Obtaining Precision-Recall Trade-Offs in Fuzzy Searches of Large Email Corpora

Kyle Porter, Slobodan Petrovic

Chapter 5

OBTAINING PRECISION-RECALL TRADE-OFFS IN FUZZY SEARCHES OF LARGE EMAIL CORPORA

Kyle Porter and Slobodan Petrovic

Abstract Fuzzy search is often used in digital forensic investigations to find words that are stringologically similar to a chosen keyword. However, a common complaint is the high rate of false positives in big data environments. This chapter describes the design and implementation of `cedas`, a novel constrained edit distance approximate string matching algorithm that provides complete control over the types and numbers of elementary edit operations considered in approximate matches. The unique flexibility of `cedas` facilitates fine-tuned control of precision-recall trade-offs. Specifically, searches can be constrained to the union of matches resulting from any exact edit combination of insertion, deletion and substitution operations performed on the search term. The flexibility is leveraged in experiments involving fuzzy searches of an inverted index of the Enron corpus, a large English email dataset, which reveal the specific edit operation constraints that should be applied to achieve valuable precision-recall trade-offs. The constraints that produce relatively high combinations of precision and recall are identified, along with the combinations of edit operations that cause precision to drop sharply and the combination of edit operation constraints that maximize recall without sacrificing precision substantially. These edit operation constraints are potentially valuable during the middle stages of a digital forensic investigation because precision has greater value in the early stages of an investigation while recall becomes more valuable in the later stages.

Keywords: Email forensics, approximate string matching, finite automata

1. Introduction

Keyword search has been a staple in digital forensics since its beginnings, and a number of forensic tools incorporate fuzzy search (or approximate string matching) algorithms that match text against keywords with typographical errors or keywords that are stringologically similar. These algorithms may be used to search inverted indexes, where every approximate match is linked to a list of documents that contain the match.

Great discretion must be used when employing these forensic tools to search large datasets because many strings that match (approximately) may be similar in a stringological sense, but are completely unrelated in terms of their semantics. Even exact keyword matching produces an undesirable number of false positive documents to sift through, where as much as 80% to 90% of the returned document hits could be irrelevant [2]. Nevertheless, the ability to detect slight textual aberrations is highly desirable in digital forensic investigations. For example, in the 2008 Casey Anthony case, in which Ms. Anthony was convicted and ultimately acquitted of murdering her daughter, investigators missed a Google search for a misspelling of the word "suffocation," which was written as "suffication" [1].

Digital forensic tools such as dtSearch [8] and Intella [24] incorporate methods for controlling the "fuzziness" of searches. While the tools use proprietary techniques, it appears that they utilize the edit distance [16] in their fuzzy searches. The edit distance – or Levenshtein distance – is defined as the minimum number of elementary edit operations that can transform a string X to a string Y, where the elementary edit operations are defined as the insertion of a character, deletion of a character and substitution of a character in string X. However, precise control of the fuzziness of searches is often limited. In fact, it may not be clear what modifying the fuzziness of a search actually does other than the results "looking" more fuzzy. For example, some tools allow fuzziness to be expressed using a value between 0 to 10, without clarifying exactly what the values represent.

The research described in this chapter has two contributions. The first is the design and implementation of a novel constrained edit distance approximate search `cedas` algorithm, which provides complete control over the types and numbers of elementary edit operations considered in approximate matches. The flexibility of search, which is unique to `cedas`, allows for fine-tuned control of precision-recall trade-offs. Specifically, searches can be constrained to the union of matches resulting from any exact edit operation combination of insertions, deletions and substitutions performed on the search term.

The second contribution, which is a consequence of the first, is an experimental demonstration of which edit operation constraints should be applied to achieve valuable precision-recall trade-offs in fuzzy searches of an inverted index of the Enron Corpus [4], a large English email dataset. Precision-recall trade-offs with relatively high precision are valuable because fuzzy searches typically have high rates of false positives and increasing recall is simply obtained by conducting fuzzy searches with higher edit distance thresholds. The experiments that were performed identified the constraints that produce relatively high combinations of precision and recall, the combinations of edit operations that cause precision to drop sharply and the combination of edit operation constraints that maximize recall without sacrificing precision substantially. These edit operation constraints appear to be valuable during the middle stages of an investigation because precision has greater value in the early stages of an investigation whereas recall becomes more valuable later in an investigation [17].

2. Background

This section discusses the underlying theory and algorithms.

2.1 Approximate String Matching Automata

A common method for performing approximate string matching, as implemented by the popular `agrep` suite [25], is to use a nondeterministic finite automaton (NFA) for approximate matching. Since `cedas` implements an extension of this automaton, it is useful to discuss some key components of automata theory.

A finite automaton is a machine that takes a string of characters X as input and determines whether or not the input contains a match for some desired string Y. An automaton comprises a set of states Q that can be connected to each other via arrows called transitions, where each transition is associated with a character or a set of characters from some alphabet Σ. The set of initial states $I \subseteq Q$ comprise the states that are active before reading the first character. States that are active check the transitions originating from themselves when a new character is being read; if a transition includes the character being read, then the state pointed to by the arrow becomes active. The set of states $F \subseteq Q$ correspond to the terminal states; if any of these states become active, then a match has occurred. The set of strings that result in a match are considered to be accepted by the automaton; this set is the language L recognized by the automaton.

