A Fast Method for Linear Space Pairwise Sequence Alignment

Chin-Chen Chang, Yu-Chiang Li, and Chia-Te Liao
Department of Computer Science and Information Engineering,
National Chung Cheng University, Chiayi, Taiwan 621, R.O.C.
E-mail:{ccc, lyc, lct90}@cs.ccu.edu.tw

Abstract: Pairwise sequence alignment is an important technique for finding the optimal arrangement between two sequences. A basic dynamic-programming strategy for sequence alignment needs $O(mn)$ time and also $O(mn)$ space. Hirschberg's divide-and-conquer method reduces the required space to roughly $2m$ ($m \leq n$), but it also doubles the computing time. On the other hand, the FastLSA approach adds extra rows and columns to generalize Hirschberg's algorithm, reducing the number of recomputations at the cost of more memory space. In this paper, we shall present an efficient linear space algorithm, called the NFLSA algorithm, to reduce the ratio of recalculation greater than FastLSA while using the same memory. The NFLSA algorithm is superior to Hirschberg's algorithm and the FastLSA algorithm in our simulations.

Keywords: sequence alignment, DNA, dynamic programming, Hirschberg's algorithm, linear space.

1. Introduction

The *nucleic* acid or *protein* sequences alignment is one of the most important research topics to explore in computational biology today. Each DNA or protein sequence is represented by a series of alphabetic characters. For example, by letters A, T, C and G, genetic biologists refer to the four nucleic acids that the DNA is composed of. Similarly, the Greek alphabet Σ is used to denote any finite set of letters, and any string s of letters from Σ is called a sequence over Σ, while $|s|$ denotes the length of s. Assume that we have two sequences s and t to align. In most cases, the lengths of s and t are different. For better comparability, we use gaps (denoted by dashes "–") inserted into s or t such that the two resulting sequences s^* and t^* have the same length. Writing down the sequences s^* and t^* one above the other, similarities and differences are easy to obtain, where $s^*, t^* \in \Sigma \cup \{-\}$ and $|s^*| \leq |s| + |t|$.

An alignment exhibits where the two sequences are similar to each other and where they differ. An optimal alignment maximizes the total match value or minimizes the total cost of mismatches and then inserts or deletes gaps. A simple similarity scoring function assigns +2 for a match, –1 for a mismatch, and –2 for each gap (called *gap penalty*). For example, consider the two strings s=CGTGTA and t=CAGGA. To obtain the maximum number of matching positions, gaps can be inserted in appropriate places as follows.

C– GTGTA
CAG– G–A

The aligned sequences match in four positions when three gaps are inserted and have the score +2 (4 matches and 3 gaps). The gap penalty is also accomplished by charging $g + ek$ for a gap of length k. The *gap-opening penalty* g is fixed for every gap, regardless of gap lengths, and e is the *gap-extension penalty* for every sequence entry in the gap [10]. Such penalties are called *affine gap penalties*. This paper uses the simplification of affine gap costs where $g = 0$.

Using naive dynamic programming to obtain the optimal two-sequence alignment requires the array size of $O(mn)$, where m and n are two lengths of two sequences, and the time complexity is quadratic. Unfortunately, a DNA sequence can be so long that the memory space of O(mn) is obviously prohibitive. For example, two 20,000-character-long sequences require 400million storage entries. Hirschberg [11] proposed the divide-and-conquer approach that performs alignment in linear space to reduce the demand on memory space, only to double the computation time. On the other hand, Charter *et al*. [6] presented the so-called FastLSA algorithm to generalize Hirschberg's algorithm. The FastLSA algorithm allows the consumption of more memory to reduce the recomputation numbers by including extra rows and columns.

In this paper, we shall propose a new fast linear space sequence alignment algorithm called *NFLSA*. The NFLSA algorithm, like the FastLSA algorithm, repeatedly subdivides each sequence into two blocks, but it differs from the FastLSA algorithm because the two blocks cannot equal to each other. By storing the middle row and column recursively, the NFLSA approach can reduce the recomputation complexity. Our experimental results show that the NFLSA is superior to Hirschberg's algorithm and also outperform the FastLSA approach with the same storage memory, especially when the two sequences have similar lengths.

The reminder of this paper is organized as follows. In Section 2, we shall present two previous linear space sequence alignments, Hirschberg's algorithm and FastLSA. Then, in Section 3, we shall propose our new fast linear space sequence alignment, called NFLSA. The experimental results and performance evaluation of the proposed algorithm will be presented in Section 4. Finally, we shall give our conclusions in Section 5.

