codes. The former procedure converts each input sequence into two numeric sequences: one is for seed searching and the other is for seed extension. The reason for using two coding types rather than just one is that the procedure for seed extension does not need to be modified at any time, whereas the type of seed coding is changed when it is necessary to do so. The latter procedure is to create a lookup table of sequence B that is used for indexing those positions with the same numerics of sequence A.

Conversion of the Input Sequences to *x*-mer-Packet Numeric Sequences

Each of the four nucleotides can be represented by two bits (e.g., C by 00, T by 01, A by 10, and G by 11). Then, we can packet each subsequence of x nucleotides (called an x-mer) together to compute a $2x$–bits numeric. To avoid the problem of which nucleotide is the starting one, each x-mer packet overlaps with the next packet in $x - 1$ nucleotides. The obtained numeric sequence is called an x-mer-packet numeric sequence. For example, if the input sequence is 5′-ATGGCATCGAATCG…-3′, the 2-mer-packet coding sequence is 6–13–15–3–8–6–1–12–11–10–6–1–12–… (fig. 2). We divide the x-mer-packet codes into two categories: one for seed searching and the other for seed extension. For seed searching, we design an 8-out-of-12-mer-packet code for obtaining conserved matching segment pairs, using the two-out-of-three rule of Wang and Li (unpublished data). That is, for each three consecutive nucleotides, we require only the first two nucleotides to be identical between the two sequences and allow the third one to be either identical or different. Thus, when counting 12 consecutive nucleotides at a time, the number of the

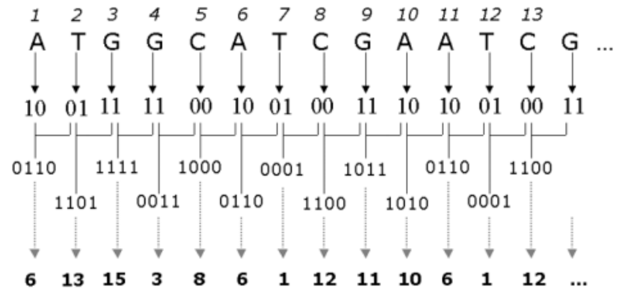


FIG. 2.—An example of a 2-mer-packet coding numeric sequence. The first row shows the positions of the nucleotides in the second row. In this paper, we denote C by $(00)_2$, T by $(01)_2$, A by $(10)_2$, and C by $(11)_2$, where the subscript 2 means that the numeric within the parentheses is represented by a binary system. Thus, when two nucleotides are packed together, one has to use a 4-bits numeric to represent them. For the example, in the figure, when the first two nucleotides, A and T, are packed together as TA, their 2-mer code is $(0110)_2$ (or 6 in a decimal system). Then, the next 2-mer packet is GT and its code is $(1101)_2$ (or 13). Using the same procedure, we obtain the 2-mer-packet coding sequence as 6–13–15–3–4–9–1–12–11–10–6–1–12–. . .

nucleotides in a packet is eight rather than 12. This strategy applies to both coding and noncoding regions because we are searching for short, highly conserved segments to be used as seeds. Note that all three possible reading frames are considered because in our encoding procedure, the first position of the x-mer moves one step to the next nucleotide in the sequence, that is, all possible x-mers are included in our encoding procedure.

For seed extension, the length of an x-mer-packet can be fixed because we allow a fixed maximum number of mismatches while deciding whether the extension procedure should be terminated or not. In this paper, our default setting is $x = 8$.

Creation of an Address Lookup Table for Seeding Codes

An address lookup table is constructed for indexing the coordinates of all x-mers in the sequence B that are identical to the seeding codes. Then, for any seeding code in sequence A we can immediately find the locations of the same numeric in sequence B from this table. The data structure of this lookup table consists of each unique seeding code and all the coordinates of the same numerics in sequence B.

Seed Extension and Anchor Searching

Searching for Ungapped Alignment Anchors

For each of the identical seeding pairs, we attempt to extend it in both directions according to the following rule: if i or fewer than i mismatches occur between the two sequences to be aligned in the immediate adjacent x-mer in the 5′ or 3′ direction, then the extension is made in that direction, and this procedure is repeated in that direction until the rule is broken. The value of i is usually less than $x/2$. All such extended segments with a score exceeding a threshold value are kept as potential alignment anchors.

Note that actually we do not need to attempt to extend every seeding pair. Each segment without gaps can always be represented as a straight line with slope 1 in the two-dimensional space. If the line is extended, it will intersect with the vertical axis and the intersection can be readily calculated. Thus, before an extension is attempted, we can calculate the intersection of a seeding pair with the Y axis, and attempt the extension only if the coordinate of the seeding pair is outside the range of the last anchor recorded in an intersection table.

For convenience, we denote sequence A by S_A and sequence B by S_B and the number of bases in a seeding packet is set to $x = 8$. In figure 3, the upper line and the lower line are S_A and S_B, respectively. Let xsp be the starting position of a seed in S_A and its 8-mer-packet numeric be i. Using i to index the coding seed lookup table, one can immediately find its related positions with the same numeric i in S_B. Then, for each starting position of such a coding seed in S_B, for example ysp, we compare the pair of next 8-mer-packet numeric at the position

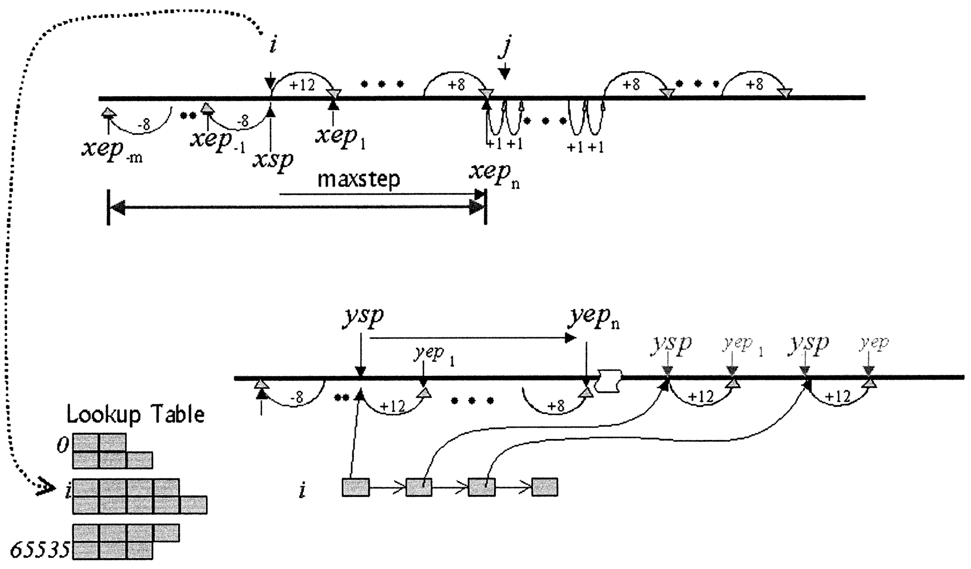


FIG. 3.—Gapless anchor searching.

