

Pattern Matching in Degenerate DNA/RNA Sequences

M. Sohel Rahman^{1,3***}, Costas S. Iliopoulos^{1,3***} and L. Mouchard^{2,4}

¹ Algorithm Design Group

Department of Computer Science, King's College London
Strand, London WC2R 2LS, England

<http://www.dcs.kcl.ac.uk/adg>

² ABISS, Université de Rouen

76821 Mont Saint-Aignan Cedex, France

³ {csi, sohel}@dcs.kcl.ac.uk

⁴ laurent.mouchard@univ-rouen.fr

Abstract. In this paper, we consider the pattern matching problem in DNA and RNA sequences where either the pattern or the text can be *degenerate* i.e. contain *sets of characters*. We present an asymptotically faster algorithm for the above problem that works in $O(n \log m)$ time, where n and m is the length of the text and the pattern respectively. We also suggest an efficient implementation of our algorithm, which works in linear time when the pattern size is small. Finally, we also describe how our approach can be used to solve the distributed pattern matching problem.

Keywords: algorithm, degenerate, DNA/RNA sequence, pattern matching.

1 Introduction

While most of the biology labs are using dedicated high-throughput equipments to produce large DNA sequences on a daily basis, the need for automatic annotation and content analysis is greater everyday. Unfortunately, the quality of the automatically-obtained sequences is sometimes questionable: among the factors that are impacting the quality, we can cite at least the intrinsic limitations of the equipments and the natural polymorphism that can be observed between individual samples (e.g. a Simple Nucleotide Polymorphism, that is a unique mutation, can either stop the traduction of a mRNA into a protein sequence, or create a binding site for a protein complex that will prevent the complete formation of the functional protein [6, 18]). Analyzing these uncertain sequences

* Supported by the Commonwealth Scholarship Commission in the UK under the Commonwealth Scholarship and Fellowship Plan (CSFP).

** On Leave from Department of CSE, BUET, Dhaka-1000, Bangladesh.

*** Supported by EPSRC and Royal Society grants.

M. Kaykobad and Md. Saidur Rahman (Eds.): WALCOM 2007, pp. 109–120, 2007.

is therefore much more complicated than the traditional problem, where given a pattern \mathcal{P} and a text \mathcal{T} , the pattern matching problem is to find all the occurrences of \mathcal{P} in \mathcal{T} . There exist, for decades, efficient algorithms that solve this problem [2, 17] but in our case, some positions in the pattern or the text can contain a set of characters (like the IUB alphabet for example [23]), instead of a single letter. Most of the existing exact methods are useless, or have to be adapted in a dramatic way, reducing their efficiency. Designing new efficient algorithms that can tackle these specific requirements is therefore a must.

In this paper, we present an asymptotically faster algorithm for finding patterns, where either the pattern or the text can be *degenerate* i.e. each symbol is a *set of characters*. Our algorithm for DNA and RNA sequences works in $O(n \log m)$ time, where n and m is the length of the text and the pattern respectively. We also suggest an efficient implementation of our algorithm, which works in $O(n + m + n \lceil \frac{m}{w} \rceil + \lceil \frac{n}{w} \rceil)$ time, where w is the word size of the target machine. It is easy to observe that for patterns having length similar to the word size (i.e. $m \sim w$) this running time would be linear. It may also be noted here that there are numerous practical examples of such cases since typical word size now a days is as high as 64. Finally, we also show how our approach can be used to solve the distributed pattern matching problems introduced in [9] and handled in [12]. The rest of the paper is organized as follows. In Section 2, we present the vocabulary and the notions that will be used in this paper, in Section 3, we present a brief literature review. In Sections 4 and 5, we present our approach and in Section 6 we show how we can use our idea to solve the distributed pattern matching problem. Finally, we conclude in Section 7.

2 Preliminaries

In what follows, we are considering a finite alphabet Σ . For DNA sequences, we have $\Sigma = \{A, T, C, G\}$ and for RNA sequences $\Sigma = \{A, U, C, G\}$. Assume that we are given a text $\mathcal{T} = \mathcal{T}_1 \dots \mathcal{T}_n$ of length n and a pattern $\mathcal{P} = \mathcal{P}_1 \dots \mathcal{P}_m$ of length m .