2. Related Works

Needleman and Wunsch [16] proposed a dynamic programming method to solve the alignment problem. Smith and Waterman [20] presented a modification of alignment algorithm to find a highest scoring alignment of arbitrary regions in the two sequences. This method consists of two parts. It requires a matrix, also called the *full-matrix* algorithm. The first part, the *Fill-Score* phase, is responsible for filling up the full matrix from the top-left corner to the bottom-right corner by using a simple scoring function. A simple similarity function pseudo-code for matrix computation will be shown later. Figure 1 shows all the scores in the full matrix and the backtracking paths for *s*=CGTGTA and *t*=CAGGA. The scores get propagated from left to right horizontally, top to down vertically and top-left to bottom-right diagonally. After that, the second part of the algorithm, namely the *TrackPath* phase, is responsible for backtracking from the bottom-right corner to the top-left corner to find the optimal paths. A diagonal path means a match, a horizontal path needs the insertion of a gap in the left-hand side sequence, and a vertical path needs the insertion of a gap in the top sequence. The following strings are all optimal alignments in Figure 1. They all have the maximum score +2.

C–GTGTA CGTGTA CGTGTA CGTGTA
CAG–G–A CAGG–A CA–GGA C–AGGA

	–	C	G	T	G	T	A
–	0	–2	–4	–6	–8	–10	–12
C	–2	2	0	-2	-4	-6	-8
A	–4	0	1	-1	-3	-5	-4
G	–6	-2	2	0	1	-1	-3
G	–8	-4	0	1	2	0	-2
A	–10	-6	-2	-1	0	1	2

FIG. 1. Alignment matrix and optimal paths for *s*=CGTGTA and *t*=CAGGA

A basic two-sequence dynamic algorithm pseudo-code for computing the matrix is stated as following:

```
Similarity(s, t)
1.  m := |s|
2.  n   := |t|
3.  for ( i := 0; i <= m; i++)
4.      a[i, 0] := i*Gap_Penalty
5.  for ( j := 1; j <= n; j++)
6.      a[0, j] := j*Gap_Penalty
7.  for ( i := 1; i <= m; i++)
8.      for ( j := 1; j <= n; j++)
9.          a[i, j] := max( a[i-1, j] + Gap_Penalty,
                           a[i-1, j-1] + Score(s[i], t[j]),
                           a[i, j-1] + Gap_Penalty )
10. return a[m, n]
```

The full-matrix method requires $O(mn)$ in both time and space. Unsatisfied with the time and space the full-matrix method calls for, researchers came to the ideas of faster algorithms. Therefore, some greedy techniques have been proposed [9, 22, 23]. Usually, the space limitation is a priority over the time constraint, especially for long sequences or multiple sequences alignment. If we generalize the full-matrix algorithm into multiple sequences alignment, the time and space complexities will both become $O(n^d)$, where n is the sequence length and d is the number of sequences. In this case, even the main memory is not enough for a few sequences of short lengths. So far, several papers have presented strategies that reduce space consumption [3, 5-7, 11-13, 15]. For multiple sequences alignment, most researchers consider finding heuristic methods [1, 14, 19, 21] or approximation algorithms [2, 17, 18].

To reduce the space requirement, Hirschberg [11] proposed the first linear-space algorithm for sequence alignment. First, he uses two rows to compute the score of the alignment with ease, since the score can be derived from the current and the previous row at any time, stopping at the middle row (row r). Afterwards, he reverses the procedure and starts with the end row, again stopping at the middle row (row r + 1). Therefore, the optimal score can be found by picking out the maximum sum between the two middle rows and locating the point O_a on the optimal path as shown in Figure 2. Then, the problem is divided into two sub-problems half the size of the original problem. The algorithm recursively divides the two sub-problems into four sub-sub-problems and obtains the two points (O_b and O_c as shown in Figure 2) on the optimal path. Similarly, the total size of all the sub-sub-problems is a fourth of the original size. Hirschberg's algorithm is round $2mn$ cell computations, and the space complexity is $2m$ ($m \leq$

n). Figure 2 shows the problem split into two sub-problems and further split into four sub-sub-problems. The curve indicates the final optimal alignment path.

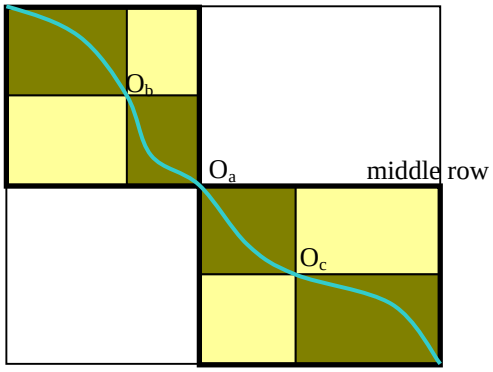


FIG. 2. Hirschberg's algorithm splits the problem into sub-problems

Hirschberg's algorithm works well for affine gap penalties [15]. Many researches have been built on the basis of Hirschberg's algorithm. For example, Eppstein *et al*. [8] combined a sparse dynamic programming algorithm with Hirschberg's algorithm, and Myers and Miller [15] also made use of it to develop a linear-space version of full-matrix algorithm.

