Approximate search with constraints on indels with application in SPAM filtering

Ambika Shrestha Chitrakar and Slobodan Petrović
Norwegian Information Security Laboratory, Gjøvik University College
ambika.chitrakar2@hig.no, slobodan.petrovic@hig.no

Abstract

Finding distorted occurrences of search pattern(s) in the search string by applying constraints on elementary edit operations (indels (insertions/deletions) and substitutions) is a new category of the approximate string search problem that we introduce in this paper. The constraint on the total number of indels can improve the search efficiency when one knows the probabilities of these edit operations for the distorted pattern/text. Two approximate search algorithms with such a constraint, CRBP-Indels and Sankoff-Indels, are presented here and their performances are evaluated for different probabilities of insertions, deletions, and substitutions. The experimental results show that CRBP-Indels has better performance over Sankoff-Indels when the number of indels is greater than the number of substitutions. However, the Sankoff-Indels algorithm is better if the number of substitutions is greater than the number of indels. Possible application of these algorithms is in SPAM filtering for detection of deliberately distorted SPAM-words. In such a scenario, the number of indels applied on the original SPAM-words must be limited in order to maintain their intelligibility.

1 Introduction

Approximate search assumes finding distorted occurrences of the search pattern(s) (P) in the search string (T). Distortion can be modeled by means of applying so-called elementary edit operations on strings - deletions, insertions and substitutions of symbols. Sometimes, it is useful to introduce various kinds of constraints into this model. These can include total numbers of some elementary edit operations (indels (insertions/deletions) and substitutions), lengths of runs of elementary edit operations and so on. The constraints have so far only been used to compute the distance [8, 9, 11], but not in approximate search. Approximate search is applied in cases where patterns appear in the search string with some level of distortion. For example, in SPAM E-mail filtering, if the spammer deliberately modifies the SPAM-words in order not to be discovered, approximate search can still detect these words. If we know the maximum probability of the words being modified in terms of the number of character insertions, deletions, or substitutions, then applying approximate search with constraints on edit operations could contribute to improvement of the efficiency in detection. This is the new scenario and the corresponding approximate search problem that we introduce in this paper.

Presented at the Norwegian Information Security Conference 2015 (NISK-2015)

We present two algorithms, Sankoff-Indels and CRBP-Indels, to solve the new problem of approximate search with the constraint on the total number of indels. The Sankoff-Indels algorithm is based on the classical dynamic programming approach to string editing [3,6,11] whereas the CRBP-Indels exploits the bit-parallelism phenomenon [4,6,7] and as such is expected to be faster on average than the Sankoff-Indels algorithm. The paper compares the performance of these algorithms for different probabilities of edit operations on the test search patterns. The assumed application scenario is SPAM filtering with tolerance on the total number of indels applied on SPAM-words limited in advance in order to maintain the intelligibility of the distorted words.

The structure of the paper is the following. Section 2 provides review of related work on the approximate search problem. Section 3 explains the two approximate search algorithms, Sankoff-Indels and CRBP-Indels. Details about the experiments and their results are given in Section 4 and Section 5 provides discussion on the experimental results. Section 6 concludes the paper.

2 Background and Related Work

Dynamic programming (also used in computing edit distance [5]) is the basic technique that has been widely exploited for solving the problem of approximate search without constraints [3, 6, 11]. Another way to solve the problem is by applying inherent bit-parallelism of computer words [4,6,7]. This section contains basic information related to constrained edit distance and bit-parallelism. Edit distance or Levenshtein distance [5] is a distance function that is computed by determining the minimum number of insertions, deletions and substitutions of symbols needed to transform one string into another.

To solve the problem of approximate search with constraints (which we shall do in the next section), we first need to solve the problem of computing the constrained edit distance. Sankoff [10] is considered to be the first to study the constrained string editing, where the total number of indels represented the constraint. To compute edit distance with such a constraint, Sankoff and Kruskal [11] presented an algorithm based on the classical dynamic programming. Their constrained edit distance computation formula is given below:

Initialization:
Let m and n be the lengths of the two given strings a and b respectively. Let K be the maximum permitted number of indels. Then

$$w_{00}^0 = 0, w_{i0}^0 = w_{0j}^0 = \infty \text{ for } i = 1,\ldots,m, j = 1,\ldots,n. \tag{1}$$

Recurrence:
Optimum alignment uses k indels and ends with:

$$w_{ij}^k = min \begin{cases} d(a_i,\phi) + w_{i-1,j}^{k-1}, & deletion \\ d(a_i,b_j) + w_{i-1,j-1}^{k}, & substitution \text{ (this may be a match if } w_{i-1,j-1}^k \text{ is 0)} \\ d(\phi,b_j) + w_{i,j-1}^{k-1}, & insertion \\ w_{i,j}^{k-1}, & \text{Optimum alignment uses} < k \text{ indels} \end{cases}$$

where, $i = 1\ldots m, j = 1\ldots n, 0 \leq k \leq K$

(2)

Here, a_i and b_j are the prefixes of strings a and b of lengths i and j respectively. The w_{ij}^k is the minimum length of any alignment in a set of alignments between a_i and b_j using exactly k indels, $0 \leq k \leq K$. The w_{ij}^k is ∞ if this set is empty [11]. $d(a_i, \phi)$, $d(a_i, b_j)$, and $d(\phi, b_j)$ are the elementary edit distances assigned to deletions, substitutions, and insertions, respectively.

