Efficient Techniques for Document Sanitization

Venkatesan T. Chakaravarthy
IBM India Research Lab
vechakra@in.ibm.com

Himanshu Gupta
IBM India Research Lab
higupta8@in.ibm.com

Prasan Roy
Aster Data Systems, USA
prasan.roy@asterdata.com

Mukesh K. Mohania
IBM India Research Lab
mkmukesh@in.ibm.com

ABSTRACT

Sanitization of a document involves removing sensitive information from the document, so that it may be distributed to a broader audience. Such sanitization is needed while declassifying documents involving sensitive or confidential information such as corporate emails, intelligence reports, medical records, etc. In this paper, we present the ERASE framework for performing document sanitization in an automated manner. ERASE can be used to sanitize a document dynamically, so that different users get different views of the same document based on what they are authorized to know. We formalize the problem and present algorithms used in ERASE for finding the appropriate terms to remove from the document. Our preliminary experimental study demonstrates the efficiency and efficacy of the proposed algorithms.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Unauthorized access

General Terms

Algorithms, Performance, Security

Keywords

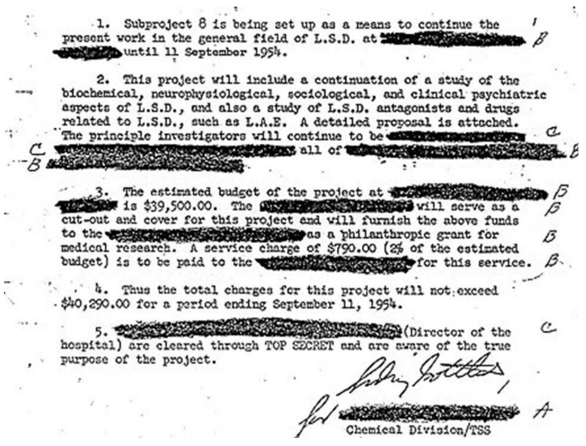
Sanitization, Redaction, Anonymization

1. INTRODUCTION

Sanitization (syn. *redaction*) of a document involves removing sensitive information from the document, in order to reduce the document's classification level, possibly yielding an unclassified document [16].

A document may need to be sanitized for a variety of reasons. Government departments usually need to declassify documents before making them public, for instance, in response to Freedom of Information requests. In hospitals, medical records are sanitized to remove sensitive patent information (patient identity information, diagnoses of deadly diseases, etc.). Document sanitization is also critical to companies who need to prevent malafide or inadvertent disclosure of proprietary information while sharing data with outsourced operations.

Example. Figure 1 shows an example U.S. government document that has been sanitized prior to release [16]. This sanitized document gives limited information (such as the purpose and the funding amount) on an erstwhile secret medical research project, while hiding the names of the funding sources, principal investigators and their affiliation.



1. Subproject 8 is being set up as a means to continue the present work in the general field of L.S.D. at [redacted] until 11 September 1954.

2. This project will include a continuation of a study of the biochemical, neurophysiological, sociological, and clinical psychiatric aspects of L.S.D., and also a study of L.S.D. antagonists and drugs related to L.S.D., such as L.A.E. A detailed proposal is attached. The principle investigators will continue to be [redacted] all of [redacted].

3. The estimated budget of the project at [redacted] is $39,500.00. The [redacted] will serve as a cut-out and cover for this project and will furnish the above funds to the [redacted] as a philanthropic grant for medical research. A service charge of $790.00 (2% of the estimated budget) is to be paid to the [redacted] for this service.

4. Thus the total charges for this project will not exceed $40,290.00 for a period ending September 11, 1954.

5. [redacted] (Director of the hospital) are cleared through TOP SECRET and are aware of the true purpose of the project.

[illegible]

for [redacted]
Chemical Division/TSS

Figure 1: A sanitized document

Traditionally, documents are sanitized manually by qualified reviewers. However, manual sanitization does not scale as the volume of data increases. The US Department of Energy's OpenNet initiative [15], for instance, needs to sanitize millions of documents each year. Given the amount of effort involved and limited supply of qualified reviewers, this is a tall order.

In this paper, we propose ERASE,[1] a system for performing document sanitization automatically. Highlights of the system include:

- ERASE can sanitize documents efficiently, making it possible to scale up to large data volumes. The low sanitization latency also enables processing of emails and dynamic webpages *on-the-fly*.

- The extent of sanitization performed by ERASE can be configured to adapt to the access privileges of an intended reader, so that different users can get different sanitized versions of the document based on their current authorization status.

[1]Efficient RedAction for Securing Entities

- ERASE can be configured to integrate seamlessly with the existing authentication mechanisms (database access-control lists) in an organization to determine the access privileges of an user.

- ERASE makes an effort to sanitize a document while causing the *least distortion* to the contents of the document. Minimal content distortion has been considered as one of the principal requirements of document sanitization [14].

Contributions. Our specific contributions in this paper are as follows.

- We present a principled approach to sanitization of unstructured text documents. While sanitization of structured relational databases has been addressed earlier [13, 8], we believe ERASE is the first work to provide a principled solution in an unstructured free-text domain.

- We device an algorithm that sanitizes a document while removing the minimum number of terms. We propose nontrivial pruning strategies that make the search practical.

- We present an alternative algorithm that achieves very reasonable performance even for large documents; the improvement in efficiency is achieved by relaxing the least-distortion property. We show that this alternative approach, in practice, is similar to the optimal algorithm in terms of the quality of the result.

