String Join Using Precedence Count Matrix

Xia Cao Anthony K. H. Tung Beng Chin Ooi Kian-Lee Tan Shuai Cheng Li
Department of Computer Science
National University of Singapore
Email: {caoxia,atung,ooibc,tankl,lisc}@comp.nus.edu.sg

Abstract

In this paper, we propose a filter-and-refine string join algorithm. While the filtering phase can rapidly prune away strings that are not joinable, the refinement phase employs a comprehensive algorithm to remove the remaining false alarms. The efficiency of the proposed scheme lies in the use of the precedence count matrix (PCM) for computing the edit distance between two sequences. With PCM, the complexity of sequence comparison is a constant time. We also evaluated the proposed sequence join algorithm, and our study shows that it outperforms the known techniques.

1 Introduction

Many applications manipulate string data, for example computational genomics, computational finance, and text and audio processing. One of the most frequently used and expensive operations is the *string join* that combines data from two datasets with *similar* string values on the join attribute. The similarity between two strings is typically determined by the *edit distance*.

In this paper, we study the problem of string join in the context of genomic applications, for example in *sequencing by hybridization*, a sequence is assembled from a set of smaller and overlapping subsequences [3, 5, 10, 11]. In this context, two sequences are joinable if a prefix of one sequence is similar to a suffix of the other. For the sequence S_1 and S_2 with the length m and n, the best suffix-prefix match of the pairs takes time $O(mn)$ [3]. Researchers have started to consider some approaches to speed up string join by skipping the dynamic programming computation for those unattractive pairs [3, 1, 2, 8, 6, 4]. A survey was done in [9] to present an overview of the current techniques to cope with the problem of approximate string matching. Here, we propose an efficient filter-and-refine sequence join algorithm for this purpose. In the filtering phase, the proposed scheme can rapidly prune away sequences that are not joinable. In the refinement phase, a more comprehensive alignment scheme is used to filter out the false alarms.

In the filtering phase, the key operation is to determine the similarity between two sequences. We propose to use the **precedence count matrix** to estimate a lower bound for the edit distance between two sequences. Given the PCMs of two sequences, we derive an efficient algorithm for computing a lower bound for the edit distance between the two sequences. The complexity of this algorithm is $O(|\Sigma|^2 log|\Sigma|)$ where Σ is the alphabet of the sequences. We conducted experiments to evaluate the proposed sequence join algorithm, and our results show that it outperforms existing techniques by a wide margin.

In the next section, we introduce the PCM and the algorithm for approximating the edit distance of two sequences using their PCMs. Section 3 presents the proposed sequence join algorithm. Results from a performance study will be reported in Section 4. Finally, we conclude in Section 6.

2 Approximating Edit Distance Using Precedence Count Matrix

Definition 2.1 Precedence Count Matrix
Let alphabet Σ be the set of characters {A,C,G,T} and Q be a sequence formed from the symbols in Σ. The precedence count matrix of Q, denoted as PCM_Q is a $|\Sigma| \times |\Sigma|$ matrix where each element, represented as $PCM_Q[a, b]$, $a \in \Sigma$, $b \in \Sigma$, is the number of unique occurrences of a preceding b (not necessary consecutive) in the sequence Q.

For ease of discussion, denote $Diag(PCM_Q) = \{PCM_Q[a, a] | a \in \Sigma\}$ as the **diagonal** of the matrix and other elements in the matrix which are not part of $Diag(PCM_Q)$ will be referred to as **non-diagonal elements**. Before we describe the algorithm, we will first highlight the following two properties of the PCM.

