Approximate Global Alignment of Sequences

Tamer Kahveci [a], Venkatakrishnan Ramaswamy [a], Han Tao[a], Tao Li[b]
[a]Department of Computer and Information Science and Engineering, University of Florida, FL, 32611
{tamer, vr1, htao}@cise.ufl.edu
[b] ECE Department, University of Florida, FL, 32611, taoli@ece.ufl.edu

Abstract

We propose two novel dynamic programming (DP) methods that solve the the approximate bounded and unbounded global alignment problems for biological sequences. Our first method solves the bounded alignment problem. It computes the distribution of the edit distance between the remaining suffixes. For a given bound k and approximation $p\,\%$, it uses this distribution to prune the entries of the DP matrix that will lead to alignments with more than k edit operations with more than $p\%$ probability. Our second method addresses the unbounded global alignment problem. For each entry of the distance matrix, it dynamically computes an upper bound to the distance between the unaligned suffixes. This bound, along with the lower bound as computed for the bounded case, is then used to eliminate the entries of the distance matrix. According to our experimental results, our methods are up to three times faster than the competing methods for the bounded alignment and up to two times faster for the unbounded alignment, even with 100 % approximation. Our methods use only 17-68 % of the space used by the next best competitor.

1 Introduction

Comparing two sequences is one of the most fundamental problems in bioinformatics. Sequence search, multiple sequence alignment, shotgun sequence assembly, and phylogenetic analysis are only a few examples of many bioinformatics areas that require comparison of sequences.

One of the most commonly used criteria for defining the distance between two sequences is the *Edit Distance*. The edit distance between two sequences x and y, ED(x, y), is defined as the minimum number of edit operations (insertion, deletion or replacement) to transform x into y.

In this paper we focus on two fundamental sequence alignment problems:

Problem 1 *(Bounded) Given two sequences x and y, and a distance threshold k, find the optimal global alignment of x and y with at most k edit operations.*

Problem 2 *(Unbounded) Given two sequences x and y, find the optimal global alignment of x and y.*

Existing methods usually fall into one of the following two classes: 1) Tools that are too slow and require too much resources (e.g., extensive memory requirements), but have optimal quality. 2) Tools that are reasonably fast and use reasonable amount of resources, but provide almost no quality guarantees. Needleman-Wunsch method (N-W) [13] and BLAST [1] are examples to the first and the second class respectively.

In this paper, we develop a model that allows the user to trade-off efficiency with quality by appropriate choice of the approximation percentage, an input parameter. We call our methods *Lookahead* since we estimate the distance of the unaligned suffixes, before actually evaluating their edit distance. The algorithm takes an approximation percentage $p \in [0, 100]$ as input. Similar to the N-W method, we fill out a distance matrix. As each value in this matrix is computed, we calculate the distance distribution of the unaligned suffixes. This is done by maintaining a summary of the suffixes, with the help of frequency vectors [10]. We keep an entry of the distance matrix if the best traceback path that passes from that entry incurs at most k edit operations with more than $(100 - p)\,\%$ probability. We call this a *$p\,\%$ approximation* for bounded alignment. We also show that eliminating sets of entries in the distance matrix enables pruning of entire rows or columns of this matrix without any computation. Our method for the unbounded problem also computes an upper bound, $\max$, to the edit distance for each entry of M. We keep an entry of the distance matrix if the best traceback path that passes from that entry incurs at most $\max$ edit operations with more than $(100 - p)\,\%$ probability. We call this a *$p\,\%$ approximation* for unbounded alignment. Our experiments show that our methods use 17 to 68 % of the space and run up to three times faster compared to the next best competitor even with 100 % approximation. The performance and memory usage gap between Lookahead and existing methods increase even further as the approximation percentage drops.

We can summarize the contributions of this paper as follows:

1) We propose new approximate solutions to both bounded and unbounded global alignment, that allow the user to choose the quality, by an appropriate choice of approximation percentage, p..

2) Most of the existing methods rely on theoretical analysis and lack experimental validation. We provide detailed experimental analysis on real data.

3) To our knowledge, this is the first paper that profiles the detailed microarchitecture characteristics of the existing algorithms.

The rest of the paper is organized as follows. In Section 2, we give background information. In Sections 3 and 4, we discuss our methods for bounded and unbounded global alignment. In Section 5 we present experimental results. We end with a brief discussion in Section 6.

2 Background

In 2.1, we review previous work on this problem. In 2.2, we define notions of frequency space, as used in this paper.