Figure 1 shows the nondeterministic finite automaton for approximate matching A_L, where the nondeterminism implies that any number of states may be active simultaneously. The initial state of A_L is the node with a bold arrow pointing to it; it is always active as indicated

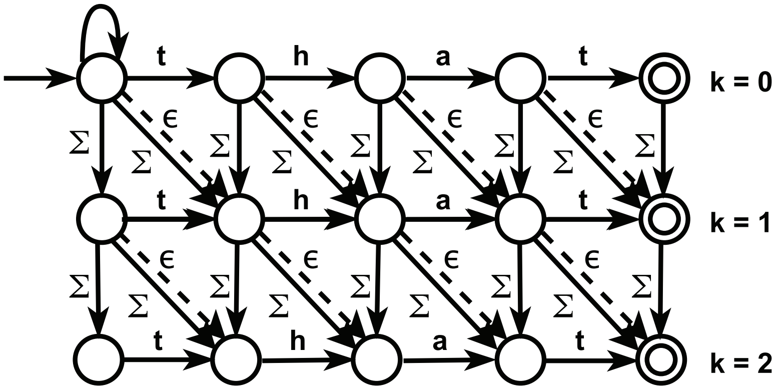


Figure 1. NFA matching the pattern "that" (allowing two edit operations).

by the self-loop. The terminal states are the double-circled nodes. Horizontal arrows denote exact character matches. Diagonal arrows denote character substitutions and vertical arrows denote character insertions, where both transitions consume a character in Σ. Since A_L is a nondeterministic finite automaton, it permits ϵ-transitions, where transitions are made without consuming a character. Dashed diagonal arrows express ϵ-transitions that correspond to character deletions. For approximate search with an edit distance threshold of k, the automaton has $k+1$ rows.

The automaton A_L is very effective at pattern matching because it checks for potential errors in a search pattern simultaneously. For every character consumed by the automaton, each row checks for potential matches, insertions, deletions and substitutions against every position in the pattern.

For common English text, it is suggested that the edit distance threshold for approximate string matching algorithms should be limited to one, and in most cases should never exceed two [9]. This suggestion is well founded because about 80% of the misspellings in English text are due to a single edit operation [5].

Let $L_{k=1}$ and $L_{k=2}$ be the languages accepted by automaton A_L with thresholds $k = 1$ and $k = 2$, respectively. The nondeterministic finite automaton A_T described in this section allows for different degrees of fuzziness that enable the exploration of the entire space between $L_{k=1}$ and $L_{k=2}$ in terms of the exact combinations of elementary edit operations applied to the search keyword. This automaton accepts the languages L_T, where $L_{k=1} \subseteq L_T \subseteq L_{k=2}$.

The automaton A_T is constructed in the following manner. The automaton that accepts $L_{k=2}$ can be viewed as the union of the languages accepted by each of its rows. For example, for an edit distance threshold of $k = 2$, the first row accepts the language comprising matches that have no edit operations performed on the keyword, the second row accepts the language of matches with one edit operation performed on the keyword and the third row accepts the language of matches with two edit operations performed on the keyword. The union of these subsets is a cover of $L_{k=2}$. An alternative cover of $L_{k=2}$ is the union of all the languages accepted by the automata for a specific number of insertions i, deletions e and substitutions s performed in a match such that $i + e + s \leq k$.

The following lemma proves the equivalence of the covers.

Lemma. Let L_k be the language accepted by an automaton such that k elementary edit operations are performed on a specified pattern. Let $L_{k=n}$ be the language accepted by the nondeterministic finite automaton for approximate matching with edit distance threshold n be equivalent to its cover $C_\alpha = \cup_{k=0}^{k=n} L_k$. Furthermore, let $L_{(i,e,s)}$ be equivalent to the language accepted by an automaton such that exactly i insertions, e deletions and s substitutions have been performed on a specified pattern. Let $C_\beta = \cup_{i,e,s:0 \leq i+e+s \leq n} L_{(i,e,s)}$. Then, $C_\alpha = C_\beta$.

Proof. For all $x \in L_k$, there exists $L_{(i,e,s)}$ such that $x \in L_{(i,e,s)}$, where $i + e + s = k$. Therefore, $C_\alpha \subset C_\beta$. For all $x \in L_{(i,e,s)}$ such that $i + e + s = k$, $x \in L_k$. Thus $C_\beta \subset C_\alpha$. □

By constraining the possible edit operations between the rows of the automaton A_T, each row of the automaton can correspond to a specific combination of i insertions, e deletions and s substitutions such that $i + e + s \leq k$ for edit distance threshold k instead of each row corresponding to some number of edit operations. This construction enables the accepted language L_T to be controlled by allowing terminal states $f \in F$ to remain in F or removing them from F. Specifically, some $L_{(i,e,s)}$ can be chosen to be not included in $C_\beta = \cup_{i,e,s:0 \leq i+e+s \leq n} L_{(i,e,s)}$.

2.2 NFA Definition

The constrained edit distance between two strings X and Y is the minimum number of edit operations required to transform X to Y given that the transformation obeys some pre-specified constraints T [19]. In general, constraints may be defined arbitrarily as long as they consider the numbers and types of edit operations. Let (i, e, s) be an element of T, the set of edit operations that constrain a transformation from string X to string Y, where (i, e, s) is an exact combination of edit operations. A_T may perform approximate searches where matches are constrained to the

