# Finding approximate occurrence of a pattern that contains gaps by the bit-vector approach

Inbok Lee[1],, Kunsoo Park[1*]

[1] School of Computer Science and Engineering, Seoul National University

[*]To whom correspondence should be addressed. E-mail: kpark@theory.snu.ac.kr

## Abstract

The application of finding occurrences of a pattern that contains gaps includes information retrieval, data mining, and computational biology. As the biological sequences may contain errors, it is important to find not only the exact occurrences of a pattern but also approximate ones. In this paper we present an $O(mnk_{\max}/w)$ time algorithm for the approximate gapped pattern matching problem, where $m$ is the length of the text, $n$ is the length of the pattern, $w$ is the word size of the target machine, and $k_{\max}$ is the greatest error bound for subpatterns.

## Introduction

Given a pattern $P$ and a text $T$, the pattern matching problem is to find all the occurrences of $P$ in $T$. The approximate pattern matching problem is to find all the positions of $T$ where $P$ appears with at most $k$ errors. The measure for the errors is the edit distance which means the minimum number of edit operations to convert one string into the other, where the edit operation is one of the following: insertion of a character, deletion of a character, and substitution of a character for another one.

In this paper, we want to find the exact and approximate occurrences of $P$ that contain *gaps*. Let $T$ be a string of length $n$ over an alphabet $\Sigma$. $T[i]$ denotes the $i$-th character of $T$. $T[i..j]$ denotes the substring of $T[i]T[i+1]...T[j]$. The pattern that we want to find from the text $T$ is $P = p_1 *_{(a_1,b_1)} p_2 *_{(a_2,b_2)} \cdots p_{l-1} *_{(a_{l-1},b_{l-1})} p_l$. Each subpattern $p_i$ $(1 \le i \le l)$ is a string over the alphabet $\Sigma$ and $m = \sum_{i=1}^{l} |p_i| + \sum_{i=1}^{l-1} b_i$. A gap $*_{(a,b)}$ is a string of consecutive wild cards or don't care characters whose length is between $a$ and $b$. We can think of a gap as a distance constraint between the neighboring occurrences of subpatterns: that is, if the pattern is $P = p_1 \cdots p_i *_{(a_i,b_i)} p_{i+1} \cdots p_l$ and there is an

This work is supported by BK 21 Project, IMT 2000 Project AB02, MOST grant M1-0309-06-0003.

occurrence $p_i$ ending at $T[j]$, the occurrences of $p_{i+1}$ should start between $[j+a+1, j+b+1]$.

**Definition 1. The exact gapped pattern matching problem** is to find all the occurrences of *P* from *T*. In **the approximate gapped pattern matching problem**, each subpattern $p_i$ has an error probability $r_i$ $(0 \le r_i < 1)$ and we allow $k_i = |p_i| * r_i$ errors for each $p_i$. Then the problem is to find all the substrings of *T* that match each subpattern $p_i$ with $k_i$ edit distance and satisfy the gap constraint.

**Example 1.** Let $P = AA*_{(2,3)}GC*_{(1,3)}TT$ and $T = GCAATTGCACTTC$. Then we can find an exact occurrence of *P* from *T*. See Figure 1.

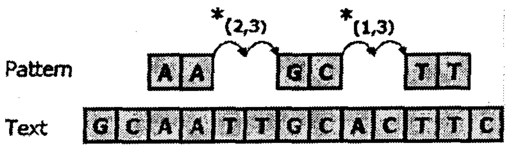

Figure 1: Finding $P = AA*_{(2,3)}GC*_{(1,3)}TT$ from $T = GCAATTGCACTTC$.

The application of this problem includes information retrieval, data mining and computational biology. We will focus on the latter one. Especially, the main application of the gapped pattern matching is to find a relationship between the protein sequences that we already know their function and the new sequence whose function we do not know. In some cases, we can't find the relationship by overall sequence alignment. We can find it by searching for motifs from the unknown protein sequence. A *motif* is a substring or a small similar subsequence that is common to many of the strings in the set. Recently people are working on building huge protein databases including PROSITE [2]. If the new sequence contains a motif that can be found from some protein family, then we can conjecture that it may have the same function as the protein family does. As the biological sequences may contain some errors due to various reasons, it is natural that we want to find not only exact matching occurrences of the pattern but also approximate one.

In this paper we show an algorithm for the approximate gapped pattern matching problem based on the bit-vector approach. Our algorithm is based on Navarro and Raffinot's bit-vector approach [5], but their algorithm has only one error bound *k* for the whole pattern *P*. Hence, it is possible that most errors occur in one subpattern. If we consider the application in biology, it is more appropriate to have an error bound $k_i$ for each subpattern $p_i$ [6].