The classical pattern matching problem consists in locating all the occurrences of \mathcal{P} in \mathcal{T} , that is, all possible i such that for all j in $[1, m]$, $\mathcal{T}[i + j - 1] = \mathcal{P}[j]$. This problem can be extended in a very interesting way by considering degenerate strings, which means that the strings \mathcal{T} or/and \mathcal{P} are built over the potential $2^{|\Sigma|} - 1$ non-empty sets of letters belonging to Σ . Note that in the literature, strings containing sets of characters are also referred to as *indeterminate* strings. In what follows, the set containing A and T will be denoted by $[AT]$ and the singleton $[C]$ will be simply denoted by C for ease of reading.

Now let us define the problems more formally.

Problem 1. We are given a degenerate text \mathcal{T} and a pattern \mathcal{P} . The problem is to find all the occurrences of \mathcal{P} in \mathcal{T} i.e. to find all i such that for all j in $[1, m]$, $\mathcal{P}[j] \in \mathcal{T}[i + j - 1]$.

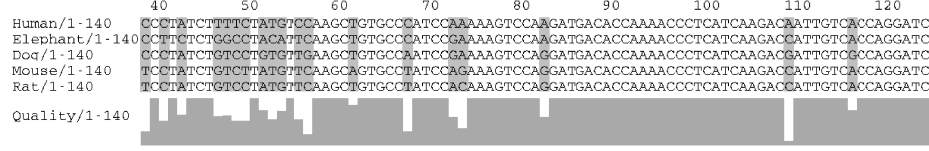


Fig. 1. An example of conserved gene sequence across species

In Figure 1, we are presenting an example of a degenerate text, that is the alignment of the five sequences, from different organisms, for one specific gene. The percentage of similarity between these five sequences is very high, most of the nucleotides being conserved (the grey columns correspond to positions where the nucleotides are species-dependent), a consensus built upon these sequences may be represented by $[CT]C[CT]T[AC]TCT[GT] \dots$. So, instead of finding a pattern separately in each of these sequences, it is natural and, perhaps, more efficient to find the pattern in the consensus sequences which gives us the motivation to solve Problem 1 efficiently.

Problem 2. We are given a text \mathcal{T} and a degenerate pattern \mathcal{P} . The problem is to find all the occurrences of \mathcal{P} in \mathcal{T} i.e. to find all i such that for all j in $[1, m]$, $\mathcal{T}[i + j - 1] \in \mathcal{P}[j]$.

The motivation to solve Problem 2 is as follows. Most of the sites the protein complexes are binding to, may be represented by degenerate words, e.g. the basic elements of a translation initiation site in *E. coli*, most conveniently represented as their DNA counterparts, may be 5'-

$$[GA][GA]GGGNNNNAN[CT]ATGNN[AT]NNNNN[CTG]$$

-where N is the don't care symbol replacing all other letters (adapted from [26]). Note also that $NAGNAG$ is the major form of alternative 3' splice sites accounting for 50% of the cases and are used for detecting intron/exon boundary [3].

Remark 1. It is clear that Problem 1 and Problem 2 differ only in whether the pattern or the text contains sets of characters. The reason we distinguish between the two problems will be clear as we proceed. We should, however, point out that this paper doesn't deal explicitly with the case where both text and pattern are degenerate.

Example 1. Suppose we have a text \mathcal{T} and a degenerate pattern \mathcal{P} :

$\mathcal{T} = A C C G G A A G T A A G T C G T A A A T$

$$\mathcal{P} = [AC] G [CGT] A A [ACGT] T$$

$\mathcal{P}[1]$ can match either A or C in the \mathcal{T} whereas $\mathcal{P}[3]$ can match either C , G or T in \mathcal{T} . Finally, $\mathcal{P}[6]$ can match any letter in the text and is named a don't care symbol. It can easily be verified that in this case we have three occurrences of \mathcal{P} in \mathcal{T} , starting at positions 3, 7 and 14.

In this paper, we present algorithms to solve both Problem 1 and 2. It may be noted here that Problem 1 can be easily reduced to the *subset matching problem* as defined below.