2.1 Background on sequence alignment

N-W method solves the unbounded alignment problem using dynamic programming as follows. Let $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_m$ be two sequences with n and m letters respectively. An $(n+1)\times(m+1)$ *distance matrix* M is filled recursively by considering the three predecessors of each entry of the matrix:

$$M_{i,0} = i,\ \ M_{0,j} = j,\ \ \forall i,j,\ \ 0 \le i \le n,\ \ 0 \le j \le m,$$

$$M_{i,j} = \min \begin{cases} M_{i-1,j} + 1, \\ M_{i,j-1} + 1, \\ M_{i-1,j-1} + v(x_i, y_j) \end{cases}$$

for $0 < i, j$, where $v(x_i, y_j) = 1$ if $x_i \neq y_j$, $v(x_i, y_j) = 0$, otherwise.

The traceback path from $M_{n,m}$ to $M_{0,0}$ defines the optimal alignment of x and y. The bounded problem can also be solved using the N-W algorithm. A *banded* version of the N-W method is used by filling only the entries $M_{i,j}$, where $|i-j| \le k$. The unbounded DP uses $O(nm)$ space and time. The space and time complexity of the bounded version is $O((n+m)k)$. In many applications, $k = O(n+m)$, thus, the complexity of the bounded version can also be as large as $O(nm)$. The quadratic time and space complexity makes the N-W algorithm impractical as the problem size gets larger. Another problem with the N-W method is that as the sequences get longer, each row (or column) of the DP matrix gets too large to fit in available cache. As a result of that each DP iteration causes cache misses.

Various methods have been developed to improve N-W by reducing the search space in the DP matrix. Ukkonen considered the banded alignment problem [17]. As the DP matrix is computed, his method approximates the distance of unaligned suffixes as the difference of their lengths. Let d' be the approximated value for a DP matrix entry. If the distance value of that entry in the DP matrix exceeds $k - d'$, then that entry is eliminated from further comparison. This method has two main problems. First, it uses a very loose bound to avoid false dismissals. Second, it does not work for the unbounded alignment problem. Hadlock's method [8] is similar to Ukkonen's. Fickett considered the unbounded alignment problem [5]. He starts with a small distance bound k and computes the DP matrix entries within this bound. If no solution is found within this bound, he iteratively increases the value of k and recomputes the distance matrix. This method has two problems. First, although, it computes only the non-redundant DP entries, the entries in the periphery are usually computed twice (once in each of the two consecutive iterations). Second, it has a poor data locality since the DP matrix is computed in an irregular order (i.e., order depends on the distribution of the distances in the DP matrix). Similar to Hadlock, K-T algorithm [15] explores the DP matrix greedily. This method has the same disadvantages as Fickett's and Hadlock's method. It also fails to find all the paths that lead to the best alignment since only the path closest to the diagonal is considered.

Although these above-mentioned methods reduce the search space, they are not commonly used in existing sequence analysis tools. This is because, the additional data structures they need for book keeping are usually larger than the space saved. Furthermore, the reduction in running time due to pruning the search space usually do not compensate the time spent for pruning these entries.

A number of methods have been developed for faster pattern searching [2, 12, 18]. However, these methods work well only for very short sequences with significant similarity. For a given range query or a nearest neighbor query, early pruning methods such as the use of k-grams [7], indexing the edit distance based on reference points [6, 14] or index structures on some feature vectors of sequences [10] have been proposed to avoid redundant comparison of sequences from a database of sequences. However, these methods only produce a candidate set of sequences from a database. Once the candidate set is generated, these methods still need to align the query sequence with the candidate sequences. Existing database search methods do not reduce the alignment cost at this step. Several heuristic methods, such as MUMmer [4] and AVID [3], have been developed for fast global alignment of large sequences. However, these methods do not provide any quality guarantees.

2.2 Background on frequency space

Let x be a sequence from the alphabet $\Gamma = \{\alpha_1, \alpha_2, \cdots, \alpha_\gamma\}$. Let n_i be the number of occurrences of the character α_i in x for $1 \le i \le \gamma$. The *frequency vector* of a sequence x is computed as the number of occur-

rences of each of the letters [10]. The *frequency distance*, FD$(f(x), f(y))$, between the frequency vectors $f(x)$ and $f(y)$ of two sequences x and y is defined as the minimum number of increment and decrement operations needed to transform $f(x)$ into $f(y)$. Unlike edit distance, frequency distance can be computed in $O(\gamma)$ time as follows. First $f(x) - f(y)$ is computed. FD$(f(x), f(y))$ is computed as the larger of the number of increments and decrements. An important property of the frequency distance is that it is a lower bound to the edit distance, i.e., FD$(f(x), f(y)) \leq$ ED$(x, y), \forall x, y$.