## Previous works

The simplest approach is to represent the gapped pattern as a regular expression and find the occurrences by regular expression matching. For example, a pattern $LIV*_{(1,2)}LM$ that contains a gap $*_{(1,2)}$ can be represented as a regular expression $LIV*(*|\varepsilon)LM$. Then we use the well-known regular pattern matching algorithm, that is, create a nondeterministic finite state automata (NFA) and either convert it into a deterministic finite state automata (DFA) or just use the NFA. See [3] for more details.

This approach is too general, as the set of pattern that contain gaps is a proper subset of strings that can be represented by the regular expression. Furthermore, finding the approximate matching occurrences of a regular expression is difficult and time-consuming. So we need a more specialized approach for the gapped pattern matching problem.

Navarro and Raffinot [5] presented an $O(mn/w)$-time algorithm for the exact gapped pattern matching problem with bounded size gaps and an $O(kmn/w)$-time algorithm for the approximate one, where $k$ is the number of errors and $w$ is the word size of the target machine. Their algorithm uses the bit-vector simulation of NFA. We will discuss it later.

Akutsu showed an $O(m'n\log n)$ time algorithm for the approximate gapped pattern matching problem in [1] where $m'=\sum_{i=1}^{l}|p_i|$ (It considers a gap $*_{(a,b)}$ as a character, where Navarro and Raffino's algorithm considers it as $b$ characters). It uses a combination of a balanced search tree and the traditional dynamic programming approach for the approximate pattern matching problem. The error bound $k$ is for the whole the whole pattern $P$, which is the same as Navarro and Raffinot's.

Lee, Iliopoulos, Apostolico and Park showed an $O(nl+m')$-time algorithm for the exact gapped pattern matching problem and an $O(m'n)$-time algorithm for the approximate one [4]. They first proposed an error model where each subpattern has its own error bound.

## Algorithm

### Navarro and Raffinot's approach

Navarro and Raffinot proposed a bit-vector approach for the gapped pattern matching problem. First, build the NFA for the pattern $P$. For example, assume $P = LIV*_{(1,2)}LM$. Figure 2 is the NFA for $P$.

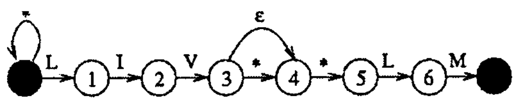

Figure 2: The NFA for $P = LIV*_{(1,2)}LM$

Then they simulate this NFA with a bit-vector $D = D_m...D_0$. There is a one-to-one relation between each bit of D and each state of the NFA: if the NFA is at the state $i$, then the $i$-th bit of $D$ is 1. Note that since an NFA can be at more than one state, there may be two or more 1's in $D$. Originally $D = 0^m$. The next step is to build the bitmask $B$. The $i$-th bit of $B[x]$ is 1 if and only if $P_i = x$ : in our example, $B[M]=1011001$, $B[L]=0110011$, $B[M]=1011001$, and $B[M]=1011001$. (Note that * is a "wild card" which matches any character in $\Sigma$).

Scanning $T$ from left to right ($T_j$ is the current character.), we update $D$ with the following equation.

$$D' \leftarrow ((D << 1) \mid 0^{m-1}1) \,\&\, B[T_j] \qquad (1)$$

Equation 1 is straightforward. To reach the state $i$ with the current character $T_j$, first the NFA should be at the state $i-1$: that is, the $(i-1)$-th bit of $D$ should be 1. If the transitive function $\delta(i-1,T_j) \to i$ is defined and the $i$-th bit of $B[T_j]$ is 1, equation 1 sets the $i$-th bit of $D$ to 1. And the NFA should be always be at the state 0: the bitwise OR operation against $0^{m-1}1$ guarantees this condition.

In the above example, if the NFA is at the state 3, by the $\varepsilon$-transitions, it should be at the state 4 simultaneously. To handle these $\varepsilon$-transitions, we use two additional bitmasks. A *gap-initial state* is the state where an $\varepsilon$-transition leaves. We create a bitmask $I$ for the gap-initial states: in the above example, $I = 00001000$ since the state 3 is a gap-initial sate. A *gap-final state* is the last state reached by an $\varepsilon$-transition. We create a bitmask $F$ for the gap-final sates: in the above example, $F = 00100000$ since the state 5 is the gap-final state. Then after updating $D$, we simulate by the following equation.

$$D' \leftarrow D \mid ((F-(D \,\&\, I)) \,\&\, {\sim} F) \qquad (2)$$

First we check whether the NFA is at the gap-initial states by computing $D \,\&\, I$ . If so, computing $(F-(D \,\&\, I)) \,\&\, {\sim} F$ yields the states reached by the $\varepsilon$-transitions. If not, by the bitwise AND operation against ${\sim} F$ , we remove undesired 1's.