Problem 3. We are given a degenerate text \mathcal{T} and a degenerate pattern \mathcal{P} . The problem is to report all the occurrences of \mathcal{P} in \mathcal{T} . The pattern is said to occur at text position i if for all j in $[1, m]$, $\mathcal{P}[j] \subseteq \mathcal{T}[i + j - 1]$.

Remark 2. Note that Problem 3 is different from the natural extension of Problem 1 and 2 where both the text and pattern are degenerate. The difference comes from the different notion of match used in Problem 3. As an example let $\mathcal{P}[i] = [AC]$ and $\mathcal{T}[j] = [CT]$. Although we have $\mathcal{P}[i] \cap \mathcal{T}[j] = [C] \neq \emptyset$ according to Problem 3 its not a match because $\mathcal{P}[j] \not\subseteq \mathcal{T}[i + j - 1]$. Note however that we have a match if $\mathcal{T}[j] = [ACT]$.

Problem 3 was first introduced in [7] where the authors presented a $O((n + s)\sqrt{m \log m})$ time⁵ algorithm to solve the subset matching problem where s is the sum of the text and pattern set sizes. Later, in [8], a $O(n \log^3 n)$ algorithm to solve the problem was also devised.

It is easy to note that Problem 1 can be seen as a special case of Problem 3 where each of the position of the pattern \mathcal{P} is a singleton set. Therefore, we can easily solve Problem 1 using the algorithms presented in [7, 8]. However, inspired by a technique used in [7] to solve a subproblem, we will present an approach to solve both Problem 1 and 2 which will give an asymptotically faster algorithm for the cases where the alphabet size is small. As a result the algorithms we present will be best suited for biological problems, more precisely for DNA and RNA sequences.

3 Literature Review

The well-known *don't care matching problem*, where a don't care character can match any character of the alphabet, can be seen as a special case of our problem as is evident from Section 2. Don't care matching problems and variants thereof have received tremendous attention in stringology research. Although, these algorithms can't be directly generalized to solve the more general problem we wish to handle, we present a brief literature review of the problem because we need to solve a restricted version of the problem in our algorithm.

⁵ the authors also presented a $O((n + s) \log^3 m)$ randomized algorithm for the same problem.

Fischer and Paterson [11] introduced and solved the don't care matching problem in $O(n \log m \log \Sigma)$ time⁶. Since their (deterministic) algorithm in 1974, the only improvements were by Muthukrishnan and Palem [21], who reduced the constant factor. And in fact the string matching with don't cares problem is proved to be at least as hard as the boolean convolution problem [22]. Indyk [15], on the other hand gave a randomized algorithm that also involved convolutions, running in time $O(n \log n)$. Kalai [16] presented a slightly better but much simpler randomized algorithm which runs in $O(n \log m)$ time. Pinter [25] on the other hand used the Aho-Corasick algorithm [2] to solve the problem. Unfortunately, however, the time complexity, in the limiting case, can be as high as $O(mn)$ [24].

There have been a number of attempts to handle the degenerate words or the indeterminate string as is sometimes mentioned in the literature, although mostly from a practical context. Abrahamson, dealt with a similar problem in [1] where he presented an algorithm that runs in $O(n + m\sqrt{n} \log n \sqrt{\log \log n})$ time. However, this problem can be tackled efficiently, in practice, by the bit-mapping technique, originally proposed in [10], reinvented in [4, 29] and available in the *agrep* utility [28], until recently virtually the only practical algorithm available for indeterminate pattern-matching. The algorithm presented in [4] can solve the problem in $O(\lceil mb/w \rceil n)$ time given a preprocessing time of $O(\lceil mb/w \rceil (m|\Sigma| + |\Sigma|))$. Here w is the length of the word in the target machine and b is the number of bits to represent each individual "state" (see [4] for detail). Very recently Holub et al. [13] presented practically efficient algorithms for the same problem and a number of variants thereof. The algorithms of [13] are based on the Sunday variant [27] of the Boyer-Moore algorithm [5], in practice [14, 19] one of the fastest exact pattern-matching algorithms.

Recently, Lee et al. [20] cleverly used a bit masking technique to handle the sets of characters for problems where the alphabet size is small. The motivation comes from the fact that in biological applications the alphabet size is very small and can be considered constant. And notably, the degenerate words are frequently found in DNA and RNA sequences and hence is very much relevant with respect to the biological problems. In this paper we also try to exploit the advantage of a small alphabet size to handle the degenerate words in an efficient way.