- We present a preliminary experimental study that shows that the proposed techniques are practical on realistic data.

Overview. ERASE models public knowledge as a database of entities (persons, products, diseases, etc.). Each entity in this database is associated with a set of terms related to the the entity; this set is termed the *context* of the entity. For instance, the context of a person entity could include the firstname, the lastname, the day, month and year of birth, the street and city of residence, the employer's name, spouse's name, etc. This database could be structured or unstructured – wikipedia, a directory of employees and projects in an organization, a compendium of diseases, etc. all fit in the proposed notion of an entity database. The only requirement is that the database is able to provide the context of a given entity, and the set of entities containing a given term in their context. This context database need not be compiled manually – it can be a existing database, or can be extracted automatically using an information extraction system such as Snowball [1].

Some of the entities in the database are considered *protected*; these are the entities that need to be protected against identity disclosure. For instance, in a database of diseases, certain diseases (such as AIDS) can be marked as protected – we are interested in protecting the disclosure of these diseases, it does not matter if the any other disease (such as Influenza) is revealed. The set of protected entities is derived according to the access privileges of the adversary. The set of entities that need to be hidden from the adversary are declared protected.

ERASE assumes an adversary that knows nothing about an entity apart from what appears in the entity's context, and has bounded inference capabilities. Specifically, given a document, the adversary can match the terms present in the document with the terms present in the context of each of the protected entities. If the document contains a group of terms that appear together only in the context of a particular entity, then the adversary gets an indication that the entity is being mentioned in the given document. We term this a *disclosure*.[2]

ERASE attempts to prevent disclosure of protected entities by removing certain terms from the document – these terms need to be selected such that no protected entity can be inferred as being mentioned in the document by matching the remaining terms with the entity database.[3]

A simplistic approach is to locate "give-away" phrases in the document and delete them all. To the best of our knowledge, most prior work on document sanitization has followed this approach, and has focused on developing more accurate ways of locating such phrases in the document's text [12, 6, 14]. We believe this is an overkill. For instance, in an intelligence report, removing all names, locations, etc. would probably leave the report with no useful content at all.

In contrast, ERASE makes an effort to sanitize a document while causing the *least distortion* to the contents of the document; indeed, this is considered one of the principal requirements of document sanitization [14]. Towards this goal, ERASE identifies the *minimum* number of terms in the document that need to be removed in order for the document to be sanitized.

Illustrative Example. We illustrate our approach using an anecdotal real-life example. We created a database of 2645 diseases obtaining information from the website *wrongdiagnosis*.[4] Each disease is an entity with the associated context consisting of symptoms, tests, treatments and risk factors. The website offers a classification of the diseases. We declared as protected entities the diseases under the following categories: sexual conditions, serious conditions (conditions related to heart, thyroid, kidney, liver, ovary), cancer conditions, and mental conditions. The number of protected entities were 550. The privacy parameter K was set to 10 (see Section 3 for the semantics of this parameter).

We created a document by taking a paragraph out of a communication between a doctor and a patient from another website.[5] The document is shown in Figure 2, where the relevant terms found in the entity contexts are shown in bold.

" Let's look at the immediate facts. You have a number of symptoms, namely **weight loss**, **insomnia**, **sweating**, **fatigue**, **digestive problems** *and* **headaches**. *These may or may not be related to* **sexually transmitted diseases**, *but you know you have been exposed to* **gonorrhoea** *and you know you may have been exposed to* **hepatitis B** *and* **HIV**. *Your symptoms are significant and need full investigation in the near future. "*

Figure 2: Illustrative Example

The document was sanitized using ERASE; the terms deleted as a result are shown underlined in Figure 2. Observe that the sanitization has differentiated between the generic symptoms and symptoms specific to the kind of diseases that appear in the protected entity set, and deleted the latter. Furthermore, though sweating is a common symptom associated with hundreds of diseases, the combination of sweating, weight loss and fatigue reveals a protected en-

[2]Adversaries such as automated *named-entity taggers* [5, 4] and keyword-based search engines fit the assumptions well. Our future work involves considering richer adversaries [9].

[3]Note that the adversary does not know which entities in the database are protected, otherwise the adversary can focus only on the protected entities whenever sanitization is present.

[4]www.wrongdiagnosis.com

[5]www.netdoctor.co.uk

tity (in this case, HIV) and so, one of these terms must be deleted to sanitize the document; consequently, the system removed the term "sweating". We note that a different protected entity set might remove a different set of symptoms.

Organization. The rest of the paper is organized as follows. We discuss the related work in Section 2. The document sanitization is formalized in Section 3, and its complexity is discussed in Section 4. Next, in Section 5, we discuss algorithms for computing optimal solutions. Section 6 presents efficient greedy heuristics. This is followed by a preliminary experimental study in Section 7. Finally, in Section 8, we present our conclusions and directions for future work.

2. RELATED WORK

Prior work on automatically sanitizing free-text medical records [12, 6, 14] has focused largely on locating phrases representing name, address and other "identifiers" in the document's free-flow text and removing them; this removal happens irrespective of the context in which such phrases appear. Moreover, such systems are only as good as the underlying parser that tries to locate such phrases; in case the parser fails to locate certain identifiers (which could happen due to bad grammar and punctuation, noisy text, and other problems which hinder accurate parsing), then the identifier is not removed at all. In contrast, ERASE works by finding terms common in the entities' context and the document, and does not rely on a parser. Since ERASE distinguishes the protected entities from the rest, it is able to selectively retain certain harmless terms. Further, ERASE does not remove a term just because it is a name or an address – it considers other terms that co-occur with it in the document and tries to take a globally optimal decision.