Each frequency vector v defines an equivalence class, which is defined as the set of all sequences whose frequency vectors are equal to v. Let u and v be two frequency vectors for sequences of length w. Let U and V be their equivalence classes respectively. It is shown that the distance distribution between the sequences in U and V can be approximated using a normal distribution $N(\mu, \sigma^2)$ [11], where

$\mu = w - \frac{\sum_{i=1}^{\gamma} u_i \cdot v_i}{|q|}$, and

$\sigma^2 \approx \sum_{i=1}^{\gamma} (\frac{u_i \cdot v_i}{w} \cdot (1 + \frac{(u_i - 1)(v_i - 1)}{w - 1} - \frac{u_i \cdot v_i}{w}))$.

3 Approximate bounded alignment

We develop a new model that lets the user choose the quality of the bounded alignment. The user picks a bound k, and an approximation percentage p, where $0 \leq p \leq 100$. Our method eliminates the entries that will lead to alignments with more than k edit operations with more than $p\%$ probability. It then selects the best alignments from the remaining alignments.

Similar to N-W, we fill a distance matrix M. Let d be the edit distance obtained for an entry of M at an intermediate step of the alignment. If $k < d$, then this entry can not be a part of the solution since the edit distance is greater than the bound. For $d \leq k$, we calculate the probability that it will be a part of a solution as follows. We compute the frequency vectors, u and v, of the unaligned suffixes at intermediate steps of the alignment. We then calculate the distance distribution $N(\mu, \sigma^2)$ between u and v. Let $\bar{d}$ be a real number such that

$$\int_{\bar{d}}^{\infty} f(x)\, dx = p/100,$$

where $f(x)$ is the probability density function for the normal distribution with mean μ and variance σ^2. This means that there is a $p\,\%$ chance that the edit distance between the suffixes is at least $\bar{d}$. If $k < d + \bar{d}$, we conclude that the corresponding entry will not be a part of the solution with $p\,\%$ probability.

We have shown that the frequency vectors can be used to eliminate entries of distance matrix based on a probability distribution. However, this is useful only if the additional cost incurred due to computation of the distance distribution is significantly small. Section 2.2 discusses how μ and σ^2 can be computed in constant time, However, this computation requires the frequency vectors of the unaligned suffixes. We propose to compute the frequency vectors incrementally, as follows. As the distance matrix is traversed row-wise (or column-wise), the suffix of one of the sequences remains unchanged, while the suffix of the other sequence decreases by one letter. Therefore, the frequency vectors at any given step of the dynamic programming can be incrementally computed in constant time by decreasing the count of one letter by one.

Eliminating isolated entries of the DP matrix would not result in a technique that works faster in practice, because the overhead necessary in deciding elimination would be much more than computing the entry itself. Next, we show how a whole block of entries may be eliminated under certain conditions, which are frequently encountered in this formulation.

Optimization 1 (Distance Prune) *If $M_{i,j} + \bar{d} > k$, then do not store the entry $M_{i,j}$.*

Optimization 1 allows us to prune entries in M even when $M_{i,j} \leq k$. This reduces the space usage of our algorithm. However, this space reduction is obtained at the expense of additional computation of $\bar{d}$. Next, we discuss how the running time can be reduced by avoiding redundant computations of $M_{i,j}$ and $\bar{d}$.

The traceback path always follows one of the three directions: 1) up by one, 2) left by one, or 3) diagonally up-left by one. This limitation in path allows us to avoid computation of some of the entries of M.

Optimization 2 (Path Prune) *A traceback path passing through $M_{i,j}$ also passes through one of the pruned entries if one of the following conditions holds:*

1. *$M_{i-1,s}$ are pruned $\forall s$, $0 \leq s \leq j$.*
2. *$M_{r,j-1}$ are pruned $\forall r$, $0 \leq r \leq i$.*

Thus, there is no need to compute or store $M_{i,j}$ and $\bar{d}$.

Figure 1 shows an example where entries of M are pruned by using Optimization 2. For example, $M_{3,1}$ is eliminated since all the paths from (3, 1) to (0, 0) should cross either (2, 0) or (2, 1). A similar reasoning can be made for $M_{1,4}$. This optimization allows us to eliminate entire columns under a pruned entry or entire rows to the right of a pruned entry without even computing the value of M for those entries. Thus, both space usage and running time of our algorithm are reduced.

We reduce the space requirement and running time through early elimination of distance matrix entries at the expense of additional computation for the distance prediction $\bar{d}$. Next, we discuss how we minimize this additional cost.