Chao *et al*. [5] also divided a problem into several sub-problems, but they did not follow Hirschberg's equal-height sub-problem method. Chao and Miller [3] proposed a method that constructs some best nonintersecting alignment from fragments in linear space. Besides, Chao *et al*. [4] also provided an overview of linear-space sequence alignment algorithms.

Korf and Zhang [13] introduced a new reduced-spaces sequence alignment algorithm called *divide-and-conquer frontier A** (*DCFA**). This algorithm eliminates portions of the full matrix that cannot be the optimal alignment. However, the DCFA* method is slower than Hirschberg's algorithm. Later, Davidson [7] developed his ideas from Hirschberg's algorithm and DCFA* to accelerate the performance.

In 2000, Charter *et al*. [6] introduced the FastLSA algorithm. The FastLSA algorithm works by bisecting both rows and columns and saving the middle row and column. By doing so, the alignment can be done with fewer values recalculated. In Figure 3(a), 3/4 of the matrix (areas A, B and C) is computed, and the middle row and column stored. If the sub-problem size is smaller than the threshold *TH*, then the full matrix can directly be used to solve it. The last quarter of the matrix (dash line rectangle D) is solved recursively. To backtrack the optimal path, the FastLSA approach uses the stored rows and columns only and thus recalculates fewer values than Hirschberg's algorithm. In Figure 3(b), the shadow area and the dash line rectangle area require recursive recalculation in the TrackPath phase. The algorithm can divide each dimension of the matrix into k ($k \geq 2$) equal sub-problems, not restricted to bisection. The FastLSA requires as much space as $k(m+n)$ and mn/k recomputations for the alignment of two sequences m and n in length, respectively. The algorithm is superior to Hirschberg's algorithm.

To save the memory requirement and reduce the number of recalculation, in this paper, we focus on the recalculation ratio against full matrix by using linear space such as the FastLSA algorithm.

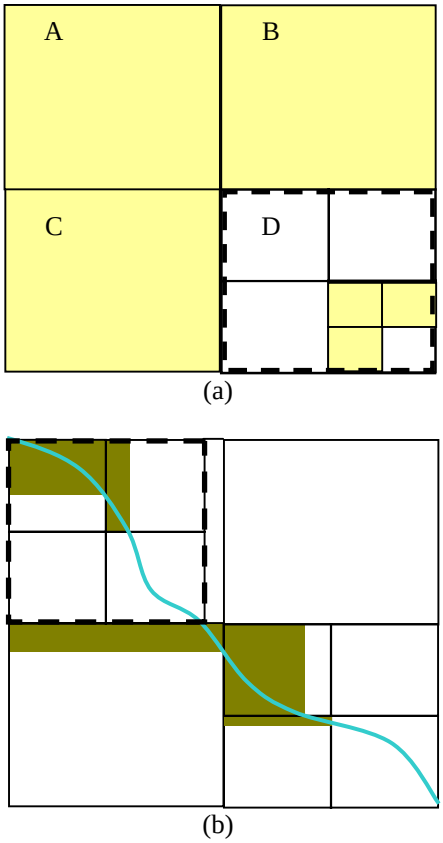


FIG. 3. The FastLSA algorithm (a) column and row bisectioned recursively (b) areas of recalculation

3. The New Fast Linear Space Sequence Alignment Algorithm

In this paper, we propose a new fast linear space sequence alignment algorithm called NFLSA. Similar to the FastLSA algorithm, the NFLSA approach also stores the middle row and column to help reduce recomputation. For k=2, the NFLSA algorithm performs the same as the FastLSA algorithm. For $k \geq 3$, the FastLSA method divides each dimension into k equal

parts and stores *k*-1 rows and *k*-1 columns recursively until the sub-problem size is smaller than the threshold *TH*. Figure 4 shows the FastLSA algorithm for *k*=3. Computing 8/9 of the matrix keeps the two rows and two columns at the boundary. The remaining 1/9 is recursively processed by the algorithm.

