On Application of Constrained Edit Distance Algorithms to Cryptanalysis and Digital Forensics

Kyle Porter and Slobodan Petrović

Faculty of Information Technology and Electrical Engineering, Norwegian University of Science and Technology Gjøvik
kyleap@stud.ntnu.no, slobodan.petrovic@ntnu.no

Abstract

This paper examines constrained edit distance algorithms and their applications to cryptanalysis and digital forensics. The constrained edit distance is an extension of the edit distance where the user may place constraints on the ways some string X is transformed into some string Y when calculating the minimum distance between strings, thereby potentially producing a different distance. If the user has a priori knowledge of which edit operations are probable or possible, then the constrained edit distance will report more accurate results than the unconstrained edit distance. We look at how constrained edit distance algorithms are implemented, how the distance can be integrated into approximate string matching algorithms, and how constraints are implemented into distance and search algorithms. Furthermore, we examine algorithms of this type which can be applied to cryptanalysis and digital forensics. Lastly, we make a small observation regarding how the Nondeterministic Finite Automaton for approximate string matching relates to Oommen's constrained edit distance formalisms, and show how we may apply basic constraints to this automaton.

1 Introduction

Identifying the similarity between strings and approximate matching of similar strings or sequences is an important problem in computer science, and this applies to information security as well. Such problems are found in cryptanalysis where the distance between unequal binary sequences can be calculated [7][18], digital forensics for approximate keyword search [10], and intrusion detection for approximately matching attack signatures [2]. The computational complexity and accuracy of algorithms of these varieties is dependent on the distance used to quantify similarity. A commonly used distance is the Levenshtein distance [9], frequently referred to as the edit distance, where it is defined as the minimum number of elementary edit operations required to transform one sequence or string X into some string Y where the elementary edit operations are the insertion of a character into X, the deletion of a character from X, and the substitution of

Presented at the Norwegian Information Security Conference 2017 (NISK 2017).

a character in X for a character in Y. While the edit distance is an effective measure of similarity in the stringological sense, it oftentimes produces far too many false positives in practical settings to the point where the application of these algorithms are simply ineffective. An extension of the edit distance, the constrained edit distance [12], combats the problem of excessive false positives by taking into account a priori knowledge about the probability or impossibility of the number and type of edit operations which are performed in the application being examined. This distance allows the user to specify arbitrary constraints regarding the number and type of edit operations that may be performed to transform some string X into Y, thereby producing a different distance than the unconstrained edit distance. In some cases where the unconstrained edit distance between strings is small, the constrained edit distance between strings can be increased or even become infinite if the transformation is impossible given some constraints.

In this paper, we study a selection of constrained edit distance algorithms which have been applied to cryptanalytic and digital forensic contexts. The algorithms chosen intend to showcase the full functionality and possibilities of the state of the art. We provide background information for the theoretical bases for the algorithms presented, an explanation of the algorithms themselves and methods to implement constraints, and examples of their applications. Lastly, we provide a novel observation on the relationship between the constrained edit distance theory established by Oommen in [12] and modern approximate string matching algorithms.

2 Background Theory and Algorithmic Methods

Levenshtein distance and Constrained Edit Distance

We begin by formally defining strings and aspects of determining similarity. We synonymously refer to a pattern, string, or sequence as defined by some arbitrary $X = x_1x_2\ldots x_N$ where finite integer N is the length of X, and x_i for $1 \leq i \leq N$ are characters from some finite alphabet Σ. Furthermore, let X_i represent the prefix composed of the first i characters of X. Let $D(X,Y)$ correspond to the general Levenshtein distance, or edit distance, between strings X and Y where it is defined as the minimum number of elementary edit operations required to transform X into Y [9]. For many approximate matching algorithms, search string X matches some contiguous substring Y in the searched text Z if $D(X,Y)$ is less than or equal to some threshold k.

The following formalisms in this subsection were first defined by Oommen [12]. To express the edit operations formally, we need to introduce the null symbol ϕ which is used for representing the deletion and insertion operations. Let $\widetilde{\Sigma} = \Sigma \cup \phi$ and $a,b \in \widetilde{\Sigma}$ where the cost function $d(a,b)$ is mapped to a positive real number and defines the cost of the following elementary edit operations:

1. $d(x_i,\phi)$ is the cost of deleting character x_i from string X.
2. $d(\phi,y_j)$ is the cost of inserting character y_j into string X.
3. $d(x_i,y_j)$ is the cost of substituting x_i for y_j in string X.