Sidebar [10], a recent initiative at Xerox PARC attempts to provide a user-friendly platform to perform document sanitization; the platform enables a user to specify rules that limit the type of information (name, address, etc.) to disclose to different types of users. These rules, however, need to be specified manually and the system, overall, suffers from the same drawbacks as the sanitization systems mentioned above.

The sanitization criterion in ERASE is similar in spirit to K-anonymization, introduced by Sweeney [13] for structured data. K-anonymization based sanitization of structured relational databases has been addressed by LeFevre et al. [8]. However, the problem formulation considered in the above setup and ERASE are different. While the goal in the K-anonymization is to anonymize a given database, in ERASE, the aim to sanitize a single document. Saygin et al. [11] propose and motivate K-anonymization based document sanitization, but focus on the special case of sanitizing against authorship detection.

There have been a few other proposals to automate document sanitization, for instance as a part of the US DOE OpenNet initiative (e.g. [15]), but these have not been reported in the technical literature and thus it is not possible to compare them with ERASE in a fair manner.

3. PROBLEM FORMULATION

In this section, we formalize the notion of document sanitization and present a precise statement of the problem.

Let E denote the set of entities. We associate each entity $e \in E$ with a set of terms $C(e)$ which collectively represent the entire knowledge the adversary has about the entity; $C(e)$ is called the *context* of the entity e. Unknown to the adversary, we are also provided with a flag that identifies the subset P of *protected* entities from among the entities E; these are the entities that need to be protected against disclosure. For instance, consider a compendium of diseases. Each disease is an entity, and its context includes the symptoms and the drugs used to cure the disease. The set E contains all the diseases in the database, and the set P is the subset containing, say, terminal diseases such as AIDS and cancer whose disclosure in a declassified patient record (the document) is not desirable.

Further, let D be the input document; in ERASE, a document is modeled as a set of terms. Without loss in generality, we assume that D contains only the terms that are present in the context of at least one protected entity, i.e. $D \subseteq \bigcup_{e \in P} C(e)$; any other term present in D is harmless and need not be removed.

Now, suppose there exists a subset $S \subseteq D$ such that S appears in the context of a protected entity $e \in P$; this set S indicates to the adversary that the protected entity e might be associated with the document. The extent of the seriousness of this indication depends upon how many entities in E, other than e, contain S in their context as well – since the adversary has no further knowledge, it cannot be sure between e and these other entities. We quantify the adversary's dilemma by considering a tunable parameter K, and say that the adversary decides that S is associated with e if the number of entities in E, other than e, that also contain S in their context is less than K; the subsets $S \subseteq D$ having this property are considered dangerous. Clearly, a sanitized document must not contain terms that form a dangerous set with respect to any protected entity e. The discussion is formalized below.

Definition 3.1 **(K-safety).** *Let $T \subseteq D$ be a set of terms. For a protected entity $\overline{e} \in P$, let $A_T(\overline{e})$ be the number of entities other than $\overline{e}$ that contain $C(\overline{e}) \cap T$ in their context. The set of terms T is said to be K-*safe *with respect to a protected entity $\overline{e}$, if $A_T(\overline{e}) \geq K$. The set T is said to be K-safe, if T is K-safe with respect to every protected entity.*

The document sanitization problem being addressed can now be stated as follows:

Problem Statement. *We assume a fixed database consisting of a set of entities E, where each entity e is associated with a set of terms $C(e)$ forming its context. A subset of entities $P \subseteq E$ are flagged as protected. Given a document D and the parameter K as input, the problem is to find the maximum cardinality subset of D that is K-safe.*

Example 1: Consider an entity set $E = \{\overline{e}_1, \overline{e}_2, \overline{e}_3, e_1, e_2, e_3, e_4\}$; with protected entities $P = \{\overline{e}_1, \overline{e}_2, \overline{e}_3\}$. The entity contexts are as follows:

$$\begin{array}{ll} C(\overline{e}_1) = \{t_1, t_2, t_3\} & C(e_1) = \{t_1, t_2, t_4, t_7\} \\ C(\overline{e}_2) = \{t_2, t_4, t_5, t_6\} & C(e_2) = \{t_3, t_5, t_6\} \\ C(\overline{e}_3) = \{t_1, t_4, t_7\} & C(e_3) = \{t_2, t_7\} \\ & C(e_4) = \{t_1, t_2, t_3, t_5, t_6, t_7\} \end{array}$$

Now, consider a document $D = \{t_1, t_2, t_4, t_5, t_6, t_7\}$. Suppose $K = 2$. The subset $T_1 = \{t_2, t_4, t_7\}$ is not K-safe, because for the protected entity $\overline{e}_2$, the set $C(\overline{e}_2) \cap T_1 = \{t_2, t_4\}$ is contained in the context of only one other entity (namely, e_1) and hence, $A_{T_1}(\overline{e}_2) = 1$. On the other hand, the subset $T_2 = \{t_1, t_5, t_6, t_7\}$ is K-safe as can be verified below:

$$\begin{array}{lll} \overline{e}_1: & C(\overline{e}_1) \cap T_2 = \{t_1\} & \text{Contained in } \overline{e}_3, e_1, e_4 \\ \overline{e}_2: & C(\overline{e}_2) \cap T_2 = \{t_5, t_6\} & \text{Contained in } e_2, e_4 \\ \overline{e}_3: & C(\overline{e}_3) \cap T_2 = \{t_1, t_7\} & \text{Contained in } e_1, e_4 \end{array}$$

Thus, we see that for all $1 \leq i \leq 3$, $C(\overline{e}_i) \cap T_2$ is contained in the context of at least two entities (other than $\overline{e}_i$) and hence, $A_{T_2}(\overline{e}_i) \geq 2$. So, T_2 is K-safe. It can be verified that D (which of size 5) is not a K-safe set. Hence, T_2 is an optimal solution. □

4. HARDNESS RESULTS

A reduction from the maximum independent set problem [7], shows that document sanitization problem is NP-hard.