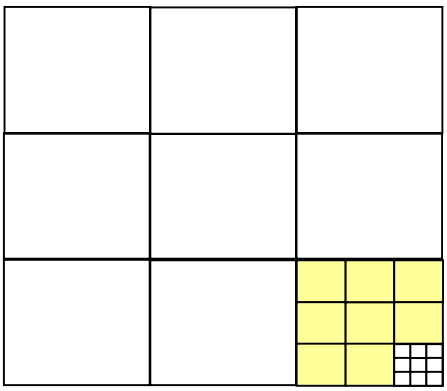
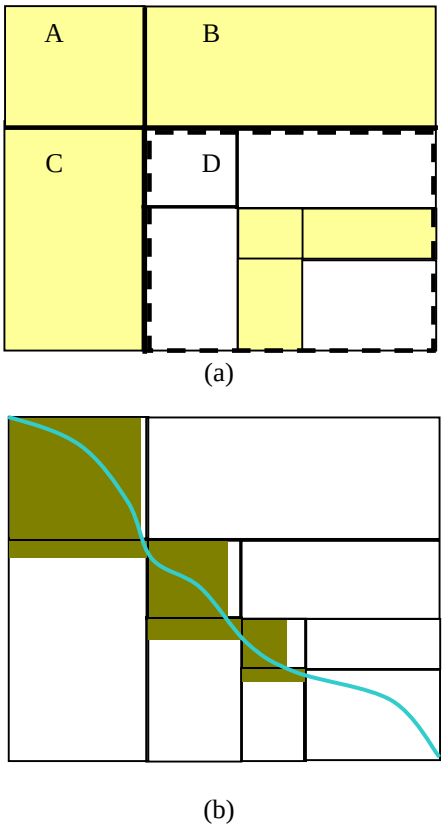


FIG. 4. The FastLSA algorithm divides each dimension into *k* equal parts and stores *k*-1 rows and *k*-1 columns recursively for *k*=3

In contrast, the NFLSA approach divides each dimension to be only one *k*th and the other two parts and then stores the boundary row and column. Figure 5(a) presents the row and column dividing each dimension into one third and two thirds, computing 5/9 of the matrix (rectangles A, B and C) and keeping the boundary row and column. The remaining 4/9 of the problem (dash line rectangle D) is solved recursively until the threshold *TH* is reached. Figure 5(b) illustrates the shadow area that requires recalculation in the TrackPath phase. In fact, the two algorithms are the same for *k*=2.

The pseudo-code of the NFLSA algorithm is shown below.

```
NFLSA(colF, colL, rowF, rowL)
// colF: Index of the first column,
// colL: Index of the last column,
// rowF: Index of the first row,
// rowL: Index of the last row,
1.  if ((colL- colF + 1)*(rowL - rowF + 1) < TH)
2.     FullMatrix(colF, colL, rowF, rowL)
3.  else
4.     colM := colF + (colL - colF)/k
5.     rowM := rowF + (rowL - rowF)/k
6.     FillScore(colF, colM, colL, rowF, rowM,
           rowL)
7.     NFLSA(colM, colL, rowM, rowL)
8.     switch (direction)
       // The direction of TrackPath phase
9.     case diagonal:
10.       NFLSA(colF, colM, rowF, rowM)
11.    case left:
12.       rowL := path.row
13.       NFLSA(colF, colM, rowM, rowL)
14.       if (direction == up)
15.          colL := path.column
16.          NFLSA(colS, colL, rowF, rowM)
17.    case up:
18.       colL := path.column
19.       NFLSA(colM, colL, rowF, rowM)
20.       if (direction == left)
21.          rowL := path.row
22.          NFLSA(colF, colM, rowF, rowL)
```



(a)

(b)

FIG. 5. NFLSA algorithm (a) divided by boundary column and row recursively in position one third (b) recomputed area

For the same *k* value, the memory requirement of NFLSA is equal to that of the FastLSA algorithm. The space requirements for aligning two sequences of length *n* by using a recursive function are as follows:

1. The FastLSA algorithm

$$f(n) = 2(k\text{-}1)n + f(\frac{1}{k}n) = 2kn\ . \qquad (1)$$

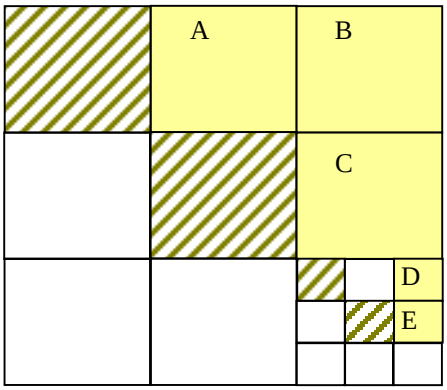
2. The NFLSA algorithm

$$f(n) = 2n + f(\frac{k-1}{k}n) = 2kn\ . \qquad (2)$$

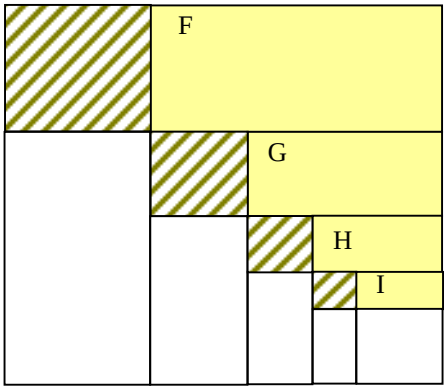
The two different recurrence equations have the same solution.