For the computation of the edit distance, we shall assume that $d(x_i,y_j) = 1$ for all $x_i,y_j \in \widetilde{\Sigma}$ except in the trivial case when $x_i = y_j$ for which $d(x_i,y_j) = 0$. We shall call such a substitution, where $x_i = y_j$, a trivial edit operation. We shall also assume that the substitution of the null symbol by the null symbol is not permitted. The purpose of altering the cost is having the ability to weigh certain operations or characters more heavily based on a priori statistical knowledge of possible edit operations, and in this fashion two transformations between X and Y requiring the same number of edit operations do not necessarily produce the same distances.

Using the provided notation we have the tools to effectively express all possible transformations from string X to Y. We illustrate such a transformation through an example. Let $X =$ "secure" and $Y =$ "scared", and let X' and Y' be transformation pair strings over $\widetilde{\Sigma}$. The pair X' and Y' describe the elementary edit operations to transform X into Y using ϕ to represent insertion and deletion operations. One possible transformation may be represented in the following way:

$$X' = \text{secure}\phi$$
$$Y' = \text{s}\phi\text{cared}$$

In this example, the character "e" is deleted from secure, the character "u" is substituted with an "a", the character "d" is inserted into the word secure, and all other substitutions do not change the characters. The sum of edit costs can quickly be calculated via $\Sigma_{j=1}^{|X'|} d(x'_j, y'_j)$. This is just one of many possible transformations from X to Y.

The constrained edit distance is defined as the minimum sum of edit operations to transform string X into Y such that the transformations obey the given constraints, where the constraints may be arbitrarily complex as long as they are defined in terms of the number and type of edit operations [12]. For example, one may place constraints on the maximum number of allowed insertions i, deletions e, or substitutions s. Other types of constraints can limit the maximum number of times an edit operation may be performed consecutively, or limit the type or combination of edit operations performed. We examine some properties of transforming strings to recognize the functionality of applying constraints. According to Oommen [12][13], let $\Gamma(X,Y)$ be the set of all possible transformation pairs (X',Y') for transforming string X into Y via elementary edit operations. The cardinality of $\Gamma(X,Y)$ is given by the following expression:

$$|\Gamma(X,Y)| = \sum_{m=max(0,|Y|-|X|)}^{|Y|} \frac{(|X|+m)!}{m!(|Y|-m)!(|X|-|Y|+m)!} \tag{1}$$

By adding constraints, we eliminate some elements of $\Gamma(X,Y)$ since there are limitations on possible transformation pairs (X',Y'). Let the constrained set of transformations of $\Gamma(X,Y)$ be denoted as $\Gamma_T(X,Y) \subset \Gamma(X,Y)$. The set of elements $\alpha \subset \Gamma(X,Y)$ which have the minimum sum of edit costs are transformations that correspond to the edit distance between X and Y. The set of elements $\beta \subset \Gamma_T(X,Y)$ which have the minimum sum of edit costs for all possible transformations under constraints T correspond to the constrained edit distance between X and Y. Since $\Gamma_T(X,Y) \subset \Gamma(X,Y)$, β potentially contains fewer elements than the unconstrained case α if the sums of edit costs are equal (fewer possible transformations), or α and β may now correspond to different edit cost summations.

When setting constraints appropriately, we can eliminate transformations on X which we know are unlikely or impossible to occur within an application. This also changes which strings Y are considered to be similar under the unconstrained edit distance. For example, let the constraints be a maximum of one deletion and one insertion when calculating the edit distance between strings X and Y. $\Gamma_T(X,Y)$ is empty for any strings Y such that $|Y| < |X| - 1$ or $|Y| > |X| + 1$. This quality is useful for search, for if we have a priori knowledge of which types of errors may occur on some string X, then we can apply constraints that align to these probabilities, which then in turn can lead to some constrained edit distances between strings that require unlikely transformations to become increased or even infinite.