Theorem 4.1. *The document sanitization problem is NP-hard, even when the security parameter K is fixed to be 1.*

PROOF. We present a reduction from the maximum independent set problem. Let G be the input graph having m edges over a set of n vertices V. Construct a database containing $m+n$ entities. For each edge (u,v), add an entity e_{uv} with the associated context $C(e_{uv}) = \{u,v\}$. For each vertex v, add an entity e_v with the associated context $C(e_v) = \{v\}$. The entities corresponding to the edges are declared protected. Let $D = V$ be the document. Set the parameter $K = 1$.

We observe that a set $T \subseteq V$ is an independent set if and only if T is K-safe. To see this, suppose T is an independent set. Let e_{uv} be a protected entity with the associated context $C(e_{uv}) = \{u,v\}$, where (u,v) is some edge in G. Since, T is an independent set it cannot contain both u and v. Thus, $C(e_{uv}) \cap T$ is one of $\{u\}$ or $\{v\}$ or $\emptyset$. In all the three cases, we see that $A_T(e_{uv}) \geq 1$. Thus, T is K-safe. On the other hand, suppose T is not an independent set. Then, T contains some vertices u and v such that (u,v) is an edge in G. Let e_{uv} be the corresponding protected entity having context $C(e_{uv}) = \{u,v\}$. Then, $C(e_{uv}) \cap T = \{u,v\}$ and this set is contained in the context of only e_{uv} and not in the context of any other entity. So, $A_T(e_{uv}) = 0$. We see that T is not K-safe. We have shown that T is an independent set if and only T is K-safe.

From the above observation, it follows that finding the maximum independent set in G is equivalent to finding the largest K-safe subset of D. We conclude that the sanitization problem is NP-hard. □

Further, a known inapproximability result for the independent set problem [17] implies the following strong inapproximability result for the sanitization problem.

Corollary 4.2. *For any $\epsilon > 0$, if sanitization problem can be approximated within a factor of of $n^{1-\epsilon}$, then NP=P. Here, n denotes the size of the input document. The result is true, even if the parameter K is fixed to be 1.*

5. COMPUTING OPTIMAL SOLUTIONS

A naive way of computing the optimal solution is to enumerate all the K-safe subsets of the given document and pick the one with the maximum cardinality. In this section, we describe more efficient algorithms.

5.1 A Levelwise Algorithm

It is easy to see that a set of terms is K-safe only if all its subsets are K-safe. In other words, the K-safe subsets of a document D satisfy the apriori criterion [2, 3]. Thus, a levelwise algorithm can be used to enumerate the maximal K-safe subsets of the given document, and consequently to find the maximum cardinality K-safe subset.

The algorithm proceeds in a levelwise manner. Starting with safe subsets of cardinality $r = 1$, in each iteration it generates the candidate subsets of cardinality $r+1$ based on the K-safe subsets of cardinality r found in the previous iteration; a subset of cardinality $r+1$ is a candidate only if all it subsets of cardinality r are K-safe. The algorithm terminates after none of the subsets considered in an iteration are found to be K-safe, or after $r = |D|$. This algorithm can be readily instrumented to output, on termination, a maximum cardinality safe subset of D (the last candidate subset that was successfully flagged as safe).

```
Procedure BestFirst()
Input: A document $D$ and parameter $K$.
Output: An optimal $K$-safe subset of $D$.
Begin
    Let $\mathcal{T} = \{\langle\emptyset, 0\rangle\}$
    While (not terminated)
        // Remove the pair having the maximum upperbound
        Let $\langle s, r\rangle = \text{argmax}_{\langle s', r'\rangle \in \mathcal{T}} \text{UB}(\langle s', r'\rangle)$
        Remove $\langle s, r\rangle$ from $\mathcal{T}$
        If $r = n$ then output($s$) and halt
        Add $\langle s, r+1\rangle$ to $\mathcal{T}$
        If the set $s' = s \cup \{t_{r+1}\}$ is $K$-safe then
            Add $\langle s', r+1\rangle$ to $\mathcal{T}$.
    End
End
```

Figure 3: The BestFirst Algorithm

Observe that when all entities are protected, this algorithm is essentially the apriori algorithm for frequent itemset mining [2] on the entity database with minimum support K. Frequent itemset mining is a well-studied problem [3] and any of the several alternative algorithms for the problem can be readily applied to the document sanitization problem. The drawback with this frequent-itemset mining approach, as applied to document sanitization, is that it computes all the maximal K-safe subsets, which is not necessary; for document sanitization, it is sufficient to find one of the maximum safe subsets. In the next section, we present a more efficient branch and bound strategy that systematically prunes may choices and converges on an optimal solution quickly.