The technique of exploiting so-called bit-parallelism for string matching was first introduced in exact search by Baeza-Yates [2]. The principal advantage of using this technique is its simplicity (use of bitwise logical operations, shifts, and additions only). The bit-parallelism combines the states of searches on simulated deterministic finite automata running in parallel into a computer word (e.g., 32-bits, 64-bits) and updates these states in a single operation, due to which it reduces the number of operations performed by an algorithm by a factor of t, where t is the number of bits in a computer word [7]. In the case of approximate search, such efficiency of the algorithms was achieved by applying the concept of bit-parallelism to simulate a specific Nondeterministic Finite Automaton (NFA) assigned to the search pattern (P). Then the search text is fed into this NFA as input, symbol by symbol. Wu and Manber [13] used bit-parallelism to simulate such an NFA in a row-wise fashion. They also developed a piece of software called Agrep [12] based on this algorithm. A detailed analysis of bit-parallelism techniques in search can be found in [4].

3 Sankoff-Indels search and CRBP-Indels search

We first define an approximate search algorithm based on the basic indels constrained edit distance calculation algorithm by Sankoff and Kruskal [11] and we call it Sankoff-Indels search. A small modification in the initialization phase has been performed in order to yield the search algorithm that permits maximum K indels while finding occurrences of the search pattern in the search string, following approximately the same lines as in [7] for the unconstrained search case. The modification of the initialization phase (equation (5)) is given below:

$$\text{indels level} = k \text{ , where } 0 \leq k \leq K \tag{3}$$

$$w_{00}^0 = 0, w_{i0}^k = \infty \text{ for } i = 1, \ldots, m. \tag{4}$$

$$w_{0j}^k = 0 \text{ for j} = 1, \ldots, n \tag{5}$$

Here, w indicates the distance or minimum length of alignment between two strings a and b, w_{00}^0 indicates the distance between strings a and b at position 0 with indels 0, and similarly, w_{ij}^k indicates the distances between two strings with k indels at the position i and j of strings a and b, respectively. If we place the search pattern in rows and the search string in columns, the distance of all the cells in the 0^{th} row and all the columns, for all k is considered to be 0 as stated in the equation (5). This indicates that the search can begin at any position of the search string, much like in [7].

Algorithm 1 solves the search problem with the constraint on the total number of indels by using a three-dimensional array, where all the cells in the row i and the column j contain distances $w_{ij}^0, \ldots, w_{ij}^K$ for each possible number of indels k (equation (3)).

The $p_1 p_2 \ldots p_m$ in Algorithm 1 is the search pattern of length m, $t_1 t_2 \ldots t_n$ is the search string of length n, and K is the maximum permitted number of indels. In the initialization phase, the distance value for all the cells in the 0^{th} row or 0^{th} column of the array $w[m, n, K]$ are assigned. The cell at $w[0, 0, 0]$ contains 0 distance, the cells in the 0^{th} column contain ∞, and the cells in the 0^{th} row contain 0 as the distance for all k (equation (3)). In practice,

Algorithm 1 Sankoff-Indels

1: **procedure** SANKOFF-INDELS($p = p_1 p_2 ... p_m$, $t = t_1 t_2 ... t_n$, K)
2: *Initialization*:
3: $w[0,0,0] \leftarrow 0$
4: **for** $i \in 1...m$ and **for** $k \in 0...K$ **do** $w[i,0,k] \leftarrow \infty$
5: **for** $j \in 1...n$ and **for** $k \in 0...K$ **do** $w[0,j,k] \leftarrow 0$
6: *Recurrence*:
7: **for** $i \in 1...m$, **for** $j \in 1...n$, and **for** $k \in 0...K$ **do**
8: **if** $k-1 < 0$ **then** $del \leftarrow ins \leftarrow opt \leftarrow \infty$
9: **else**
10: $del \leftarrow w[i-1,j,k-1]$
11: $ins \leftarrow w[i,j-1,k-1]$
12: $opt \leftarrow w[i,j,k-1]$
13: $subs \leftarrow w[i-1,j-1,k]$
14: **if** $p_{i-1} = t_{j-1}$ **then** $ds \leftarrow 0$ ▷ ds is the substitution distance
15: **else**
16: $ds \leftarrow 1$
17: $w[i,j,k] \leftarrow \text{Min}(1+del, ds+subs, 1+ins, opt)$
18: $w[m,n,k]$ is the distance at position m, n with k indels, $0 \leq k \leq K$

∞ is replaced with a very large number and the distance 0 for all the cells in the 0^{th} row indicates that the search can be started at any position in the search string.