4 Our approach

We first discuss our approach for Problem 1 where we are given a pattern \mathcal{P} and a text \mathcal{T} and \mathcal{T} can contain sets of characters. The basic idea of our approach is as follows. We consider each position in \mathcal{T} and \mathcal{P} as a class or a set (for \mathcal{P} each position is a set of one character i.e. a singleton set). For each character in the alphabet, we construct a smaller similar problem, namely a *restricted* don't care matching problem, to be defined shortly, and solve them separately. The

⁶ The running time reported in [11], $O(n \log^2 m \log \log m \log |\Sigma|)$, is slightly higher because they do not use the RAM model.

results are then combined to get the final result. Let us first formally define the restricted don't care matching problem.

Problem 4. We are given a text \mathcal{T} and a pattern \mathcal{P} over $\Sigma \cup \{*\}$, where Σ is such that $|\Sigma| = 2$ and $*$ is a don't care character. The problem is to report all the occurrences of \mathcal{P} in \mathcal{T} .

Remark 3. Since $|\Sigma| = 2$, using the algorithm of [11] we can solve Problem 4 in $O(n \log m)$ time.

Algorithm 2

Input: A text \mathcal{T} such that \mathcal{T} may contain sets of characters and a pattern \mathcal{P} which does not contain sets of characters.

output: The set of indices where \mathcal{P} occurs in \mathcal{T} .

- 1: **for** $a \in \Sigma$ **do**
 - 2: Construct \mathcal{T}' where for all $i, 1 \leq i \leq n, \mathcal{T}'[i] = \begin{cases} 1, & \text{if } \mathcal{T}[i] = a \\ 0, & \text{if } \mathcal{T}[i] \neq a. \end{cases}$
 - 3: Construct \mathcal{P}' where for all $i, 1 \leq i \leq m, \mathcal{P}'[i] = \begin{cases} 1, & \text{if } \mathcal{P}[i] = a \\ *, & \text{if } \mathcal{P}[i] \neq a. \end{cases}$
 - 4: Solve the don't care matching problem for the text \mathcal{T}' and the pattern \mathcal{P}' . Let M_a denote the set of indices where \mathcal{P}' occurs in \mathcal{T}' i.e. $M_a = \{i \mid \mathcal{P}' \text{ occurs at position } i \text{ of } \mathcal{T}'\}$
 - 5: **end for**
 - 6: Compute $M = \bigcap_{a \in \Sigma} M_a$
 - 7: **return** M
-

Now we are ready to formally state the algorithm in the form of Algorithm 2. The analysis of Algorithm 2 is straightforward. Step 2 and Step 3 can be done implicitly, while performing Step 4. Therefore in Step 1 we basically perform Step 4 $|\Sigma|$ times requiring $O(|\Sigma|n \log m)$ time. Step 6 can be done incrementally in the loop and hence the total running time would remain the same i.e. $O(|\Sigma|n \log m)$.

Remark 4. It is easy to verify that, if \mathcal{P} contains sets of characters instead of \mathcal{T} , we just need to swap the definitions of \mathcal{T}' and \mathcal{P}' in Step 2 and Step 3 respectively.

Remark 5. We would also like to note that, if both \mathcal{T} and \mathcal{P} are degenerate the algorithm wouldn't work properly.

Example 2. Suppose we are given a text \mathcal{T} and a pattern \mathcal{P} as follows:

$$\mathcal{T} = AA[CGT]G[AG]T[AT]CG[AG]TATC[ACGT]C[GT]GTAA[CG]$$

$$\mathcal{P} = ACGGTA$$

How the algorithm works is illustrated in Figure 2, 3, 4 and 5.