5.2 A Best-First Search Algorithm

Given a document D containing n terms, our goal is to find the optimal solution (the largest K-safe subset of D). Consider a complete binary tree of depth n such that each level represents a term and each root-to-leaf path represents a subset of D. The BestFirst algorithm performs a pruned-search over this tree by expanding the most promising thread in each iteration. The algorithm is briefly discussed below.

Let $D = t_1 t_2 \ldots t_n$ be the sequence of terms in the given document. For $i \leq j$, we use $D_{i,j}$ to denote the substring $t_i t_{i+1} \ldots t_j$. Of particular interest are the prefixes $D_{1,r}$, for $r \leq n$. The algorithm maintains a collection $\mathcal{T}$ of K-safe subsets of the prefixes of the document: each element in $\mathcal{T}$ is a pair $\langle s, r\rangle$ such that $s \subseteq D_{1,r}$.

We say that a K-safe set $T \subseteq D$ *extends* $\langle s, r\rangle$, if $T \cap D_{1,r} = s$. Along with each pair $\langle s, r\rangle$, the algorithm keeps an upper-bound $\text{UB}(\langle s, r\rangle)$ on the largest K-safe subset extending $\langle s, r\rangle$. In the next section, we present mechanisms for computing these upper-bounds; for now, notice that an element $\langle s, n\rangle$ cannot be extended any further and so, the largest subset extending $\langle s, n\rangle$ is simply s itself – thus, without loss of generality, we assume that $\text{UB}(\langle s, n\rangle) = |s|$.

The algorithm, shown in Figure 3, proceeds iteratively as follows. In each iteration, it picks the pair $\langle s, r\rangle$ in $\mathcal{T}$ having the maximum upperbound value $\text{UB}(\langle s, r\rangle)$, and considers the two possible ways of extending it: (i) including the term t_{r+1}, or (ii) excluding the term t_{r+1}. In the first case, the algorithm checks whether the set $s' = s \cup \{t_{r+1}\}$ is K-safe and if so, it adds the pair $\langle s', r+1\rangle$ to the collection $\mathcal{T}$. In the second case, s is guaranteed to be K-safe and so, it directly adds the pair $\langle s, r+1\rangle$ to the collection $\mathcal{T}$. The algorithm terminates when the chosen pair is of the form $\langle s, n\rangle$, and outputs the set s as the solution.

Theorem 5.1. *The algorithm BestFirst outputs an optimal K-safe subset of the input document.*

PROOF. Let T^* be the set output by the algorithm. This means that the pair chosen in the last iteration is $\langle T^*, n\rangle$. Consider any K-safe subset $T \subseteq D$ and we shall show that $|T^*| \geq |T|$. First, observe that in any iteration, $\mathcal{T}$ contains a pair of the form $\langle s', r'\rangle$ such that T extends $\langle s', r'\rangle$. Consider the last iteration and let $\langle s, r\rangle$ be the pair in $\mathcal{T}$ such that T extends $\langle s, r\rangle$.

Since $\mathrm{UB}(\langle s, r\rangle)$ is an upperbound on the size of any K-safe subset achievable by extending $\langle s, r\rangle$, we have: $\mathrm{UB}(\langle s, r\rangle) \geq |T|$. Also, since the algorithm chooses the pair having the maximum upperbound, we further have: $\mathrm{UB}(\langle s, r\rangle) \leq \mathrm{UB}(\langle T^*, n\rangle)$. Combining these observations with our assumption that $\mathrm{UB}(\langle T^*, n\rangle) = |T^*|$, we get $|T^*| \geq |T|$, proving the theorem. $\square$

The BestFirst algorithm essentially performs an exhaustive search. In the worst case, its running time is exponential in the size of the input document.

5.2.1 Computing Upper-bounds

Let $E^*(\langle s, r\rangle)$ denote the size of the largest K-safe set that extends $\langle s, r\rangle$; in this section we show how to compute a non-trivial upper-bound $UB(\langle s, r\rangle)$ on $E^*(\langle s, r\rangle)$.

Any set that extends the pair $\langle s, r\rangle$ includes the set s, by definition, and can at most include all of $D_{r+1,n}$. This leads to the following naive upperbound:

$$\mathrm{UB}_{naive}(\langle s, r\rangle) = |s| + (n - r).$$

Clearly, $\mathrm{UB}_{naive}(\langle s, r\rangle) \geq E^*(\langle s, r\rangle)$ and thus, $\mathrm{UB}_{naive}(\langle s, r\rangle)$ is an upperbound on $E^*(\langle s, r\rangle)$. This upperbound, however, is very weak and is not likely to provide significant pruning of the search space. We now derive a stronger upper-bound on $E^*(\langle s, r\rangle)$.

For $r \leq n$, let $A^*(D_{r,n})$ denote the size of the optimal K-safe subset of the suffix $D_{r,n}$. Notice that for any pair $\langle s, r\rangle$, $E^*(\langle s, r\rangle) \leq |s| + A^*(D_{r+1,n})$. Below, we discuss an approach for computing upperbounds on $A^*(D_{r,n})$ for every suffix $D_{r,n}$ and thereby obtain the required upperbounds on $E^*(\langle s, r\rangle)$. The approach is based on relaxing the notion of K-safety to *local K-safety*.

Fix an integer $L \leq n$. For $r \leq n - L + 1$, we call the sequence of L consecutive terms $D_{r,r+L-1}$ as an *L-window*. A set of terms $T \subseteq D$ is said to be *L-local K-safe*, if T is K-safe with respect to every L-window, i.e., for all r, the set $T \cap D_{r,r+L-1}$ is K-safe. While the notion of K-safety imposes a constraint with respect to the whole document, the notion of L-local K-safety is a relaxation that considers only the L-windows. The two notions coincide when $L = n$.