Constrained Edit Distance and Dynamic Programming

Common methods of implementing the constrained edit distance are either through dynamic programming or automata theoretic methods. Dynamic programming methods have been shown to be considerably more flexible than finite automata implementations, but the methods utilizing automata can conduct simpler fast approximate string matching. Calculating the constrained edit distance utilizing dynamic programming is based on the methods for calculating the unconstrained edit distance, first described by Wagner and Fisher in 1974 [19].

Oommen developed a dynamic programming method for calculating the constrained edit distance with constraints defined in the number and type of edit operations that may be performed to transform some string X into string Y [12]. The essence of this algorithm is dependent on the recursive function $W(i,e,s)$ for the constrained edit distance associated with editing prefixes X_{e+s} to Y_{i+s} subjected to the constraint that exactly i insertions, e deletions, and s substitutions are performed.

Theorem 1 *[12] For any two strings X and Y, let $W(i,e,s)$ be defined by:*

$$\begin{aligned} W(i,e,s) = min[&\{W(i-1,e,s)+d(\phi,y_{i+s})\}\\ &\{W(i,e-1,s)+d(x_{e+s},\phi)\}\\ &\{W(i,e,s-1)+d(x_{e+s},y_{i+s})\}] \end{aligned}$$

For all feasible triples (i,e,s), and $W(0,0,0)=0$.

Using the inherent constraints of the relationship between possible insertions, deletions, and substitutions all feasible values of i,e,s are used to fill out the entire $W(\cdot,\cdot,\cdot)$ array. An example of an inherent constraint is if string X is being transformed into an equal length string Y and one insertion operation must take place; then one deletion operation must take place as well. Theorem 1 describes how to calculate the values of $W(i,e,s)$, and in Theorem 2 we see the relationship between the constrained edit distance $D_T(X,Y)$ and the $W(\cdot,\cdot,\cdot)$ array for some constraints T expressed in terms of the number of insertions i.

Theorem 2 *[12] The quantity $D_T(X,Y)$ is related to the elements of the array $W(i,e,s)$ as follows:*

$$D_T(X,Y) = min_{i\in T} W(i,N-M+i,M-i) \tag{2}$$

Essentially, from the possible i,e, and s constraints the minimum number of necessary non-trivial edit operations is returned. We give an example here using the words "secure" and "scared". After inputting these strings into an algorithm for computing $W(\cdot,\cdot,\cdot)$, some transformations and their corresponding edit costs are given in Table 1.

Table 1: Transformations between “secure” and scared” given various constraints

Constraints (i,e,s)	**Transformation**	**Sum of Edit Operation Costs**
(0,0,6)	secure scared	5
(1,1,5)	secureϕ sϕcared	3
(2,2,4)	secuϕreϕ sϕcϕared	4
(3,3,3)	secuϕre$\phi\phi$ sϕcϕarϕed	6
⋮	⋮	⋮

If our constraint was that the transformation must contain more than one insertion, then the constrained edit distance between “secure” and “scared” would be 4. It is important to note that the number of substitutions is for general substitutions, but the cost is only applied if the letters being substituted differ.

Applying complex constraints requires modification of the recursive function $W(i,e,s)$, as we shall see later in the paper. However, if the constraints are relatively simple, such as requiring a precise number of specific edit operations, we simply do not consider values $W(i,e,s)$ which do not fit the constraints. Calculating $W(\cdot,\cdot,\cdot)$ and a corresponding constrained edit distance $D_T(X,Y)$ have $O(|X||Y|min(|X|,|Y|))$ time complexity. The flexibility and practicality of the algorithm are given by the facts that the user can modify the cost function operations, apply more complex constraints, and can produce the optimal edit transformation by backtracking through $W(\cdot,\cdot,\cdot)$ in $O(max(|X|,|Y|))$ time [12].

Automata Theory and Approximate Matching

In order to discuss search algorithms utilizing the constrained edit distance, we first need to discuss bit-parallel implementations of the Nondeterministic Finite Automaton (NFA) for approximate string matching. We first go over some necessary components of formal language theory.