Text	A	A	[CGT]	G	[AG]	T	[AT]	C	G	[AG]	T	A	T	C	[ACGT]	C	[GT]	G
A	1	1	0	0	1	0	1	0	0	1	0	1	0	0	1	0	0	0
C	0	0	1	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0
G	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	0	1	1
T	0	0	1	0	0	1	1	0	0	0	1	0	1	0	1	0	1	0

Fig. 2. Text \mathcal{T} and \mathcal{T}' for each $a \in \Sigma$ of Example 2

Pattern	A	C	G	G	T	A
A	1	*	*	*	*	1
C	*	1	*	*	*	*
G	*	*	1	1	*	*
T	*	*	*	*	1	*

Fig. 3. Pattern \mathcal{P} and \mathcal{P}' for each $a \in \Sigma$ of Example 2

Text	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
M_A		✓				✓		✓			✓			✓				
M_C		✓					✓						✓					
M_G	✓	✓					✓											
M_T	✓	✓	✓				✓		✓		✓		✓					
M	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fig. 4. Intermediate matching solutions (M_a for each $a \in \Sigma$) and the final solution M of Example 2

A	C	G	G	T	A												
A	A	[CGT]	G	[AG]	T	[AT]	C	G	[AG]	T	A	T	C	[ACGT]	C	[GT]	G
A	C	G	G	T	A												

Fig. 5. Occurrence of \mathcal{P} in \mathcal{T} of Example 2

We now present our results in the form of following theorem and corollaries.

Theorem 1. *Problem 1 and 2 can be solved in $O(|\Sigma|(\chi + n))$ time where χ is the running time to solve Problem 4.*

Remark 6. Note that the $O(n)$ factor in the running time in Theorem 1 comes from the intersection operation required in Step 6.

Corollary 1. *Problem 1 and 2 can be solved in $O(|\Sigma|n \log m)$ time.*

Corollary 2. *For DNA and RNA sequences Problem 1 and 2 can be solved in $O(n \log m)$ time.*

In the rest of this section we discuss a practically efficient implementation technique of our approach.

5 Implementation

In Section 4 we have presented a $O(n \log m)$ time algorithm for both Problem 1 and 2. Our algorithm makes use of the Fast Fourier Transformation (FFT) technique [11]. Since FFT algorithms have large hidden constants, in real applications they don't perform well. It is easy to see that our algorithms also suffer from

the same problem inspite of the good theoretical bound. In this section, on the other hand, we present a technique to implement our approach without employing FFT. As will be seen from the analysis, the resulting algorithm runs in linear time for sufficiently short patterns.

As before we first consider Problem 1 and then show the changes that need be done to solve Problem 2. The key idea of our implementation depends on the following interesting observation.

OBSERVATION 1 *Suppose S_1 is a string over the alphabet $\{1, 0\}$ and S_2 is a string over the alphabet $\{1, *\}$ where ‘ $*$ ’ is the don’t care character. Assume also that $|S_1| = |S_2|$. Now S_2 matches S_1 if and only if $S'_2 \wedge S_1 = S'_2$ where ‘ \wedge ’ is the bitwise ‘and’ operator and $S'_2[i] = \begin{cases} 1, & \text{if } S_2[i] = 1 \\ 0, & \text{if } S_2[i] = * \end{cases}$.*

Based on Observation 1 we can present a straightforward algorithm (Algorithm 3) to implement Algorithm 2. In what follows we use the idea of factor as follows. Given a string S , a factor of S , denoted by $S[i..j]$ is the substring $S[i]S[i+1]...S[j]$ where $1 \leq i \leq j \leq |S|$.

Algorithm 3

```

1: for  $a \in \Sigma$  do
2:   Construct  $T'$  where for all  $i, 1 \leq i \leq n, T'[i] = \begin{cases} 1, & \text{if } T[i] = a \\ 0, & \text{if } T[i] \neq a. \end{cases}$ 
3:   Construct  $P'$  where for all  $i, 1 \leq i \leq m, P'[i] = \begin{cases} 1, & \text{if } P[i] = a \\ 0, & \text{if } P[i] \neq a. \end{cases}$ 
4:    $M_a = \epsilon$  {Each  $M_a$  is a bit string and ‘+’ with respect to it indicates concatenation}
5:   for  $i = 1$  to  $n - m + 1$  do
6:     if  $P' \wedge T'[i..i+m-1] = P'$  then
7:        $M_a = M_a + '1'$ 
8:     else
9:        $M_a = M_a + '0'$ 
10:    end if
11:  end for
12: end for
13: Compute  $M = \bigwedge_{a \in \Sigma} M_a$ 
14: Return  $M$ 