Define $A^*_L(D)$ to be the size of the largest L-local K-safe subset of D. Similarly, let $A^*_L(D_{r,n})$ denote the size of the largest L-local K-safe subset of the suffix $D_{r,n}$. Every K-safe subset is also L-local K-safe; this implies that for any $r \leq n - L + 1$, $A^*(D_{r,n}) \leq A^*_L(D_{r,n})$. Thus, for any pair $\langle s, r\rangle$, with $r \leq n - L$, $E^*(\langle s, r\rangle) \leq |s| + A^*_L(D_{r+1,n})$. Thus, we get the following upperbound on $E^*(\langle s, r\rangle)$:

$$\mathrm{UB}_L(\langle s, r\rangle) = |s| + A^*_L(D_{r+1,n})$$

Clearly, UB_L is a much tighter upper-bound than UB_{naive}.

Moreover, $A^*_L(D_{r,n})$ for all $r \leq n - L + 1$ can be computed efficiently (in time linear in n) for small L using a dynamic programming strategy, presented next.

Efficiently Computing $A^*_L(D_{r,n})$

We now present a dynamic programming algorithm that runs in $O(2^L n)$ steps and computes $A^*_L(D_{r,n})$, for all $r \leq n - L + 1$. The algorithm is efficient when L is small.

For each $r \leq n - L + 1$ and $x \subseteq D_{r,r+L-1}$, let $A^*_L(D_{r,n}|x)$ denote the size of the largest L-local K-safe subset of $D_{r,n}$, among those sets $T \subseteq D_{r,n}$ satisfying $T \cap D_{r,r+L-1} = x$. If x is not K-safe, then $A^*_L(D_{r,n}|x)$ is not well-defined and we declare it to be $-\infty$.

Let us consider the two consecutive L-windows $D_{r,r+L-1}$ and $D_{r+1,r+L}$. Now, for each $T \subseteq D_{r,n}$ such that $T \cap D_{r,r+L-1} = x$, we can either have $T \cap D_{r+1,r+L} = x - t_r$, or $T \cap D_{r+1,r+L} = (x - t_r) \cup t_{r+L}$. This leads to the following:

$$A^*_L(D_{r,n}|x) = |x \cap \{t_r\}| + \max \left[\begin{array}{c} A^*_L(D_{r+1,n}|x - t_r), \\ A^*_L(D_{r+1,n}|(x - t_r) \cup t_{r+L}) \end{array} \right]$$

The algorithm first computes $A^*_L(D_{n-L+1,n}|x)$ for each $x \subseteq D_{n-L+1,n}$, as follows. If x is K-safe, then set $A^*_L(D_{n-L+1,n}|x) = |x|$, else set $A^*_L(D_{n-L+1,n}|x) = -\infty$. Based on the above recurrence relation, it then computes $A^*_L(D_{r,n}|x)$ for all $1 \leq r \leq n - L$ and $x \subseteq D_{r,r+L-1}$. Finally, $A^*_L(D_{r,n})$ is computed as:

$$A^*_L(D_{r,n}) = \max_{x \subseteq D_{r,r+L-1}} A^*_L(D_{r,n}|x)$$

Clearly, there is a trade-off in choosing L. Larger values of L lead to a tighter upper-bound, and therefore better search-space pruning in the BestFirst algorithm. However, the complexity of computing the upper-bounds on the other hand, the would be take more time. We shall discuss how to choose L in the experimental section.

5.2.2 Testing for K-safety.

Recall that the BestFirst algorithm (ref. Figure 3) needs to check for K-safety of a set in each iteration. A simple-minded implementation the K-safety of a set S would involve checking, for each protected entity $\overline{e}$, whether $C(\overline{e}) \cap S$ appears in the context of at least K other entities. This can be expensive for large S.

However, the BestFirst algorithm invokes the K-safety tester on sets of the form $s = s' \cup \{t\}$, where s' is a K-safe set and t is a new term. Thus, it is sufficient to only focus on protected entities containing t. To efficiently retrieve all entities containing t in their context, we exploit an inverted index. This index maps each term to the set of entities that contain the term, and is created in a preprocessing step.

6. GREEDY HEURISTICS

In this section, we present an efficient heuristic that scales well with increase in document size. The heuristic aims to ensure that only a minimum number of terms are removed to sanitize the document, but occasionally removes a larger number of terms. The idea is to iteratively delete terms from the document until it is K-safe, where the term to delete in each iteration is chosen by estimating the amount of progress made by deleting a term. Towards that end, we first discuss a simple characterization of a set X such that deleting X from D yields a K-safe set; i.e., we characterize sets X such that $D - X$ is K-safe.

For an entity e, let $C_D(e) = C(e) \cap D$ denote the context of e restricted to the terms in the document D. Consider a protected entity $\overline{e}$. For any other entity e, we call the set $C_D(\overline{e}) - C_D(e)$ a *blocker* of $\overline{e}$; informally, the set $C_D(\overline{e}) - C_D(e)$ "blocks" the entity $\overline{e}$ from becoming a subset of e. Let $B(\overline{e})$ denote the set of all blockers of $\overline{e}$:

$$B(\overline{e}) = \{C_D(\overline{e}) - C_D(e) \ : \ e \in E \text{ and } e \neq \overline{e}\}.$$

Consider a set $X \subseteq D$. From the definition of K-safety, the set $D - X$ is K-safe with respect to a protected entity $\overline{e}$, if $C_D(\overline{e}) - X$ is contained in the context of at least K other entities; equivalently,

X must contain at least K blockers of $\overline{e}$. We say that X *covers* a blocker b, if $b \subseteq X$. Then, the above observation leads to the following proposition.