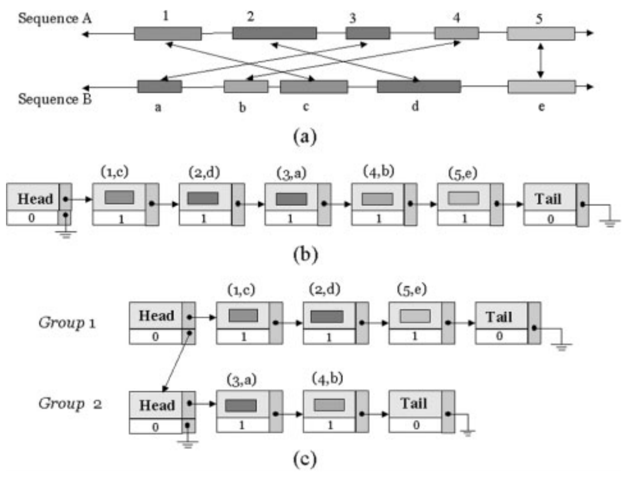


FIG. 4.—The procedure for obtaining consistent anchors in the first iteration. (*a*) The rectangles in sequences A and B represent the matching pairs obtained at the second stage. (*b*) These matching pairs are stored in a linking list starting at a head node and ending at a tail node. The numeric in the smaller rectangle at the bottom of each node represents the level of this node. For each matching pair, the starting positions in sequence A and sequence B and its length are saved in the linking node. (*c*) By using the LIS algorithm, these matching pairs can be divided into two groups. In the linking list of each group, the matching pairs are ordered and consistent in both sequences A and B.

$xsp+8$ of S_A and the position $ysp+8$ of S_B. Let *r* be the allowed maximum mismatches within a pair of *x*-mers. If the number of mismatches of the pair of *x*-mers is less than *r*, the forward extending process can go ahead. Otherwise, the forward process is terminated, and the backward extension process is started. Once both of the forward and backward processes are terminated, we record the obtained *xsp*, *xep*, *ysp*, and *yep* in one of the linking lists.

A 4-mer-k-Mismatches Lookup Table

In a large-scale sequence alignment, the number of the above extension operations can be up to billions or trillions of times. So if we calculate the numbers of matches or mismatches between two *x*-mers base by base in each operation, it will take much time. To overcome this problem, a 4-mer-*k*-mismatches table for all possible pairs of 4-mers is constructed in advance ($k = 0, \ldots 4$). For each pair of two 8-mers to be compared, we divide it into two pairs of 4-mers and the number of nucleotide differences between each pair of 4-mers can be readily found in the table. Then, the number of mismatches between the two 8-mers is the sum of the mismatches in the comparisons of two 4-mer pairs. Thus, each comparison of two 8-mers requires only two accesses to this lookup table and one addition operation. If *x* is less than 8 (but > 4), it can still be regarded as an 8-mer represented by a 16-bit numeric with the first $2(8 - x)$ bits equal to 0 and the number of differences between any two *x*-mers can be computed as above. Note that we do not construct an 8-mer-*k*-mismatches lookup table because the total number of possible 8-mer pairs is $2^{32} = {\sim}420 \times 10^6$, whereas the total number of possible 4-mer pairs is only $2^{16} = 65{,}536$.

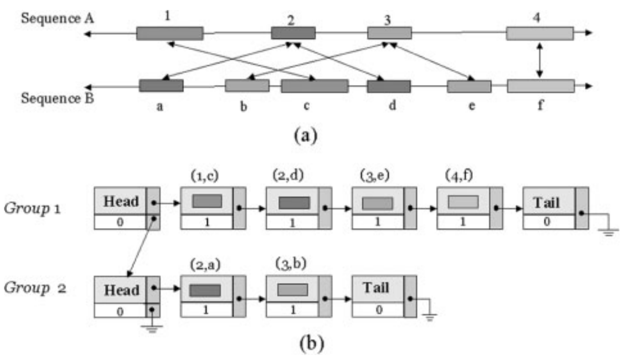


FIG. 5.—A case of sequence duplication. (*a*) The matching pairs (2,a) and (3,b) are duplicated with (2,d) and (3,e), respectively. (*b*) After using the LIS algorithm, the pairs (2,d) and (3,e) are recorded in group 1 since they are orderly consistent with (1,c) and (4,f). The other duplicated pairs (2,a) and (3,b) are stored in group 2. Thus, these duplication blocks can also be detected by our program.

Sorting the Anchors

As in other approaches (e.g., Delcher et al. 1999), the obtained anchors are usually unordered and can be inconsistent (fig. 4*a*). Thus, we have to sort these anchors according to their coordinates and scores. Using the LIS algorithm, several sets of consistent and ordered anchors can be obtained. Each set is called a group in our system. For example, the two sequences to be aligned are shown in figure 4*a*, and the matching pairs, (1,c), (2,d), (3,a), (4,b), and (5,e), that are obtained by the anchor searching procedure are shown in figure 4*b*. Here, we use a link list to store these pairs at the implementation stage. After sorting by the LIS procedure, two consistent groups, group 1 and group 2, are obtained as shown in figure 4*c*. Note further that if there are some blocks duplicated either in one sequence or in both sequences, one set of the duplicated matching pairs will be recorded in group 1 and the other set of the pairs will be stored in group 2, as an example shown in figure 5.

Iterative Procedure

The iterative procedure of finding alignment anchors was proposed by Batzoglou et al. (2000). It starts with a stringent criterion for finding alignment anchors to cut the two input sequences into shorter segments. Then, a less stringent criterion is used to find additional anchors between anchors, and so on. In figure 4*c*, two groups of consistent anchors are obtained from the first round of anchor searching. Let us call them level 1 anchors. We can then change the criterion, say by increasing the number of allowed mismatches and/or decreasing the minimum acceptable length for an anchor. For example, (1,c) and (2,d) in figure 6*a* are two pairs of level 1 anchors. Using a less stringent criterion, we find a set of three matching pairs, (α,*x*), (β,*z*), and (γ,*y*). After LIS sorting, (α,*x*) and (β,*z*) are found to be two consistent pairs. Then, we insert these two matching pairs into between (1,c) and (2,d) as shown in figure 6*c*. The pairs (α,*x*) and (β,*z*) are called level 2 anchors. After scanning all pairs at level 1, all the anchors are stored in a link list. Figure 7 shows the anchors finally obtained.