```

Let us now analyze the running time of Algorithm 3. Step 2 and Step 3 requires $O(n + m)$ time. In the ‘For loop’ of Step 5, we perform bitwise ‘and’ operations for every position of the text⁷. What we plan to do is to perform the bitwise ‘and’ operations word by word. So, if w is the word size of the target machine then we get a running time of $O(n \lceil \frac{m}{w} \rceil)$ instead of $O(nm)$. Thus in total Step 5 can be performed in $O(|\Sigma|n \lceil \frac{m}{w} \rceil)$ time. Similarly, Step 13 can

⁷ We can do better than checking each position by applying techniques of [27, 5] and thereby improve our average running time

be performed in $O(|\Sigma|\lceil \frac{n}{w} \rceil)$ time. Therefore the overall running time should be $O(n + m + |\Sigma|n\lceil \frac{m}{w} \rceil + |\Sigma|\lceil \frac{n}{w} \rceil)$. In many practical cases we can assume $w \sim m$ which implies a linear running time for DNA and RNA sequences (since $|\Sigma| = 4$). So far we have only considered Problem 1. From the Observation 1 it should be clear that to handle Problem 2 the only change that need be done is in the ‘if’ statement of Step 6. The complete ‘if’ statement, modified to handle Problem 2 is given in Figure 6.

```

if  $\mathcal{P}' \wedge \mathcal{T}'[i..i + m - 1] = \mathcal{T}'[i..i + m - 1]$  then
     $M_a = M_a + '1'$ 
else
     $M_a = M_a + '0'$ 
end if
    
```

Fig. 6. Modification to handle Problem 2

We now summarize our results in the form of following theorem and corollary.

Theorem 2. *Problem 1 and 2 can be solved in $O(n + m + |\Sigma|n\lceil \frac{m}{w} \rceil + |\Sigma|\lceil \frac{n}{w} \rceil)$ time*

Corollary 3. *For DNA and RNA sequences Problem 1 and 2 can be solved in $O(n + m + n\lceil \frac{m}{w} \rceil + \lceil \frac{n}{w} \rceil)$ time.*

6 Other Applications

The approach we take to solve the problems in this paper can be applied in other contexts as well especially when the alphabet size is small. In this section we attack another related problem in pattern matching, namely the *Distributed Pattern Matching Problem* and show how we can solve it using our approach. A family of distributed pattern matching problems were first introduced in [9] and then handled in [12]. We first define the problems below.

Problem 5. We are given s texts $\mathcal{T}^i = \mathcal{T}^i[1]\mathcal{T}^i[2]...\mathcal{T}^i[n]$, $i \in \{1, \dots, s\}$ of equal length n over the alphabet Σ and a pattern $\mathcal{P} = \mathcal{P}[1]\mathcal{P}[2]...\mathcal{P}[m]$ of length $m \leq n$ over the same alphabet Σ . The problem is to find all occurrences of the pattern \mathcal{P} in the texts \mathcal{T}^i , $i \in \{1, \dots, s\}$, such that the various parts of the pattern \mathcal{P} can be located in consecutive positions of different texts, i.e. find all positions $k \in \{1, \dots, n - m + 1\}$, such that for each $j \in \{1, \dots, m\}$, there exists an integer $l_j \in \{1, \dots, s\}$, such that $\mathcal{P}[j] = \mathcal{T}^{l_j}[k + j - 1]$.

Problem 6. We are given a text $\mathcal{T} = \mathcal{T}[1]\mathcal{T}[2]...\mathcal{T}[n]$ of length n over an alphabet Σ and r patterns $\mathcal{P}^i = \mathcal{P}^i[1]\mathcal{P}^i[2]...\mathcal{P}^i[m]$, $i \in \{1, \dots, r\}$ of equal length $m \leq n$. The problem is to find all occurrences \mathcal{P} in \mathcal{T} such that each symbol of the found string matches a symbol from any pattern \mathcal{P}^i located at the corresponding

position, i.e. find all positions $k \in \{1, \dots, n-m+1\}$, in the text \mathcal{T} such that for each $j \in \{1, \dots, m\}$, there exists an integer $l_j \in \{1, \dots, r\}$ such that $\mathcal{P}^{l_j}[j] = \mathcal{T}[k+j-1]$.