Proposition 6.1. *Let $X \subseteq D$. The set $D - X$ is K-safe if and only if for every protected entity $\overline{e}$, X covers at least K blockers of $\overline{e}$ (i.e., $|\{b : b \in B(\overline{e})$ and $b \subseteq X\}| \geq K$).*

PROOF. Consider a set $X \subseteq D$. Suppose $D - X$ is K-safe. So, by the definition of K-safety, for any protected entity $\overline{e}$, the set $C_D(\overline{e}) - X$ is contained in the context of at least K other entities. For any entity e, the set $C_D(\overline{e}) - X$ can be contained in $C(e)$ if and only if $C_D(\overline{e}) - C_D(e) \subseteq X$; in other words, X must contain the blocker $C_D(\overline{e}) - C_D(e)$. It follows that for any protected entity $\overline{e}$, X contains at least K blockers $\overline{e}$.

The converse direction is proved in a similar way. Suppose X contains at least K blockers of any protected entity $\overline{e}$. Consider any protected entity $\overline{e}$. If X contains the blocker $C_D(\overline{e}) - C_D(e)$, for some entity e, then it means that $C_D(\overline{e}) - X \subseteq C_D(e)$. It follows that $C_D(\overline{e}) - X$ is contained in the context of at least K entities (other than $\overline{e}$). Notice that $C_D(\overline{e}) \cap (D - X) = C_D(\overline{e}) - X$. So, $D - X$ is K-safe with respect to $\overline{e}$. We conclude that $D - X$ is K-safe. $\square$

Example 2: We continue Example 1 (see Section 3) and illustrate the notion of blockers. The blockers for the three protected entities (with respect to the document D) are shown in Figure 4. In the table, the entry $[\overline{e}_i, e]$ gives the blocker $C_D(\overline{e}_i) - C_D(e)$. Write $T_1 = D - X_1$, so that $X_1 = \{t_1, t_5, t_6\}$. Notice that for the protected entity $\overline{e}_2$, X_1 contains only one blocker of $\overline{e}_2$ and hence, T_1 is not K-safe. On the other hand, the set T_2 can be written as $D - X_2$, where $X_2 = \{t_2, t_4\}$. The set X_2 contains at least two blockers of each protected entity $\overline{e}_i$. So, T_2 is K-safe. $\square$

Our goal is to construct a small set X that covers at least K blockers for each protected entity. We start with an empty set X and iteratively include terms in X until the set has the required property. In each iteration, the main task lies in choosing a suitable term for including in X. This is done by assigning a score for each term that estimates the amount of progress made by including the term; then, the term with the highest score is chosen. In the next section, we consider different scoring functions.

6.1 Term Scoring Functions

We start with a simple scoring approach and then, propose refinements that lead to solutions of better quality. In the following, E is the set of all entities in the entity database, and $P \subseteq E$ is the set of protected entities.

6.1.1 Frequency Based Scoring (BFreq)

When we choose a term t, we make some progress towards covering any blocker that contains t. A simple scoring function is to count the number of blockers in which a term t appears, and use this count as the score for t. Formally:

$$BFreq(t) = \sum_{\overline{e} \in P} |\ \{\ b : b \in B(\overline{e}) \text{ and } t \in b\}\ |.$$

This results in the heuristic of iteratively picking a term that appears in the maximum number of blockers.

A drawback with BFreq is that it only considers the number of blockers hit by a term, but not the size of the blockers. A term t hitting a large blocker b does not help much in covering the blocker, as we also need to choose all the remaining terms in b to cover it. Based on this intuition, we obtain our next scoring function by refining BFreq.

6.1.2 Blocker Size Based Scoring (BSize)

In BFreq, a term t gets a score of one for hitting a blocker b, irrespective of the size of the blocker. Instead, we need to assign higher score to a term that hits smaller blockers and a lower score to a term that hits larger blockers. One straightforward approach to achieve this scoring is to assign the term a weight of $1/|b|$ for *each* blocker b hit by t.

However, consider two blockers containing the term t – one with ten terms, none of which have been chosen earlier, and another with fifteen terms, five of which have been chosen earlier – clearly, both should be assigned the same weight. Let $\hat{X}$ denote the set of terms not deleted so far. Our next scoring function considers only the terms in $\hat{X}$ and also takes into account the sizes of the blockers:

$$BSize(t) = \sum_{\overline{e} \in P} \sum_{b: b \in B(\overline{e}), t \in b} \frac{1}{|b \cap \hat{X}|}$$

Note that unlike BFreq, this has the overhead of having to maintain the scores at the end of each iteration.

6.1.3 Scoring Based on Top K Blockers (BTop)

The scoring function BSize takes *all* the blockers in $B(\overline{e})$ into account. This would be fine if X was required to cover all the blockers; however, X is required to cover only K blockers. Clearly, when K is much less than E, BFreq and BSize may be quite off the mark.