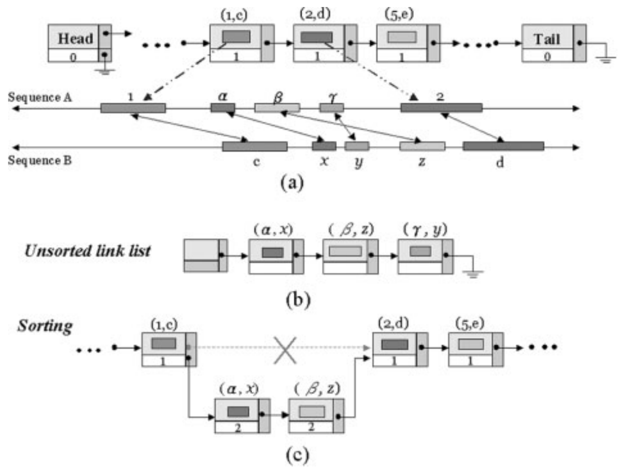


FIG. 6.—The procedure for finding anchors in the second iteration. (*a*) Between the matching pairs (1,c) and (2,d), which were obtained in the first iteration, (α,*x*), (β,*z*), and (γ,*y*) are the matching pairs obtained by using a less stringent criterion. (*b*) These three unsorted matching pairs are stored in a linking list. (*c*) After using the LIS algorithm, the matching pairs (α,*x*) and (β,*z*) are consistent and ordered. Then, these two matching pairs are inserted into between (1,c) and (2,d).

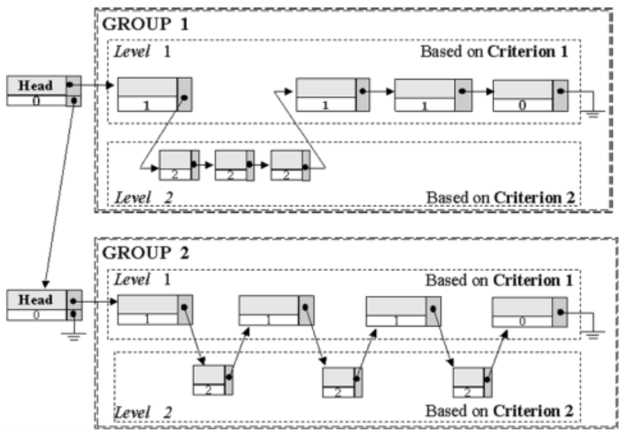


FIG. 7.—All anchors obtained in the first and the second iterations are stored in a hierarchical linking list. In this case, there are two groups of anchors and the anchors in each group are consistent and ordered.

Search for Inversions

To find inversions in either of the two sequences, we convert the complementary strand of sequence A to an inverted *x*-mer-packet numeric sequence. Since the address lookup table of sequence B has been constructed at the first stage, we can apply the table and same procedures to the inverted numeric sequence to find inversions.

Alignment of Interanchor Subsequences

At the last stage, we align the subsequences between each pair of two neighboring anchors in the same group by developing a program for the algorithm of Smith and Waterman (1981). The default scores of match, mismatch, open gap penalty, and gap extension penalty are set to 1, –1, –5, and –0.2, respectively. These scores can be adjusted to users' demand.

Results

Our program was implemented in C++, and all the following results were run on a Pentium IV (2.7 GHz) with 1G RAM under LINUX. To test our program and compare it with other programs, we used two sets of test sequence data. The first set is a pair of human and mouse genomic sequences (GenBank Accession numbers U47924 and AC002397). After removing the repeats and low-complexity regions with RepeatMasker (A. F. A. Smit and P. Green, RepeatMasker at http://ftp.genome.washington.edu/RM/RepeatMasker.html), the length of the human sequence is 142 kb and that of the mouse sequence is 157 kb. The other set of sequences were downloaded from http://baboon.math.berkeley.edu/avid/ (Bray et al. 2003) The set includes the pairs of genomic sequences from chicken/human, cat/human, cow/human, dog/human, pig/human, rat/human and chimp/human, representing various degrees of conservation (table 1). In total, there are 117 sequences with over 50 Mb. About 18 Mb of the interspersed repeats and low-complexity regions were detected in these sequences with RepeatMasker; the detected repeats were masked with lower case letters.

Identical Rate of Aligned Base Pairs

We now consider the quality of an alignment. The number of alignment anchors and the sum of unaligned lengths have been used for comparing different alignment algorithms (Delcher et al. 1999; Wang and Li, unpublished data), but they may not be good criteria for two reasons. First, in some programs, two neighboring anchors with few gaps or mismatches in between are merged into a single anchor. The number of anchors is usually smaller in such a program than those that align the subsequences between these anchors at the final stage. Second, the unaligned lengths are dependent on the parameter, the maximum aligned length ($L_{\max}$), which is the maximum length of the segments that are to be aligned between two adjacent anchors. If we set $L_{\max}$ to a larger value, the sum of unaligned lengths is usually shorter, and vice versa. However, setting $L_{\max}$ to a larger value may not lead to a good quality alignment because the quality of an alignment usually decreases with increasing $L_{\max}$. Therefore, the maximum aligned length and the quality of the alignment should be considered together.

In PipMaker (Schwartz et al. 2000), the program gives pips (percent identity plots), each of which represents the position and percent identity (from 50% to 100%) of a gap-free segment of the alignment. However, this percent identity represents only the mean value of the identities in an aligned segment. If there are two aligned segments with the same percent identity, but one is several hundreds of base pairs long and the other is, say, only 30 base pairs long, we do not know how to compare the qualities of these two aligned segments.

In Waterston et al. (2002), the alignments are filtered after they have been generated to retain only those portions of the alignments that score above a certain threshold according to a suitable scoring matrix (Bray, Dubchak, and

Table 1
The Test Sequence Data

Species	Entire Length (bp)	Total Length of Repeats	Unknown Nucleotides
Chicken[1]	1,254,651	11,058	0
Cat[2]	1,857,802	500,882	0
Cow[3]	2,573,067	965,350	0
Dog[4]	1,530,045	363,656	0
Pig[5]	8,309,185	2,365,016	0
Rat[6]	4,503,238	1,567,593	0
Chimp[7]	1,100,481	521,100	0
Human[1]	1,034,310	208,395	139,873
Human[2]	2,440,783	945,100	32,943
Human[3]	3,241,677	1,144,713	78,958
Human[4]	2,495,979	1,050,190	28,909
Human[5]	12,741,966	5,262,676	100,381
Human[6]	6,628,016	2,690,037	55,610
Human[7]	890,574	435,912	0

NOTE.—The data were downloaded from http://baboon.math.berkeley.edu/avid/ (Bray, Dubchak, and Pachter 2003). Each set of the human sequences corresponds to the sequence above with the same superscript number. There are 117 sequences with a total length of over 50 Mb. About 18 Mb of the interspersed repeats were detected by using the RepeatMasker program (http://ftp.genome.washington.edu/RM/RepeatMasker.html, Smit and Green). In addition, 1.4% of the human sequences are unknown bases marked by Ns.