Property 2.1 Occurrence Count Property
Let $N_Q(a)$ denote the number of occurrences of a character a in a sequence Q. Then $PCM_Q[a, a] = f(N_Q(a))$ where $f(n) = \frac{n(n-1)}{2}$. Conversely, given $PCM_Q[a, a]$,

Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04)
1099-3371/04 $ 20.00 © 2004 IEEE

we will have $N_Q(a) = f'(PCM_Q[a,a])$ where $f'(n)$ is the inverse of the function $f(n)$. [1]

Property 2.2 Reverse Sum Property
Given a sequence Q and any two characters a and b, $a \neq b$, $PCM_Q[a,b] + PCM_Q[b,a] = N_Q(a) \times N_Q(b)$.

Given the PCMs of two sequences Q and R, an algorithm for approximating the minimum edit distance between Q and R is sketched in Algorithm 1.

Algorithm 1 Estimate Edit Distance

Input: PCM_Q and PCM_R
Output: Lower bound of edit distance between Q and R.
Method:

1. Compute $\mathcal{D}_1$, the minimum number of operations (insert, delete or replace) required to transform $Diag(PCM_Q)$ into $Diag(PCM_R)$. An algorithm in [7] can be adopted for this step. Let this set of operations be denoted as $OPER$ and let the transformed precedence count matrix of Q be PCM'_Q.
2. Compute the maximum impact that $OPER$ has on other **non-diagonal** elements of PCM'_Q. Let the new precedence count matrix be PCM''_Q.
3. we need to compute $\mathcal{D}_2$, the minimum number of operations needed to adjust PCM''_Q such that its other non-diagonal elements are the same as PCM_R. This must be done **while keeping the diagonal unchanged**.

2.1 Adjusting Diagonal Elements

The diagonal elements directly correspond to the number of occurrences of each character in the sequence. We can just adopt an algorithm from [7]. Denote vector V as $V[a] = f'(PCM_R[a,a]) - f'(PCM_Q[a,a])$, $a \in \Sigma$. Then it is not difficult to deduct the following lemma from the algorithm in [7].

Lemma 2.1 $\mathcal{D}_1 = (\sum_{a\in\Sigma} |V[a]| + ||R| - |Q||)/2$.

Intuitively, to transform Q into R, we need to delete or insert at least $||R| - |Q||$ characters, and then perform at least $(\sum_{a\in\Sigma} |V[a]| - ||R| - |Q||)/2$ replacement operations.

2.2 Computing Maximum Impact

In this phase, our aim is to assess how the various edit operations in step 1 impact the non-diagonal values of PCM'_Q and derive PCM''_Q. Note that an operation has an impact only if it brings the non-diagonal values of PCM'_Q closer to the non-diagonal values of PCM_R. We assess this impact individually for each non-diagonal value

[1] $PCM_Q[a,a] = 0$, the value $N_Q(a)$ can be determined with the frequency of other characters.

	$V[a] > 0$ Insert a	$V[a] < 0$ Delete a	$V[b] > 0$ Insert b	$V[b] < 0$ Delete b
Case (I) $PCM_R[a,b] \geq PCM'_Q[a,b]$	+	0	+	0
Case (II) $PCM_R[a,b] \leq PCM'_Q[a,b]$	0	-	0	-

Figure 1. Assessing Impact of Edit Operations on Non-Diagonal Element $PCM'_Q[a,b]$

SubCases	Conditions	Computing PCM''_Q
(1)	$V[a] > 0$ $V[b] > 0$	$PCM''_Q[a,b] = min\{PCM_R[a,b],$ $PCM'_Q[a,b] + N_R[a]N_R[b] - N_Q[a]N_Q[b]\}$
(2)	$V[a] \leq 0$ $V[b] > 0$	$PCM''_Q[a,b] = min\{PCM_R[a,b],$ $PCM'_Q[a,b] + V[b]N'_Q[a]\}$
(3)	$V[a] > 0$ $V[b] \leq 0$	$PCM''_Q[a,b] = min\{PCM_R[a,b],$ $PCM'_Q[a,b] + V[a]N'_Q[b]\}$
(4)	$V[a] \leq 0$ $V[b] \leq 0$	$PCM''_Q[a,b] = PCM'_Q[a,b]$

Figure 2. Subcases for Case (I)

$PCM'_Q[a,b]$, where $a \neq b$. Two cases are shown in Figure 1 together with the edit operations involved and their potential impact.

