Chapter 1

OBTAINING VALUABLE PRECISION-RECALL TRADE-OFFS FOR FUZZY SEARCHING LARGE E-MAIL CORPORA

Kyle Porter and Slobodan Petrović

Abstract Fuzzy search is used in digital forensics to find words stringologically similar to a chosen keyword, but a common complaint is its high rate of false positives in big data environments. This work describes the design and implementation of *cedas*, a novel constrained edit distance approximate string matching algorithm which provides complete control over the type and number of elementary edit operations that are considered in an approximate match. The flexibility of this search algorithm is unique to *cedas*, and allows for fine-tuned control of precision-recall trade-offs. Specifically, searches can be constrained to a union of matches resulting from any exact edit operation combination of i insertions, e deletions, and s substitutions performed on the search term. By utilizing this flexibility, we experimentally show which edit operation constraints should be applied to achieve valuable precision-recall trade-offs for fuzzy searching an inverted index of a large English e-mail dataset by searching the Enron corpus. We identified which constraints produce relatively high combinations of precision and recall, the combinations of edit operations which cause precision to sharply drop, and the combination of edit operation constraints which maximize recall without greatly sacrificing precision. We claim these edit operation constraints are valuable for the middle stages of an investigation as precision has greater value in the early stages and recall becomes more valuable in the latter stages.

Keywords: E-mail forensics, fuzzy keyword search, edit distance, constraints, 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 support fuzzy search (or ap-

proximate string matching) algorithms to match keywords which have typographical errors or are stringologically similar to a keyword. These algorithms may be applied to approximately searching inverted indexes, where every approximate match is linked to a list of documents which contain it.

Great discretion must be used when utilizing these tools in exceedingly large datasets, as many strings that approximately match may be similar in a stringolocial sense, but are completely unrelated semantically. Even exact keyword matching produces an undesirable number of false positive documents to sift through, where maybe 80%-90% of the returned document hits are irrelevant [2]. Nevertheless, being capable of detecting slight textual aberrations has shown to be an important factor in past investigations. In the 2008 Casey Anthony case, in which she was convicted and ultimately acquitted of murdering her daughter, investigators missed a Google search for a misspelling of the word suffocation, written as "suffication" [1].

Current digital forensic tools such as dtSearch [7] and Intella [8] include methods for controlling the *fuzziness* of the search, and while these tools are proprietary technology, it appears that some of the tools utilize the edit distance [18] for their fuzzy searches to some extent. The edit distance, or Levenshtein distance, is defined as the minimum number of elementary edit operations to transform some string X into some string Y, where the elementary edit operations are defined as the insertion of a character, the deletion of a character, and substitution of a character in a string. However, precise control of the fuzziness of search is often limited, and it may not be clear what modifying the fuzziness of a search actually does other than the results "looking" more fuzzy. For instance, some tools may allow fuzziness to be assigned to a number between 0 to 10, without any indication of what these numbers represent.

This work has two contributions. The first of which describes the design and implementation of *cedas*, a novel Constrained Edit Distance Approximate Search algorithm which provides complete control over the type and number of elementary edit operations that are considered in an approximate match. The flexibility of this search algorithm is unique to *cedas*, and allows for fine-tuned control of precision-recall trade-offs. Specifically, searches can be constrained to a union of matches resulting from any exact edit operation combination of i insertions, e deletions, and s substitutions performed on the search term. Our second contribution, which is a consequence of the first, is that we experimentally show which edit operation constraints should be applied to achieve valuable precision-recall trade-offs for fuzzy searching an inverted index of a large English e-mail dataset by searching the Enron corpus [6]. We consider

valuable precision-recall trade-offs to be those with relatively high precision, since fuzzy searching typically has a high rate of false positives and increasing recall is simply performed by conducting fuzzy searches with higher edit distance thresholds. From our tests, we identified the constraints which produce relatively high combinations of precision and recall, the combinations of edit operations which cause precision to sharply drop, and the combination of edit operation constraints which maximize recall without greatly sacrificing precision. We claim these edit operation constraints are valuable for the middle stages of an investigation as precision has greater value in the early stages and further along an investigative timeline recall becomes more valuable [19].

2. Background Theory and Algorithm Implementation

2.1 Approximate String Matching Automata

A common method of performing approximate string matching, as is implemented by the popular approximate string matching suite *agrep* [24], is by using the Nondeterministic Finite Automaton (NFA) for approximate matching as seen in Figure 1. The automaton that *cedas* implements is an extension of this automaton, therefore we go over some necessary components of automata theory.

A finite automaton is a theoretical machine that uses characters from a string X as input, and can be used to determine whether the input contains a match for some desired string Y. Any finite automaton A can be defined with the following components. An automaton is made up of 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 states $I \subseteq Q$ make up the initial states that are *active* before reading the first character. States which are active check the transitions originating from themselves when a new character is being read, and if the transition includes the character being read then the state that the arrow is pointing to becomes active. The set of states $F \subseteq Q$ make up the terminal states, where if any of these states are made active then it signals that a match has occurred. The set of strings that cause a match are considered to be accepted by the automaton, and this set is denoted as some language L.