Any finite automaton A may be defined by its set of states Q, the set representing the alphabet of characters Σ, the set of initial states $I \subseteq Q$, the set of terminal states $F \subseteq Q$, and arrows from state to respective state representing the transition relationships between the states. Each arrow is associated with some set of characters from Σ representing which characters may trigger a transition from one state to the next. Upon beginning to read in some string of characters, the automaton is active in its initial states. For each character input from the string, the active states may transfer to other states if the input character is included in the set of characters associated with the transition. If the sequence of contiguous characters input from a string into an automaton allows the traversal from an initial state to a terminal state, then the string is considered to be a match.

Figure 1 shows a common automaton for approximate string matching. This automaton is nondeterministic, meaning that any number of the states in Q may be active simultaneously. Furthermore, such nondeterministic automata allow for ε-transitions, where transitions are made without needing a prerequisite character. The initial state is represented by the node with a bolded arrow pointing to it, and is always active as indicated by the self-loop. The terminal states are represented by the double-circled nodes, the horizontal transitions represent exact character matches, vertical transitions represent insertions, solid diagonal transitions represent substitutions, and dashed diagonal transitions represent deletions via ε-transitions.

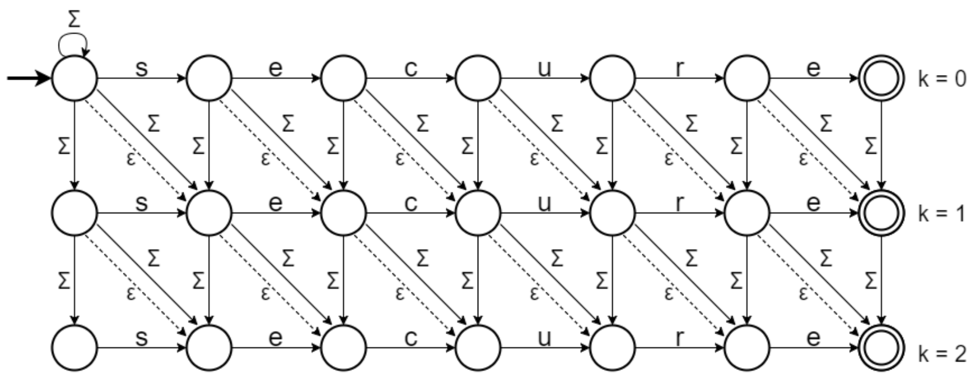

Figure 1: NFA for approximate string matching. Matches the pattern “secure” allowing two edit operations.

For an approximate search with an edit distance threshold k, this automaton has $k+1$ rows. The first row in the automaton represents a machine that performs exact matching of the search pattern against the searched input text Y, the second row is a machine that matches the search pattern in Y with one edit operation performed on the pattern, etc. The primary advantage of these theoretical machines is that they can determine potential errors in the pattern simultaneously, where for every input character each row checks for potential matches, insertions, deletions, and substitutions against each position of the pattern. Constraints can be applied to this type of automaton by modifying the transitions or counting potential edit operations, as we shall see in Section 3.

Due to the fact that these automata are nondeterministic, they must be simulated in practical applications. An efficient form of simulation is via bit-parallelism, wherein bit-vectors represent each row of the automaton and are updated by way of basic bitwise operations which correspond to the automaton’s transition relations. The bitwise operations update all the bits of a bit-vector at once, and therefore update the states of a row of an automaton simultaneously. This parallelism reduces the number of operations a search algorithm performs at most by w bits in the computer word (32 or 64 bits) containing the bit-vector if the length of the pattern is less than or equal to w [4].

Here we introduce some fundamentals of bit-parallelism based algorithms, see for example [11]. Each row of the automaton for pattern X is represented as a binary vector of length $|X|$, and we create a table of Boolean vectors $B[t_j]$ of the same size, called *bit-masks*, as representations for incoming characters t_j for comparison. These bit-masks represent the positions of a character within the search pattern. For example, Table 2 gives the bit-masks for the pattern “secure”, where any character not present in the pattern is represented by the * symbol.