In Figure 1, '+' means that inserting a or b can affect Case (I), '$-$' means that deleting a or b can affect Case (II), and '0' means that there is no influence on both Case (I) and Case (II). Note that $V[a] > 0$ corresponds to an insertion of at least one character a in sequence Q and $V[a] < 0$ corresponds to a deletion of at least one character a in Q. Figure 2 shows the four subcases for Case (I) and how PCM''_Q is to be computed for each of them.

2.3 Adjusting Non-Diagonal Elements

This phase will proceed to calculate the minimum number of edit operations that are needed to transform PCM''_Q into PCM_R. We only need to adjust the non-diagonal values of PCM''_Q to be the same as those of PCM_R in the minimum number of operations since $Diag(PCM''_Q) = Diag(PCM_R)$. This must also be done while ensuring that the diagonal values of PCM''_Q remain the same, failing which we undo the effect from earlier edit operations.

Theorem 2.1 *Given that p pairs of replacement operations are performed to reduce the difference between $PCM''_Q[a,b]$ and $PCM_R[a,b]$, then the maximum reduction in the difference is $p * (N_R(a) + N_R(b) - p)$.*

Theorem 2.2 *The lower bound of edit distance between two DNA sequences Q and R based on PCM is $\mathcal{D}_1 + \mathcal{D}_2$. $\mathcal{D}_1 + \mathcal{D}_2$ can be computed in time complexity $O(|\Sigma|^2 log|\Sigma|)$.*

With Theorem 2.1, we now describe the algorithm for the last phase in Algorithm 2.

3 Approximate DNA Sequence Join

In this section, we will describe how the PCM is useful in DNA sequence join. We assume there are two sets

Algorithm 2 Phase 3: Adjust Non-diagonal Elements

Input: PCM''_Q, PCM_R
Output: $\mathcal{D}_2$
Method: $\mathcal{D}_2 = 0$;

1. Find all different non-diagonal elements between PCM''_Q and PCM_R, and compute the minimum number of operations $MinOpr$ for each pair according to Theorem 2.1.
2. Find the non-diagonal element $[a, b]$ with maximum $MinOpr$; $\mathcal{D}_2 = \mathcal{D}_2 + MinOpr$.
3. Set $PCM''_Q[a, b]$ and other affected non-diagonal elements to be the same as the ones in PCM_R.
4. Go to 2 until all the non-diagonal elements in PCM''_Q are adjusted to the same as PCM_R.

of sequences $PSet$ and $SSet$, which are called prefix and suffix DNA sequence sets respectively. The nested loops approach is used to join two DNA sequence sets.

We will use $Q[i, j]$ to denote the subsequence of Q that includes entry in position i through j. Denote i^{th} suffix of a sequence Q (i.e. $Q[i, |Q| - 1]$) as $suf(Q, i)$. Similarly, denote the j^{th} prefix of the sequence Q(i.e. $Q[0, j]$) as $pre(Q, j)$.

Let P be a sequence in $PSet$ and S be a sequence in $SSet$. Our objective here is to find all pairs of P and S in which there exist i, j, $min((|S| - i), (j + 1)) \geq Minlen$ such that $edit(suf(S, i), pre(P, j)) \leq e$. Here, $Minlen$ and e are user specified threshold and $edit(S, P)$ denotes the edit distance of the two sequences S and P. The method can be described as follow:

1. **Transformation**
   In this step, for each DNA sequence, we generate two sets of PCMs. For each S in $SSet$, the first set of PCMs corresponds to the suffixes of S, i.e., each suffix of S results in a PCM. For each P in $PSet$, the first set of PCMs corresponds to the prefixes of P. For each sequence (either from $SSet$ or $PSet$), the second set of PCMs is generated in the same manner as follows. A set of w-tuples is obtained from a sequence by placing a sliding window of size w over the sequence. Each such w-tuple is transformed into a PCM.
2. **Filtering**
   Potential candidates of DNA sequence joins are formed by using PCM as part of the filter. Three filtering techniques for efficient DNA sequence joins based on PCM are proposed in the following:
   - **Distance Filtering**
     In the Distance Filtering scheme, the candidates of sequence joins are obtained by using the distance function based on the PCMs, which is the lower bound of edit distance.
     Firstly, for each prefix $pre(P, j)$ and suffix $suf(S, i)$ with length no shorter than $Minlen$, the distance between the corresponding PCMs of the prefix and suffix is computed. If the distance is greater than a given threshold e, this pair of prefix and suffix is not candidate. Otherwise, the PCMs of the disjoint sub-windows of the prefix and the suffix will be used as another layer of filtering. If all the distances between all the corresponding PCMs of the sub-windows are not greater than $e + ||pre(P, j)| - |suf(S, i)||$, $pre(P, j)$ and $suf(S, i)$ will be accepted as a candidate, or else this pair will be filtered out.
   - **Length Filtering**
     The sequence length can be used as a filter for sequence joins. If $||S| - |P|| > e$ then $edit(S, P) > e$.
3. **Verification**
   For two DNA sequences of length m and n, the edit distance computed by dynamic programming with time complexity $O(mn)$ can be used to process the candidates pairs generated to obtain the final results pairs of DNA sequence joins.

4 Experimental Results

We implemented and evaluated the proposed PCM method with the filtering schemes. As references, we also compared our scheme against the q-grams method (denoted qgram) and frequency vector (denoted FV) method.

Under qgram, an auxiliary file that stores the q-grams information will be created beforehand. We also used the length filtering, as well as the count filtering technique proposed in [8]. For the FV method, we used the *Frequency Distance* proposed in [7] as the distance filtering in our implementation. In addition, the length filtering is also deployed. We also looked at two integrated strategies: PCM+qgram method and FV+qgram method. Both methods extend their base method (i.e., PCM and FV respectively) by using qgram method as a further filter for the candidates pairs generated by the respective base methods.

We randomly generated two datasets, prefix data set and suffix data set, from a complete sequence in $ecoli.nt$ sequence database. Each sequence data set consists of 1000 DNA sequences with the length varying between 200 and 300. In the experiment, we study the effect of e on the five join algorithms. We vary e from 1 to 5 for $Minlen = 40$ and $q = 3$. The size of q was set 3 for q-grams method since it always gives the best performance for our algorithm. We note that this is consistent with the observation given in [8]. The sliding window size w for PCM and FV is set as 40.

The filtering rate of the schemes is shown in Figure 3. The efficiency of the schemes is shown in Figure 4 and Figure 5 for the filtering and total processing, respectively. The results show that qgram is very effective but least efficient. We also observe that PCM method is generally superior over FV method. However, though they provide fast filtering, the inability to prune away dissimilar sequences

e	PCM	PCM+qgram	FV	FV+qgram	qgram
1	99.9985%	100%	99.81 %	100 %	100%
2	99.42 %	99. 998%	96.66%	99.998%	99.997%
3	92.47 %	99.986%	84.98%	99.98%	99.98%
4	75.17 %	99.918%	66.39%	99.91%	99.90%
5	54.42%	99.387%	47.25%	99.36%	99.28%

Figure 3. Filtering Rate for Minlen=40

results in high refinement computation overhead. On the whole, they still outperform qgram. This is because the distance function based on PCM can cause better effect of filtering with low cost of computation. Finally, we note that the integrated methods, PCM+qgram and FV+qgram outperform qgram in terms of both effectiveness and efficiency, with PCM+qgram being slightly more superior.