In the recurrence phase, all the remaining cells of the array are filled in by using the recurrence formula (2) for constrained distance calculation. This is done $K+1$ times for every cell for all k and during this process one need to check if $k-1 < 0$ or not. If the value is smaller than 0 then the distance for insertion, deletion, and optimum alignment should be ∞ (step 8). Algorithm 1 considers error cost as 1 regardless of the edit operation and for each edit operation, it adds this cost to the distance. In the case of substitution operation, the cost remains 0 if the characters that are being compared are the same (a match). The value in a particular (i, j)-cell for all the possible numbers of indels is the minimum of the four values (insertion, deletion, substitution, or optimum alignment), see the equation (2). Moreover, the distance at a particular k at cell (i, j) can be obtained from the distance value from the array $w[i, j, k]$.

The next algorithm that we introduce here is the CRBP-Indels algorithm based on the row-wise bit-parallelism (RBP) algorithm by Wu and Manber [13] but with the constraint on the total number of indels introduced in their scheme. The operation of the CRBP-Indels algorithm is illustrated on an expample: the search pattern is "threat", permitting up to L = 2 errors (the corresponding NFA is shown in Fig. 1) and the search string is "trett". The operation of the NFA from Fig. 1 is shown in Fig. 2.

The automaton from Fig. 1 contains $L+1$ rows, labeled from 0 to L, where L is the error threshold (indels and substitutions). Every row denotes the number of errors seen i.e., the first row represents the exact search (0 errors) and the second and the third rows represent approximate search with 1 and 2 errors, respectively. The automaton is considered as a two-dimensional array and all the nodes/states are labeled with the corresponding row and column numbers (S_{ij}) along with the maximum number of allowed indels (e.g., C_{00}, C_{01} etc.). The state S_{00} is the initial state. The self-loop at S_{00} allows a match to start anywhere in the search string. Every column denotes a matching prefix of the pattern, and the rightmost states (double-circled) at each row are the final states that represent a match of the whole pattern when it is active (a state becomes active when it can be reached after a prefix match, see for example [7]). The arrows represent the transitions

from one state into another and in Fig. 1, horizontal, vertical, dashed-diagonal, and solid-diagonal arrows represent a match, insertion, deletion, and substitution transition, respectively.

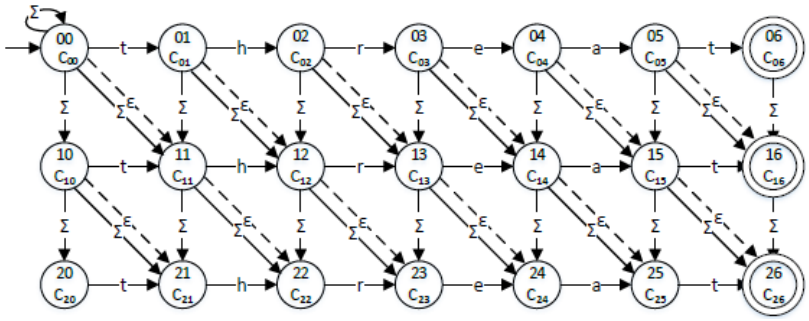


Fig. 1: The NFA that searches for the pattern "threat", permitting up to 2 character insertions, deletions, or substitutions

The CRBP-Indels algorithm uses the automaton from Fig. 1 in such a way that the state S_{ij} stores the maximum number of allowed indels C_{ij}. The C_{ij} for each state is checked to determine if a vertical and dashed-diagonal transition can be initiated from that state or not. If there is a possibility of an indel operation from the state S_{ij} and the C_{ij} is greater than 0 in that state, then indels operation can be carried out. In the case of a successful indels transition, the C_{ij} is decreased by 1 in the targeted state. This process is continued until the end of the input stream to the automaton. We say there is an occurrence if any final state of the automaton is reached by following this algorithm.

Algorithm 2 shows the implementation of the concepts introduced above. In the algorithm presented by Navarro and Raffinot [7] for row-wise bit-parallelism [13], each state is represented by a bit (0 or 1) and Shift as well as the logical operations (AND and OR) are performed over the combination of bits in each row based on the update formula. The update formula to obtain the new rows $R_0^{'}$ and $R_i^{'}$ at position j of the search string from the current R_0 and R_i values, is given below [4, 7, 13]:

$$R_0^{'} \leftarrow ((R_0 << 1)|0^{m-1}1)\&B[t_j] \tag{6}$$

$$R_i^{'} \leftarrow ((R_i << 1)\&B[t_j])|R_{i-1}|(R_{i-1} << 1)|(R_{i-1}^{'} << 1)|1 \tag{7}$$

The update formula is followed for all the input characters of the search string and we say there is a match when the last bit of any row is active. Note that the algorithm skips the bits of the 0^{th} column. A similar process is followed in CRBP-Indels algorithm but it also introduces a buffer to store C_{ij} for each active bit of $L+1$ rows.

In Algorithm 2, the parameter P is the search pattern of length m, T is the search string of length n, L is the number of rows in the automaton, and K is the maximum number of insertions/deletions allowed to find the occurrences of the search pattern in the search text.