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 represented by the node with a bolded arrow pointing to it, where it is always active as indicated by the self-loop, and the terminal states are

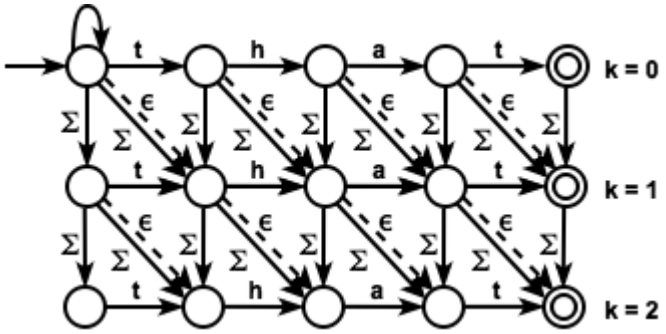

Figure 1. NFA A_L that matches the trivial pattern "that" allowing for two edit operations.

represented by the nodes which are double-circled. Horizontal arrows represent exact character matches. Diagonal arrows represent character substitutions, and vertical arrows represent character insertions where both transitions accept any character in Σ. Since this is an NFA, it also allows for ϵ-transitions, where transitions are made without needing a prerequisite character. The dashed diagonal arrows are ϵ-transitions, which represent character deletions. For approximate search with an edit distance threshold of k, this automaton has $k + 1$ rows.

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

In practice, 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 for common English text [12]. This suggestion is well founded, as it has long been known that about 80% of misspellings in English text are due to a single edit operation [10]. Let the languages accepted by the automaton A_L with threshold $k = 1$ and $k = 2$ be $L_{k=1}$ and $L_{k=2}$ respectively. The nondeterministic finite automaton A_T provided in this work allows for different degrees of fuzziness that allow the user explore the entire space between $L_{k=1}$ and $L_{k=2}$ in terms of exact combinations of elementary edit operations applied to the search keyword. The languages this automaton accepts we formally define as L_T, for which $L_{k=1} \subseteq L_T \subseteq L_{k=2}$.

The construction of this type of automaton is made possible in the following way. The automaton accepting $L_{k=2}$ can be seen as a union of the languages accepted by each one of its rows. For example, for an edit

distance threshold of $k = 2$, the first row accepts the language of matches that has 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 $L_{k=2}$. We can describe an alternative cover of $L_{k=2}$ as a 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 $i + e + s \leq k$. We prove the equivalence of the covers below.

LEMMA 1 *Let L_k be the language accepted by an automaton such that k elementary edit operations are performed on a specified pattern and let the language accepted by the nondeterministic finite automaton for approximate matching be denoted $L_{k=n}$ 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, and 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 some $L_{(i,e,s)}$ 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 $C_\beta \subset C_\alpha$.

Q.E.D.

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 rather than each row corresponding to some number of edit operations. By this construction, we then control the automaton's accepted language L_T by allowing terminal states $f \in F$ to remain in F or removing them from F. That is, we can choose not to include some $L_{(i,e,s)}$ in $C_\beta = \cup_{i,e,s:0\leq i+e+s\leq n} L_{(i,e,s)}$.

2.2 Defining the NFA for *cedas*

The constrained edit distance between strings X and Y is the minimum number of edit operations to transform X to Y given that the transformation obeys some prespecified constraints T [21]. In general, constraints may be defined arbitrarily as long as they are defined in the number and type of edit operations. Let $(i, e, s) \in T$, where T is the set of edit operations a transformation from string X to string Y is constrained to, and where elements (i, e, s) are defined by some exact combination of edit operations. A_T may perform approximate searches