In [12] the authors solve the above problems by constructing the Nondeterministic Finite Automata (NFA) for them and then simulating them using the Shift-Or algorithm [4]. For both Problem 5 and 6 the running time achieved in [12] is, $O((M + |\Sigma| + N)\lceil \frac{m}{w} \rceil)$. Here for Problem 5, $M = m$ and $N = ns$ and for Problem 6, $M = mr$ and $N = n$. We, on the other hand employ the same technique we use in Section 4. But before that we need to do some preprocessing as follows. In what follows we consider Problem 5. We are given s texts $\mathcal{T}^i, i \in \{1, \dots, s\}$ of equal length n and a pattern \mathcal{P} of length $m \leq n$. We construct a string \mathcal{T} from the given strings where each position j of \mathcal{T} is a set of characters comprising of the characters at position j of each of $\mathcal{T}^i, 1 \leq i \leq s$. In particular we construct a new (degenerate) string \mathcal{T} such that

$$\mathcal{T}[j] = [\mathcal{T}^1[j]\mathcal{T}^2[j] \dots \mathcal{T}^s[j]], \quad 1 \leq j \leq n.$$

It is easy to see that now we can use Algorithm 2 and 3 to solve Problem 5 by supplying \mathcal{T} and \mathcal{P} as input text and pattern respectively.

To solve Problem 6, the only change that need be done is in the preprocessing step described above. For this one we need to do the same preprocessing as described above the only difference being that the preprocessing is to be applied on the set of patterns instead of the text. The preprocessing step would take $O(ns)$ for Problem 5 and $O(mr)$ for Problem 6. So we get the following theorem.

Theorem 3. *Problem 5 can be solved in $O(ns + \xi)$ time and Problem 6 can be solved in $O(mr + \xi)$ time, where ξ is the running time of Algorithm 2.*

Using the result of Corollary 1 and Theorem 2 we get the following Corollaries.

Corollary 4. *Problem 5 can be solved in $O(ns + |\Sigma|n \log m)$ time and Problem 6 can be solved in $O(mr + |\Sigma|n \log m)$ time.*

Corollary 5. *Problem 5 and Problem 6 can be solved in $O(\mathcal{R} + n + m + |\Sigma|n\lceil \frac{m}{w} \rceil + |\Sigma|\lceil \frac{n}{w} \rceil)$ time, where $\mathcal{R} = ns$ and mr for Problem 5 and Problem 6 respectively.*

7 Conclusion

In this paper, we have presented an asymptotically faster algorithm for pattern matching, where either the text or the pattern can be degenerate. Our approach is alphabet dependent and hence is best suited for problems where the alphabet size is small. Therefore our algorithm is particularly suitable for DNA and/or RNA sequences. It may be noted here that degenerate words are frequently existent in problems with DNA and RNA sequences. In particular, we have presented an $O(n \log m)$ algorithm for pattern matching in DNA or RNA sequences where either the pattern or the text may be degenerate. We also present an efficient implementation of our algorithm that works in $O(n + m + n\lceil \frac{m}{w} \rceil + \lceil \frac{n}{w} \rceil)$ time. It

is easy to observe that for sufficiently short patterns this will work in linear time. We also consider a variant of pattern matching problem namely the distributed pattern matching problem and show how our approach can be used to solve this problem efficiently.

Finally, one other interesting aspect of our technique lies in the fact that if the text has sets of characters in the original problem then, in the subproblems created by our approach, the pattern contains don't care characters and vice versa. This aspect can be exploited to solve a number of pattern matching problems as follows. In the literature almost all don't care matching problems and variants thereof consider only patterns to have don't cares. So, if we need to solve a pattern matching problem, where the text contains don't cares instead of the pattern, we can apply our technique to transform the problem to problems where patterns have the don't cares instead of the text.