Pachter 2003). This method can be used to compare the alignments obtained with a common scoring system. However, because the default scoring systems of different programs are different, it may not be logical to recalculate the scores of the alignment results of different programs according to any particular scoring matrix and filter out those alignments under a single threshold. For example, the default scoring parameters of BlastZ were match = 1, mismatch = –1, gap open = –6, and gap extension = –0.02 in Schwartz et al. (2000). However, in their recent paper (Schwartz et al. 2003), the substitution scoring matrix has been adjusted as follows:

$$\begin{array}{ccccc} & A & C & G & T \\ A & 100 & -200 & -100 & -200 \\ C & -200 & 100 & -200 & -100 \\ G & -100 & -200 & 100 & -200 \\ T & -200 & -100 & -200 & 100 \end{array}$$

The gap open score is –2000, and the gap extension score is –50. In contrast, the default scoring parameters used in MUMmer were match = 1 and the other penalties, including mismatch and gap, were zeroes. The scales of these scoring systems are very different. Thus, we did not use this approach to compare our result with the other programs. In VISTA, Mayor et al. (2000) also defined a conserved region with a percentage and a length cutoff. Conserved segments with percent identity x and length y are defined to be regions in which every contiguous subsegment of length y was at least x% identical to its paired sequence. These segments are merged to define the conserved regions. In what follows, we define the identical rate per aligned base pair and use it as a measure of the quality of an alignment. Although the definition is almost the same as a conserved region defined by Mayor et al. (2000), we do not set the cutoff threshold to filter out the lower conserved regions. Moreover, we use it as a measure of the quality of an alignment and also consider its statistical distribution.

Let $S(i)$ be the ith position of alignment S. The identical rate per aligned base pair is defined as the proportion of the matching (identical) pairs in an aligned segment with $S(i)$ at the center of the segment; in this paper, the default length of the segment is set to 21. (Note that it is not suitable to use a long segment; that is, a large window size, because the number of windows may become very small.) Thus, the identical rate does not refer to a particular site, but to the average similarity of the sequence segment. If the identical rate is close to 1, the region is very highly conserved. On the other hand, if the rate is below 25%, the region contains mostly gaps, because even for an alignment of two random sequences, the identical rate is approximately 25% if the four nucleotides are about equally frequent.

Since the length of the segment is set to 21, all possible calculated identical rates are from 0% to 100% with a step of 5%. Thus, we can accumulate the number of all $S(i)$ values with the same identical rate over the alignment and obtain a distribution in which the X-axis represents the identical rate and the Y-axis represents the number of base pairs with the same identical rate. This distribution is called the total number of sites for each identical rate. If the two sequences are highly conserved, the peak of the distribution will be very close to the right end (100%). In the remaining regions, the distribution will be close to zero. If numerous successive gaps are inserted into one of the two aligned sequences, the peak of the distribution with highly identical rates will still be close to the right end, but a second peak of the distribution will appear at the left end (0%). Thus, in an alignment of two highly homologous sequences, the distribution could have two peaks, one close to 1 and the other close to 0.

In addition, we also calculate the percentage of aligned sites for each identical rate and call this normalized distribution the accumulated percentage for each identical rate. In most cases, finding a relatively small number of highly conserved regions or a large number of moderately or highly conserved regions is a trade-off problem. Thus, we can analyze the quantity and quality of the final aligned result based on the distributions of the total number and the percentage of aligned sites for each identical rate, respectively.

Alignment Anchors

In GS-Aligner, the number of allowed mismatches within an 8-mer extension and the minimum score of an acceptable anchor are two parameters used at the stage of finding anchors, and the maximum aligned length ($L_{\max}$) is the parameter used at the final stage for aligning interanchor subsequence pairs; the scoring system is used at both stages. At the stage of finding anchors, the scores of a match and a mismatch are set to 1 and –1, respectively. Since we do not allow any gap within an anchor, there is no penalty at this stage for gap opening and extension. The allowed mismatches within an 8-mer extension are set to 2 and 3 at the first iteration and the second iteration, respectively. Thus, the average identical rate of each candidate anchor is expected to be at least 75% in the first iteration and 62.5% in the second iteration.

How to decide the minimum score of an acceptable anchor is based on the computational result obtained by Wang and Li (unpublished results). In their analysis, the expected number of matching pairs that appear by chance with a score over 30 is only 0.0001, when aligning two DNA sequences that are both 100 kb long. In our case, the minimum scores of an acceptable anchor are set to 35 and 25 at the first iteration and the second iteration, respectively. At the stage for the final alignment, the default scores of match, mismatch, open gap penalty, and gap extension penalty are set to 1, –1, –5, and –0.2, respectively. The maximum aligned length is set to several values ($L_{max} = 100$, 1,000, and 5,000) for comparison (see below). All parameters, of course, can be adjusted by users themselves to suit their purpose.

Applying our program to the first set of test sequences, we found 372 anchors, of which 239 were found at the first iteration and 133 were found at the second iteration. The total length of the anchors is 31,360 bp, of which 80% are obtained at the first iteration and 20% are obtained at the second iteration. The identical rates of the anchors at the first iteration and the second iteration are 89% and 81%, respectively. The average identical rate of all the anchors is 85%.

Total Numbers of Aligned Sites for Different Identical Rates

Figure 8*a* shows the total number of sites on the vertical axis and the identical rate on the horizontal axis for three different maximum aligned lengths (L_{max} = 100, 1,000, and 5,000). We can see that the total numbers of aligned sites with identical rates over 85% are almost the same for different lengths of L_{max}, whereas the total number of sites with identical rates less than 85% varies with L_{max}. Thus, if we want to increase the number of conserved regions that do not have a high identical rate (e.g., under 80% but above 40%), the L_{max} has to be set to a larger one. We consider an identical rate of 40% as acceptable, but an alignment segment with an average identical rate lower than 40% should be taken with caution. When L_{max} is set to a large value, there may be many aligned regions with an average identical rate lower than 40%. One should be very cautious in accepting such regions. Figure 8*b* shows the accumulated percentage for each identical rate of figure 8*a*. The quality of the aligned sites by using L_{max} = 100 is better than the other ones because over 70% of its aligned sites have an identical rates higher than 80%. However, we can see from figure 8*a* that it has the smallest number of sites with an identical rate higher than 40%. Thus, we should consider not only the quality but also the ‘‘quantity’’ of an alignment.