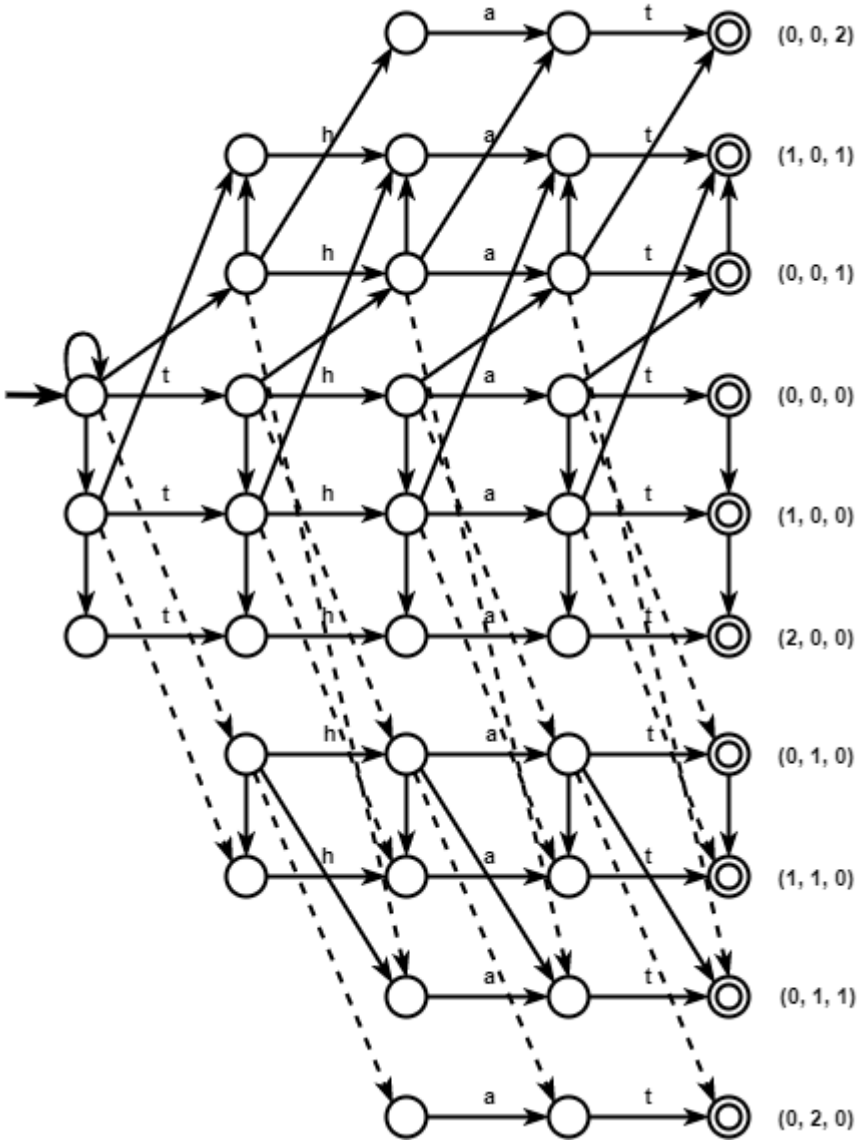

Figure 2. A_T Automaton for the word "that".

where matches are constrained to the allowed edit operation combinations in T. For instance, we may constrain our search to approximate matches that were derived from edit operation combinations $(0, 0, 0)$, $(1, 0, 1)$, and $(0, 2, 0)$ only. This corresponds to accepting the language $L_{(0,0,0)} \cup L_{(1,0,1)} \cup L_{(0,2,0)}$.

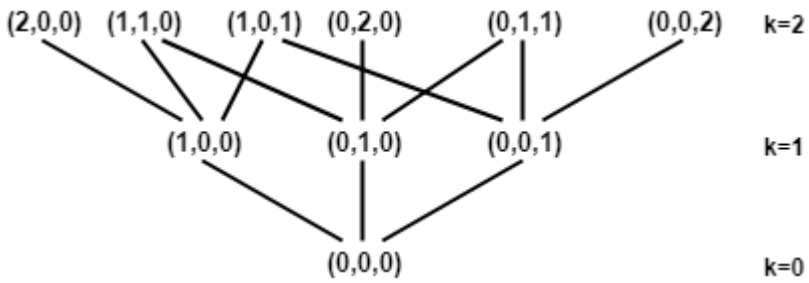

Figure 3. Diagram D for the partially ordered multisets (i, e, s)

Figure 2 shows the constrained edit distance NFA A_T. It uses the same symbolic conventions as the NFA from Figure 1 except now substitutions and insertions are represented 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 accepting the language $L_{(i,e,s)}$ we examine the diagram in Figure 3 of a partially ordered set of multisets, which describes the edit operation transpositions connecting each row. We first need the following definitions.

DEFINITION 2 *[14] If X is a set of elements, a multiset M drawn from the set X is represented by a function count M or C_M defined as $C_M : X \to N$ where N represents the set of non-negative integers.*

For each $x \in X$, $C_M(x)$ is the characteristic value of x in M and it indicates the number of occurences of the elements x in M. A multiset M is a set if $C_M(x) = 0$ or 1 for all $x \in X$.

DEFINITION 3 *[14] Let M_1 and M_2 be two multisets selected from a set X, then M_1 is a sub multiset 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 sub multiset 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 we examine is composed of elements (i, e, s), where the variables denote that the multiset contains i insertions, e deletions, and s substitutions and furthermore the cardinality of the multisets is less than or equal to 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, $M_1 \sim M_2$ defines M_1 is related to M_2 by the fact that $M_1 \subset M_2$. The partially ordered multiset diagram in Figure 3, which we refer to D, models the edit operation transitions between rows

Table 1. Bit-masks for the word "that".

Character t_j	**Bit-mask** $B[t_j]$
a	01000
h	00100
t	10010
*	00000

of 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 from D, every $R_{(i,e,s)}$ has a specific edit operation transition being sent to it from a specific row. In this way, each row $R_{(i,e,s)}$ is able to determine which and how 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).