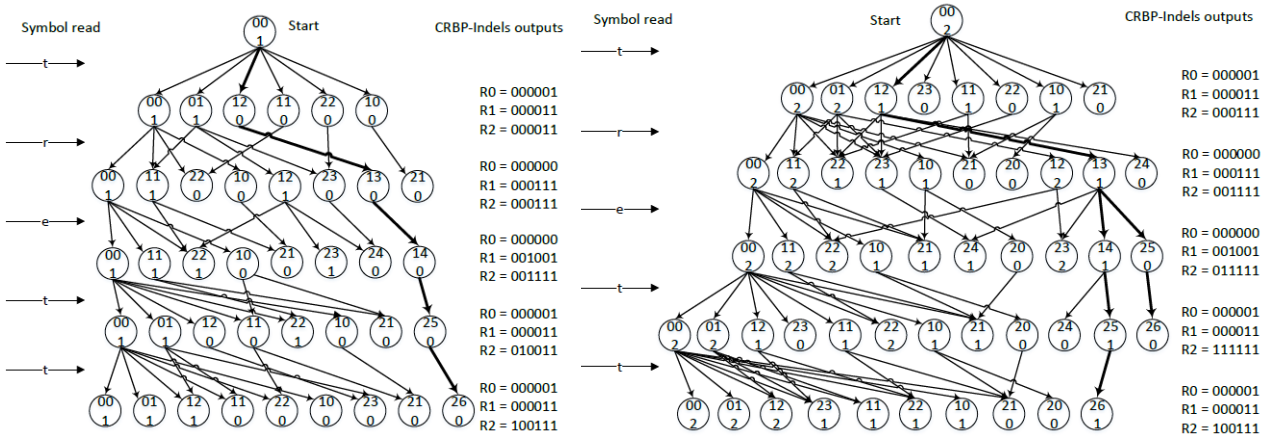
The preprocessing (step 2) computes the bit mask $b_m...b_1$ for each character of the search pattern P. The bit mask has the j^{th} bit set to 1 if p_j is the same as the character of the search pattern for which the bit mask is applied. This bit mask computation process is similar to the bit mask process of RBP [7, 13].

Algorithm 2 CRBP-Indels

1: **procedure** CRBP-INDELS($P = p_1p_2...p_m$, $T = t_1t_2...t_n$, L, K)
2: *Preprocessing*: bit mask $B[c] \leftarrow b_m...b_1$ for all the characters c in p : j^{th} bit set to 1 if $p_j = c$, $B[*] = 0$
3: *Initialization*:
4: **for** i ∈ 0...L **do**
5: R_i : set the i starting bits to 1
6: buffer $RBuff_i$: for all the active bits of R_i, set the buffer value as $K - i$
7: *Input symbol read*:
8: **for** pos ∈ 1...n **do**
9: $R_0^{'} \leftarrow ((R_0 << 1)|0^{m-1}1)\&B[t_{pos}]$
10: $oldRBuff \leftarrow RBuff_0$
11: $newRBuff$: for all active bits of $R_0^{'}$, set the buffer value as K
12: **for** i ∈ 1...L **do**
13: $R_i^{'} \leftarrow ((R_i << 1)\&B[t_{pos}])|R_{i-1}|(R_{i-1}^{'} << 1)|(R_{i-1} << 1)|1$
14: $newR = 0^m$ ▷ holds the updated bits
15: **for** all active bits of $R_i^{'}$ at position $j : j > 1$ **do**
16: **if** horizontal transition **then**
17: $nRRBuff_j \leftarrow RBuff_{i,j-1}$ ▷ $RBuff_{i,j-1} = C_{i,j-1}$ of t_{pos-1}
18: $nRRBuff_0 \leftarrow K - i + 1$, $newR_j \leftarrow$ '1'
19: **if** vertical transition and $oldRBuff_j > 0$ **then** ▷ $oldRBuff_j = C_{i-1,j}$ of t_{pos-1}
20: **if** $nRRBuff_j < oldRBuff_j - 1$ **then** $nRRBuff_j \leftarrow oldRBuff_j - 1$
21: $nRRBuff_0 \leftarrow K - i$, $newR_j \leftarrow$ '1'
22: **if** dashed-diagonal transition and $newRBuff_{j-1} > 0$ **then** ▷ $newRBuff_{j-1} = C_{i-1,j-1}$ of t_{pos}
23: **if** $nRRBuff_j < newRBuff_{j-1} - 1$ **then** $nRRBuff_j \leftarrow newRBuff_{j-1} - 1$
24: $nRRBuff_0 \leftarrow K - i$, $newR_j \leftarrow$ '1'
25: **if** solid-diagonal transition **then**
26: **if** $nRRBuff_j < oldRBuff_{j-1}$ **then** $nRRBuff_j \leftarrow oldRBuff_{j-1}$
27: $newR_j \leftarrow$ '1'
28: **if** active bit of $R_i^{'}$ at position 1 **then**
29: **if** vertical transaction and $oldRBuff_j > 0$ **then**
30: **if** $nRRBuff_0 < oldRBuff_j - 1$ **then** $nRRBuff_0 \leftarrow oldRBuff_j$ - 1
31: $newR_j \leftarrow$ '1'
32: $newRBuff \leftarrow nRRBuff$ ▷ $newRBuff$ holds C_{ij} of $newR$
33: $oldR \leftarrow R_i$, $R_i \leftarrow newR$
34: $oldRBuff \leftarrow RBuff_i$, $RBuff_i \leftarrow newRBuff$
35: **if** $newR$ & $10^{m-1} \neq 0^m$ **then** report an occurrence at pos