	0	1	2	3	4	5	6
0	⋆		⋆	•	○	○	○
1	⋆		⋆	•	○	○	○
2	•	•	⋆		⋆	•	○
3	○	○	•	⋆		⋆	•
4	○	○	○	•	•	⋆	⋆
5	○	○	○	○	○	•	⋆

Figure 1. An illustration of the optimizations applied to a sample distance matrix M. The symbol • shows the entries of M deleted by Optimization 1. The symbol ○ shows the entries of M deleted because the traceback path from any of these entries to $M_{0,0}$ has to cross an entry marked with • (Optimization 2). The remaining entries cannot be deleted. $\bar{d}$ is computed only for the entries marked with • and ⋆.

Assume that we fill the matrix M row-wise. (i.e., first row is filled at the beginning, then the second row, and so on.) Optimization 2 says that we can prune an entry (i, j) only if a consecutive run of entries are pruned either (1) on the $(i-1)$th row from $(i-1, 0)$ to $(i-1, j)$, or (2) on the $(j-1)$th column from $(0, j-1)$ to $(i, j-1)$. Thus, if an entry (i, j) is not eliminated, then elimination of entries in the remaining row will not provide any pruning for the rows below. Therefore, at each row, we stop computing the lower bounding distance as soon as an entry is not pruned. Similar observation can be made for the columns too. Consider Figure 1. Each row is bounded on the left and right by • symbols. The lower bounding distance is computed only for 1) the entries that are marked with •, and 2) the two entries at each row (shown with ⋆ symbol) that are neighbors to • symbols and inside the region bounded by • symbols. A simple analysis shows that we only need to compute the lower bounding distance $O(m + n)$ times.

4 Unbounded global alignment

The traditional N-W method fills the entire distance matrix M in order to find the global alignment. Our method for bounded alignment does not work for this problem since there is no prespecified distance threshold k. We solve this problem by computing another distance function $D(x, y)$. This distance function has to satisfy two criterion:

1. $\text{ED}(x, y) \leq D(x, y), \forall x, y$.
2. $D(x, y)$ can be computed in $O(1)$ time.

Using $k = D(x, y)$ in our bounded local alignment in Section 3, we can find the best global alignment since the edit distance of the best alignment is guaranteed to be within the bound k. We develop our DP method by dynamically updating the value of k as follows:

$$k = k_{0,0} = D(x, y),$$

$$k = k_{i,j} = \min\{k, M_{i,j} + D(\bar{x}[i+1], \bar{y}[j+1])\}$$

Here, $\bar{x}[i+1]$ and $\bar{y}[j+1]$ correspond to suffixes of x and y starting from positions $i+1$ and $j+1$ respectively. This leads to our third optimization.

Optimization 3 (Implicit Distance Prune) *If $M_{i,j} + \bar{d} > k_{i,j}$, then do not store the the entry $M_{i,j}$.*

Similar to Optimization 1, Optimization 3 eliminates entries of M without sacrificing optimality of the result. We also apply Optimization 2 to further reduce the computation and space cost of our algorithm.

The pruning capability of our unbounded alignment method highly depends on the upper bound function $D(x, y)$ between sequences x and y. A lower value of $D(x, y)$ provides a closer approximation to the actual distance. Thus, it results in a narrow band for our alignment algorithm. In this section, we will discuss how to find a better bound without paying a significant additional cost. The idea is to compute a sample alignment (not necessarily an optimal one) quickly between x and y. This alignment will then be used to predict a lower bound to any pair of suffixes of x and y in $O(1)$ time.

Given sequences x and y, of lengths n and m respectively, we align them by coupling their letters starting from the last one. (i.e., $x[n]$ with $y[m]$, $x[n-1]$ with $y[m-1]$, etc.) Such an initial alignment and its edit distance can be computed in $O(m+n)$ time. This upper-bounding distance function can be incrementally computed in $O(1)$ time for any given pair of suffixes as follows. As the entries of distance matrix is computed row-wise or column-wise, the upper bound distance for each entry can be incrementally computed from the previous entry by just checking the first letter of the suffix. If it is a match, then moving to the neighboring entry inserts a gap. Thus the upper bound distance increases by one. If it is aligned with a gap then removal of this letter from the suffix eliminated the number of edit operations by one. Thus the upper bound distance decreases by one. Otherwise, the upper bound distance stays the same.

Assume that the letters are distributed uniformly across the sequences. The possibility for a given pair of letters to be the same is $1/\gamma$, where γ is the alphabet size. Thus, the expected number of letter pairs that match is $\min\{m, n\}/\gamma$. For DNA sequences, this would be $25\ \%$ the length of the shorter sequence. We conclude that the expected upper bound distance is $\max\{m, n\} - \min\{m, n\}/\gamma$.

Figure 2(a) presents the matrix M for unbounded alignment of sequences ATAC and AGATC. It also shows the lower bounds computed for each entry of M and the minimum of all the upper bounds computed up to that entry. Here, we fill the matrix in row-wise order. Note that one can also traverse the matrix column-wise or diagonal-wise.