Since this automaton is nondeterministic, it cannot be implemented directly in von Neumann architecture.

2.3 Bit-parallel Implementation of A_T

Bit-parallelism allows for an efficient form of NFA simulation. This method uses bit-vectors to represent each row of the automaton, where the vectors are updated via basic logical bitwise operations which correspond to the automaton's transition relations. As bitwise operations update every bit in the bit-vector simultaneously, it is likewise updating the states in the row of an automaton simultaneously. If the lengths of the bit-vectors are less than or equal to the number of bits w in a computer word, then this parallelism reduces the number of operations a search algorithm performs at most by w [13].

For our bit-parallel NFA simulation, we require that each row of the automaton for search pattern X is represented as a binary vector of length $|X| + 1$, and this also requires that our input characters are represented by vectors of the same size. Input characters t_j are handled by *bit-masks*, where we create a table of bitmasks $B[t_j]$, and each bit-mask represents the positions of the character t_j in the pattern X. For example, when $X =$ "that", the bit-masks are shown in Table 1. Characters which are not present in the pattern are represented by the * symbol.

Our bit-parallel algorithm for simulating the A_T automaton, seen in Algorithm 1, is an extension of the simulation of the NFA for approx-

imate string matching using the unconstrained edit distance, first implemented by Wu and Manber [25]. Therefore, the components of the simulation of A_T are similar, the primary changes being the transition relationships between rows and the number of rows. The rows of the automaton are represented by bit vectors $R_{(i,e,s)}$ and their updated values are denoted by $R'_{(i,e,s)}$. $R^B_{(i,e,s)}$ represents Boolean values for whether or not row $R_{(i,e,s)}$ will report a match, which simulates if $R_{(i,e,s)}$ has a terminal state. For some i, e, and s, the value $R^B_{(i,e,s)}$ is true if $(i, e, s) \in T$.

Each row is updated by first checking if the input character is an exact match for that row by calculating $((R'_{(i,e,s)} << 1)\ \&\ B[t])$. This value is then bitwise OR'd 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 some row $R_{(i,e,s)}$. As mentioned previously, incoming transitions for any row $R_{(i,e,s)}$ may be determined by checking against the incoming relations to the multiset (i, e, s) of diagram D from Figure 3. After updating all rows, matches are checked by bitwise AND'ing each of the allowed rows and determining if any bit representing the terminal states is set to 1.

2.4 Complexity Performance Evaluation and Limitations

The additional flexibility in specifying fuzziness comes at a cost in terms of speed and space. The time and space complexities of *cedas* can be defined in the number of multisets in D for an edit distance threshold k. This number is equal to the number of rows in our algorithm, 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, its 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 this algorithm should not be applied to extremely 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 approximate search over an index of e-mails, as the values of k should be low and the index has acted as a type of data reduction. We show in the experimentation that our algorithm runs approximately six times slower than *agrep* with an edit distance threshold of $k = 2$.

Algorithm 1: *cedas* NFA update algorithm

Initialize all rows R' to 0 except $R'(0,0,0) \leftarrow 0^{|p|}1$;
for *each input character* t **do**
  $R_{(0,0,0)} \leftarrow R'_{(0,0,0)}$;
  $R'_{(0,0,0)} \leftarrow ((R'_{(0,0,0)} << 1)\ \&\ B[t]) \mid 0^{|p|}1$;
  $R_{(1,0,0)} \leftarrow R'_{(1,0,0)}$;
  $R'_{(1,0,0)} \leftarrow ((R'_{(1,0,0)} << 1)\ \&\ B[t]]) \mid R_{(0,0,0)}$;
  $R_{(0,1,0)} \leftarrow R'_{(0,1,0)}$;
  $R'_{(0,1,0)} \leftarrow ((R'_{(0,1,0)} << 1)\ \&\ B[t]]) \mid (R'_{(0,0,0)} << 1)$;
  $R_{(0,0,1)} \leftarrow R'_{(0,0,1)}$;
  $R'_{(0,0,1)} \leftarrow ((R'_{(0,0,1)} << 1)\ \&\ B[t]]) \mid (R_{(0,0,0)} << 1)$;
  $R_{(0,1,1)} \leftarrow R'_{(0,1,1)}$;
  $R'_{(0,1,1)} \leftarrow ((R'_{(0,1,1)} << 1)\ \&\ B[t]) \mid (R_{(0,1,0)} << 1) \mid (R'_{(0,0,1)} << 1)$;
  $R_{(1,0,1)} \leftarrow R'_{(1,0,1)}$;
  $R'_{(1,0,1)} \leftarrow ((R'_{(1,0,1)} << 1)\ \&\ B[t]) \mid (R_{(1,0,0)} << 1) \mid R_{(0,0,1)}$;
  $R_{(1,1,0)} \leftarrow R'_{(1,1,0)}$;
  $R'_{(1,1,0)} \leftarrow ((R'_{(1,1,0)} << 1)\ \&\ B[t]) \mid R_{(0,1,0)} \mid (R'_{(1,0,0)} << 1)$;
  $R_{(2,0,0)} \leftarrow R'_{(2,0,0)}$;
  $R'_{(2,0,0)} \leftarrow ((R'_{(2,0,0)} << 1)\ \&\ B[t]) \mid R_{(1,0,0)}$;
  $R_{(0,2,0)} \leftarrow R'_{(0,2,0)}$;
  $R'_{(0,2,0)} \leftarrow ((R'_{(0,2,0)} << 1)\ \&\ B[t]) \mid (R'_{(0,1,0)} << 1)$;
  $R_{(0,0,2)} \leftarrow R'_{(0,0,2)}$;
  $R'_{(0,0,2)} \leftarrow ((R'_{(0,0,2)} << 1)\ \&\ B[t]) \mid (R_{(0,0,1)} << 1)$;
  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