Comparison with Other Methods

Since we focused on aligning two fairly divergent genomic sequences in this paper, the ability to align less conserved regions and the execution time are two main issues we are concerned with when we compare the performance of our program with others. To date, several alignment tools have been proposed in the literature. They include MUMmer (Delcher et al. 1999), Glass (Batzoglou et al. 2000), BlastZ (Zhang et al. 2000), WABA (Kent and Zahler 2000), SSAHA (Ning, Cox, and Mullikin 2001), Dialign (Morgenstern et al. 2002), Blat (Kent 2002), and Avid (Bray, Dubchak, and Pachter 2003). Blat and SSAHA were designed for searching large databases, so we did not include them in our comparison. Glass, WABA, and Dialign are not time efficient for our test data because these programs could not complete the alignments within a few hours. Therefore, only MUMmer, BlastZ, and Avid are compared with our program (GS-Aligner) below.

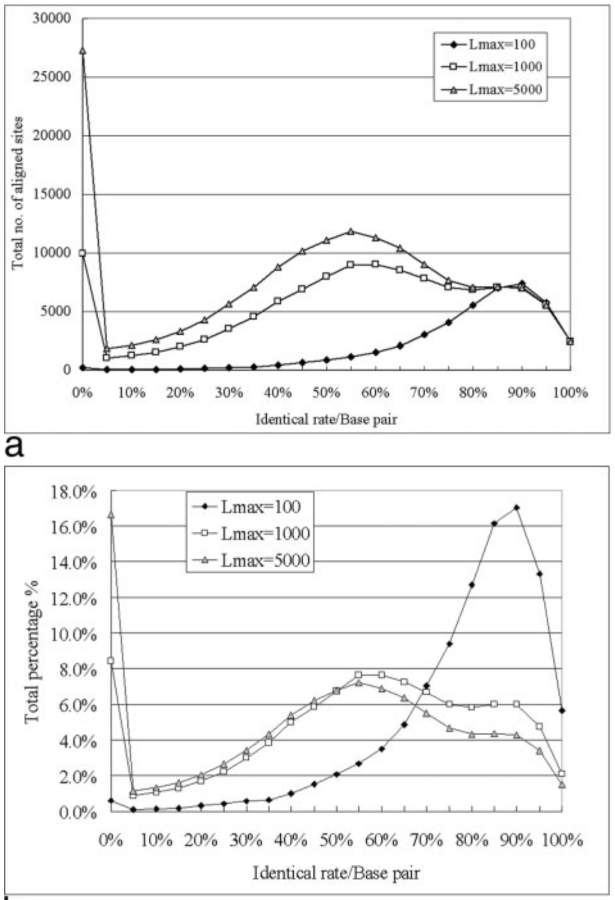
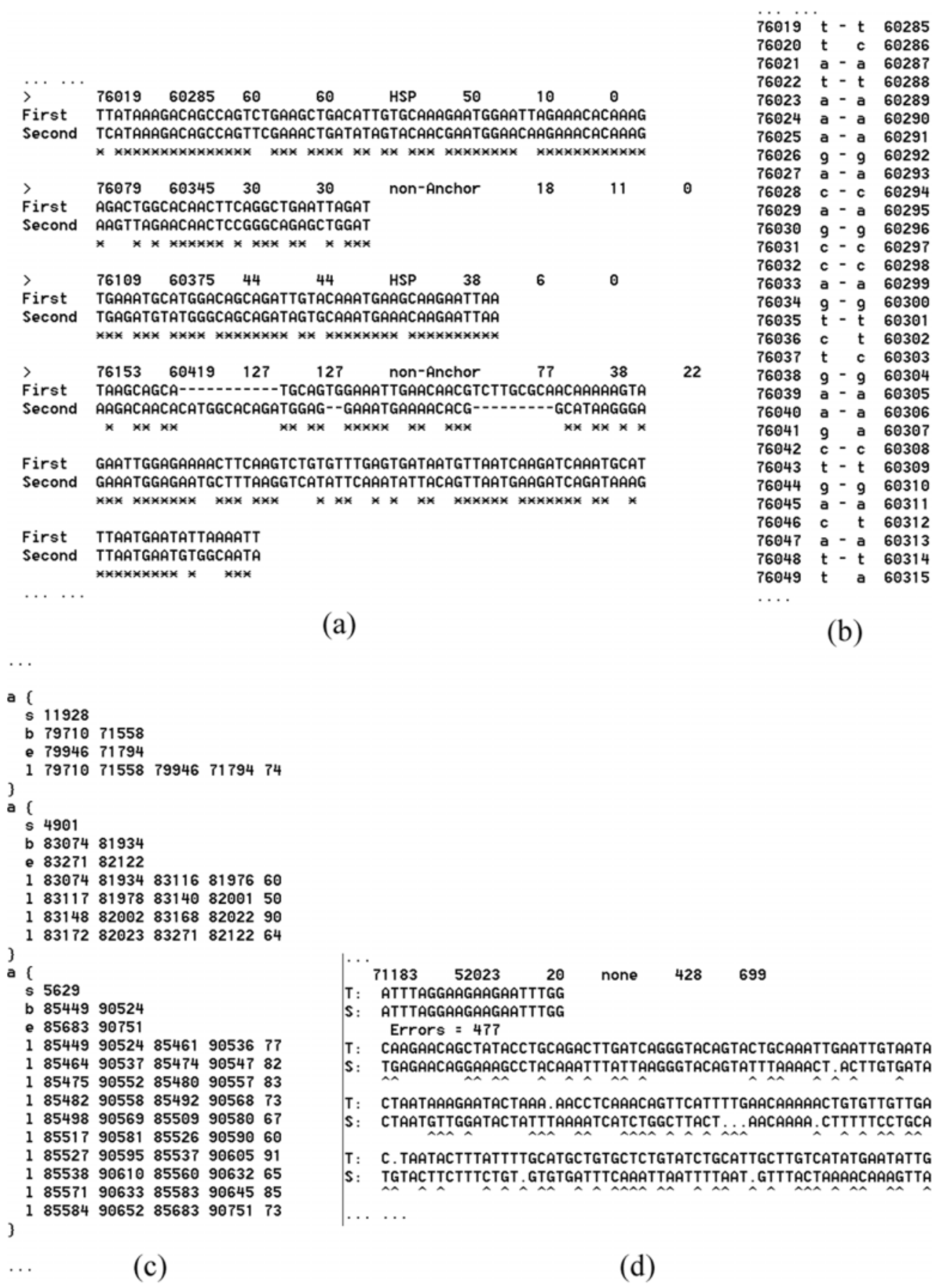


FIG. 8.—(*a*) The total number of sites versus the identical rate with different L_{max}'s for GS-Aligner. (*b*) The total percentage diagram of (*a*).

