Implementation of Bit-Vector Algorithm for Approximate String Matching on Rhodopsin Protein Sequence

Yessica Nataliani
Department of Information Systems
Satya Wacana Christian University
Salatiga, Indonesia, 50711

Theophilus Wellem
Department of Information Systems
Satya Wacana Christian University
Salatiga, Indonesia, 50711

ABSTRACT
Approximate string matching has been used in many applications such as, text retrieval, spell checker and DNA sequence matching in computational biology. In this paper, we implemented bit-vector algorithm using MATLAB for approximate string matching on Rhodopsin protein sequence of class Aves. Our experiments on real data of Rhodopsin protein sequences demonstrate that the algorithm can work as expected. The experiment results shows that the Rhodopsin protein sequence of the species in same genus is more approximately match each other compared to the species from different genus in the same family, Furthermore, for the species from different genus in the same family, its Rhodopsin protein sequence is more approximately match each other compared to the species from different family in the same order.

General Terms
Algorithm, Implementation, Experimentation

Keywords
Bit-vector, DNA, Rhodopsin, string matching

1. INTRODUCTION
Approximate string matching has been used in many applications such as, text retrieval, spell checker and DNA sequence matching in computational biology. DNA sequences can be regarded as strings over the alphabet A, C, G, T. Searching a pattern or patterns in DNA sequences can be done using approximate string matching techniques.

Several algorithms have been proposed for searching patterns in DNA sequences. Cheng et al. [1] developed an algorithm for approximate string matching in DNA sequences. They proposed an indexing structure that can improve the searching efficiency of suffix array, and also a parallel computing technique for indexing and searching DNA sequences in PC clusters. Liu et al. [2] proposed FAAST (fast algorithm for approximate string matching) for solving the *k-mismatch* problem of the approximate string matching and tested the algorithm using simulated dataset and real ribosomal fungal/bacterial DNA sequences from NCBI DNA sequence database. Basic Local Alignment Search Tool (BLAST) developed by Altschul et al. [3] is a popular tool used by researchers for comparing a pattern in a DNA sequence with a database of DNA sequences. FASTA [4] is also a tool for DNA and protein sequence alignment. It uses heuristic algorithm to do the searching.

In this paper, we implemented the bit-vector algorithm described in [5] for approximate string matching on Rhodopsin protein sequence of class Aves. Rhodopsin is a pigment in photoreceptor cells of retina. This pigment is responsible for the perception of light (light-absorber) [6]. The objective of this research is to find out whether a Rhodopsin protein sequence from a species is approximately match to the Rhodopsin protein sequence of other species from the same and different genus, family and order.

The remainder of this paper is organized as follows. In Section 2, we provide the background of approximate string matching. Section 3 discusses the bit-vector algorithm and related work, and Section 4 explains the implementation of bit-vector algorithm and testing method used. Results of the experiments and discussion are presented in Section 5. Finally, we conclude our work in Section 6.

2. APPROXIMATE STRING MATCHING
Generally, string matching techniques can be divided into: 1) exact string matching and 2) approximate string matching. The exact string matching problem is to find all occurrences of a given pattern P in a string T, while the approximate string matching tries to find substrings in the string T that are within a predefined distance (called edit distance) from the given pattern P [7].

There are two variants of approximate string matching problem, *k-mismatch* problem and *k-difference* problem. We focus on the *k-difference* problem in this paper. Given a pattern $P = p_1\ p_2\ \ldots\ p_m$ with length m, a string $T = t_1\ t_2\ \ldots\ t_n$ with length n, and a positive integer threshold $k \geq 0$, the *k-difference* problem is to find all substrings of T ending at t_j (or all positions j in T), whose edit distance, $d(T, P)$, to P is at most k (i.e., $d \leq k$). The edit distance (Levenshtein distance [8]), d between two strings is defined as the minimum number of edits (insertion, deletion, and substitution/replacement of a single character) needed to make the two strings equal [7]. For example, we have a string $x = abca$ and a pattern $y = abba$ to be matched in x. In this case, $d(x, y)$ is equal to 1 since we need one substitution in y (replace its third character with 'c') in order to make x equal to y. Using the same method, we can easily compute that $d(abca, cca) = 2$.