Table 2: Bit-mask table for the word "secure"

Character t_j	**Bit-mask** $B[t_j]$
c	000100
e	100010
r	010000
s	000001
u	001000
*	000000

We examine the bit-parallel construction of the NFA from Figure 1 to understand the unconstrained matching case, as the constrained cases are extensions of this algorithm. Each row of the automaton is represented by a binary vector R_i, $0 \leq i \leq k$ where k is the total number of allowed errors (non-trivial edit operations), and the goal is to update each row R_i as defined by the automaton. A clarification needs to be made regarding the direction of the automaton in Figure 1 versus their simulated counterparts. By convention, automaton graphs typically traverse from left to right; however the vectors simulating these automata typically traverse from right to left, see [11]. The NFA from Figure 1 is simulated by Algorithm 1, where the initial state of each row is $R_i = 0^{|X|-i}1^i$:

Algorithm 1 [11] Bit-parallel simulation of the NFA for approximate string matching.

```
    $R'_0 \leftarrow ((R_0 << 1) \mid 0^{|X-1|}1)$ & $B[t_j]$
    $R'_i \leftarrow ((R_i << 1)$ & $B[t_j]) \mid R_{i-1} \mid (R_{i-1} << 1) \mid (R'_{i-1} << 1)$
```

The first line of this algorithm represents a character match with no errors taken place. The subsequent rows take in account for i errors, where each of the components is bitwise OR'd with different transition relationships. Explicitly from left to right, the first expression represents exact character matches, the second represents an insertion, the third a substitution, and the fourth a deletion. A match is accounted for if any of the rows have the $|X|$th bit from the left side equal to 1, the bits representing the terminal states. This algorithm was first implemented by Wu and Manber and runs in $O(k\lceil |X|/w \rceil |Y|)$ time [21], but more modern algorithms have made improvements to reduce the time complexity to $O(\lceil (|X|-k)(k+1)/w \rceil |Y|)$ (see for example [8]).

3 Applications of Constrained Edit Distance Algorithms

This section is split into subsections for applications of constrained edit distance algorithms implemented through dynamic programming and bit-parallel NFA simulation.

Dynamic Programming Applications

Perhaps the most extensively used application of the constrained edit distance has been in cryptanalytic generalized correlation attacks against stream ciphers with irregularly clocked linear feedback shift registers (LFSRs), where the algorithms being used are based on the $W(i,e,s)$ function from Theorem 1 with modifications. For a thorough review of correlation attacks see Siegenthaler [18]. Here we describe the essentials necessary to show how the constrained edit distance is applied to generalized correlation attacks. Correlation attacks are attacks on nonlinear combination stream ciphers where the

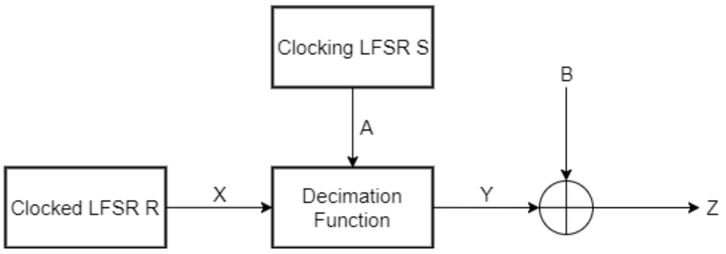

Figure 2: Clock-controlled stream cipher model from Golić and Mihaljević [7]

goal is to use the correlation between the LFSR R_j and the ciphertext to reduce the number of total guesses of the key significantly. The Hamming distance is used to quantify the similarity between the LFSR output and the ciphertext for standard correlation attacks, but when the sequences are of different lengths then the Hamming distance cannot be used. For stream ciphers employing irregularly clocked LFSRs, this is the case, and the edit distance is an appropriate distance.

Work by Golić and Mihaljević [7] introduced the concept of a generalized correlation attack, and described the procedure to attack a simple clock controlled stream cipher using the constrained edit distance. Figure 2 shows this model, where a clocked LFSR R produces a binary sequence X, and a clocking LFSR S produces sequence A which is used as input with X into a decimation function that outputs the sequence Y. The bits in sequence Y are given by the bit characters $y_n = x_{f(n)}$ where $f(n) = n + \Sigma_{j=1}^{n} a_j$ for $n = 0, 1, 2, \ldots$, which is just X with deletions. This decimated sequence Y is then XOR'd with noise B, such as plaintext, producing the ciphertext Z. In terms of string editing, X may be transformed into Z through a number of deletions and substitutions, where the maximum number of consecutive deletions is 1. Golić and Mihaljević generalized the constrained edit distance solution for applying constraints of a maximum of E consecutive deletions.