Figure 2(b) shows the entries of M eliminated by Optimizations 3 and 2. An entry can be eliminated without

		A	G	A	T	C
	0 (1, 4)	1 (2, 4)	2 (3, 4)	3 (5, 4)	4 (7, 4)	5 (9, 4)
A	1 (3, 3)	0 (1, 3)	1 (1, 3)	2 (3, 3)	3 (5, 3)	4 (7, 3)
T	2 (5, 3)	1 (3, 3)	1 (2, 3)	2 (3, 3)	2 (3, 3)	3 (5, 3)
A	3 (7, 3)	2 (5, 3)	2 (4, 3)	1(2, 2)	2 (2, 2)	3 (4, 2)
C	4 (9, 2)	3 (7, 2)	3 (6, 2)	2 (4, 2)	2 (3, 2)	2 (2, 2)

(a)

		A	G	A	T	C
					●	○
A					●	○
T	●					●
A	○	●	●			●
C	○	○	○	●	●	

(b)

Figure 2. (a) The unbounded alignment of sequences ATAC and AGATC with 100 % approximation. At each entry, $dist(lb, ub)$ has the following meaning: $dist$ = The value $M_{i,j}$ computed for the distance matrix by the N-W algorithm. lb = the lower bound to the distance of the alignment that passes from entry (i, j). ub = the minimum upper bound to the distance of the alignment that passes from any entry computed so far. (Entries are traversed in row-wise order.) (b) The entries of M eliminated by our method (shown with ● and ○). Entries $M_{i,j}$ marked with ● are eliminated following from Optimization 3 (i.e., $lb > ub$). Entries $M_{i,j}$ marked with ○ are eliminated following from Optimization 2.

sacrificing the optimality of the result if the lower bound is greater than the minimum upper bound for that entry. For example, at the entry for the letter pair (C, G), the lower bound and the minimum upper bound are six and three respectively (see Figure 2(b)).

Our algorithm for unbounded sequence alignment does not introduce any additional space usage since k is updated dynamically. We only keep one integer that stores the last value of $k_{i,j}$. Our method incurs additional computation since we compute an upper bounding function $D(x, y)$ at each iteration. This is, however, a negligible cost since 1) $D(x, y)$ is computed in constant time, and 2) $D(x, y)$ is not computed for the entries pruned by Optimization 2.

4.1 Extensions to other alignment problems

In Sections 3 and 4 we discussed our methods for two most common problems: bounded and unbounded sequence alignment. We discussed the edit distance as the similarity criteria. The algorithms we proposed are very general, thus can be applied to more complicated distance/similarity measures, where the cost of each gap is computed using gap-open and gap-extend penalties. Kahveci et. al., showed that lower and upper bounds to such scoring methods can be computed efficiently [9]. However, the distance distribution for these models needs to be developed.

Another extension of our method is the alignment of a short query sequence q to a subsequence of a long sequence x. This problem is also known as *pattern search*. Pattern search problem can be mapped to the methods we describe here. If a pattern search query seeks for the best alignment, then it is similar to the unbounded alignment. If a pattern search query asks for all the alignments above some quality threshold, then it is similar to the bounded alignment. For pattern searching, the computation of lower and upper bounds cannot use the entire suffix of x. This is because q is aligned to a subsequence of x. Instead, the prefix (with t letters) of the suffix of x must be used, where t is the number of letters of q which are not yet aligned.

Extension of our algorithm to local alignment is nontrivial. This is because a subsequence is aligned from both sequences. One needs to predict the length of the subsequences aligned from both sequences.

Table 1. The mean and the standard deviation of the distances between sequences in each dataset.

Seq. Len.	500	1000	2000	4000	8000
Mean	269	533	1060	2121	4202
Std. Dev	9.59	16.15	26.02	35.2	55.02

5 Experimental evaluation

We used five datasets from human chromosome 18 by randomly selecting 100 non-overlapping subsequences of lengths 500, 1000, 2000, 4000, and 8000. These sequences are then modified with 5 % mutation probability. Table 1 lists the mean and the variance of the edit distance between the sequences in each of the datasets.

We implemented bounded and unbounded versions of the traditional N-W algorithm, Ukkonen's [17] and Fickett's [5] algorithms, and our Lookahead method using C. All programs were compiled with *gcc* using the maximum optimization level. The experiments were performed on an Intel Pentium M machine (with 1.4 GHz clock frequency and 512 MB DDR SDRAM) running GNU/Linux operating system.

5.1 Bounded global alignment

The first set of experiments examines the space and time usage for bounded alignment. We use error thresholds of k = 5, 10, 25, 50, 75 and 90 % of the sequence length. For each dataset, we align every sequence to every other

sequence in our experiments.