In this paper, we used only input sequences that have already been repeat-masked. (In Bray, Dubchak, and Pachter [2003], the input sequences to BlastZ were not masked for repeats, so all the execution times for BlastZ were much longer than the ones shown in this paper.) In addition, because the alignment results for different programs were in different formats (see fig. 9), we converted the results from different programs into a consistent format, so that we could calculate their total numbers of aligned sites for different identical rates. Table 2 summarizes the execution times of different programs for



```
... ...
>       76019   60285   60      60      HSP     50      10      0
First   TTATAAAGACAGCCAGTCTGAAGCTGACATTGTGCAAAGAATGGAATTAGAAACACAAAG
Second  TCATAAAGACAGCCAGTTCGAAACTGATATAGTACAACGAATGGAACAAGAAACACAAAG
        * ***************  *** **** ** ** *** ********  ************

>       76079   60345   30      30      non-Anchor      18      11      0
First   AGACTGGCACAACTTCAGGCTGAATTAGAT
Second  AAGTTAGAACAACTCCGGGCAGAGCTGGAT
        *   * * ****** * *** **  * ***

>       76109   60375   44      44      HSP     38      6       0
First   TGAAATGCATGGACAGCAGATTGTACAAATGAAGCAAGAATTAA
Second  TGAGATGTATGGGCAGCAGATAGTGCAAATGAAACAAGAATTAA
        *** *** **** ******** ** ******** **********

>       76153   60419   127     127     non-Anchor      77      38      22
First   TAAGCAGCA-----------TGCAGTGGAAATTGAACAACGTCTTGCGCAACAAAAAGTA
Second  AAGACAACACATGGCACAGATGGAG--GAAATGAAAACACG---------GCATAAGGGA
         *  ** **           ** **  *****  **  ***          ** ** * *

First   GAATTGGAGAAAACTTCAAGTCTGTGTTTGAGTGATAATGTTAATCAAGATCAAATGCAT
Second  GAAATGGAGAATGCTTTAAGGTCATATTCAAATATTACAGTTAATGAAGATCAGATAAAG
        *** *******  *** ***    * **  * *  **  ****** ******* **  *

First   TTAATGAATATTAAAATT
Second  TTAATGAATGTGGCAATA
        ********* *   ***
... ...
```

(a)

```
... ...
76019  t - t  60285
76020  t   c  60286
76021  a - a  60287
76022  t - t  60288
76023  a - a  60289
76024  a - a  60290
76025  a - a  60291
76026  g - g  60292
76027  a - a  60293
76028  c - c  60294
76029  a - a  60295
76030  g - g  60296
76031  c - c  60297
76032  c - c  60298
76033  a - a  60299
76034  g - g  60300
76035  t - t  60301
76036  c   t  60302
76037  t   c  60303
76038  g - g  60304
76039  a - a  60305
76040  a - a  60306
76041  g   a  60307
76042  c - c  60308
76043  t - t  60309
76044  g - g  60310
76045  a - a  60311
76046  c   t  60312
76047  a - a  60313
76048  t - t  60314
76049  t   a  60315
....
```

(b)

```
...

a {
  s 11928
  b 79710 71558
  e 79946 71794
  l 79710 71558 79946 71794 74
}
a {
  s 4901
  b 83074 81934
  e 83271 82122
  l 83074 81934 83116 81976 60
  l 83117 81978 83140 82001 50
  l 83148 82002 83168 82022 90
  l 83172 82023 83271 82122 64
}
a {
  s 5629
  b 85449 90524
  e 85683 90751
  l 85449 90524 85461 90536 77
  l 85464 90537 85474 90547 82
  l 85475 90552 85480 90557 83
  l 85482 90558 85492 90568 73
  l 85498 90569 85509 90580 67
  l 85517 90581 85526 90590 60
  l 85527 90595 85537 90605 91
  l 85538 90610 85560 90632 65
  l 85571 90633 85583 90645 85
  l 85584 90652 85683 90751 73
}
...
```

(c)

```
...
   71183    52023     20    none    428    699
T:  ATTTAGGAAGAAGAATTTGG
S:  ATTTAGGAAGAAGAATTTGG
     Errors = 477
T:  CAAGAACAGCTATACCTGCAGACTTGATCAGGGTACAGTACTGCAAATTGAATTGTAATA
S:  TGAGAACAGGAAAGCCTACAAATTTATTAAGGGTACAGTATTTAAAACT.ACTTGTGATA
    ^^       ^^ ^^   ^  ^ ^  ^^ ^           ^ ^^   ^ ^ ^    ^

T:  CTAATAAAGAATACTAAA.AACCTCAAACAGTTCATTTTGAACAAAAACTGTGTTGTTGA
S:  CTAATGTTGGATACTATTTAAAATCATCTGGCTTACT...AACAAAA.CTTTTTCCTGCA
         ^^^ ^      ^^^  ^^   ^^^^ ^ ^ ^ ^^^       ^  ^ ^ ^^ ^^

T:  C.TAATACTTTATTTTGCATGCTGTGCTCTGTATCTGCATTGCTTGTCATATGAATATTG
S:  TGTACTTCTTTCTGT.GTGTGATTTCAAATTAATTTTAAT.GTTTACTAAAACAAAGTTA
    ^^  ^ ^    ^ ^ ^ ^^  ^ ^ ^^^^ ^^  ^ ^^  ^ ^  ^^^ ^ ^^  ^^  ^

... ...
```

(d)

FIG. 9.—Different formats of the final alignment results obtained using different programs. (*a*) GS-Aligner; (*b*) Avid; (c) BlastZ; (*d*) MUMmer.

aligning the nonhuman sequences with the human sequences. All programs were executed by using the default parameters; for BlastZ we also set the parameter *c* to 2 since turning the chaining opinion on can speedup its execution time (Bray, Dubchak, and Pachter 2003). In the comparisons of the cat and cow sequences with the human sequences, the execution times of GS-Aligner are slightly shorter than the others, but, for the other species, either MUMmer is faster or BlastZ is faster. However, it should be noted that BlastZ is only a local alignment program because it searches for local similarity and gives alignments of local segments but does not sort the segments into consistent orders. Skipping the sorting step is expected to save some execution time. MUMmer is a fast program for closely related sequences such as human versus chimpanzee sequences. On average, we cannot say which program is the fastest except that for Avid, the execution times of each species pair are almost twice those for the other programs. Note that as long as the differences in execution time are not large, the execution time should not matter much because the major determining factors should be the quality and quantity of alignment, which are

Table 2
The Execution Times Required by Different Alignment Programs for Aligning a Set of Sequence Pairs from a Nonhuman Species to the Human Sequences

	BlastZ	BlastZ (c = 2)	Avid	MUMmer	GS-Aligner
Chicken	8.81	8.90	31.18	5.81	16.89
Cat	41.92	40.95	88.45	43.45	36.1
Cow	49.28	46.55	125.71	49.77	43.39
Dog	41.76	38.62	112.13	39.42	38.81
Pig	172.89	160.97	497.08	151.16	214.08
Rat	83.57	76.17	248.99	65.77	126.18
Chimp	17.1	11.14	77.04	4.14	12.43

NOTE.—The unit of execution time is a second. In BlastZ, we used not only the default setting but also the chaining option (c = 2) for speeding up the performance (Bray, Dubchak, and Pachter 2003). The other methods were executed with their default setting. Comparing our results with those of Bray, Dubchak, and Pachter (2003), we found some differences. In their results, the execution times for BlastZ were very long because the input sequences to the program were unmasked for the repeats.

the ultimate goal of sequence alignment and will be discussed below.