References

1. K. Abrahamson. Generalized string matching. *SIAM Journal of Computing*, 16(6):1039–1051, 1987.
2. A. Aho and M. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18:333–340, 1975.
3. M. Akerman and Y. Mandel-Gutfreund. Alternative splicing regulation at tandem 3' splice sites. *Nucl. Acid. Res.*, 34(1):23–31, 2006.
4. R. Baeza-Yates and G.H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35:74–82, 1992.
5. R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.
6. L. Cartegni and A.R. Krainer. Disruption of an SF2/ASF-dependent exonic splicing enhancer in SMN2 causes spinal muscular atrophy in the absence of SMN1. *Nature Genetics*, 30:377–384, 2002.
7. R. Cole and R. Hariharan. Tree pattern matching and subset matching in randomized $o(n \log^3 m)$ time. In *Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 245–254, 1999.
8. R. Cole, R. Hariharan, and P. Indyk. Tree pattern matching and subset matching in deterministic $o(n \log^3 n)$ time. In *Twenty Ninth Annual Symposium on the Theory of Computing*, pages 66–75, 1997.
9. T. Crawford, C.S. Iliopoulos, and R. Raman. String matching techniques for musical similarity and melody recognition. *Computing in Musicology*, 11:227–236, 1998.
10. B. Domolki. An algorithm for syntactical analysis. *Computational Linguistics*, 3:29–46, 1964.
11. M.J. Fischer and M.S. Paterson. String matching and other products. in *Complexity of Computation*, R.M. Karp (editor), *SIAM AMS Proceedings*, 7:113–125, 1974.
12. J. Holub, C.S. Iliopoulos, B. Melichar, and L. Mouchard. Distributed pattern matching using finite automata. *Journal of Automata, Languages and Combinatorics*, 6(2):191–204, 2001.
13. J. Holub, W. F. Smyth, and S. Wang. Fast pattern-matching on indeterminate strings. In *16th Australasian Workshop on Combinatorial Algorithms*, pages 415–428, 2005.

14. A. Hume and D. Sunday. Fast string searching. *Software Practice & Experience*, 21(11):1221–1248, 1991.
15. P. Indyk. Faster algorithms for string matching problems: matching the convolution bound. In *39th Symposium on Foundations of Computer Science*, 1998.
16. A. Kalai. Efficient pattern-matching with don't cares. In *Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 655–656, 2002.
17. D. E. Knuth, J. H. Morris Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350, 1977.
18. J.B.J. Kwoka, G. M. Hallidaybc, W. S. Brooksbd, G. Doliose, H. Laudonf, O. Murayamagand M. Halluppa, R. F. Badenhopac, J. Vickersh, R. Wange, J. Naslundf, A. Takashimag, S.E. Gandyi, and P.R. Schofieldacj. Presenilin-1 mutation l271v results in altered exon 8 splicing and alzheimer's disease with non-cored plaques and no neuritic dystrophy. *Journal of Biological Chemistry*, 278(9):6748–6754, 2003.
19. T. Lecroq. Experimental results on string matching algorithms. *Software Practice & Experience*, 25(7):727–765, 1995.
20. I. Lee, A. Apostolico, C. S. Iliopoulos, and K. Park. Finding approximate occurrence of a pattern that contains gaps. In *14th Australasian Workshop on Combinatorial Algorithms*, pages 89–100, 2003.
21. S. Muthukrishnan and K. Palem. Non-standard stringology: Algorithms and complexity. In *26th Symposium on the Theory of Computing, Canada*, 1994.
22. S. Muthukrishnan and H. Ramesh. Non-standard stringology: Algorithms and complexity. *Information and Computation*, 122(1):140–148, 1995.
23. Nomenclature Committee of the International Union of Biochemistry (NC-IUB). Nomenclature for incompletely specified bases in nucleic acid sequences. recommendations 1984. *Eur J Biochem*, (150):1–5, 1985.
24. R.Y. Pinter. Private communication.
25. R.Y. Pinter. Efficient string matching with dont care patterns. In A. Apostolico and Z. Galil, editors, *Combinatorial algorithms on words, NATO Advanced Science Institute Series F: Computer and System Sciences*, volume 12, pages 11–29, 1985.
26. G. D. Stormo, T. D. Schneider, L. Gold, and A. Ehrenfeucht. Use of the 'perceptron' algorithm to distinguish translational initiation sites in E. coli. *Nucl. Acids Res.*, (10):2997–3011, 1982.
27. D. M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, 33(8):132–142, 1990.
28. S. Wu and U. Manber. Agrep- a fast approximate pattern-matching tool. pages 153–162, 1992.
29. S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.