There are several well-known techniques for approximate string matching, for example, dynamic programming [9, 10] and bit-vector algorithm [5]. Using the classic dynamic programming approach [9], the solution to the approximate string matching problem is computed using an $(m+1) \times (n+1)$ dynamic programming (DP) matrix, $C[0..m, 0..n]$ in which each element $C[i, j]$ is the minimum edit distance between pattern $p_1\ p_2\ \ldots\ p_i$ and any substring of T (ending at t_j). After

		String									
		b	a	c	a	b	b	b	b	a	c
	0	0	0	0	0	0	0	0	0	0	0
b	1	0	1	1	1	0	0	0	0	1	1
b	2	1	1	2	2	1	0	0	0	1	2
b	3	2	2	2	3	2	1	0	0	1	2
a	4	3	**2**	3	**2**	3	**2**	**1**	**1**	**0**	**1**

Pattern

Fig 1: DP matrix example

the computation process, all locations j such that $C[m, j] \leq k$ are the solutions to the problem. For example, we want to find the pattern *bbba* in the text *bacabbbbac* with $d \leq 2$ (i.e., $k = 2$, less than 2 differences), Figure 1 shows the DP matrix. As shown in the figure, we can find 7 locations (shaded in the last row) in the string that approximately match to the pattern. The computation can be done in $O(mn)$ time and $O(m)$ space [11, 5]. Later, Ukkonen [10] improved this classic dynamic programming approach, and reduced its running time to $O(kn)$ time. More complete survey on approximate string matching can be found in [12] and [10].

3. BIT-VECTOR ALGORITHM

Myers [5] proposed a bit-vector algorithm for approximate string matching. The idea of this algorithm is to process the DP matrix using bit-parallelism. To parallelize the DP matrix, the differences (deltas) between consecutive rows and columns of the DP matrix, $C[i, j]$ are used (instead of their absolute values) to encode the DP matrix using bit-vectors [13]. Let

$$\Delta h_{i,j} = C[i, j] - C[i, j-1] \in \{-1, 0, +1\}$$

$$\Delta v_{i,j} = C[i, j] - C[i-1, j] \in \{-1, 0, +1\}$$

$$\Delta d_{i,j} = C[i, j] - C[i-1, j-1] \in \{0, +1\}$$

where $\Delta h_{i,j}$, $\Delta v_{i,j}$, and $\Delta d_{i,j}$ are the horizontal adjacency property, vertical adjacency property, and diagonal property, respectively. These delta vectors are encoded as bit-vectors using the following Boolean variables,

$$VP_{i,j} = (\Delta v_{i,j} = +1)$$

$$VN_{i,j} = (\Delta v_{i,j} = -1)$$

$$HP_{i,j} = (\Delta h_{i,j} = +1)$$

$$HN_{i,j} = (\Delta v_{i,j} = -1)$$

$$D0_{i,j} = (\Delta d_{i,j} = 0)$$

Then we have these equivalences,

$$HN_{i,j} \Leftrightarrow VP_{i,j-1} \text{ AND } D0_{i,j}$$

$$VN_{i,j} \Leftrightarrow HP_{i-1,j} \text{ AND } D0_{i,j}$$

$$HP_{i,j} \Leftrightarrow VN_{i,j-1} \text{ OR NOT } (VP_{i,j-1} \text{ OR } D0_{i,j})$$

$$VP_{i,j} \Leftrightarrow HN_{i-1,j} \text{ OR NOT } (HP_{i-1,j} \text{ OR } D0_{i,j})$$

$$D0_{i,j} \Leftrightarrow (p_i = t_j) \text{ OR } VN_{i,j-1} \text{ OR } HN_{i-1,j}$$

where

(1) $HN_{i,j}$ is the horizontal negative delta vector.

(2) $VN_{i,j}$ is the vertical negative delta vector.

(3) $HP_{i,j}$ is the horizontal positive delta vector.

(4) $VP_{i,j}$ is the vertical positive delta vector.

(5) $D0_{i,j}$ is the diagonal zero delta vector.

(6) (i, j) denotes row i and column j.

(7) p denotes the pattern.

(8) t denotes the text string.

The bit-vector algorithm for approximate string matching consists of 2 stages: 1) preprocessing and 2) searching, scoring and output. Let Σ is the alphabet and $B[c] \mid \forall c \in \Sigma$ is a bit-vector. The preprocessing stage does the computation of bit-vectors. These two stages are shown in Algorithm 1 and Algorithm 2. We refer the reader to [5], [7] and [13] for the details of the algorithm.