For the approximate gapped pattern matching problem, they duplicate the above NFA $k$ times and link the states by the following rules. Figure 3 is an example.

- The downward arrow via $\Sigma$ means an "insertion".
- The diagonal arrow via $\Sigma$ means a "substitution".
- The diagonal arrow via ε means a "deletion".

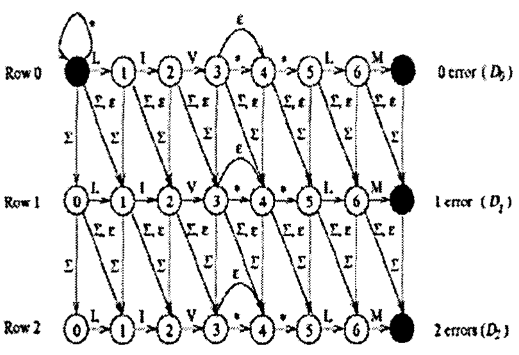

Figure 3: The NFA for $P = LIV *_{(1,2)} LM$ with $k = 2$ edit distance.

We keep $k+1$ words $D_0, D_1, \ldots, D_k$ for representing the rows of the NFA. We update $D'_i$ from $D_i$ , $D_{i-1}$ , and $D'_{i-1}$ by the following equations.

$$D'_i \leftarrow (D_i \ll 1) \,\&\, B[T_j] \qquad (3)$$

$$D'_i \leftarrow D'_i \mid D_{i-1} \mid ((D_{i-1} \ll 1) \mid 0^{m-1}1) \mid (D'_{i-1} \ll 1) \qquad (4)$$

$$D'_i \leftarrow D'_i \mid ((F-(D'_i \,\&\, I)) \,\&\, {\sim} F) \qquad (5)$$

The first and last equations are easy to understand. The middle one is slightly harder. $D_{i-1}$ means the vertical arrows via $\Sigma$ (insertions). $(D_{i-1} \ll 1) \mid 0^{m-1}1$ means the diagonal arrows via $\Sigma$ (substitutions). $D'_{i-1} \ll 1$ means the diagonal arrow via $\varepsilon$ (deletions).

**Our algorithm**

As we mentioned above, we deal with the approximate gapped pattern matching problem where each subpattern has its own error bound. Moreover, the gap constraint must be conserved:

we do not allow errors in gaps. If we need to allow $k$ errors in gaps, we replace the gap constraint $*_{(a,b)}$ with $*_{(a-k,b+k)}$. (In gaps, only insertions and deletions are important while substitutions are not.) We build short NFAs for each subpattern $p_i$, duplicate it $k_i$ times, and link the states by the same rules as we did in Navarro and Raffinot's algorithm. Finally we link these NFAs by the gap constraint. Figure 4 is an example. Note that the resulting NFA is not in a rectangular shape, but in a rather complex one.

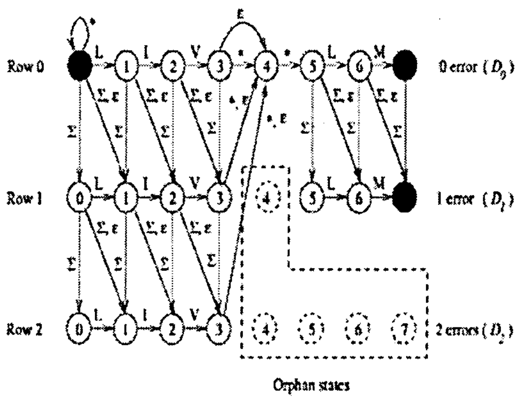

Figure 4: The NFA for $P = LIV *_{(1,2)} LM$ with $k_1 = 2$ and $k_2 = 1$ .

Before describing our algorithm, we first define the orphan states. An *orphan state* is a state where no transition reaches. The bits representing these orphan states should be set to 0. To do so, we create $k+1$ bitmasks $O_0, O_1, ..., O_k$. The $i$-th bit of $O_j$ is set to 0 if the state $i$ of the row $j$ in the NFA is an orphan state, and 1 otherwise. In the above example, $O_0 = 11111111$ , $O_1 = 11101111$ , and $O_2 = 00001111$ . Then equation 4 should be modified as follows.

$$\begin{aligned} D'_i \leftarrow (D'_i \mid D_{i-1} \mid \\ (((D_{i-1} \& O_i) << 1) \mid 0^{m-1}1) \mid \\ ((D'_{i-1} \& O_i) << 1)) \& O_i \end{aligned} \qquad (6)$$

The meaning of equation 6 is as follows.