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

3. Experiment Methodology

We tested *cedas*'s flexibility in specifying fuzziness in the context of an investigation into the Enron e-mail dataset [6], and evaluate the effectiveness of different constraints. We converted the e-mails to their mbox format to make processing them easier, and only examined their body content.

To search the data, we created an inverted index of the e-mails to obtain a database of index tokens using the *mkid*[1] program. After deduplicating the tokens in the index, we output all tokens into a single text file which we searched with *cedas*. It is important to note that the choice of indexing algorithm will affect the list of tokens since they may interpret delimiters differently, and therefore will affect any search. Our list of tokens amounted to 460800 different words.

We used 28 different keywords related to the 2001 Enron scandal. This list of keywords was not compiled by a digital forensic investigator, so the choice of keywords could still be improved. Keywords were chosen that were relevant to the case, but would not obviously produce an overwhelming number of false positives. Furthermore, no keyword was chosen that contained less than six characters. Unconstrained fuzzy searching with an edit distance threshold of $k = 2$ for small keywords produced massive lists of words to manually analyze, oftentimes exceeding 10000 words. This can be seen as a limitation of our research. The list of search keywords may be found in Table 2 (many of which were obtained from Rodger Lepinsky's webpage on data science and the Enron corpus [17]).

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* occurs if a keyword is found as a substring of an index token with an allowed tolerance of edit operations as specified

by the constraints defined in terms of $(i, e, s) \in T$. All approximate matches found in the index were returned in a list to the user. 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 accepts $L_{(0,0,0)}, L_{(1,0,0)}, L_{(0,1,0)}, L_{(0,0,1)}$, but we potentially allow the inclusion of all possible combinations of languages $L_{(i,e,s)}$ such that $i + e + s = 2$. The effectiveness of a search for each constraint on each keyword was measured in terms of precision and recall derived from the list of returned approximate matches. To understand the overall effectiveness of each constraint, we took the average precision and recall results for all keywords under each constraint using the harmonic mean. The harmonic mean was chosen as the arithmetic mean tended to produce overly optimistic results for what one should expect for fuzzy searches under the various chosen constraints.

3.1 Interpreting Match Results

Precision and recall have been used to gauge the effectiveness of approximate string matching algorithms before [3] [22]. Precision is defined as the proportion of retrieved items that are relevant, and recall is the proportion of total relevant items retrieved [15]. As recall increases, precision tends to decrease, and they may be expressed as follows:

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

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

These statistics are useful, but not perfect since the concept of relevance is subjective. We define relevant terms as being either variations of the original term (such as obtaining litigating when searching for litigation), closely related to the original term in a semantic sense (such as obtaining legalese when searching for illegal), or misspellings of the original term. However, if the keyword is a substring in the examined index token and is clearly unrelated, then it is not considered relevant. For example, if the search term is "audit" and we get a hit for "AudiTalk", then that hit is irrelevant. It was for this reason that the classification of relevant versus irrelevant hits for approximate hits was a manual process.

One more shortcoming of our statistics is that we cannot exactly calculate the true precision and recall for each keyword since the number of relevant words for each keyword in the Enron dataset is simply unknown. Our compromise is the following: we let the number of total relevant items be equal to the ones we identified for each keyword for

Table 3. Shorthand for which $(i, e, s) \in T$.

(i,e,s)	**Label**
$(1, 1, 0)$	ie
$(1, 0, 1)$	is
$(0, 1, 1)$	es
$(2, 0, 0)$	ii
$(0, 2, 0)$	ee
$(0, 0, 2)$	ss

unconstrained approximate matching at edit distance threshold $k = 2$. This means that for our calculations, unconstrained approximate matching with $k = 2$ produces 100% recall, which is not necessarily true.

Lastly, it is important to note that approximate string matching results are highly dependent on the specific data being matched and the keywords being used [9]. Therefore, utilizing *cedas* on an inverted index which was not derived from an English e-mail corpus may produce different results.