Algorithm 1 Preprocessing

```
for c ∈ Σ do
    B[c] = 0^m
end for
for j ← 1 to m do
    B[p_j] = B[p_j] | 0^(m−j) 10^(j−1)
end for
VP = 1^m
VN = 0^m
score = m
```

Algorithm 2 Searching, scoring and output

```
for pos ← 1 to n do
    X = B[t_pos] | VN
    D0 = ((VP + (X & VP)) ⊕ VP) | X
    HN = VP & D0
    HP = VN | ~ (VP | D0)
    X = HP << 1
    VN = X & D0
    VP = (HN << 1) | ~ (X | D0)
    {Scoring and output:}
    if (HP & 10^(m−1)) ≠ 0^m then
        score+ = 1
    else
        if (HN & 10^(m−1)) ≠ 0^m then
            score– = 1
        end if
    else
        if (HP & 10^(m−1)) = 0^m then
            score = score
        end if
    end if
    if score ≤ k then
        print(pos)
    end if
end for
```

4. IMPLEMENTATION AND TESTING

We implemented the bit-vector algorithm described in Section 3 using MATLAB. The algorithm requires 3 inputs: the string (T), pattern (P), and threshold/maximum difference constant (k). The string is the Rhodopsin protein sequence of a species and the pattern is the Rhodopsin protein sequence from one other species. After reading the inputs, the pattern is then extracted to get the all different characters of the pattern. Here, we get the A, C, G, and T characters. Therefore, we get the alphabet $\Sigma = \{A, C, G, T\}$. The bit-vector $B[c]$ is then created according to preprocessing stage described in previous section. The flowchart of the algorithm implementation is shown in Figure 2.

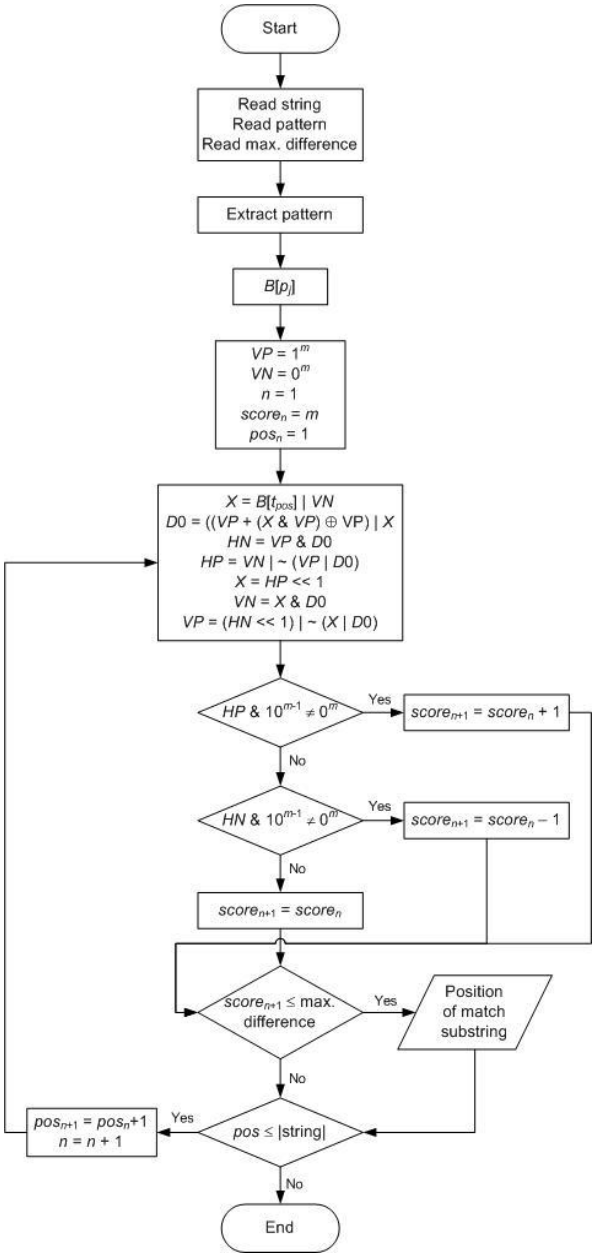

Fig 2: Application flowchart

The species data and Rhodopsin sequence data are taken from Universal Protein Resource (UniProt) Consortium website [14] and DNA Data Bank of Japan (DDBJ) website [15], respectively. The species samples are taken from class *Aves*, superorder *Neognathae*, and order *Galliformes*. Of the 244 species taken, there were 25 species which have Rhodopsin protein. Some of the species are as follows: *Crax alector* (black curassow), *Aepypodius arfakianus* (wattled brushturkey), *Megapodius eremita* (Melanesian scrubfowl), *Gallus gallus* (red junglefowl), and *Colinus cristatus* (crested bobwhite). These 25 species are from 19 genera, one subfamily of *Phasianidae* (*Perdicinae*), and five families (*Cracidae*, *Megapodiidae*, *Numididae*, *Odontophoridae* and *Phasianidae*). The complete list of family, genus, and species used in the experiments is shown in Table 1.