In the initialization phase (steps 3 to 6), the 0^{th} row contains all inactive bits and the rows greater than 0 contain starting active bits equal to the number of the row. For example, if P = "threat", the row 1 will have 000001 bits with $C_{ij} = 1$ for the active bit, when the maximum allowed number of indels is 2. Since the 0^{th} row does not contain any error, the buffer value of active state at 0^{th} row should be the maximum possible number of indels K and the value decreases by 1 for the active states of the immediately lower row.

The update formula is applied for each row for all the input symbols one after another. All 1s in $R_0^{'}$ should contain the total number of allowed indels (steps 9 to 11). When the update function ($R_i^{'}$, see equation (7)) is applied for the rows greater than 0, all the possible transitions to these active states should be checked and the bits for these active states should be updated based on the performed transitions and availability of the C_{ij} of the states that are responsible to initiate the transitions (steps 12 to 34). In order to check if there was a horizontal transition or not to get the active state S_{ij}, the algorithm checks if S_{ij} is still active after the horizontal transition only. The state is kept active in the final



(a) With maximum indels 1 (b) With indels 2

Fig. 2: Computation of NFA 1 for the search string "trett"

result if it is active, otherwise it is converted to the inactive state. The C_{ij} from where the horizontal transition was initiated is then copied to the buffer of the state S_{ij}. The process is similar to the check if there was a substitution transition. The algorithm checks the C_{ij} value of the state from where indels transition can be initiated for S_{ij}, in order to determine if there was any indels transition to get S_{ij} as an active state. In this case, the C_{ij} of $S_{i-1,j}$ or $S_{i-1,j-1}$ should be greater than 0. After successful indels transition, the C_{ij} of the S_{ij} should be less than 1 from the C_{ij} of the $S_{i-1,j}$ or $S_{i-1,j-1}$. We do not need to check all these processes for the 1^{st} bit of the row after applying the update formula. The reason is that the bit at the position 1 is always active since the update formula for $R_i^{'}$ uses OR-ing with 1. The buffer value of the 1^{st} bit in an individual transition except a horizontal one becomes (K - number of rows), whereas it is (K - number of rows + 1) for a horizontal transition. If the active status of the 1^{st} bit can be because of an insertion transition then its buffer value should be updated by following the rule of an insertion transition. In case of multiple transitions to a single state, the maximum number of C_{ij} among all the possible transitions is stored in that state. After following all these updates, a match occurs if the last bit of any row is active (step 35).

Fig. 2a shows the operation of the automaton from Fig. 1 with maximum $K = 1$ indels and 1 substitution error, whereas Fig. 2b shows the operation with $K = 2$ indels only. In both figures 2a and 2b, the left part with an arrow shows the input symbol to the automaton, the middle part is the operation of the automaton, and the right-most part is the result of the CRBP-Indels algorithm for that input symbol. Each state S_{ij} stores the maximum number of allowed indels C_{ij}. For example, S_{00} at the top contains the maximum number of allowed indels = 1 in Fig. 2a.

In Fig. 2a, S_{00} is the initial state and when the input symbol "t"is read, there are several possibilities for the transitions: a self loop to the state 00, a horizontal transition to the state 01, a deletion transition to the state 11, and an insertion transition to the state 10. Nondeterministically, the machine splits in four and follows different choices in parallel. Here, the deletion transition exits for one state and the machine splits again to handle the deletion transition separately. Since we reduce the number of indels by 1 for each indels transition, the state from where indels transition is initiated should have indels value greater than 0 in order to have successful indels transitions. The state becomes inactive when there are no outgoing transitions. After reading the first symbol, the active states are 00, 01, 12, 11, 22, and 10. These states are also active in the output of the

CRBP-Indels (Algorithm 2) for the first input symbol "t".

Similar procedures are followed for the rest of the input symbols based on the availability of the C_{ij} in all the active states and the possible transition functions from those states. Note that the same state can be reached by following various transitions. In that case, the state stores the maximum number of indels among all the possible transitions to that state. We say there is an occurrence of the search pattern when the final state of the automaton in any row is active. For example, S_{26} is a final state and it is active with $C_{26} = 0$ when input symbol "t"(the last character of the search string) is read in Fig. 2a. Therefore, there is an occurrence of the search pattern at that symbol. The buffer value at each state indicates the number of allowed indels left at that state.