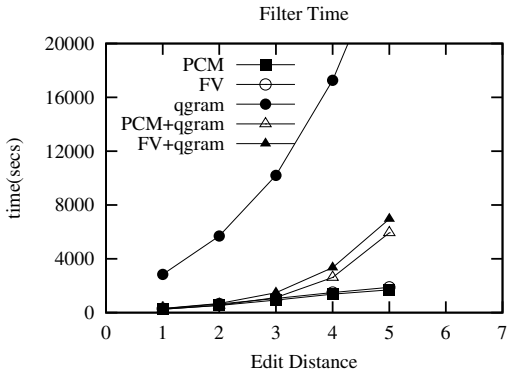

Figure 4. Filter Time vs Edit Distance(DataSet Size:1000, Minlen=40)

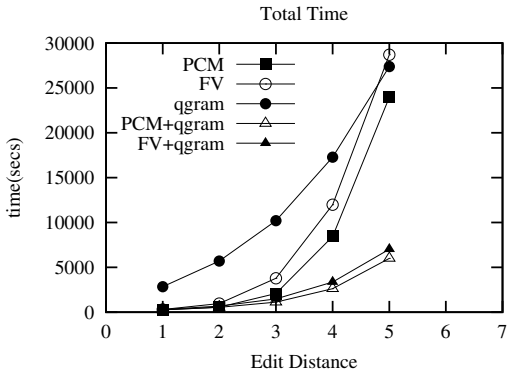

Figure 5. Total Time (Filter and Refinement) vs Edit Distance(DataSet Size:1000,Minlen=40)

We also note that with increasing e values, the performance of all schemes degenerates. This is expected since more false positives are being retained in the filtering step.

The effects of Minlen l are also studied on the schemes on the same data set. As expected, the results show that all the schemes are less effective for small Minlen as the number of false candidates increases with smaller Minlen. The relative performance of the five methods is consistent with the results of the earlier experiments: PCM+qgram is the best in terms of both filtering rate and total running time, followed by FV+qgram, PCM, and qgram is superior over FV when Minlen decreases.

5 Conclusions

In this paper, we have proposed a filter-and-refine string join algorithm for genomic applications. In the filtering phase, strings that are not joinable are pruned away rapidly. The refinement phase employs an efficient algorithm to remove the remaining false alarms. The proposed scheme employs the precedence count matrix (PCM) to compute the edit distance between two DNA sequences efficiently. We have evaluated the proposed sequence join algorithm, and our study shows that it outperforms known techniques.

References

[1] T. Chen and S. Skiena. Trie-based data structures for sequence assembly. In *Technical Report: Department of Computer Science*, Stony Brook N.Y, 1996.

[2] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *In Proc. of the 1998 ACM SIGMOD Conf. on Management of Data*, pages 201–212, 1998.

[3] D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.

[4] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *In Proc. of the 1998 ACM SIGMOD Conf. on Management of Data*, pages 237–248, 1998.

[5] R. Idury and M. S. Waterman. A new algorithm for dna sequence assembly. *Journal of Computational Biology*, 2:291–306, 1995.

[6] L. Jin, C. Li, and S. Mehrotra. Efficient similarity string joins in large data sets. In *Technical Report: Department of Information and Computer Science*, University of California, 2002.

[7] T. Kahveci and A. K. Singh. An efficient index structure for string databases. In *Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01)*, pages 351–360, Italy, Roma, Sept. 2001.

[8] 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 *Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01)*, pages 491–500, Italy, Roma, Sept. 2001.

[9] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, 33:31 – 88, 2001.

[10] M. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen. Algorithms for some string matching problems arising in molecular genetics. In *Proc. of the 9th IFIP World Computer Congress*, pages 59–64, 1983.

[11] M. Peltola, H. Soderlund, and E. Ukkonen. Sequaid: a dna sequence assembly program based on a mathematical model. *Nucleic Acids Research*, 12:307–321, 1984.