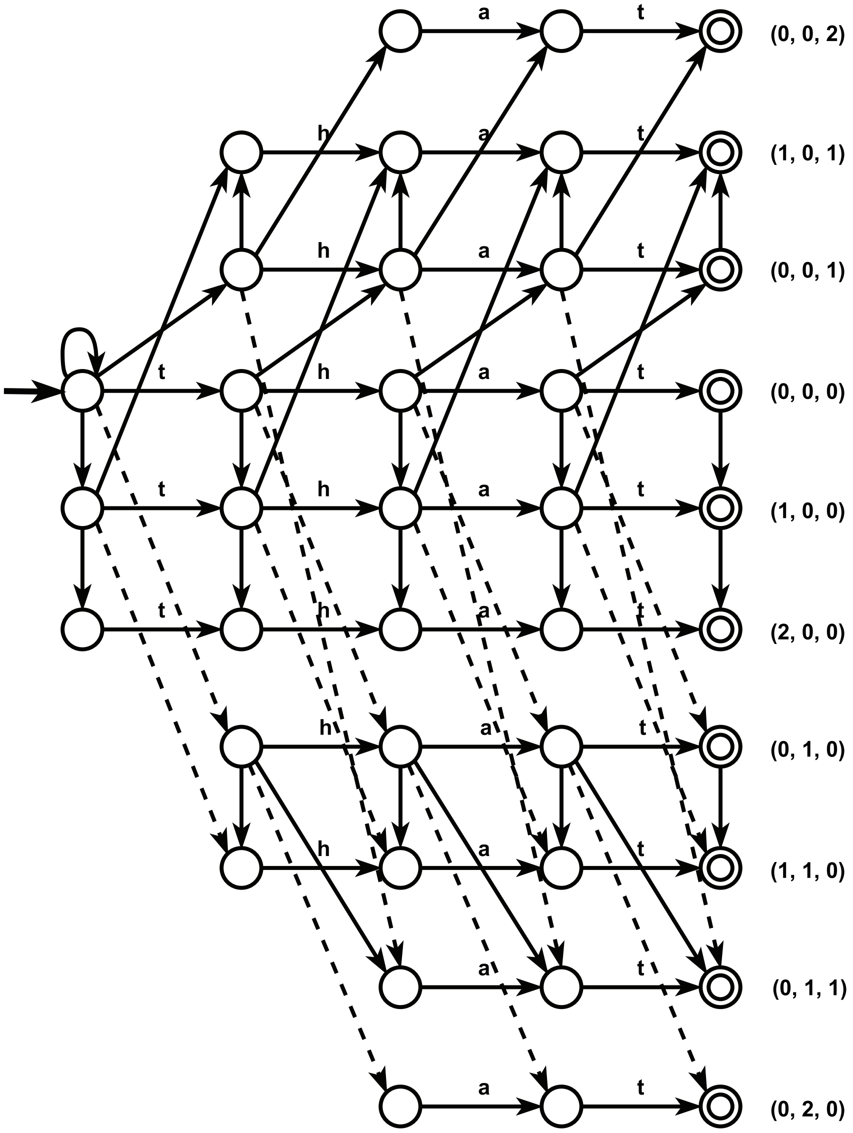


Figure 2. NFA matching the pattern "that."

allowed edit operation combinations in T. For example, the search may be constrained to approximate matches derived from the edit operation combinations $(0,0,0)$, $(1,0,1)$ and $(0,2,0)$. The corresponding accepted language is $L_{(0,0,0)} \cup L_{(1,0,1)} \cup L_{(0,2,0)}$.

Figure 2 shows the constrained edit distance nondeterministic finite automaton A_T. It uses the same symbol conventions as the nondeter-

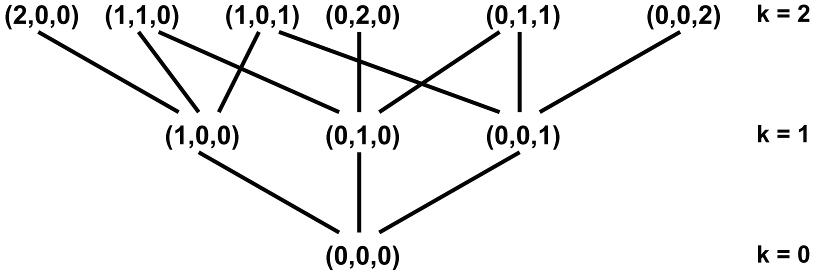


Figure 3. Partially ordered multisets (i, e, s).

ministic finite automaton in Figure 1, except that substitutions and insertions are expressed by diagonal and vertical transitions, respectively, where the transitions may go up or down. In order to ensure that each row $R_{(i,e,s)}$ of the automaton A_T corresponds to the accepted language $L_{(i,e,s)}$, it is necessary to engage the notion of a partially ordered set of multisets, which describes the edit operation transpositions that connect each row. The following definitions [12] are required:

Definition. Let X be a set of elements. Then, a multiset M drawn from set X is expressed by a function count M or C_M defined as $C_M : X \rightarrow N$, where N is the set of non-negative integers. For each $x \in X$, $C_M(x)$ is the characteristic value of x in M, which indicates the number of occurrences of elements x in M. A multiset M is a set if $C_M(x) = 0$ or 1 for all $x \in X$.

Definition. Let M_1 and M_2 be multisets selected from a set X. Then, M_1 is a submultiset of M_2 ($M_1 \subseteq M_2$) if $C_{M_1}(x) \leq C_{M_2}(x)$ for all $x \in X$. M_1 is a proper submultiset of M_2 ($M_1 \subset M_2$) if $C_{M_1}(x) \leq C_{M_2}(x)$ for all $x \in X$ and there exists at least one $x \in X$ such that $C_{M_1}(x) < C_{M_2}(x)$.

The set of multisets considered here comprises the elements (i, e, s), which implies that the multiset contains i insertions, e deletions and s substitutions. The cardinality of the multisets is no greater than the edit distance threshold k. The partial ordering of this set of multisets is the binary relation $\sim$, where for multisets M_1 and M_2, $M_1 \sim M_2$ means that M_1 is related to M_2 via $M_1 \subset M_2$.

Figure 3 presents the partially ordered multiset diagram D. Diagram D models the edit operation transitions between the rows of automaton A_T, where each multiset element (i, e, s) corresponds to row $R_{(i,e,s)}$ and each row of D corresponds to a sum of edit operations. As seen in D, every $R_{(i,e,s)}$ has a specific edit operation transition sent to it from a specific row. In this way, each row $R_{(i,e,s)}$ can determine which (and how

Table 1. Bit-masks for the word “that.”