To analyze the performance, let's consider the

two sequences *s* and *t*. The number of calculations of Hirschberg's algorithm is two times, that of the FastLSA algorithm is 1 + 1/*k* times, and that of the NFLSA algorithm is 1 + 1/2(*k*-1). Figure 6(a) and Figure 6(b) show the recalculated areas for *k*=3 by the FastLSA and NFLSA algorithms, respectively. The slash areas require recalculation in the equal case (*s* = *t*). In the worst scenario, the A, B, C, D and E areas require recalculation recursively by the FastLSA algorithm as shown in Figure 6(a). In contrast, Figure 6(b) presents the F, G, H and I areas to be recomputed by the NFLSA algorithm.



(a)



(b)

FIG. 6. Recomputation area (a) the recomputation area of the FastLSA algorithm for *k*=3 (b) the recomputation area of the NFLSA algorithm for *k*=3

Performance Analysis

In this paper, the performance is evaluated in terms of the number of calculating entries. We shall use the equal case and worst scenario to analyze the computation time.

In worst scenario, the computation time of FastLSA is $(\frac{2k^2-k-2}{k^3-k^2+k+2}+1)mn$ notated as FastLSA_w in Equation (3). For NFLSA, the the computation time NFLSA_w is $(\frac{k-1}{k}+1)mn$. The variable *m* and *n* are the lengths of the sequences *s* and *t*, respectively. In the equal case that the two sequences are the same, $\text{FastLSA}_{s=t}$ and $\text{NFLSA}_{s=t}$ present the computation time of algorithm FastLSA and NFLSA, respectively.

If *k*>2, the FastLSA and NFLSA methods' calculation time are as follows:

$$\text{FastLSA}_w = (\frac{k-2}{k^2}+\frac{1}{k+1})mn*\frac{1}{1-(\frac{k-2}{k^2}+\frac{1}{k+1})}+mn$$

$$=(\frac{2k^2-k-2}{k^3-k^2+k+2}+1)mn \qquad (3)$$

$$\text{FastLSA}_{s=t}= \frac{1}{k+1}mn*\frac{1}{1-1/(k+1)}+mn$$

$$= (\frac{1}{k}+1)mn \qquad (4)$$

$$\text{NFLSA}_w = \frac{k-1}{2k-1}mn*\frac{1}{1-(k-1)/(2k-1)}+mn$$

$$= (\frac{k-1}{k}+1)mn \qquad (5)$$

$$\text{NFLSA}_{s=t} = \frac{1}{2k-1}mn*\frac{1}{1-1/(2k-1)}+mn$$

$$= (\frac{1}{2(k-1)}+1)mn \qquad (6)$$

If *k*=3, In Equation (3), the term $(\frac{k-2}{k^2}+\frac{1}{k+1})mn$ is the number of entries in the gray area of Figure 6(a). In Equation (4), the term $\frac{1}{k+1}mn$ is the number of entries in the slash area of Figure 6(a). These term $\frac{k-1}{2k-1}mn$ and $\frac{1}{2k-1}mn$, in Equation (5) and (6), are for the numbers of entries in the gray area and slash area in Figure 6(b), respectively.

For $\text{FastLSA}_{s=t}$ example, let *k*=3, slash area in Figure 6(a) is 1/4 *mn*. All of the recalculated area is 1/3 *mn*. Therefore, the time complexity of $\text{FastLSA}_{s=t}$ is (1/3 +1)*mn*.

Figure 7 shows the analysis results of relative computation time and how the three methods compare. In the equal case scenario, the performance of the NFLSA algorithm is better than those of the others. The FastLSA algorithm performs best in the worst scenario, and the experimental results show that its performance in the worst scenario is very close to that in the equal case scenario [6]. In real-life situations, the optimal path in the1 TrackPath stage is round diagonal, and the performance is quite close to that in the equal case. Therefore, the NFLSA approach is very practical. If long trailing gaps exist, the performance of the NFLSA algorithm is likely to drop. To avoid these bad cases, we can limit the sequence length difference.