There are two occurrences in Fig. 2b: one at the 4^{th} position of the search string and another at the 5^{th} position. The occurrences in both positions are due to the final state S_{26} but in both cases the buffer values are different. We say, there is an occurrence of a search pattern with 2 indels at the search string position 4, since C_{ij} of S_{26} is 0 at that position. Similarly, we say, there is an occurrence of the search pattern with 1 indel and 1 substitution at the position 5 of the search string, since the C_{ij} of S_{26} is 1 at that position.

4 Experimental work

In our experiment, we used a text taken from a paper about harassment and threatening E-mails [1]. In that paper, there is a part containing words related to harassment and threats of the size 12,430 characters without spaces. The same paper also contains parts that are not related to these E-mails. The materials from the paper [1] can be condidered as a SPAM scenario and if we want to use a SPAM filter, such a filter should be capable of detecting keywords related to harassment and threat.

To illustrate a SPAM-filter scenario with the constraints on the total number of indels, we randomly selected some keywords from the text and modified them by adding insertion (i), deletion (e), and substitution (s) errors. We performed this operation four times by varying the probability of insertions, deletions, and substitutions in the selected keywords. Therefore, we had four groups of patterns with the following probabilities of errors (p_i=10%, p_e=10%, p_s=5%), (p_i=5%, p_e=5%, p_s=5%), (p_i=5%, p_e=5%, p_s=10%) and (p_i=5%, p_e=5%, p_s=20%). We also added few patterns in all the pattern groups that are not available in the text. The keywords selected from the text are the following: threateningemail, phonenumber, staffmembers, frightening, moreevidence, harassingemail, involve, theauthoritiesneedtoexamine, complexemailtracingtechniques, annoyingemails, and spammersandharassingemailers. The four groups of patterns with different probabilities of constraints on insertions, deletions, and substitutions are listed in Table 1. These pattern groups are chosen in such a way that the first two groups contain a number of indels greater than the number of substitutions and the last two groups contain a number of indels equal or smaller than the number of substitutions.

The experiment includes the implementation of the two approximate search algorithms, CRBP-Indels and Sankoff-Indels, introduced in Section 3. Search patterns are the inputs to these algorithms and the output is their running time to process the search patterns in order to find their (distorted) occurrences in the search text. The running times of these algorithms for different groups of patterns are taken by following the steps of Algorithm 3.

Patterns with (p_i=10%,p_e=10%,p_s=5%)	Patterns with (p_i=5%,p_e=5%,p_s=5%)
tkhreatninkgemik	t!hreateningema!
pkhonenumbk	p!honenumb!
sktaffmembek	s!taffmembe!
stringstringstringstring	stringstringstringstring
fkrightenik	f!righteni!
mkoreevidenk	m!oreeviden!
hkarassingemak	h!arassingema!
moneykjalksdfjlkasdfkljasdflkadsflksdafld	moneykjalksdfjlkasdfkljasdflkadsflksdafld
iknvole	involve
tkheauthritkiesnedtoekxaink	t!heauthoritiesneedtoexami!
police	police
ckomplexmaikltraingtekchiquek	c!omplexemailtracingtechniqu!
aknnoyingemaik	a!nnoyingemai!
creditcardnumber	creditcardnumber
skpammerandkharasingekmalerk	s!pammersandharassingemaile!
Patterns with (p_i=5%,p_e=5%,p_s=10%)	Patterns with (p_i=5%,p_e=5%,p_s=20%)
tShreate#ingemaS	trhrertenirgemal
pShonenumbS	prhonrnumbr
sStaffmembeS	srtaffrember
stringstringstringstring	stringstringstringstring
fSrighteniS	frrigrtenig
mSoreevidenS	mroreeridenr
hSarassingemaS	hrarrssirgemrl
moneykjalksdfjlkasdfkljasdflkadsflksdafld	moneykjalksdfjlkasdfkljasdflkadsflksdafld
involvS	involvr
tSheauthoSitiesneeStoexamiS	trhearthorrtiesreedtrexamre
police	police
cSomplexeSailtraciSgtechniqSs	cromrlexrmairtraringrechriqus
aSnnoyingemaiS	arnnryinremars
creditcardnumber	creditcardnumber
sSpammersSndharassSngemailes	srparmerrandrararsinremarles

Table 1: List of four groups of patterns with varying i, e, and s

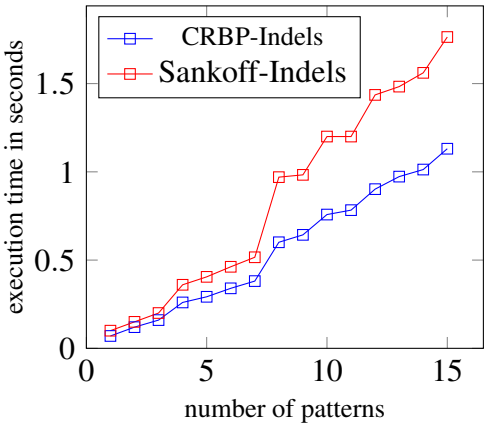
Algorithm 3 Performing the experiment

1: execute the algorithms, CRBP-Indels and Sankoff-Indels with only one pattern from a group
2: iterate the process 11 times
3: take the median running time output from each algorithm
4: add another pattern from the same group
5: repeat steps 2 to 4 until the last pattern of that group is evaluated
6: repeat steps 1 to 5 until all the patterns of all the groups are evaluated