Character ($\mathbf{t_j}$)	**Bit-Mask** ($\mathbf{B[t_j]}$)
a	01000
h	00100
t	10010
*	00000

many) elementary edit operations are being considered in a match due to the partial ordering. For example, $R_{(1,0,0)}$ only has insertion transitions going to it from $R_{(0,0,0)}$, and $R_{(1,1,0)}$ only has deletion transitions going to it from $R_{(1,0,0)}$ (where one insertion has already taken place) and it only has insertion transitions going to it from $R_{(0,1,0)}$ (where one deletion has already taken place).

Finally, since the automaton is nondeterministic, it cannot be implemented directly using a von Neumann architecture.

2.3 Bit-Parallel Implementation

Bit-parallelism allows for an efficient simulation of a nondeterministic finite automaton. The method uses bit-vectors to represent each row of the automaton, where the vectors are updated via basic logical bitwise operations that correspond to transition relations of the automaton. Because bitwise operations update every bit in the bit-vector simultaneously, it updates the states in the row of the automaton simultaneously. If the lengths of the bit-vectors are not greater than the number of bits w in a computer word, then the parallelism reduces the maximum number of operations performed by a search algorithm by w [10].

In the bit-parallel nondeterministic finite automaton simulation, each row of the automaton for the search pattern X is expressed as a binary vector of length $|X| + 1$; this also requires the input characters to be expressed as vectors of the same size. Input characters t_j are handled by bit-masks. Thus, a table of bit-masks $B[t_j]$ is created, where each bit-mask represents the positions of the character t_j in pattern X. Table 1 shows the bit-masks when X = “that.” Characters that are not present in the pattern are expressed using the “*” symbol.

Algorithm 1 presents the bit-parallel simulation of automaton A_T. The algorithm is an extension of the simulation of the nondeterministic finite automaton for approximate string matching using the unconstrained edit distance; this was first implemented by Wu and Manber [26]. Therefore, the components of the A_T simulation are similar, the primary modifications being the transition relationships between rows

Algorithm 1: NFA update algorithm.

```
Initialize all rows $R'$ to 0 except $R'(0,0,0) \leftarrow$ 0x00000001,
      $R'(0,1,0) \leftarrow$ 0x00000002, $R'(0,2,0) \leftarrow$ 0x00000004;
for each input character $t$ do
      $R_{(0,0,0)} \leftarrow R'_{(0,0,0)}$;
      $R'_{(0,0,0)} \leftarrow ((R'_{(0,0,0)} << 1)\ \&\ B[t])\ |$ 0x00000001;
      $R_{(1,0,0)} \leftarrow R'_{(1,0,0)}$;
      $R'_{(1,0,0)} \leftarrow ((R'_{(1,0,0)} << 1)\ \&\ B[t]])\ |\ R_{(0,0,0)}$;
      $R_{(0,1,0)} \leftarrow R'_{(0,1,0)}$;
      $R'_{(0,1,0)} \leftarrow ((R'_{(0,1,0)} << 1)\ \&\ B[t]])\ |\ (R'_{(0,0,0)} << 1)$;
      $R_{(0,0,1)} \leftarrow R'_{(0,0,1)}$;
      $R'_{(0,0,1)} \leftarrow ((R'_{(0,0,1)} << 1)\ \&\ B[t]])\ |\ (R_{(0,0,0)} << 1)$;
      $R_{(0,1,1)} \leftarrow R'_{(0,1,1)}$;
      $R'_{(0,1,1)} \leftarrow ((R'_{(0,1,1)} << 1)\ \&\ B[t])\ |\ (R_{(0,1,0)} << 1)\ |\ (R'_{(0,0,1)} << 1)$;
      $R_{(1,0,1)} \leftarrow R'_{(1,0,1)}$;
      $R'_{(1,0,1)} \leftarrow ((R'_{(1,0,1)} << 1)\ \&\ B[t])\ |\ (R_{(1,0,0)} << 1)\ |\ R_{(0,0,1)}$;
      $R_{(1,1,0)} \leftarrow R'_{(1,1,0)}$;
      $R'_{(1,1,0)} \leftarrow ((R'_{(1,1,0)} << 1)\ \&\ B[t])\ |\ R_{(0,1,0)}\ |\ (R'_{(1,0,0)} << 1)$;
      $R_{(2,0,0)} \leftarrow R'_{(2,0,0)}$;
      $R'_{(2,0,0)} \leftarrow ((R'_{(2,0,0)} << 1)\ \&\ B[t])\ |\ R_{(1,0,0)}$;
      $R_{(0,2,0)} \leftarrow R'_{(0,2,0)}$;
      $R'_{(0,2,0)} \leftarrow ((R'_{(0,2,0)} << 1)\ \&\ B[t])\ |\ (R'_{(0,1,0)} << 1)$;
      $R_{(0,0,2)} \leftarrow R'_{(0,0,2)}$;
      $R'_{(0,0,2)} \leftarrow ((R'_{(0,0,2)} << 1)\ \&\ B[t])\ |\ (R_{(0,0,1)} << 1)$;
      if $(((R'_{(0,0,0)}\ \&\ R^B_{(0,0,0)})\ |\ (R'_{(0,0,1)}\ \&\ R^B_{(0,0,1)})\ |\ (R'_{(0,1,0)}\ \&\ R^B_{(0,1,0)})\ |$
            $(R'_{(1,0,0)}\ \&\ R^B_{(1,0,0)})\ |\ (R'_{(0,1,1)}\ \&\ R^B_{(0,1,1)})\ |\ (R'_{(1,0,1)}\ \&\ R^B_{(1,0,1)})\ |$
            $(R'_{(1,1,0)}\ \&\ R^B_{(1,1,0)})\ |\ (R'_{(2,0,0)}\ \&\ R^B_{(2,0,0)})\ |\ (R'_{(0,2,0)}\ \&\ R^B_{(0,2,0)})\ |$
            $(R'_{(0,0,2)}\ \&\ R^B_{(0,0,2)}))\ \&$ (0x00000001 $<< n-1$)) then
            Match is found;
      end
end
```

and the number of rows. The rows of the automaton are denoted by bitvectors $R_{(i,e,s)}$ and their updated values are denoted by $R'_{(i,e,s)}$. $R^B_{(i,e,s)}$ represents the Boolean values for whether or not row $R_{(i,e,s)}$ reports a match, which occurs when $R_{(i,e,s)}$ has a terminal state. For some i, e and s, the value $R^B_{(i,e,s)}$ is true when $(i, e, s) \in T$.