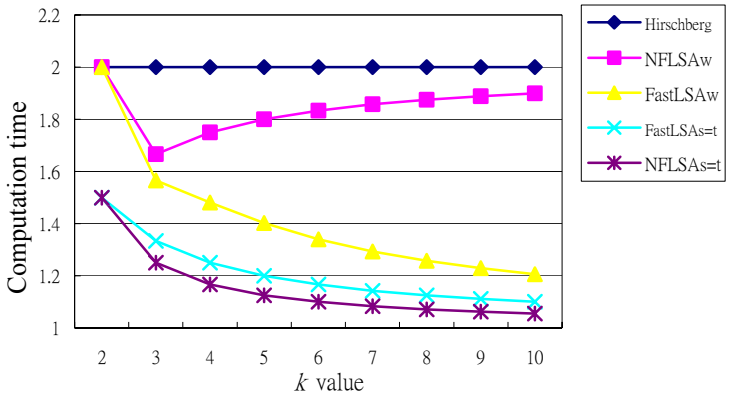


FIG. 7. Analysis on computation time

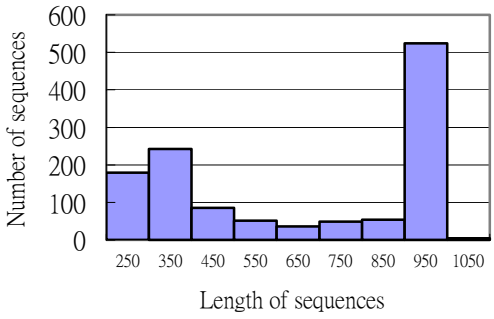
4. Experimental Results

We have evaluated the performances of NFLSA, Hirschberg's algorithm, and FastLSA, respectively. These algorithms have been applied to process three datasets of sequences, all of which are extracted from Genbank's Musculus DNA. The characteristics of the datasets used are given in Table 1. Differences in sequence length do not occur more than five times in the same dataset. Figure 8 (a), (b) and (c) show the distribution of sequence lengths in dataset 1, 2 and 3, respectively. All the sequences were aligned with each other in the same dataset. All the experiments were performed on a 1.5GHz Pentium IV PC with 384 MB of memory, running under Windows 2000 professional. The algorithms were coded in Visual C++ 6.0.

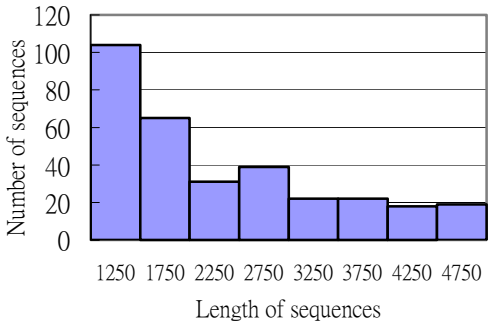
Figure 9, Figure 11 and Figure 12 compare the algorithms by the average recomputation ratios. The average recomputation ratio is calculated as the total number of recomputation operations divided by the total size of all the full matrices. The x-coordinate is the k value, and the y-coordinate is the average recomputation ratio. The full-matrix approach is not shown in these figures, since the y-coordinate value is zero. Hirschberg's algorithm requires a recomputation ratio less than the theoretical value 100% in our experiments.

TABLE 1. Datasets

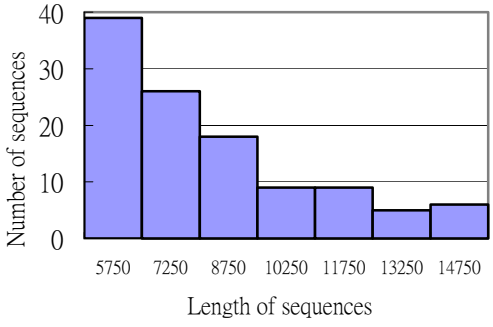
Dataset	Number of sequences	Max length	Min length
1	1224	1000	200
2	320	4964	1000
3	112	15000	5004



(a)



(b)



(c)

FIG. 8. Distribution of sequence lengths (a) Dataset 1 (b) Dataset 2 (c) Dataset 3

For dataset 1, FastLSA and NFLSA are compared for k=2 to 10 in Figure 9. The NFLSA algorithm is better. For k=4, the NFLSA approach outperforms FastLSA by 6.87%. Figure 10 shows the distribution of recalculation ratios of dataset 1 for k=3. The x-coordinate is the recomputation ratio of alignment, and the y-coordinate is the number of alignment tasks. For the alignment tasks, most of the recomputation ratios by the FastLSA algorithm lie somewhere between 0.29 and 0.34, and the figure for the NFLSA algorithm are almost all in the range from 0.19 to 0.29. Even though the NFLSA algorithm has the largest value 0.40, the average recomputation ratio is lower than that of the FastLSA algorithm by about 8%. In contrast to the total alignment number 748476 (1224*1223/2), the influence of the only one largest value is of little significance.

For dataset 2 and database 3, the performances of FastLSA and NFLSA for k=2 to 10 are shown in Figure 11 and Figure 12 individually. The NFLSA algorithm still performs better. In Figure 12, for k=10, the NFLSA algorithm only requires a recalculation ratio of 5.69%. In contrast, the FastLSA approach requires 9.84%. The result is close to the analyses of the equal case in Figure 7.

5. Conclusions

In recent years, computational molecular biology has become one of the most popular interdisciplinary fields. Sequence alignment is an important technique widely used in computational molecular biology. To reduce the memory space requirement and running time, many sequence alignment techniques have been developed and closely studied.