4. Experimental Results

4.1 Precision and Recall

The results in this section reflect how effective each set of constraints T was for search in terms of precision and recall, and furthermore which constraints produce valuable results for investigating an English e-mail corpus. Figure 4 shows the precision-recall trade-off curve derived from our experiments. Data points labeled $k = 1$ and $k = 2$ represent the results for unconstrained fuzzy searching with edit distance thresholds set 1 and 2 respectively. Table 3 provides further notation for interpreting results where the labels in the right column represent if some $(i, e, s) \in T$. All data points with the labels shown in the table assume that $(0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1) \in T$.

As was to be expected, the application of constraints for fuzzy searching the Enron inverted index obtained a combination of higher recall than an unconstrained fuzzy search with an edit distance threshold at $k = 1$ and better precision than an unconstrained fuzzy search with an edit distance threshold at $k = 2$. However, we are primarily interested in precision-recall trade-offs which are useful in an investigation. As mentioned in the introduction, a common complaint is the number of false positives that fuzzy searching produces, and also that early on in an investigation precision is valued more than recall [19]. This implies that

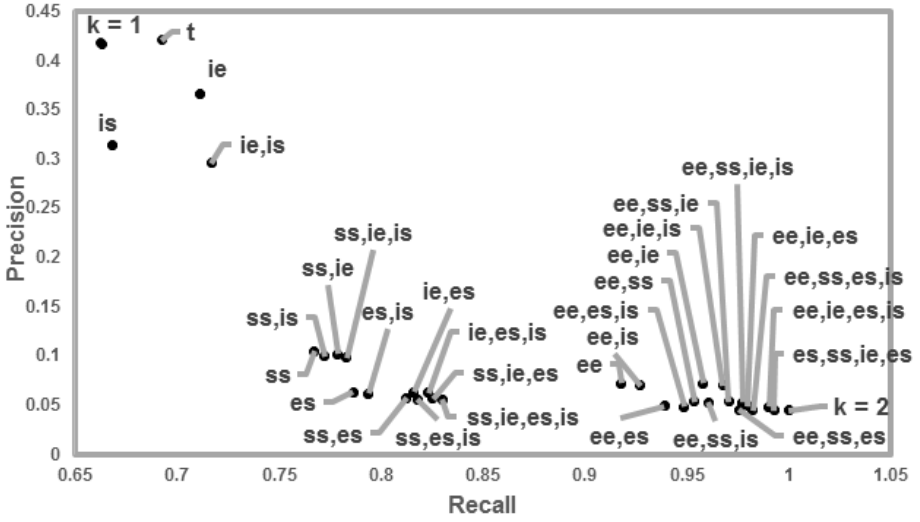

Figure 4. Precision-recall tradeoff curve applying various constraints

the constraints that produced results with relatively high precision are most useful for an investigation, as increasing recall is easily performed 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 is made up of 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 to determine an approximate match. 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 $k = 1$, one would constrain their fuzzy searches to edit operations which do not include the previously mentioned edit operation combinations. Furthermore, the data points in the cluster mostly show a marked improvement of recall. We claim the constraints in this cluster are useful for the middle stages of an investigation, as they have relatively high precision and recall.

Data points in this cluster that include a single insertion and deletion (ie) have very beneficial precision-recall tradeoffs. To ensure that the fuzzy search with these constraints was simply not due to transposition edit operations, for which adjacent characters may be swapped, we per-

Table 4. Run Time for *cedas* compared to *agrep*.

Average Runtime agrep	**Average Runtime cedas**
0.0477142857	0.2795714286

formed the same tests but used the *nrgrep* [20] 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 we can see that ie and t do not correspond to the same results.

4.2 Run-Time

To evaluate the speed of *cedas*, we timed the unconstrained fuzzy search with an edit distance threshold of $k = 2$ for every keyword on the Enron inverted index. The average of these results were taken and compared to the results of the same tests using *agrep*.

From the results in Table 4 , it can be seen that *cedas* runs nearly six times slower than agrep at edit distance threshold $k = 2$. It is important to note that the *cedas* implementation has not been optimized, so therefore it has a potential to become faster. For comparisons between *cedas* and *agrep* regarding flexibility of applying constraints see Section 5.

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. To be certain of this would require additional testing, but one can see that it is quite easy to transform words into other words by using many substitutions or deletions. By limiting the application of deletion and substitution edit operations we preserve much of the structure of the original word. For this reason, if somewhat high precision is needed, we would suggest using a fuzzy search that does not include the edit operations of two deletions, two substitutions, or a single deletion and single substitution.

To maximize the recall for the fuzzy searching results without sacrificing precision as seen by applying many of the edit operation combinations, we would suggest 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), where for the sake of brevity we refer to the language accepted by this automaton to be $L_{-(ee,ss,es)}$. The precision and recall of this language corresponds to data point ie,is. If more precision is needed, with nearly as much recall, 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 corresponds to data point ie). Ultimately, users of *cedas* should choose constraints accordingly for precision-recall trade-offs they are willing to tolerate.