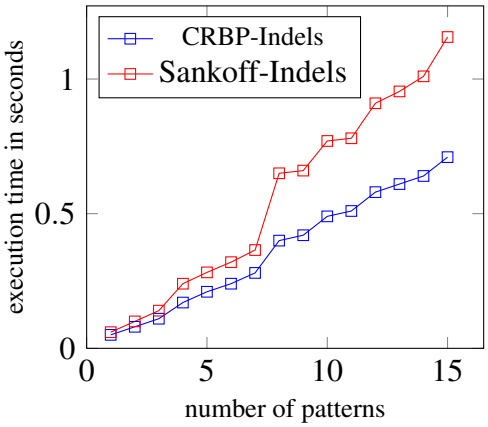
Figures 3 and 4 show the execution times of these two algorithms (in seconds) for each group of patterns. The horizontal axis indicates the number of patterns from a group and the vertical axis indicates the running time in seconds for the patterns.

The two sub-figures of Fig. 3 show the results of using CRBP-Indels and Sankoff-Indels algorithms for the first two pattern groups with the number of indels greater than the number of substitution errors. In both cases, CRBP-Indels has performed better than Sankoff-Indels, but there is a small difference in speed if the patterns are short. However, we can see a great difference in performance with longer patterns and the patterns that are not available in the text. For example, the 8th pattern (moneykjalksdfjlkasdfkljasdflkadsflksdafld) is random and is not available in the text. We can see that at this point, in both subfigures 3a and 3b, there is a great difference

between the performance of CRBP-Indels and Sankoff-Indels. CRBP-Indels algorithm has achieved better performance than the Sankoff-Indels algorithm on the pattern 8. This result shows that CRBP-Indels algorithm performs well when the number of indels is greater than the number of substitution errors, with long patterns, and with patterns that are not available in the text.

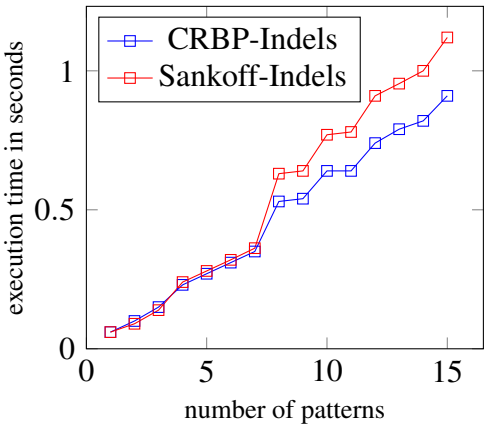


(a) p_i=10%, p_e=10%, p_s=5%

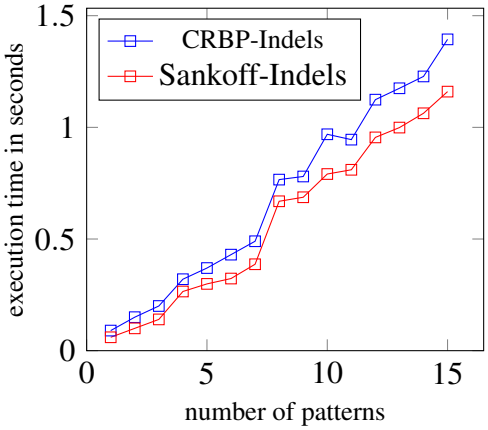


(b) p_i=5%, p_e=5%, p_s=5%

Fig. 3: Performance of CRBP-Indels and Sankoff-Indels when there are less substitutions than indels.



(a) p_i=5%, p_e=5%, p_s=10%



(b) p_i=5%, p_e=5%, p_s=20%

Fig. 4: Performance of CRBP-Indels and Sankoff-Indels when there are more substitutions than indels.

The two sub-figures of Fig. 4 show different results than the results of Fig. 3. They show the search results of the last two groups of patterns by using CRBP-Indels and Sankoff-Indels algorithms. In the Fig. 4a, the performance of both algorithms is similar until the pattern number 7. After that, CRBP-Indels outperforms the Sankoff-Indels algorithm. In the Fig. 4b, the Sankoff-Indels algorithm performs better with all the patterns. This shows that the Sankoff-Indels performs better for very small number of indels compared to the number of substitution errors.

5 Discussion

Based on the experiments and the results given in Section 4, we can see the correlation between the performance of the algorithms (CRBP-Indels and Sankoff-Indels) with the

choice of the number of indels and the number of substitutions, lengths of the patterns, and the patterns that are not available in the text.