Conceptually, the constrained edit distance algorithm applied in this attack is computed by means of dynamic programming. However, only deletions and substitutions are permitted in the function $W(\cdot,\cdot)$ and the recursion only calculates edit distances for which the maximum number of consecutive deletions is E. The prefixes of the strings X and Z being examined are X_{e+s} and Z_s where $|X| = M$ and $|Z| = N$.

Theorem 3 *[7] The partial constrained edit distance $W(e,s)$ satisfies the recursion where d_0 is the cost of deletion*

$$\begin{aligned} W(e,s) = &min\{W(e-e_1, s-1) + e_1 d_0 + d(x_{e+s-e_1}, z_s) : \\ &\{0, e - min\{M-N, sE\}\} \leq e_1 \leq min\{e, E\}\} \end{aligned}$$

for $1 \leq s \leq N$, $0 \leq e \leq min\{M-N, (s+1)E\}$, and, for $s = 0$ and $0 \leq e \leq min\{M-N, E\}$, $W(e,0) = ed_0$.

The constraint of E is performed in the recursion by determining the minimum distance between prefixes considering up to E deletions in Z. Even more advanced constraints can be implemented in this scenario by further modifying the recursion function as shown by Petrović and Golić [16]. In addition to having a maximum consecutive number of deletions E or insertions I, they simultaneously applied constraints in which between consecutive runs of substitutions there can be at most one consecutive run of deletions and/or at most one consecutive run of insertions. Similar constructions were used to extend these results to arbitrary combining functions without memory [5], and then to alternating step generators [6].

These methods and algorithms have been applied to the field of digital forensic search as well. In [14], the constrained edit distance with constraints on maximum length consecutive runs of deletions and insertions has been used for a digital forensic search preselection phase, in which large fragments of the total search space have their constrained edit distance calculated between the fragment and a search pattern. When the distance returned was beneath a prespecified threshold, the fragment was considered to be worth investigating. By using the constrained edit distance rather than the unconstrained edit distance, a significant data reduction of data needing closer inspection was obtained. A similar data reduction approach was applied to a preselection phase of selecting relevant Snort signature rules given an input string utilizing only a constraint on the maximum consecutive runs of deletions [15].

NFA Simulation Applications

Constrained edit distance applications utilizing NFA simulation are typically used in approximate string matching scenarios. This distance was introduced into approximate search by Chitrakar and Petrović [1] and applied the algorithm for spam filtering with a priori knowledge of the probabilities of errors which may be introduced into spam keywords. The constraint in this case sets a maximum number of possible *indels*, defined as the total sum of insertions and deletions. The method of implementation uses the NFA for approximate matching and is an extension of Algorithm 1, where it associates an integer counter to every state that represents how many insertion or deletion transitions can still be performed on the substring being analyzed. If a state is traversed to via an insertion or deletion, then that state's counter is reduced by one, and when a state is reached for which the counter is 0 then the algorithm no longer permits insertion or deletion transitions from that state. A similar algorithm which sets constraints on the maximum number of allowed insertions, deletions, and substitutions using counters was used for approximate search in intrusion detection in which the probability of edit transformations on attack signatures was previously known [2]. Both methods saw reductions in false positive matches compared to the unconstrained cases under the specified conditions.

A question regarding constrained edit distance NFA algorithms is how we may apply arbitrary constraints, such as a maximum consecutive run of a particular type of edit operation, so that we may apply more flexible constraints to approximate search. Such developments could lead to faster generalized correlation attack cryptanalysis. We discuss this in the next section.