Figure 3(a) compares the ratio of the filled entries of the DP matrix to the size of the matrix of Lookahead, N-W and Ukkonen algorithm for sequence length of 8000. The approximation is set to 100 % to guarantee that the Lookahead method obtains the same alignment as the N-W or Ukkonen algorithm. Among the three techniques, Lookahead consistently fills the smallest amount of the DP matrix for all error rates. For 10 % error rate, the space usage of Lookahead is only 3% of that of N-W and 17% of that of Ukkonen's method. As the error rate increases, all the methods use more space. When error rate is 90 %, Lookahead uses only 68 % of the space used by the competing methods.

Figure 3(b) shows the average running time to align a pair of sequences of length 8000 for varying error rates. Our method significantly outperforms both N-W and Ukkonen's algorithm for all error rates. Ukkonen's method outperforms N-W for small error rate. However, as the error rate increases, their performance converge to each other. The speedup of our method over the next fastest method ranges from 1.75 to 5. The largest speedup is observed for small error bounds.

5.2 Unbounded global alignment

The second set of experiments examines the space and time usage for unbounded alignment. For each dataset, we align every sequence to every other sequence in our experiments. The experiments are run using Lookahead, N-W, and Fickett's algorithm. We run Lookahead with varying approximation percentages.

Figure 4(a) shows the ratio of the number of entries in distance matrix filled to the total number of entries in the matrix. The ratio is always 1.0 (one) for the N-W method since it fills the entire DP matrix for unbounded alignment. Lookahead uses significantly less memory compared to N-W and Fickett's method for all query lengths even when it has 100 % approximation. The memory usage of Lookahead varies between 67% and 75 % of that of the Fickett's method. With 99 % approximation, Lookahead fills approximately 35 % of the DP matrix. Thus the memory usage of Lookahead is approximately 45 % of that of Fickett's method.

Figure 4(b) plots the average running time for the experiments in Figure 4(a). Fickett's method runs slower than N-W although it computes fewer entries. This is because the amount of DP matrix computations avoided does not compensate for the additional computation to prune these entries. On the other hand, Lookahead is faster than both competing methods even at 100 % approximation for all query lengths. The total running time of Lookahead further drops when the approximation is set to 99 %.

Table 2 shows the ratio of the number of entries filled to the total size of the DP matrix and running time of Lookahead for varying approximation percentages for sequences of length 8000. In We calculate the *accuracy* of an alignment as $(1 - \frac{D-D^*}{D^*}) \times 100$, where D and D^* are the predicted and correct edit distances, respectively. We report the average values for space, time, and accuracy over all queries in Table 2. N-W and Fickett's method fill 100 % and 72 % of the DP matrix respectively. N-W and Fickett's method run in 1080 and 1160 milliseconds respectively. The table shows that as the approximation percentage drops, the memory usage and the running time of Lookahead drops. The saving in space is much greater than the saving in running time. This is because Lookahead spends additional time to compute the distance distributions.

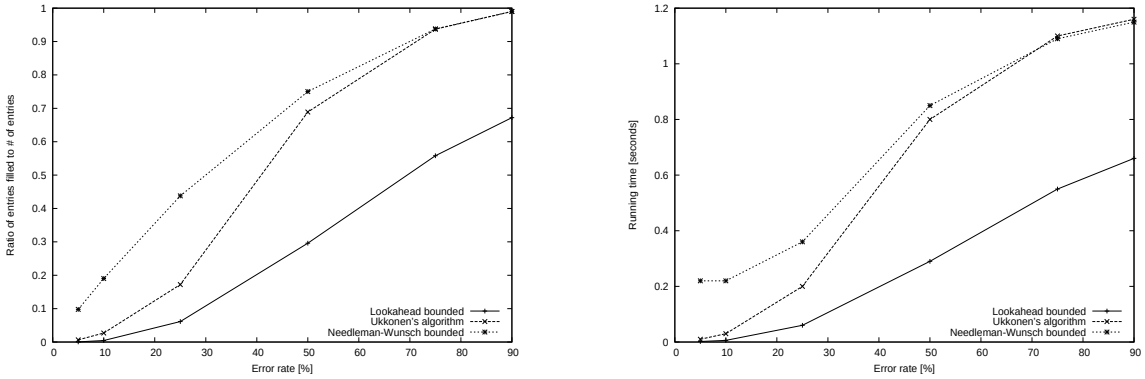
Table 2. The percentage of the number of DP matrix entries filled, running time, and the accuracy of Lookahead for varying approximation percentages for sequences of length 8000. For the same experiment, N-W and Fickett's method run in 1080 and 1160 milliseconds respectively.