From figures 3a and 3b, we can observe that the performance of CRBP-Indels is better than the performance of Sankoff-Indels when the search is performed with larger number of indels than the substitution errors. In contrast, figures 4a and 4b show that the performance of Sankoff-Indels is better than the CRBP-Indels when the number of indels is much smaller than the number of substitution errors. The reason for this can be that the algorithm by Sankoff and Kruskal [11] for sequence matching considers the number of indels only and the substitution errors are handled internally. The same is true in the case of CRBP-Indels but this algorithm also checks if there was a substitution error or not and copies the allowed numbers of indels to its buffer so that it can be used for the next row. Moreover, Sankoff-Indels is based on the algorithm which is designed specifically to compare the long sequences that contain very small numbers of indels compared to the numbers of substitutions, for example, the comparison of human and E. coli FS RNA [11]. Since the CRBP-Indels is based on the RBP algorithm [13], its performance may be affected by the number of rows or errors in total rather than the number of indels only.

The length of the search pattern also affects the performance of these algorithms. From figures 3a and 3b, we can see that the performance of CRBP-Indels is better for longer patterns (e.g., pattern number 12 and 15) than the Sankoff-Indels, whereas from figures 4a and 4b, we can see that the performance of Sankoff-Indels is similar or little better than CRBP-Indels for the same long patterns. This shows that keeping the number of indels smaller than the number of substitutions, the length of the pattern does not affect a lot the performance of CRBP-Indels but its performance can be worse for long patterns when the number of indels is very low compared to the number of substitutions. There is a very small difference in the performance of both algorithms for patterns with smaller length.

Matching the patterns that are not available in the text may also affect the performance of these algorithms. For example, in figures 3a, 3b, and 4a, we can see that the performance of CRBP-Indels for the pattern 8 is much better than the Sankoff-Indels but Fig. 4b shows that the performance of both algorithms is similar for the same pattern. This shows that the CRBP-Indels with a larger number of indels than the number of substitutions, can perform well in a data set where patterns are not available. It is useful when the pattern is not an attack vector and it has to be compared with all the attack signatures in a database (of SPAM-words).

6 Conclusion

The main objective of this paper was to identify a method that exploits bit-parallelism for solving the new problem of approximate search with constraints on indels and compare the results with a solution based on the classical dynamic programming approach. We introduced CRBP-Indels (an algorithm based on row-wise bit-parallelism but with constraints on indels that we introduced) and Sankoff-Indels (an algorithm based on the classical dynamic programming solution to solve the approximate sequence-matching problem (again with constraints on indels)). The performances of these two algorithms were compared and the results show that the performance of CRBP-Indels is better than the performance of Sankoff-Indels when the number of indels is greater than the number of substitutions. In such a case, the CRBP-Indels performs well even for longer patterns and for patterns that are not available in the search string. However, the performance

of CRBP-Indels gradually decreases with the increase in the number of substitutions compared to the number of indels.

CRBP-Indels can be used in cases where there is a higher probability of using indels than the substitutions. SPAM-filtering is one of the possible application scenarios. Another application could be in file carving in digital forensics. The concept of CRBP-Indels can also be used to support other types of constraints related to limiting the allowed number of individual edit operations, and lengths of runs of edit operations.

References

[1] http://www.easyemailsearch.com/dealing-with-email-harassment.html. [Accessed: July 2015].

[2] Ricardo A. Baeza-yates. *Efficient Text Searching*. PhD thesis, Dept. of Computer Science, University of Waterloo, May 1989. Research Report CS-89-17.

[3] Carl Barton, CostasS. Iliopoulos, and SolonP. Pissis. Average-case optimal approximate circular string matching. In Adrian-Horia Dediu, Enrico Formenti, Carlos Marín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications*, volume 8977, pages 85–96. Springer International Publishing, 2015.

[4] Simone Faro and Thierry Lecroq. Twenty years of bit-parallelism in string matching. In J. Holub, B. W. Watson, and J. Žďárek, editors, *Festschrift for Bořivoj Melichar*, pages 72–101. 2012.

[5] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics-Doklady*, 10(8):707–710, 1966. Translated from Doklady Akademii Nauk SSSR, 163, 4, 845–848, 1965.

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

[7] Gonzalo Navarro and Mathieu Raffinot. *Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, New York, NY, USA, 2002.

[8] B.J. Oommen. Constrained string editing. *Information Sciences*, 40(3):267 – 284, 1986.

[9] Slobodan V. Petrović and Jovan DJ. Golić. String editing under a combination of constraints. *Information Sciences*, 74(1 - 2):151 – 163, 1993.

[10] David Sankoff. Matching sequences under deletion/insertion constraints. *Proceedings of the National Academy of Sciences*, 69(1):4–6, January 1972.

[11] David Sankoff and Joseph B. Kruskal. *Time warps, string edits and macromolecules : the theory and practice of sequence comparison*. Addison Wesley, Reading, MA, 1983.

[12] Sun Wu and Udi Manber. Agrep - a fast approximate pattern-matching tool. In *In Proc. of USENIX Technical Conference*, pages 153–162, 1992.

[13] Sun Wu and Udi Manber. Fast text searching allowing errors. *Commun. ACM*, 35(10):83–91, October 1992.