4 Discussion

Oommen's Formalisms Applied to Approximate Matching NFA

We would like to examine other ways we may implement constraints to the NFA for approximate matching. A novel observation we make in this paper is describing this NFA in terms of Oommen's string editing formalisms. The automaton allows for the set of (X', Y') transformation pairs for all strings X and Y over $\widetilde{\Sigma}$ such that the sum of edit operations is less than or equal to k. The transformation pairs X' and Y' can be obtained by traversing the automaton. Horizontal transitions are edits considered by (x_i, y_j) pairs where $x_i = y_j$, vertical transitions are considered by (ϕ, y_j) pairs, ε-transitions are considered by (x_i, ϕ) pairs, and diagonal transitions are considered by (x_i, y_j) pairs for $x_i \neq y_j$. We give an example for strings $X = secure$ and $Y = secare$. Figure 3 shows the possible transformations from X to Y for $k \leq 2$, which can be traced on the automaton in the following ways.

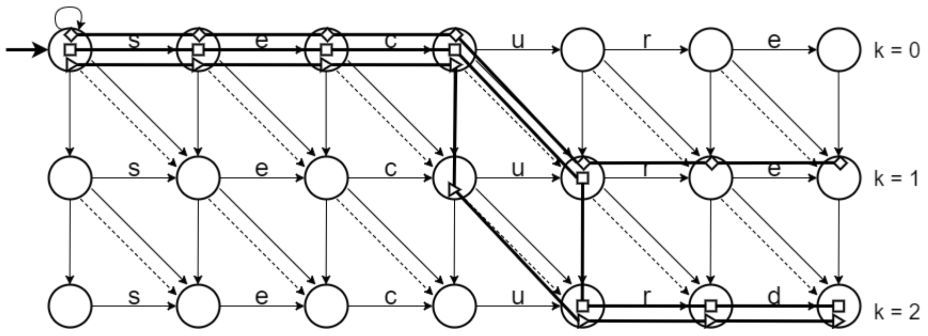

Figure 3: NFA for approximate matching with possible transformations between “secure’ and “secare”.

The line with diamond dots corresponds to the transformation pair $(secure, secare)$, the line with square dots corresponds to $(secu\phi re, sec\phi are)$, and the line with triangular dots corresponds to $(sec\phi ure, seca\phi re)$. It is intuitive to see that by disabling transitions in the automaton we are then by definition applying constraints to $\Gamma(X,Y)$ where $D(X,Y) \leq 2$, but the problem is how to translate more complex constraints as seen performed in the dynamic programming method of Oommen into transitions in an NFA. Challenges which come about are that Oommen’s $W(\cdot,\cdot,\cdot)$ array is a large memory matrix of the possible ways to transform one prefix into another, whereas algorithms for simulating the NFA for approximate matching are typically memoryless. Furthermore, the nondeterministic nature of NFA makes it difficult to see at which stage in the transformation certain edit operations took place since how a string was transformed can be ambiguous. The implementation of complex constraints into the NFA for approximate matching are considered to be open problems.

The Automata Processor

While NFA simulation via bit-parallelism is quite fast, it is limited by the fact that the length of the search pattern must be less than or equal to the size of a computer word w for the Levenshtein approximate search algorithms to run optimally. Recent hardware developments such as the Automata Processor (AP) [3], which is a native hardware implementation for nondeterministic finite automata, allows for approximately 1.5 million NFA states and thousands of NFAs to be used in parallel. Most importantly, all of these states can process an input symbol and access successor states in a single clock cycle [20]. This hardware architecture has already been used for intrusion detection. An application called Fast-SNAP [17], used AP to scan network data streams for 4312 Snort signature rules simultaneously, for which the throughput was 10.3 Gbps. These results highlight the importance of implementing constraints into approximate matching NFA utilizing automata theoretic methods.

5 Summary

In this paper, we discussed the methods for implementing the constrained edit distance, how different constraints can be implemented, and looked at how constrained edit distance algorithms have been applied to digital forensics and cryptanalysis.

The constrained edit distance computation may be implemented as string distance algorithms where they are built on a dynamic programming algorithm by Oommen [12]. Such algorithms implement their constraints by simply choosing which exact edit operations took place in the transformation from string X to Y or more complicated constraints may be implemented by modifying the recursive algorithm to only consider transformations with some maximum consecutive runs of specific edit operations. These algorithms have been applied to cryptanalysis, being used in generalized correlation attacks against stream ciphers with irregularly clocked LFSRs [5][6][7]. Additionally, the constrained edit distance may be used in data reduction for preselection phases of pattern searching in intrusion detection rule databases [15] and digital forensic search spaces [14]. Primary advantages of this dynamic programming method are the abilities to set flexible constraints, and to reconstruct the sequence of edit operations.