5. Related Work

Applications of fuzzy search algorithms to digital forensic can be found in such products as dtSearch [7] and Intella [8]. In these tools, the variables for setting the tolerated fuzziness do not always directly correlate with the edit distance threshold. Whether or not these algorithms used a constrained edit distance algorithm to achieve these results is unknown since the technology is proprietary, and could possibly be combining the edit distance measure with other types of distance measures, natural language processing, or information retrieval techniques.

The open source tools which have been used for fuzzy searching in digital forensics are *agrep* [24] and *nrgrep* [20]. These tools are suites of approximate matching algorithms, which do not only include edit distance based approximate matching, but also prefix matching, regular expression matching, amongst other options. They can be considered as the cutting edge of bit-parallel NFA implementations for approximate matching in terms of speed and utility. *cedas*'s advantage over their edit distance matching algorithms is its flexibility in constraining edit operations. *agrep* cannot implement any specific type of constraints, therefore specifying its fuzziness in terms of edit distance operations is limited to setting the edit distance threshold. *nrgrep* is more flexible in that it allows the user to not only set an edit distance threshold, but also allows for transpositions, and lets the user define a subset of the edit operations the search will permit. This last factor can be seen as an application of constraining edit operations, which produces a subset of the possible edit operation constraints as *cedas*. For instance, this type of matching cannot return matching results equivalent to $L_{-(ee,ss,es)}$.

Other constrained edit distance search algorithms exist as well. Chitrakar and Petrović have produced multiple constrained edit distance search algorithms utilizing row-based bit-parallelism. The first described edit operation constraints in terms of the maximum number of allowed *indels* [4], the sum insertions and deletions, and the second algorithm described edit operation constraints in terms of the maximum allowed number of specific insertions, deletions, and substitutions permitted in a match [5]. These algorithms also produce a subset of possible constraints that *cedas* can produce, where constraints cannot be specified to return the language $L_{-(ee,ss,es)}$. Their experiments show these algorithms to be nearly as fast as *agrep*.

6. Conclusion

In this work we presented our novel algorithm *cedas*, a constrained edit distance fuzzy search algorithm for which we mathematically showed can perform approximate search where possible transformations on the search term are constrained to any set of edit operation combinations in the form of exactly i insertions, e deletions, s substitutions. The algorithm is a bit-parallel simulation of a nondeterministic finite automaton, in which the rows of the automaton are not defined by the number of elementary edit operations considered, but by both the number and types of edit operations. This flexibility in defining edit operation constraints to approximate search is unique to *cedas*.

Using this algorithm, we performed constrained edit distance fuzzy searches for a list of keywords in an inverted index of the Enron e-mail corpus where approximate matches were returned. The average precision and recall resulting from searches applying each edit operation combination constraint showed which constraints were the most valuable for fuzzy searching an English e-mail dataset. As a common complaint of fuzzy searching is the number of false positives it produces, we considered edit operation constraints that allowed for high precision to valuable.

From our experiments, we found that to avoid the drop in precision which is commonly perceived in unconstrained fuzzy searching at an edit distance threshold of k = 2, one should constrain the fuzzy search to not include any matches in which two deletions, two substitutions, or a substitution and deletion have been performed on the approximate match of a keyword. 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 from fuzzy searching at an unconstrained edit distance threshold of $k = 1$ while also improving recall. To maximize the recall of fuzzy searching without experiencing a significant reduction in precision, for the combination of edit operations (i insertions, e deletions, s substitutions), the fuzzy search 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 for the middle stages of an investigation as precision has greater value in the early stages and recall becomes more valuable further along an investigative timeline [19].

The flexibility of *cedas* comes at a cost. Its worst-case space complexity is cubic in k and worst-case time complexity is $O(k^3n)$ 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 our experimentation, we found that the average time it took 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 using *agrep* to about 0.2796 seconds using *cedas*. It is important to note that the implementation of *cedas* has not been optimized, and ways to reduce the space and time consumption would be to dynamically generate the rows of the automaton which were necessary as opposed to simply removing terminal state from specific rows.

Alternatively, if the automaton *cedas* simulates was implemented into hardware architecture such as the Automata Processor [11], which can directly implement NFA's, it could potentially run in linear time. Tracy et. al. [23] found that the NFA for approximate matching (Figure 1) can run in worst-case linear time on the Automata Processor where the hardware could maximally contain an NFA with a search pattern length of 2730 characters with an edit distance threshold of 4. The fastest bit-parallel NFA simulations of the same type of automaton require that the search pattern lengths do not exceed lengths of about thirty characters to preserve optimal results with a worst-case time complexity of $O(\lceil (m-k)(k+1)/w \rceil n)$ [16], where m is the length of the search pattern, k is the edit distance threshold, and n is the length of the input.