Figure 10 shows the total number of aligned sites for all species pair sets with different programs. Obviously, the total number of aligned sites for Avid is more than the total number of aligned sites for the other programs. This is not surprising because Avid is an end-to-end global alignment program; that is, it aligns the sequence pair from the head to the end. Therefore, more sites are aligned. However, the alignment also includes low-quality regions (i.e., regions with identical rates lower than 25%). We shall therefore no longer consider Avid here (but see the next paragraph). Note that the number of aligned sites with an identical rate under 65% but above 40% obtained by using GS-Aligner is larger than the number of such sites for MUMmer, BlastZ, and BlastZ (c = 2), although the number of sites with above 65% for GS-Aligner is slightly less than that obtained by BlastZ.

In table 3, we calculated the accumulation of the total number of aligned sites with an identical rate ≥40% for each species pair set aligned by different programs. For the chicken-human comparison, GS-Aligner has the largest

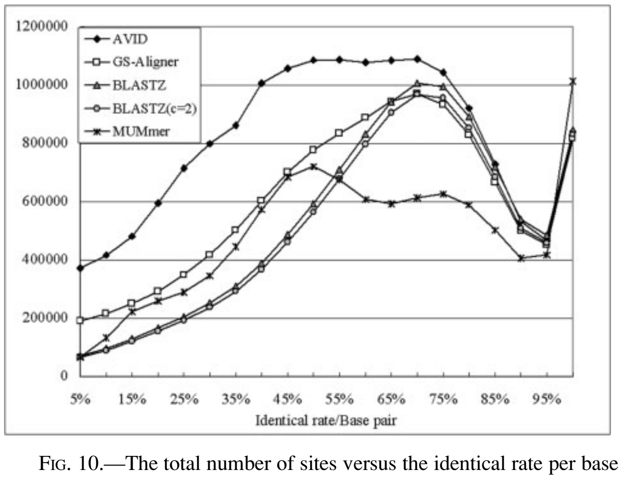


FIG. 10.—The total number of sites versus the identical rate per base pair for different alignment programs and the entire data set in table 1.

Table 3
The Total Numbers of Sites with an Identical Rate ≥40% for Different Sets of Species Pairs by Different Alignment Programs

	BlastZ	BlastZ (c = 2)	Avid	MUMmer	GS-Aligner
Chicken	53,787	53,787	286,692	36,079	132,289
Cat	1,138,689	1,128,986	1,298,260	1,115,350	1,136,062
Cow	1,155,931	1,115,987	1,334,115	1,057,653	1,161,122
Dog	985,879	915,678	1,088,586	825,193	962,171
Pig	3,581,145	3,463,908	4,784,860	2,974,904	3,867,111
Rat	1,550,491	1,492,215	2,363,081	1,114,516	1,795,581
Chimp	962,821	860,028	857,202	893,869	855,460
Total	9,428,743	9,030,589	12,012,796	8,017,564	9,909,796

number of aligned sites than the other programs (except for Avid, which, as mentioned above, will no longer be discussed until the next paragraph). For the cat-human, cow-human, dog-human, pig-human, and rat-human comparison, the total number of aligned sites with ≥40% identical rate is the highest for GS-Aligner (with a total of 8,922,047 sites), second highest for BlastZ (8,412,135), third highest for BlastZ (c = 2) (8,116,774), and lowest for MUMmer (7,087,616). Thus, for fairly divergent sequences such as those between mammalian orders or between a mammal and a nonmammal vertebrate, GS-Aligner seems to perform better than the other programs. For closely related sequences such as chimpanzee versus human sequences, BlastZ (without the chaining option) aligned more sites than the other programs (table 3). However, a close examination revealed that the some aligned segments were highly overlapping. These ambiguous regions disappeared and the total number of aligned sites with ≥40% identical rate was decreased by ~10% (almost 100k) when the chaining option was turned on. Thus, BlastZ did not perform better than GS-Aligner even for the chimpanzee-human pair. For the chimpanzee-human pair MUMmer aligned slightly more sites than GS-Aligner, but many segments aligned by MUMmer contained fragments of repeat elements because MUMmer aligned repeats masked into lowercase letters. Therefore, overall GS-Aligner appeared to perform better than the other program compared.

We now consider the question of alignment reliability. The most common approach is to calculate the significance values (*P*-value or *E*-value) of each aligned segment pair (Altschul et al. 1991). However, how to choose a scoring system for a fair comparison is still an open problem. We therefore did a simple comparison as follows. We generated three pairs of random sequences, each with 100 kp, to test the programs under comparison. As expected, no alignment results were obtained by using MUMmer, BlastZ, or GS-Aligner, but unexpectedly, Avid could align each random sequence pair and reported end-to-end alignments. (The parameter with Avid used in this case is *–nm = both*, which specifies the program not to mask the sequences.) Indeed, the total number of aligned sites with an identical rate above 50% is over 10,000 bp in each case (fig. 11). Thus, how to justify the alignment results obtained by Avid is a serious problem. This result reflects the fact that a global (end-to-end) alignment of

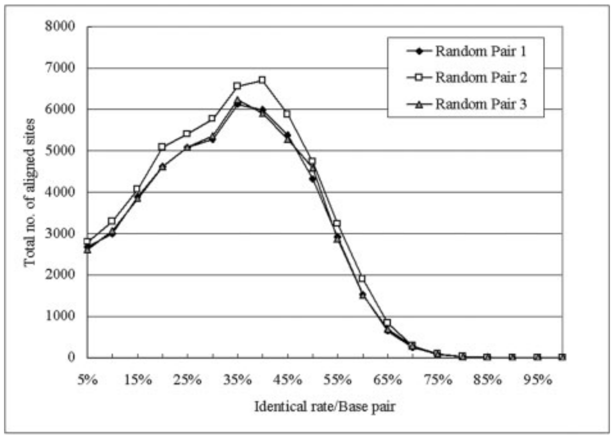


FIG. 11.—The total number of sites versus the identical rate for the alignments of three random sequence pairs obtained by Avid.

genomic sequences is doomed to frequently aligned unrelated regions and, worse, to produce misleading results (Miller 2001). Why did the anchor-based programs, such as MUMmer and GS-Aligner, not align the random sequence pairs? The reason is that the anchors found in these programs are usually long enough so that few occur by chance, even in a long, random sequence pair. Accordingly, no alignment was reported in the above test cases.