Alternatively, the constrained edit distance may be implemented inside algorithms which simulate the nondeterministic finite automaton for approximate string matching. The algorithms we examined simulate this automaton by utilizing bit-parallel approaches [4], and implement their constraints by applying integer counters to each active automaton state representing the maximum number of edit-operations the substring under analysis may still apply. Algorithms of this variety have been used in spam detection [1] and intrusion detection [2]. Additionally, new hardware such as the Automata Processor [3] allows for direct implementation of NFA, which can make searching even faster, and has already been applied to intrusion detection [17].

Lastly, we made a novel observation that the NFA for approximate matching using the Levenshtein distance can be described in terms of Oommen's string editing formalisms. By this definition, it is clear to see that a constrained edit distance may be implemented into this NFA by eliminating or modifying transitions. Translating complex constraints, such as maximum consecutive runs of specific edit operations, which have been performed with Oommen's algorithm into transition relations in an NFA remains to be an open problem.

References

[1] Ambika Shrestha Chitrakar and Slobodan Petrovic. Approximate search with constraints on indels with application in spam filtering. *Norsk informasjonssikkerhetskonferanse (NISK)*, pages 22–33, 2015.

[2] Ambika Shrestha Chitrakar and Slobodan Petrovic. Constrained row-based bit-parallel search in intrusion detection. *Norsk informasjonssikkerhetskonferanse (NISK)*, pages 68–79, 2016.

[3] Paul Dlugosch, Dave Brown, Paul Glendenning, Michael Leventhal, and Harold Noyes. An efficient and scalable semiconductor architecture for parallel automata processing. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3088–3098, 2014.

[4] Simone Faro and Thierry Lecroq. Twenty years of bit-parallelism in string matching. *Festschrift for Borivoj Melichar*, pages 72–101, 2012.

[5] Jovan Dj Golić. Edit distance correlation attacks on clock-controlled combiners with memory. In *Australasian Conference on Information Security and Privacy*, pages 169–181. Springer, 1996.

[6] Jovan Dj Golić and Renato Menicocci. Edit distance correlation attack on the alternating step generator. In *Annual International Cryptology Conference*, pages 499–512. Springer, 1997.

[7] Jovan Dj Golić and Miodrag J Mihaljević. A generalized correlation attack on a class of stream ciphers based on the Levenshtein distance. *Journal of Cryptology*, 3(3):201–212, 1991.

[8] Heikki Hyyrö. Improving the bit-parallel NFA of Baeza-Yates and Navarro for approximate string matching. *Information Processing Letters*, 108(5):313–319, 2008.

[9] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[10] Sanjeeb Mishra. Keyword indexing and searching for large forensics targets using distributed computing. Master's thesis, University of New Orleans Theses and Dissertations. 510, 2007. http://scholarworks.uno.edu/td/510.

[11] Gonzalo Navarro and Mathieu Raffinot. *Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.

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

[13] B John Oommen. Recognition of noisy subsequences using constrained edit distances. *IEEE transactions on pattern analysis and machine intelligence*, (5):676–685, 1987.

[14] Slobodan Petrovic and Katrin Franke. Improving the efficiency of digital forensic search by means of the constrained edit distance. In *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on*, pages 405–410. IEEE, 2007.

[15] Slobodan Petrovic and Katrin Franke. A new two-stage search procedure for misuse detection. In *Future Generation Communication and Networking (FGCN 2007)*, volume 2, pages 418–422. IEEE, 2007.

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

[17] Indranil Roy, Ankit Srivastava, Marziyeh Nourian, Michela Becchi, and Srinivas Aluru. High performance pattern matching using the automata processor. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 1123–1132. IEEE, 2016.

[18] Thomas Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on computers*, 1(C-34):81–85, 1985.

[19] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[20] Ke Wang, Elaheh Sadredini, and Kevin Skadron. Hierarchical pattern mining with the automata processor. *International Journal of Parallel Programming*, pages 1–36, 2017.

[21] Sun Wu and Udi Manber. Fast text searching: allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.