Other improvements that can be made to *cedas*, which can be found in other search tools, are to allow for prefix matching or character specific fuzziness.

Acknowledgments

The research leading to these results has received funding from the Research Council of Norway program IKTPLUSS, under the R&D project "Ars Forensica - Computational Forensics for Large-scale Fraud Detection, Crime Investigation & Prevention", grant agreement 248094/O70.

Notes

1. http://www.gnu.org/software/idutils/manual/idutils.html

References

[1] Associated Press, Casey Anthony detectives missed 'suffocation' search, 2012 (`https://www.usatoday.com/story/news/nation/2012/11/25/casey-anthony-suffocation-google/1725253/`).

[2] N.L. Beebe and J.G. Clark, Digital Forensic Text String Searching: Improving Information Retrieval Effectiveness by Thematically Clustering Search Results, *Digital Investigation*, vol. 4, pp. 49–54,

2007.

[3] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar and S. Fienberg, Adaptive Name Matching in Information Integration, *IEEE Intelligent Systems*, vol. 18, no. 5, pp. 16–23, 2003.

[4] A.S. Chitrakar and S. Petrović, Approximate Search with Constraints on Indels with Application in SPAM Filtering, *Norsk Informasjonssikkerhetskonferanse*, pp. 22–33, 2015.

[5] A.S. Chitrakar and S. Petrović, Constrained Row-Based Bit-Parallel Search in Intrusion Detection, *Norsk Informasjonssikkerhetskonferanse*, pp. 68–79, 2016.

[6] W. Cohen, Enron Email Corpus, 2015 (`https://www.cs.cmu.edu/~./enron/`).

[7] dtSearch, Search features - search types (`https://www.dtsearch.com/PLF_Features_2.html`).

[8] Intella, Individual solutions, 2017 (`https://www.vound-software.com/individual-solutions`).

[9] R. Da Silva, R. Stasiu, V.M. Orengo and C.A. Heuser, Measuring Quality of Similarity Functions in Approximate Data Matching, *Journal of Informetrics*, vol. 1, no. 1, pp.35–46, 2007.

[10] F.J. Damerau, A Technique for Computer Detection and Correction of Spelling Errors, *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, 1964.

[11] 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, no. 12, pp. 3088–3098, 2014.

[12] Elastic, Elasticsearch reference: Fuzzy query, 2017 (`https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-fuzzy-query.html`)

[13] S. Faro and T. Lecroq, Twenty Years of Bit-Parallelism in String Matching, *Festschrift for Borivoj Melichar*, pp. 72–101, 2012.

[14] K.P. Girish and J.J. Sunil, General Relations Between Partially Ordered Multisets and Their Chains and Antichains, *Mathematical Communications*, vol. 14, no. 2, pp. 193–205, 2009.

[15] P.A. Hall and G.R. Dowling, Approximate String Matching, *ACM Computing Surveys*, vol. 12, no. 4, pp. 381–402, 1980.

[16] H. Hyyrö, Improving the Bit-Parallel NFA of Baeza-Yates and Navarro for Approximate String Matching, *Information Processing Letters*, vol. 108, no. 5, pp 313–319, 2008.

[17] R. Lepinsky, Analyzing Keywords in Enron's Email, 2013 (`https://rodgersnotes.wordpress.com/2013/11/24/analyzing-keywords-in-enrons-email/`).

[18] V.I. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions, and Reversals, *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[19] D. Lillis and M. Scanlon, On the Benefits of Information Retrieval and Information Extraction Techniques Applied to Digital Forensics, *Advanced Multimedia and Ubiquitous Engineering*, Springer Singapore, pp. 641–647, 2016.

[20] G. Navarro, NR-grep: A Fast and Flexible Pattern-Matching Tool, *Software: Practice and Experience*, vol. 31, no. 13, pp. 1265–1312, 2001.

[21] B.J. Oommen, Constrained String Editing, *Information Sciences*, vol. 40, no. 3, 267–284, 1986.

[22] T. Rees, Taxamatch, an Algorithm for Near ('Fuzzy') Matching of Scientific Names in Taxonomic Databases, *PLoS One*, vol. 9, no. 9, 2014 (`https://doi.org/10.1371/journal.pone.0107510`).

[23] T. Tracy II, M. Stan, N. Brunelle, J. Wadden, K. Wang, K. Skadron and G. Robins, Nondeterministic Finite Automata in Hardware - the Case of the Levenshtein Automaton, *Architectures and Systems for Big Data, in conjunction with ISCA*, 2015.

[24] S. Wu and U. Manber, Agrep - a Fast Approximate Pattern-Matching Tool, *Usenix Winter 1992 Technical Conference*, pp. 153–162, 1992.

[25] S. Wu and U. Manber, Fast Text Searching: Allowing Errors, *Communications of the ACM*, vol. 35, no. 10, pp. 83-91, 1992.