Each row is updated by first checking if the input character is an exact match for the row by computing $((R'_{(i,e,s)} << 1)\ \&\ B[t])$. This value is then bitwise ORed with potential transition relationships. $R_{(i,e,s)}$ checks for insertions, $(R'_{(i,e,s)} << 1)$ checks for deletions and $(R_{(i,e,s)} << 1)$ checks for substitutions from a row $R_{(i,e,s)}$. As mentioned above, incoming transitions for a row $R_{(i,e,s)}$ may be determined by checking against the incoming relations to the multiset (i, e, s) in diagram D in

Figure 3. After updating all the rows, matches are checked by bitwise ANDing each of the allowed rows and determining if any bit representing the terminal states is set to one.

2.4 Evaluation

The additional flexibility in specifying fuzziness comes at the cost of time and space. The time and space complexities of `cedas` can be specified in terms of the number of multisets in diagram D for an edit distance threshold k. This number is equal to the number of rows in Algorithm 1, which is $f(k) = \frac{1}{6}(k+1)(k+2)(k+3)$. Therefore, for keyword searches shorter than 64 characters on an x64 architecture, the worst-case space complexity is cubic in k and the worst-case time complexity is $O(k^3 n)$, where n is the length of the text searched. Obviously, this implies that the algorithm should not be applied to time sensitive tasks with massive throughput such as intrusion detection, or applied to a live search of massive forensic data with high edit distance thresholds k. However, this is sufficient for specifying the fuzziness of approximate searches over an index of emails, because the values of k should be low and the index acts as a data reduction mechanism. The experimentation described in the next section demonstrates that the algorithm runs approximately six times slower than `agrep` with an edit distance threshold of $k = 2$.

3. Experimental Methodology

The flexibility of `cedas` was evaluated by specifying fuzziness in the context of an investigation of the Enron email dataset [4] and assessing the effectiveness of the constraints. The email messages were converted to the mbox format to simplify processing. Only the contents of the email bodies were examined.

In order to search the data, an inverted index of the emails was created using the `mkid` program [11], which yielded a database of index tokens. After deduplicating the tokens in the index, all the tokens were output to a single text file that was searched using `cedas`. Note that the choice of indexing algorithm affects the list of tokens because different algorithms may interpret delimiters differently and, therefore, would affect any search. The list of tokens used in the experiments totaled 460,800 unique words.

Twenty-eight different keywords related to the 2001 Enron scandal were used. This list of keywords was not compiled by a digital forensic investigator, so the choice of keywords could be improved. Keywords were chosen that were relevant to the case, but would not obviously

Table 2. Keyword list.

Word Length	Keywords
6	Cuiaba
7	BlueDog, BobWest, corrupt, illegal, launder, Sarzyna, scandal
8	bankrupt, Backbone, Fishtail, Margaux1, Shutdown, subpoena, Velocity, unlawful
9	collusion, Whitewing, Yosemite
10	Catalytica, conspiracy, KennethLay, litigation, reputation, suspicious
>10	ArthurAndersen, illegitimate, talkingpoints

produce an overwhelming number of false positives. Furthermore, no keyword that contained less than six characters was chosen.

Unconstrained fuzzy searching with an edit distance threshold of $k = 2$ for small keywords produces massive lists of words (often exceeding 10,000 words) that have to be analyzed manually. This can be viewed as a limitation of the proposed approach. Table 2 shows the list of search keywords. Many of the words were taken from Rodger Lepinsky's webpage on data science and the Enron corpus [15].

A case-insensitive fuzzy search for each keyword was conducted on the list of index tokens, and each search was done under 64 different constraints. A match occurred if a keyword was found as a substring of an index token with the allowed tolerance of edit operations as specified by the constraints in terms of $(i, e, s) \in T$. All the approximate matches found in the index were returned in a list. The elements (i, e, s) that were possibly not included in T were those in which the sum $i+e+s$ was equal to the edit distance threshold $k = 2$. Specifically, the automaton accepted $L_{(0,0,0)}, L_{(1,0,0)}, L_{(0,1,0)}, L_{(0,0,1)}$, but the inclusion of all possible combinations of languages $L_{(i,e,s)}$ such that $i + e + s = 2$ was allowed.

The effectiveness of a search for each constraint on each keyword was measured in terms of the precision and recall derived from the list of returned approximate matches. To understand the overall effectiveness of each constraint, the average precision and recall results for all the keywords under each constraint were computed as harmonic means. The harmonic mean was chosen because the arithmetic mean produced overly optimistic results for fuzzy searches under the chosen constraints.