In our experiments, we compared the Rhodopsin protein sequence from a species to other species within the same and different genera, families, and orders. Each species' Rhodopsin protein sequence is compared to the other 24 species. Therefore, there are 600 comparisons for all species. The result then is categorized by species that belong to the same genus, the same family, and the same order.

Table 1. List of family, genus and species

Family	Genus	Species
Cracidae	*Crax*	*Crax alector*
Megapodiidae	*Aepypodius*	*Aepypodius arfakianus*
	Alectura	*Alectura lathami*
	Eulipoa	*Eulipoa wallacei*
	Leipoa	*Leipoa ocellata*
	Macrocephalon	*Macrocephalon maleo*
	Megapodius	*Megapodius eremita*
		Megapodius freycinet
		Megapodius layardi
		Megapodius pritchardii
		Megapodius reinwardt
		Megapodius tenimberensis
	Talegalla	*Talegalla fuscirostris*
Numididae	*Numida*	*Numida meleagris*
Odontophoridae	*Colinus*	*Colinus cristatus*
Phasianidae	-	*Gallus gallus*
	Alectoris	*Alectoris chukar*
	Coturnix	*Coturnix coturnix*
	Perdix	*Perdix perdix*
	Rollulus	*Rollulus roulroul*
	Gallus	*Gallus gallus*
	Pavo	*Pavo cristatus*
	Dendragapus	*Dendragapus obscurus*
	Lagopus	*Lagopus lagopus*
	Tetrastes	*Tetrastes bonasia*

Example of the matching process is shown in Figure 3. The top part of the figure shows the Rhodopsin sequence of two species from the same genus (*Megapodius freycinet* and *Megapodius layardi* from genus *Megapodius*). In this example we took only a part of the Rhodopsin protein sequence for the sake of clarity. We used $k = 0$ in this example. It means that the string is compared to the pattern with maximum difference = 0. The result shows that we have one match substring (the position is indicated with the uppercase letter). The bottom part is the matching process of two species from the same family with $k = 2$. In this case, we also have one match. As shown in the figure, the number of differences is 2 (the positions in the figure are underlined). All the experiments were executed on 1.66GHz Intel Core 2 Duo CPU, 1 GB RAM.

5. RESULTS AND DISCUSSION

As stated earlier, the purpose of this research is to find out whether a Rhodopsin protein sequence from a species is approximately match to the Rhodopsin protein sequence of other species from the same and different genus, family and order. The experiment results showed that comparing all six species from the genus *Megapodius* (total = 30 comparisons) using $k = 0$, except for *Megapodius eremita*, resulted in one match. In case of *Megapodius eremita* compared to the other five species using $k = 0$ to $k = 15$, there were no match. Only if $k = 16$, we can have one match. For $k > 16$, the number of match substring is more than one. The average of k for all comparisons is 2.67.

Comparing all the species (12 species) from different genera in family *Megapodiidae* (total = 132 comparisons), if *Alectura lathami* is compared to the other 11 species, there are 9 comparisons that need at least $k = 22$ in order to get matched substring. The average of k for all comparisons is 5.99.