Discussion

Aligning genomic sequences is often an essential step in comparative genomics. There are two major challenges. First, the sequences to be aligned are usually very long, so methods based on dynamic programming cannot effectively solve the problem within reasonable time and space complexity. Second, the sequences may not have been well conserved, but finding conserved regions with the identical rates above ~40% is necessary. In this paper, we proposed a novel program for aligning genomic sequences. The results show that our program can be used to align divergent sequences effectively and rapidly. However, alignment trade-off is still a difficult problem we have to solve. If we want to obtain a large number of conserved regions, the trade-off is that more unrelated regions will be aligned, too. In figure 8, the aligned regions with identical rates less than 40% are almost half of the amount of aligned regions. Thus, there is a need to develop criteria for judging what local alignments with an identical rate less than 40% are acceptable.

Acknowledgments

We thank Anton Nekrutenko, Haidong Wang, and two reviewers for their valuable suggestions and Wen-Yu Chung for maintenance of the program. This study was supported by NIH grants GM30998 and GM66104 and by Academia Sinica, Taiwan.

Literature Cited

Altschul, S. F., W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. 1990. Basic local alignment search tool. J. Mol. Biol. **215**:403–410.

Altschul, S. F., T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. 1997. Gapped Blast and PSI-Blast: a new generation of protein database search programs. Nucleic Acids Res. **25**:3389–3402.

Ansari-Lari, M. A., J. C. Oeltjen, S. Schwartz et al. (11 co-authors). 1998. Comparative sequences analysis of a gene-rich cluster at human chromosome 12p13 and its syntenic region in mouse chromosome 6. Genome Res. **8**: 29–40.

Bailey, J. A., A. M. Yavor, H. F. Massa, B. J. Trask, and E. E. Eichler. 2001. Segmental duplications: organization and impact within the current human genome project assembly. Genome Res. **11**:1005–1017.

Batzoglou, S., L. Pachter, J. P. Mesirov, B. Berger, and E. S. Lander. 2000. Human and mouse gene structure: comparative analysis and application to exon prediction. Genome Res. **10**:950–958.

Bray, N., I. Dubchak, and L. Pachter. 2003. Avid: a global alignment program. Genome Res. **13**:97–102.

Delcher, A. L., S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg. 1999. Alignment of whole genomes. Nucleic Acids Res. **27**:2369–2376.

Dubchak, I., M. Brudno, G. G. Loots, L. Pachter, C. Mayor, E. M. Rubin, and K. A. Frazer. 2000. Active conservation of noncoding sequences revealed by three-way species comparisons. Genome Res. **10**:1304–1306.

Göttgens, B., J. G. Gilbert, L. M. Barton, D. Grafham, J. Rogers, D. R. Bentley, and A. R. Green. 2001. Long-range comparison of human and mouse SCL loci: localized regions of sensitivity to restriction endonucleases correspond precisely with peaks of conserved noncoding sequences. Genome Res. **11**:87–97.

Jareborg, N., E. Birney, and R. Durbin. 1999. Comparative analysis of noncoding regions of 77 orthologous mouse and human gene pairs. Genome Res. **9**:815–824.

Lund, J., F. Chen, A. Hua, B. Roe, M. Budarf, B. S. Emanuel, and R. H. Reeves. 2000. Comparative sequences analysis of 634kb of the mouse chromosome 16 region of conserved synteny with the human velocardiofacial syndrome region on chromosome 22q11.2. Genomics **63**:374–383.

Kent, W. J. 2002. Blat—the Blast-like alignment tool. Genome Res. **12**:656–664.

Kent, W. J., and A. M. Zahler. 2000. Conservation, regulation, synteny, and introns in a large-scale C. briggsae-C. elegans genomic alignment. Genome Res. **10**:1115–1125.

Mallon, A.-M., M. Platzer, R. Bate et al. (31 co-authors). 2000. Comparative genome sequence analysis of the Bpa/Str region in mouse and man. Genome Res. **10**:758–775.

Mayor C., M. Brudno, J. R. Schwartz, A. Poliakov, E. M. Rubin, K. A. Frazer, L. S. Pachter, and I. Dubchak. 2000. VISTA: visualizing global DNA sequence alignments of arbitrary length. Bioinformatics **16**:1046–1047.

Miller, W. 2001. Comparsion of genomic DNA sequences: solved and unsolved problems. Bioinformatics **17**:391–397.

Morgenstern, B., O. Rinner, S. Abdeddaïm, D. Haase, K. F. X. Mayer, A. W. M. Dress, and H.-W. Mewes. 2002. Exon discovery by genomic sequence alignment. Bioinformatics **18**:777–787.

Needleman, S., and C. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**:443–453.

Ning, Z., A. J. Cox, and J. C. Mullikin. 2001. SSAHA: a fast search method for large DNA databases. Genome Res. **11**: 1725–1729.

Pearson, W. R., and D. J. Lipman. 1988. Improved tools for biological sequence comparison. Proc. Natl. Acad. Sci. USA **85**:2444–2448.

Schwartz, S., W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. 2003. Human-mouse alignments with BlastZ. Genome Res. **13**:103–107.

Schwartz, S., Z. Zhang, K. A. Frazer, A. Smit, C. Riemer, J. Bouck, R. Gibbs, R. Hardison, and W. Miller. 2000. PipMaker—a web server for aligning two genomic DNA sequences. Genome Res. **10**:577–586.

Smith, T., and M. Waterman. 1981. Identification of common molecular subsequences. J. Mol. Biol. **147**:195–197.

Waterston, R. H., K. Lindblad-Toh, E. Birney et al. (222 co-authors). 2002. Initial sequencing and comparative analysis of the mouse genome. Nature **420**:520–562.

Wilson, M. D., C. Riemer, D. W. Martindale et al. (12 co-authors). 2001. Comparative analysis of the gene-dense ACHE/TFR2 region on human chromosome 7q22 with the orthologous region on mouse chromosome 5. Nucleic Acids Res. **29**:1352–1365.

Zhang, Z., S. Schwartz, L. Wagner, and W. Miller. 2000. A greedy algorithm for aligning DNA sequences. J. Comput. Biol. **7**:203–214.

Naruya Saitou, Associate Editor

Accepted April 1, 2003