In this paper, we have proposed the NFLSA algorithm, which uses the same memory as FastLSA and reduces the recomputation ratio greater than FastLSA. The experimental results have shown that the NFLSA algorithm can obviously perform better. Most adverse alignments can be avoided in NFLSA by limiting the difference of sequence length within a certain ratio (e.g. 5:1).

Even though we have tried to avoid most unfavorable alignments for our NFLSA algorithm, a small number still exist. In the future, we shall focus our attention on finding an efficient testing method to determine in advance which sequence alignments will be better processed by NFLSA and which sequence alignments will be better processed by FastLSA so that we can always use the more suitable algorithm to solve different sequence alignment problem.

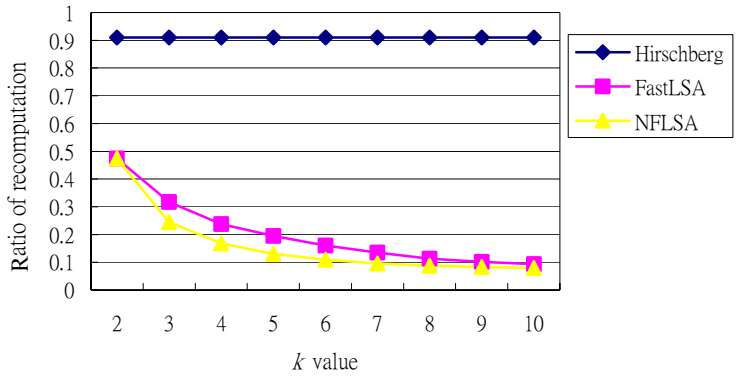


FIG. 9. Average recomputation ratio of Dataset 1

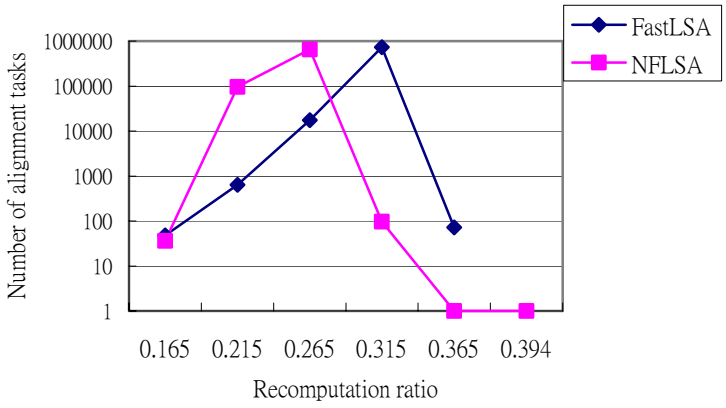


FIG. 10. The number of alignment tasks of Dataset 1 for k=3

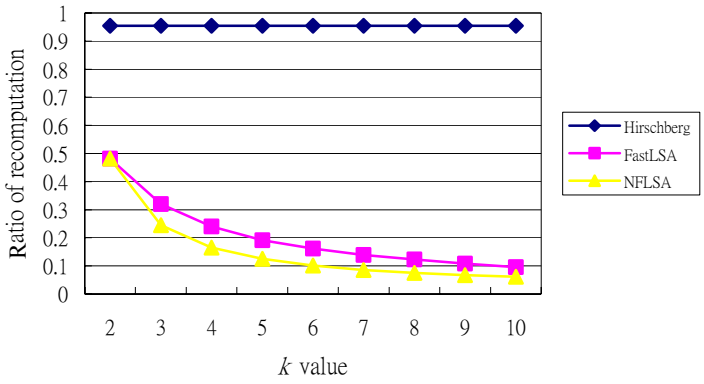


FIG. 11. Average recomputation ratio of Dataset 2

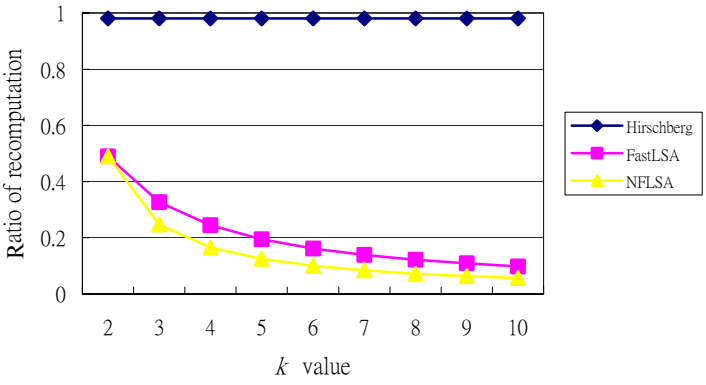


FIG. 12. Average recomputation ratio of Dataset 3

References

[1] Altschul, S.F., and Lipman, D.J., Trees, stars, and multiple biological sequence alignment. *SIAM J. Appl. Math.*, Vol. 49, No. 1, 197-209, 1989.