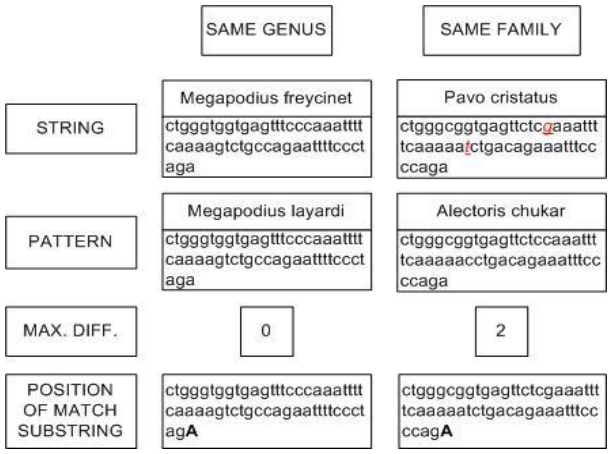

Fig 3: Example of matching process

For the comparison of all 25 species from different genera and different families in same order (*Galliformes*), there are total of 600 comparisons. From these comparisons, we found that there is one comparison (*Lagopus lagopus* compared to *Pavo cristatus*) that need at least $k = 26$ to get matched substring. The average of k for all comparisons is 11.72. Some results of the experiments are shown in Table 2.

From the experiment results, it is found that species from the same genus and family have a smaller average of k compared to the species from different genus in the same family. Also, species from different genus in same family have smaller average of k compared to species from genus in different families.

6. CONCLUSION

This paper presented the implementation of bit-vector algorithm for approximate string matching on Rhodopsin protein sequence from the species of class Aves. The experiment results showed that, in order *Galliformes*, the Rhodopsin protein sequence from the species in same genus (*Megapodius*, family *Megapodiidae*) is approximately match each other with $k = 0$ and $k = 16$. For species from other genera in same family, the Rhodopsin protein sequences approximately match with $0 \leq k \leq 22$. Furthermore, the Rhodopsin protein sequence of species from different family in same order is approximately match each other with $0 \leq k \leq 26$.

7. REFERENCES

[1] Lok-Lam Cheng, David W. Cheung, and Siu-Ming Yiu. Approximate string matching in {DNA} sequences. In *Proceedings of the Eight International Conference on Database Systems for Advanced Applications*, {DASFAA} '03, pages 303–310, Washington, DC, USA, 2003. IEEE Computer Society.

[2] Zheng Liu, Xin Chen, James Borneman, and Tao Jiang. *A Fast Algorithm for Approximate String Matching on Gene Sequences*. 2005.

[3] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[4] D J Lipman and W R Pearson. Rapid and sensitive protein similarity searches. *Science (New York, N.Y.)*, 227(4693):1435–1441, March 1985.

[5] Gene Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, 46(3):395–415, May 1999.

[6] B.J. Litmann and D.C. Mitchell. Rhodopsin structure and function. In A.G. Lee, editor, *Rhodopsin and G-Protein Linked Receptors Vol.2, Part A*, pages 1–32. JAI Press, 1996.

[7] Heikki Hyyro. Explaining and extending the bit-parallel approximate string matching algorithm of Myers. Technical report, Dept. of Computer and Information Sciences, University of Tampere, Finland, 2001.

[8] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[9] Peter H Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373, December 1980.

[10] Esko Ukkonen. Algorithms for approximate string matching. *Information Control*, 64(1-3):100–118, March 1985.

[11] Petteri Jokinen, Jorma Tarhio, and Esko Ukkonen. A comparison of approximate string matching algorithms. *Software - Practice and Experience*, 26(12):1439–1458, December 1996.

[12] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Survey*, 33(1):31–88, March 2001.

[13] D. Huson. Bit-Vector-based Approximate String Matching. http://ab.inf.uni-tuebingen.de/teaching/ws04/seqana/script/bitvector.pdf, 2004.

[14] Universal Protein Resource (UniProt) Consortium website}. http://www.uniprot.org/taxonomy.

[15] DNA Data Bank of Japan (DDBJ) website. http://www.ddbj.nig.ac.jp.

Table 2. Ezample of experiment result

Species (String)	Species (Pattern)	*k*	# match	Position in string
Megapodius freycinet	*Megapodius layardi*	0	1	52
		1	3	51, 52, 53
		2	5	50, 51, 52, 53, 54
		3	7	49, 50, 51, 52, 53, 54, 55
Megapodius freycinet	*Megapodius eremita*	0-15	0	-
		16	1	36
		17	3	35, 36, 37
Aepypodius arfakianus	*Alectura lathami*	0-21	0	-
		22	10	111, 112, 113, 151, 152, 153, 154, 737, 738, 739
Pavo cristatus	*Alectoris chukar*	0-1	0	-
		2	1	52
		3	3	51, 52, 53
		4	5	50, 51, 52, 53, 54
Dendragapus obscurus	*Colinus cristatus*	0-19	0	-
		20	2	349, 350
		21	8	141, 142, 156, 158, 348, 349, 350, 351