- After equation 3, we set the orphan states to 0: $D'_i \& O_i$.
- To handle the vertical arrows via $\Sigma$ (insertion): $D_{i-1} \& O_{ii}$.
- To handle the diagonal arrows via $\Sigma$ (substitutions): $(((D_{i-1} \& O_i) << 1) \mid 0^{m-1}1) \& O_i$.
- To handle the diagonal arrows via $\varepsilon$ (deletions): $((D'_{i-1} \& O_i) << 1) \& O_i$.

Finally, calculating the bitwise OR operation among them and applying the de Morgan's law yields equation 6.

We need to handle the transition related to the gaps. If we reached one of the state 3's of each row in Figure 4, we encountered an occurrence of *LIV* within 2 errors. Then we proceed to the next states. To do so, we create a temporary variable *G*: initially, $G = 0^m$ . It maintains the information whether the NFA reached one of the gap-initial states. We update *G* with the following equation.

$$G \leftarrow G \mid (D'_i \& O_i \& I) \qquad (7)$$

The whole algorithm is as follows.

1. $k_{\max} = \max_{1 \le i \le l} k_i$ ($l$ is the number of subpatterns.).
2. Create the bitmasks *I*, *F*, and *O*.
3. Set $D_i \leftarrow 0^m$ . $(0 \le i \le l)$
4. Scan the text *T* from left to right ($T_j$ is the current character.) and do the

```
following.
    A. Set $G = 0^m$.
    B. For $i = 0,1,\ldots k_{\max}$, compute the following equations.
        $D'_i \leftarrow (D_i << 1) \& B[T_j]$
        $D'_i \leftarrow (D'_i \mid D_{i-1} \mid$
            $(((D_{i-1} \& O_i) << 1) \mid 0^{m-1}1) \mid$
            $((D'_{i-1} \& O_i) << 1)) \& O_i$
        $G \leftarrow G \mid (D'_i \& O_i \& I)$
    C. Compute the $\varepsilon$-transitions in the row 0.
        $D'_0 \leftarrow D'_0 \mid ((F - G) \& \sim F)$
5. Report the $j$-th position of $T$ as an occurrence of $P$ if one of $D'_i$'s MSB is 1.
```

**Time complexity**

The algorithm consists of the preprocessing (step 1-3) and the main algorithm (step 4-5). It takes $O(m|\Sigma|)$ time for creating the bitmask $B$, $O(mk_{\max})$ time for $O$ and $D$, and $O(m)$ time for $I$ and $F$. It is easy to show that the main algorithm runs in $O(mnk_{\max}/w)$ time, where n is the length of the text, m is the length of the pattern, and w is the word size of the target machine. (Compare with an $O(mnk/w)$-time algorithm in [5], where $k = \sum_{i=1}^{l} k_i$. It follows that $k_{\max} \leq k$.)

## Conclusion

We showed an $O(mnk_{\max}/w)$-time algorithm for the approximate gapped pattern matching problem. It has a better time complexity than the algorithm in [5]. This algorithm can be useful when we search the protein databases.

## Acknowledgements

We would like to thank Costas Iliopoulos (Department of Computer Science, King's College London), Sung Gyoo Park and Guhung Jung (School of Biological Science, Seoul National University) for their kind help.

This work is supported by BK 21 Project, IMT 2000 Project AB02, MOST grant M1-0309-06-0003.

## References

[1] T. Akutsu, Approximate string matching with variable length don't care characters, *IEICE Transaction on Information and Systems,* 1996, E79-D 1353-1354

[2] L. Falquet, M. Pagni, P. Bucher, N. Hulo, C. J. Sigrist, K. Hofmann, and A. Bairoch, The PROSITE database, its status in 2002, *Nucleic Acids Research,* 2002, 30:235-238

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

[4] I. Lee, A. Apostolico, C. Iliopoulos, and K. Park, Finding approximate occurrence of a pattern that contains gaps, *Proceedings of the Fourteenth Australasian Workshop on Combinatorial Algorithms (AWOCA '03),* 2003, 89-100

[5] G. Navarro and M. Raffinot, Fast and simple character classes and bounded gaps pattern matching, with application to protein searching, *Proceedings of RECOMB 2001,* 2001, 231-240

[6] G.G. Sutton, O. White, M. D. Adams, and A.R. Kerlavage, TIGR Assembler: a new tool for assembling large shotgun sequencing projects, *Genome Science and Technology 1*, 1995, 9-19