3.1 Interpreting Match Results

Precision and recall have been used by researchers to gauge the effectiveness of approximate string matching algorithms [3, 20]. Precision is the proportion of retrieved items that are relevant whereas recall is the proportion of total relevant items retrieved [13]. As recall increases, precision tends to decrease. Precision and recall are expressed as:

$$\text{Precision} = \frac{|(\text{retrieved items}) \cap (\text{relevant items})|}{|(\text{retrieved items})|} \tag{1}$$

$$\text{Recall} = \frac{|(\text{retrieved items}) \cap (\text{relevant items})|}{|(\text{relevant items})|} \tag{2}$$

The precision and recall metrics are useful, but they are not perfect because the notion of relevance is subjective. Relevant terms are defined as being variations of the original term (e.g., obtaining "litigating" when searching for "litigation"), closely related to the original term in a semantic sense (e.g., obtaining "legalese" when searching for "illegal"), or misspellings of the original term. However, if a keyword is a substring of the examined index token and is clearly unrelated, then it is not considered relevant. For example, if the search term is "audit" and a hit is obtained for "AudiTalk," then the hit is not relevant. For this reason, the classification of relevant versus non-relevant hits for approximate hits is always a manual process.

Another shortcoming of the metrics is that it is not possible to compute the true precision and recall for every keyword; this is because the number of relevant words for each keyword in the Enron dataset is unknown. Therefore, a compromise was employed: the number of total relevant items was set to be equal to the items identified for each keyword for unconstrained approximate matching at the edit distance threshold $k = 2$. This implies that unconstrained approximate matching with $k = 2$ yields 100% recall, which is not necessarily true.

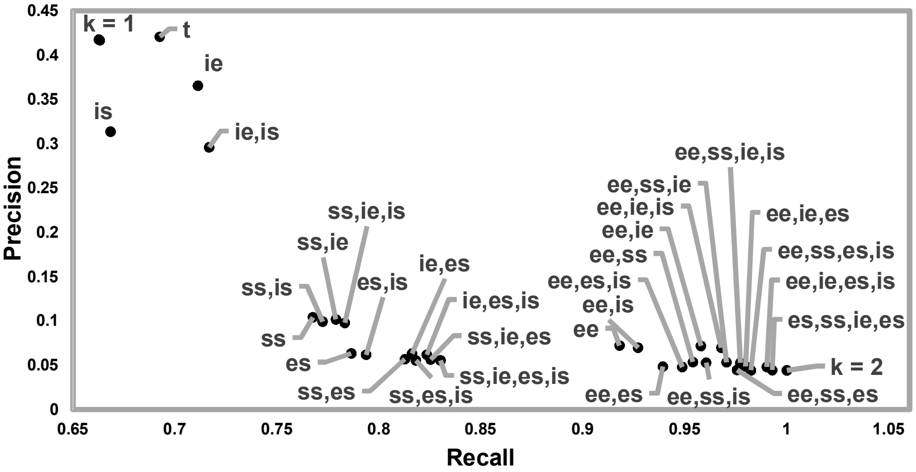
Finally, it is important to note that approximate string matching results are highly dependent on the specific data being matched and the keywords being used [6]. Therefore, utilizing `cedas` on an inverted index that was not derived from an English email corpus may produce different results.

4. Experimental Results

This section describes the experimental results.

4.1 Precision and Recall

The results in this section reflect the effectiveness of each set of constraints T in terms of precision and recall, and identify the constraints that produce valuable results for investigating an English email corpus.



(1, 1, 0) = ie; (1, 0, 1) = is; (0, 1, 1) = es; (2, 0, 0) = ii; (0, 2, 0) = ee; (0, 0, 2) = ss

Figure 4. Precision-recall trade-off curve for various constraints.

Figure 4 shows the precision-recall trade-off curve for various constraints. Data points labeled $k = 1$ and $k = 2$ represent the results for unconstrained fuzzy searches with edit distance thresholds set to one and two, respectively.

As expected, the application of constraints to fuzzy searches of the Enron inverted index resulted in higher recall than an unconstrained fuzzy search with an edit distance threshold of $k = 1$, and better precision than an unconstrained fuzzy search with an edit distance threshold of $k = 2$. However, the primary interest is in the precision-recall trade-offs that are useful in an investigation. As mentioned in the introduction, a common complaint is the number of false positives produced by a fuzzy search, and the fact that precision is valued more than recall early in an investigation [17]. This implies that the constraints that produce results with relatively high precision are most useful in an investigation, because increased recall is easily obtained by increasing the edit distance threshold.

What is immediately apparent from the data is that there are several distinct clusters of data points, where each cluster is associated with different edit operation combinations. The cluster with the highest precision comprises all the data points near data point $k = 1$, where $(0, 1, 1), (0, 0, 2), (0, 2, 0) \notin T$. This means that matches under these constraints did not include edit operations with exactly two substitutions, a substitution and deletion, or two deletions performed on the keyword.

Table 3. Average execution times for agrep and cedas.

agrep	cedas
0.0477142857 s	0.2795714286 s

It follows that, in order to preserve the precision of a fuzzy search to be near the unconstrained case of an edit distance threshold of $k = 1$, it is necessary to constrain the fuzzy search to edit operations that do not include the previously mentioned edit operation combinations. Furthermore, the data points in the cluster mostly show a marked improvement in recall. Because of the relatively high precision and recall, it can be posited that the constraints in this cluster are useful in the middle stages of an investigation.

Data points in this cluster that include a single insertion and deletion (*ie*) have very good precision-recall tradeoffs. To ensure that the results of a fuzzy search with these constraints are simply not due to the transposition edit operations for which adjacent characters may be swapped, the same tests were conducted using the nrgrep [18] algorithm to perform an unconstrained fuzzy search allowing transpositions with an edit distance threshold of $k = 1$. These results are represented by data point t and it can be seen that *ie* and t do not yield the same results.

4.2 Execution Time

The speed of cedas was evaluated by timing the unconstrained fuzzy search with an edit distance threshold of $k = 2$ for every keyword in the Enron inverted index. The average of these results was computed and compared with the average of the results obtained using agrep.

The results in Table 3 demonstrate that cedas executed nearly six times slower than agrep for an edit distance threshold $k = 2$. However, it should be noted that the cedas implementation was not optimized; therefore, it has the potential to run faster than measured.