	Approximation [%]						
	50	60	70	80	90	95	99
Space [%]	13	22	24	26	29	31	34
Time [ms]	182	294	319	345	381	405	468
Acc. [%]	93	100	100	100	100	100	100

One important observation from Table 2 is the following: The accuracy percentage is significantly higher than the approximation percentage. This happens because of two reasons. First, there may be multiple traceback paths that give the same edit distance. Therefore, even if one of the optimal solutions is discarded during approximation, others may remain, leading to an optimal solution. Second, Lookahead computes an upper bound to the edit distance. The approximation to lower bound, on the other hand, does not exceed the upper bound with probability p%. Thus the probability that it does not exceed the actual edit distance is much greater than p%.

5.3 Evaluation of hardware characteristics

To obtain a more insightful knowledge on the achieved speedups, we use the Pentium 4 hardware performance counters [16] to breakdown the performance improvement and attribute to various run-time characteristics. Table 3 lists the run-time characteristics of the experimented algorithms. For bounded alignment, we used sequences of length 32 K. The memory overhead required by Fickett was too large to run on the experimented Pentium 4 machine. Therefore, we used a smaller input data set with input sequence length of 16 K for unbounded alignment experiments. As can be seen, Lookahead significantly reduces the



(a) (b)

Figure 3. (a) The ratio of the filled entries of the DP matrix to the size of the DP matrix, and (b) the running time of Lookahead method, N-W, and Ukkonen's method for bounded alignment for varying error rates for sequences of length 8000. t % error rate corresponds to an error bound of t % of the sequence length. We used 100 % approximation in this experiment.

computation overhead by showing reduced hardware event counts in almost all the experiments. For example, compared with the N-W technique, our method reduces the total executed instructions by a factor of 3.98 for 60 % error rate. The reductions for loads, stores, branches and floating point instructions are 3.46X, 3.65X, 5.36X and 18.51X respectively. Compared with the N-W technique, Lookahead also yields fewer total cache misses, TLB misses and mispredicted branches. On a high-performance microprocessor such as Pentium 4, microarchitecture events such as cache misses and branch mispredictions imply performance penalty by causing undesirable pipeline stalls and flushes. Table 2 shows that in few cases, our methods can yield higher number in instruction cache misses, D-cache load misses, ITLB misses and mispredicted indirect branches. Nevertheless, the aggregated computation reductions result in a speedup ranging from 1.64X to 2.94X.

6 Discussion

In this paper, we addressed bounded and unbounded sequence alignment problems. We proposed two approximate DP solutions, named *Lookahead* to these problems. Lookahead computes the distribution of the edit distance between the unaligned parts of sequences. It discards the entries of the DP matrix that would not lead to the optimal solution, with probability p, for a given p, thus saving both space and time.

For the bounded alignment, the space usage of our method ranged from 17 to 68 % of that of N-W and Ukkonen's detour method even with 100 % approximation. We achieved up to 3 times speedup to the next fastest method in our experiments. For the unbounded alignment, our experiments show that Lookahead uses only 35 % and 47 % of the space used by N-W and Fickett's method respectively with 99 % approximation. We achieved almost three times speedup over N-W and Fickett's method. Further, we used the hardware performance counters to perform a detailed workload characterization of different alignment methods. We found that our method can significantly reduce the total computational requirement and yield much better statistics in terms of cache misses, TLB misses and mispredicted branches.

The algorithms we proposed can be applied to more complicated distance/similarity measures as well. In the future, we plan to extend our work to problems such as pattern search and local alignment.

References

[1] S. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[2] R. Baeza-Yates and G. Navarro. Faster Approximate String Matching. *Algorithmica*, 23(2):127–158, 1999.

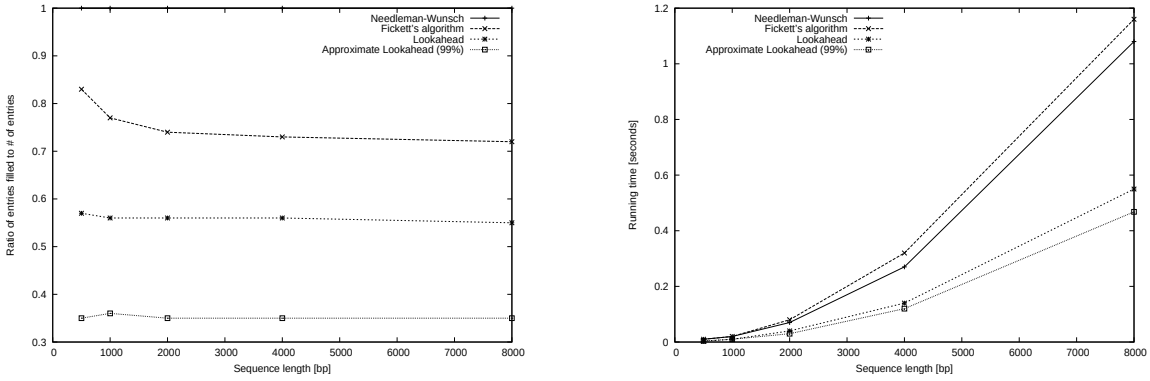
[3] N. Bray, I. Dubchak, and L. Pachter. AVID: A Global Alignment Program. *Genome Research*, 13(1):97–102, 2003.