[2] Bafna, V., Lawler, E.L., and Pevzner P.A., Approximation algorithms for multiple sequence alignment. *Theoret. Comput. Sci.*, Vol. 182, No. 1-2, 233-244, 1997.

[3] Chao, K.M., and Miller, W., Linear-space algorithms that build local alignment from fragment. *Algorithmica*, Vol. 13, No. 1-2, 106-134, 1995.

[4] Chao, K.M., Hardison, R.C., and Miller, W., Recent developments in linear-space alignment methods: A survey. *J. Comp. Biol.*, Vol. 1, No. 4, 271-291, 1994.

[5] Chao, K.M., Pearson, W.R., and Miller, W., Aligning two sequences within a specified diagonal band. *Comput. Appl. Biosci.*, Vol. 8, No. 5, 481-487, 1992.

[6] Charter, K., Schaeffer, J., and Szafron, D., Sequences alignment using FastLSA. In *Proceedings of the 2000 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences* (*METMBS 2000*), Nevada, USA, pp. 239-245, June, 2000.

[7] Davidson, A., A fast pruning algorithm for optimal sequence alignment. In *Proceedings of the 2nd IEEE International Symposium Bioinformatics and Bioengineering* (*BIBE 2001*), Maryland, USA, pp. 49-56, November, 2001.

[8] Eppstein, D., Galil, Z., Giancarlo, R., and Italiano, G.F., Sparse dynamic programming I: Linear cost functions. *J. ACM*, Vol. 39, No. 3, 519-545, 1992.

[9] Florea, L., Hartzell, G., Zhang, Z., Rubin, G.M., and Miller, W., A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Res.*, Vol. 8, No. 9, 967-974, 1998.

[10] Gotoh, O., An improved algorithm for matching biological sequences. *J. Mol. Biol.*, Vol. 162, No. 3, 705-708, 1982.

[11] Hirschberg, D.S., A linear space algorithm for computing maximal common subsequences. *Comm. ACM*, Vol. 18, No. 6, 341-343, 1975.

[12] Huang, X., and Miller, W., A time-efficient, linear-space local similarity algorithm. *Adv. Applied Math.*, Vol. 12, 337-357, 1991.

[13] Korf, R. and Zhang, W., Divide-and-conquer frontier search applied to optimal sequence alignment. In *Proceedings of the 17th National Conference on Artifical Intelligence* (*AAAI 2000*), Texas, USA, pp. 910-916, July, 2000.

[14] McNaughton, M., Lu, P., Schaeffer, J., and Szafron D., Memory-efficient A* heuristics for multiple sequence alignment. In *Proceedings of the 18th National Conference on Artifical Intelligence* (*AAAI 2002*), Alberta, Canada, pp. 737-743, July, 2002.

[15] Myers, E.W., and Miller, W., Optimal alignment in linear space. *Comput. Appl. Biosci.*, Vol. 4, No. 1, 11-17, 1988.

[16] Needleman, S.B., and Wunsch, C.D., A general method applicable to the search for similarities in the amini acid sequences of two proteins. *J. Mol. Biol.*, Vol. 48, 443-453, 1970.

[17] Pachter, L., Lam, F., and Alexandersson, M., Picking alignments from (Steiner) trees. In *Proceedings of the 6th International Conference on Computational Mo-*

lecular Biology (*RECOMB 2002*), Washington, DC, USA, pp. 246-253, April, 2002.

[18] Pevzner, P.A., Multiple alignment, communication cost, and graph matching. *SIAM J. Appl. Math.*, Vol. 52, No. 6, 1763-1779, 1992.

[19] Reinert, K., Stoye, J., and Will, T., An iterative method for faster sum-of-pairs multiple sequence alignment. *Bioinformatics*, Vol. 16, No. 9, 808-814, 2000.

[20] Smith, T.F., and Waterman, M.S., Identification of common molecular sequences. *J. Mol. Biol.*, Vol. 147, 195-197, 1981.

[21] Stoye, J., Multiple sequence alignment with the divide-and-conquer method. *Gene*, Vol. 211, No. 2, GC45-GC56, 1998.

[22] Wu, S., Manber, U., Myers, G., and Miller, W., An $O(NP)$ sequence comparison algorithm. *Inform. Process. Lett.*, Vol. 35, No. 6, 317-323, 1990.

[23] Zhang, Z., Schwartz, S., Wagner, L., and Miller, W., A greedy algorithm for alignment DNA Sequences. *J. Comp. Biol.*, Vol. 7, No. 1-2, 203-214, 2000.