4.3 Analysis and Suggestions

The gap in precision between the higher and lower data clusters is potentially shaped by the statistics of the English language. Additional experimentation is necessary to confirm this observation, but it is clearly easy to transform words to other words using many substitutions or deletions. By limiting the application of deletion and substitution edit operations, the structure of the original word is preserved. For this reason, if somewhat high precision is needed, it is appropriate to use a fuzzy search that does not include edit operations involving two deletions, two substitutions, or a single deletion and a single substitution.

To maximize the recall in the fuzzy search results without sacrificing precision as seen when applying many of the edit operation combinations, it is suggested that the set of constraints T should contain (0,0,0), (1,0,0), (0,1,0), (0,0,1), (1,1,0), (1,0,1), (2,0,0); for the sake of brevity, the language accepted by this automaton is denoted by $L_{-(ee,ss,es)}$. The precision and recall of this language correspond to the data point ie, is. If more precision is needed with nearly as much recall, then the set of constraints T should contain (0,0,0), (0,0,0), (1,0,0), (0,1,0), (0,0,1), (1,1,0), (2,0,0) (whose precision and recall correspond to data point ie). Ultimately, `cedas` users should choose constraints based on the precision-recall trade-offs they are willing to tolerate.

5. Related Work

Fuzzy search algorithms are implemented in digital forensic tools such as dtSearch [8] and Intella [24]. However, the variables for setting the tolerated fuzziness in these tools do not always correlate directly with the edit distance thresholds. Whether or not these tools employ constrained edit distance algorithms is unknown because their techniques are proprietary. Nevertheless, they appear to be combining the edit distance measure with other types of distance measures, natural language processing and/or information retrieval techniques.

The `agrep` [25] and `nrgrep` [18] open-source tools may be used for fuzzy searches in digital forensic investigations. The tools incorporate various approximate matching algorithms, including edit-distance-based approximate matching, prefix matching, regular expression matching and other options. They can be considered to represent the cutting edge of bit-parallel nondeterministic finite automaton implementations for approximate matching in terms of speed and utility. However, the primary advantage of `cedas` compared with the edit distance matching algorithms used by the tools is its flexibility in constraining edit operations. The `agrep` tool does not implement constraints; as a result, specifying fuzziness in terms of edit distance operations is limited to setting the edit distance threshold. The `nrgrep` tool is more flexible in that it allows a user to set an edit distance threshold, use transpositions and define a subset of the edit operations used in a search. The last feature essentially constrains edit operations, thereby producing a subset of possible edit operation constraints as in the case of `cedas`. However, this type of matching cannot return results equivalent to $L_{-(ee,ss,es)}$.

Other researchers have also proposed constrained edit distance search algorithms. For example, Chitrakar and Petrovic [21, 22] have specified constrained edit distance search algorithms that employ row-based bit-parallelism. One algorithm specifies edit operation constraints in terms of the maximum number of allowed indels [21] (sum of insertions and deletions). A second algorithm expresses edit operation constraints in terms of the maximum allowed number of insertions, deletions and substitutions permitted in a match [22]. These algorithms also engage subsets of possible constraints permitted by `cedas`, but they cannot specify constraints that yield the language $L_{-(ee,ss,es)}$. Experiments by Chitrakar and Petrovic reveal that their algorithms are nearly as fast as `agrep`.

6. Conclusions

This chapter has presented `cedas`, a novel constrained edit distance fuzzy search algorithm that performs approximate searches where the possible transformations on the search terms are constrained to any set of edit operation combinations with exactly i insertions, e deletions and s substitutions. The algorithm is a bit-parallel simulation of a nondeterministic finite automaton, in which the rows of the automaton are defined not by the number of elementary edit operations considered, but by the numbers and types of edit operations. This flexibility in defining edit operation constraints for approximate search is unique to `cedas`.

Experiments employed the `cedas` algorithm to perform constrained edit distance fuzzy searches for a list of keywords in an inverted index of the Enron email corpus. The average precision and recall results of searches applying various edit operation combination constraints identified the constraints that were the most valuable for fuzzy searches of an English email dataset. Because a common complaint against fuzzy search is its large number of false positives, edit operation constraints that yield high precision would be valuable in digital forensic investigations.

The experiments revealed that, in order to avoid the precision drop commonly seen in unconstrained fuzzy search at an edit distance threshold of $k = 2$, it is necessary to constrain fuzzy search to not include any matches involving two deletions, two substitutions, or a substitution and deletion. Fuzzy searches with an edit distance threshold of two and whose constraints did not include the previously mentioned edit operation combinations produced relatively high combinations of precision and recall, where the precision is somewhat reduced with an unconstrained edit distance threshold of $k = 1$ while also improving recall. To maximize the recall of a fuzzy search without significant precision reduction, the combination of edit operations (i, e, s) should be constrained to (0,0,0), (1,0,0), (0,1,0), (0,0,1), (1,1,0), (1,0,1) and (2,0,0). These findings should be useful in the middle stages of an investigation because precision has greater value in the early stages and recall becomes more valuable later in an investigation [17].

The flexibility of `cedas` comes at a cost. The worst-case space complexity of the algorithm is cubic in k and the worst-case time complexity is $O(k^3 n)$ for searching keywords of length less than 64 characters on an x64 architecture, where k is the edit distance threshold and n is the length of the searched text. During the experiments, it was discovered that the average time taken to perform an unconstrained approximate search with edit distance threshold $k = 2$ on the inverted index of the Enron dataset and return the list of approximate matches increased from about 0.0477 seconds with `agrep` to about 0.2796 seconds with `cedas`. It is important to note that the `cedas` implementation has not been optimized. In fact, the space and time requirements can be reduced by dynamically generating the rows of the automaton that are necessary instead of simply removing terminal states from specific rows.