[4] A.L. Delcher, S. Kasif, R.D. Fleischmann, J. Peterson, O. Whited, and D.L. Salzberg. Alignment of Whole Genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[5] JW Fickett. Fast optimal alignment. *Nucleic Acids Research*, 12(1):175–179, 1984.

[6] Roberto F. Santos Filho, Agma J. M. Traina, Jr. Caetano Traina, and Christos Faloutsos. Similarity Search without Tears: The OMNI Family of All-purpose Access Methods. In *International Conference on Data Engineering (ICDE)*, pages 623–630, 2001.

[7] L. Gravano, P.G. Ipeirotis, H.V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *International Conference on Very Large Databases (VLDB)*, pages 491–500, 2001.



(a) (b)

Figure 4. (a) The ratio of the marked entries of the DP matrix to the size of the DP matrix, and (b) the running time of N-W, Fickett's method and Lookahead with 100% and 99% approximation for unbounded alignment of varying length sequences.

Table 3. Run time characteristics of the experimented algorithms relative to Lookahead. The numbers are normalized to set the same values for Lookahead to 1.

		N-W			Ukkonen		Fickett
	Error Rate	30 %	60 %	∞	30 %	60 %	∞
Basic	cycles	1.68	2.94	1.93	1.64	2.25	1.76
	instructions	2.02	3.98	2.21	2.08	2.91	1.42
	micro-ops	1.9	3.75	2.21	1.87	2.62	1.37
Instruction Mix	loads	2.06	3.46	2.1	1.77	2.49	1.53
	stores	1.88	3.65	1.86	1	1.41	1.29
	branches	2.68	5.36	2.22	2.35	3.3	1.38
	floating point	1.93	18.51	1	1.96	18.72	3.66
Cache Misses	I-cache	2.22	15.45	0.86	1.08	13.02	26.56
	D-cache load	1.61	9.25	1.86	0.95	1.33	0.7
	L2 cache load	2.06	1.36	8.66	1.94	1.14	1204.82
TLB Misses	ITLB misses	1.63	1.09	8.25	1.7	0.21	8.31
	DTLB misses	8.62	33.34	2.54	5.64	2.19	27.68
Mispredicted branches	conditional	1.4	2.02	1.58	1	1.36	1.24
	indirect	2.04	1.63	8.49	2.43	0.22	0.08

[8] F. Hadlock. An efficient algorithm for pattern detection and classification. In *International Conference On Industrial And Engineering Applications Of Artificial Intelligence And Expert Systems (IEA/AIE)*, pages 645–653, 1988.

[9] T. Kahveci, V. Ljosa, and A.K. Singh. Speeding Up Whole-Genome Alignment By Indexing Frequency Vectors. *Bioinformatics*, 20(13):2122–2134, 2004.

[10] T. Kahveci and A. Singh. An Efficient Index Structure for String Databases. In *International Conference on Very Large Databases (VLDB)*, pages 351–360, Roma, Italy, September 2001.

[11] T. Kahveci and A.K. Singh. Progressive Searching of Biological Sequences. *IEEE Database Engineering Bulletin*, 27(3), 2004.

[12] E. Myers. A Sublinear Algorithm for Approximate Keyword Matching. *Algorithmica*, pages 345–374, 1994.

[13] S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48:443–53, 1970.

[14] S.C. Sahinalp, M. Taşan, J. Macker, and Z.M. Özsoyoğlu. Distance Based Indexing for String Proximity Search. In *International Conference on Data Engineering (ICDE)*, 2003.

[15] J. Spouge. Speeding up Dynamic Programming Algorithms for Finding Optimal Lattice Paths. *SIAM Journal on Applied Mathematics*, 49(5):1552–1566, 1989.

[16] B. Sprunt. The Basics of Performance Monitoring Hardware,. In *IEEE Micro*, pages 64–71, 2002.

[17] E. Ukkonen. Algorithms for Approximate String Matching. *Information and Control*, 64:100–118, 1985.

[18] S. Wu and U. Manber. Fast text searching: allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.