The implementation of `cedas` in a hardware architecture targeted for nondeterministic finite automata (e.g., Automata Processor [7]) could potentially run in linear time. Tracy et al. [23] have shown that the nondeterministic finite automaton for approximate matching (Figure 1) runs in worst-case linear time on the Automata Processor, where the hardware could maximally handle a nondeterministic finite automaton with a search pattern length of 2,730 characters with an edit distance threshold $k = 4$. The fastest bit-parallel nondeterministic finite automaton simulations of the same type of automaton require that search patterns do not exceed about 30 characters to preserve optimal results with a worst-case time complexity of $O(\lceil (m-k)(k+1)/w \rceil n)$ [14], where m is the length of the search pattern, k is the edit distance threshold and n is the length of the input. Finally, other improvements, such as those employed by other search tools, can also be made to `cedas`; these include prefix matching and character-specific fuzziness.

Acknowledgement

This research was supported by the Research Council of Norway Program IKTPLUSS under the R&D Project: “Ars Forensica – Computational Forensics for Large-Scale Fraud Detection, Crime Investigation and Prevention” (Grant Agreement 248094/O70).

References

[1] Associated Press, Casey Anthony detectives missed ‘suffocation’ search, *USA Today*, November 25, 2012.

[2] N. Beebe and J. Clark, Digital forensic text string searching: Improving information retrieval effectiveness by thematically clustering search results, *Digital Investigation*, vol. 4(S), pp. S49–S54, 2007.

[3] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar and S. Fienberg, Adaptive name matching in information integration, *IEEE Intelligent Systems*, vol. 18(5), pp. 16–23, 2003.

[4] W. Cohen, Enron Email Dataset, Machine Learning Department, Carnegie Mellon University, Pittsburgh, Pennsylvania (`www.cs.cmu.edu/~./enron`), 2015.

[5] F. Damerau, A technique for computer detection and correction of spelling errors, *Communications of the ACM*, vol. 7(3), pp. 171–176, 1964.

[6] R. da Silva, R. Stasiu, V. Moreira Orengo and C. Heuser, Measuring quality of similarity functions in approximate data matching, *Journal of Informetrics*, vol. 1(1), pp. 35–46, 2007.

[7] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal and H. Noyes, An efficient and scalable semiconductor architecture for parallel automata processing, *IEEE Transactions on Parallel and Distributed Systems*, vol. 25(12), pp. 3088–3098, 2014.

[8] dtSearch, Over 25 Federated and Concurrent Search Options, Bethesda, Maryland (`www.dtsearch.com/PLF_Features_2.html`), 2018.

[9] Elasticsearch, Fuzzy Query, Mountain View, California (`www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-fuzzy-query.html`), 2017.

[10] S. Faro and T. Lecroq, Twenty years of bit-parallelism in string matching, in *Festschrift for Borivoj Melichar*, J. Holub, B. Watson and J. Zdarek (Eds.), Prague Stringology Club, Prague, Czech Republic, pp. 72–101, 2012.

[11] Free Software Foundation, ID Database Utilities, GNU Operating System, Boston, Massachusetts (`www.gnu.org/software/idutils/manual/idutils.html`), 2012.

[12] K. Girish and J. Sunil, General relations between partially ordered multisets and their chains and antichains, *Mathematical Communications*, vol. 14(2), pp. 193–205, 2009.

[13] P. Hall and G. Dowling, Approximate string matching, *ACM Computing Surveys*, vol. 12(4), pp. 381–402, 1980.

[14] H. Hyyro, Improving the bit-parallel NFA of Baeza-Yates and Navarro for approximate string matching, *Information Processing Letters*, vol. 108(5), pp. 313–319, 2008.

[15] R. Lepinsky, Analyzing Keywords in Enron's Email, *Rodger's Notes* (`www.rodgersnotes.wordpress.com/2013/11/24/analyzing-keywords-in-enrons-email`), 2013.

[16] V. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Soviet Physics Doklady*, vol. 10(8), pp. 707–710, 1966.

[17] D. Lillis and M. Scanlon, On the benefits of information retrieval and information extraction techniques applied to digital forensics, in *Advanced Multimedia and Ubiquitous Engineering*, J. Park, H. Jin, Y. Jeong and M. Khan (Eds.), Springer, Singapore, pp. 641–647, 2016.

[18] G. Navarro, NR-`grep`: A fast and flexible pattern-matching tool, *Software – Practice and Experience*, vol. 31(13), pp. 1265–1312, 2001.

[19] B. Oommen, Constrained string editing, *Information Sciences*, vol. 40(3), pp. 267–284, 1986.

[20] T. Rees, Taxamatch, an algorithm for near ('fuzzy') matching of scientific names in taxonomic databases, *PLoS ONE*, vol. 9(9), 2014.

[21] A. Shrestha Chitrakar and S. Petrovic, Approximate search with constraints on indels with application in spam filtering, *Proceedings of the Norwegian Information Security Conference*, pp. 22–33, 2015.

[22] A. Shrestha Chitrakar and S. Petrovic, Constrained row-based bit-parallel search in intrusion detection, *Proceedings of the Norwegian Information Security Conference*, pp. 68–79, 2016.

[23] T. Tracy, M. Stan, N. Brunelle, J. Wadden, K. Wang, K. Skadron and G. Robins, Nondeterministic finite automata in hardware – The case of the Levenshtein automaton, presented at the *Fifth Workshop on Architectures and Systems for Big Data*, 2015.

[24] Vound, Individual Solutions, Evergreen, Colorado (`www.vound-software.com/individual-solutions`), 2017.

[25] S. Wu and U. Manber, `agrep` – A fast approximate pattern-matching tool, *Proceedings of the USENIX Winter Technical Conference*, pp. 153–162, 1992.

[26] S. Wu and U. Manber, Fast text searching: Allowing errors, *Communications of the ACM*, vol. 35(10), pp. 83–91, 1992.