A more sophisticated scoring function should take the parameter K into account as well. Instead of considering all the blockers of a protected entity $\overline{e}$ containing the given term t, the idea is to consider a blocker $b \in B(\overline{e})$ only if (a) b contains t, and (b) $|b \cap \hat{X}|$ is among the smallest K among all the blockers in $B(\overline{e})$ that contain t. Let the set of such blockers be denoted by $\hat{B}(\overline{e}, t)$.

Another drawback of BSize is that it considers a protected entity even after K blockers in $B(\overline{e})$ have already been covered in $\hat{X}$. This is clearly unnecessary, and can lead to removal of more terms than required from the document. Instead of considering the entire set of protected entities e, BTop considers only those protected entries for whom less than K blockers are covered by $\hat{X}$. Let the set of such protected entries be given by $\hat{P}$.

Based on the preceding discussion, the scoring function BTop can be defined as:

$$BTop(t) = \sum_{\overline{e} \in \hat{P}} \sum_{b: b \in \hat{B}(\overline{e}, t)} \frac{1}{|b \cap \hat{X}|}$$

As with BSize, the terms scores computed with BTop need to be maintained as $\hat{X}$ changes at the end of each iteration.

Our experimental study shows that BTop performs significantly better in terms of quality than the other two scoring functions and moreover, outputs near-optimal solutions. So, for the remainder of this section we focus on BTop, and discuss efficient mechanisms for implementing the heuristic.

6.2 Efficiently Implementing BTop

We start with some simple observations:

- Any protected entity $\overline{e} \in P$ that does not contain any term from D (i.e., $C(\overline{e}) \cap D = \emptyset$) can be ignored, since any subset of D is K-safe with respect to such an entity.

- Any term $t \in D$ that appears in less than K entities should always be deleted from D, because such a term cannot be part of any K-safe set. Henceforth, we assume that such terms are deleted and D contains only the remaining terms.

Entity $\overline{e}_i$	$\overline{e}_1$	$\overline{e}_2$	$\overline{e}_3$	e_1	e_2	e_3	e_4
$B(\overline{e}_1)$	-	$\{t_1\}$	$\{t_2\}$	$\emptyset$	$\{t_1, t_2\}$	$\{t_1\}$	$\emptyset$
$B(\overline{e}_2)$	$\{t_4, t_5, t_6\}$	-	$\{t_2, t_5, t_6\}$	$\{t_5, t_6\}$	$\{t_2, t_4\}$	$\{t_4, t_5, t_6\}$	$\{t_4\}$
$B(\overline{e}_3)$	$\{t_4, t_7\}$	$\{t_1, t_7\}$	-	$\emptyset$	$\{t_1, t_4, t_7\}$	$\{t_1, t_4\}$	$\{t_4\}$

Figure 4: Blockers in Example 2

- A protected entity having only one term from D in its context can be ignored; under the assumption above, all the terms in D have frequency at least K. Similarly, we can also ignore non-protected entities having at most one term from the document.[6]

The first and third observations above typically result in significant pruning of the entity set on real-life documents; this is expected since most documents are expected to focus on a few entities. The second observation results in pruning of the document as well. Let E' and P' denote the set of entities and the set of protected entities after this pruning.[7]

As for BestFirst, we build an inverted index over the database that maps each term t to the set of all entities containing t in their context. We exploit the above index to compute the sets E' and P'. Next, for each $\overline{e} \in P'$, we compute the blockers of $\overline{e}$ with respect to the entities $e \in E'$.

For each $\overline{e} \in E'$, each term t and each $i \leq |D|$, we maintain the collection $B'(\overline{e}, t, i)$ consisting of all blockers of $\overline{e}$ of size i that contains the term t. We also maintain $R(\overline{e})$, the number of blockers that need to be additionally covered; the value $R(\overline{e})$ is initialized to K and keeps decreasing as and when blockers get covered. These collections help us in computing $\hat{P}$ and in finding the smallest blockers $\hat{B}(\overline{e}, t)$ at the end of each iteration. We now consider the issue of maintaining $B'(\cdot)$ and $R(\cdot)$ in each iteration. We consider two approaches: a naive approach (Naive-BTop) and an optimized approach (Fast-BTop).

6.2.1 Naive Approach: Naive-BTop

Suppose a term t is chosen and deleted from the document. We iterate through all the protected entities $\overline{e} \in P'$ and perform the necessary updates. First, there may be blockers of $\overline{e}$ that contain only the term t; these are present in the collection $B'(\overline{e}, t, 1)$. As we have chosen t, these blockers are covered and so, we update $R(\overline{e}) = R(\overline{e}) - |B'(\overline{e}, t, 1)|$. Next, for each term t and number i, we select all the blockers in $B(\overline{e}, t, i)$ that contain the term t. These blockers have shrunk in size by one; so , we move them to the collection $B(\overline{e}, t, i-1)$. Suitable indices are maintained to perform the above operation quickly.

6.2.2 Optimized Approach: Fast-BTop

The update operations in Naive-BTop are costly and inefficient. We can improve the performance by conducting the updates in a lazy manner. The main idea is that a blocker b of large size does not contribute the score of any term, until sufficiently many of its terms are deleted and it becomes one of the K smallest blockers. And so, we can delay updating such blockers. We present a brief sketch of the implementation.

Suppose a term t is deleted. Let t be the r^{th} term to be deleted, so that we are in the r^{th} iteration. Consider a protected entity $\overline{e} \in P'$. For a term $t' \in C(\overline{e}) \cap D$, let $\ell_{t'}$ denote the size of the Kth smallest blocker of $\overline{e}$, among those blockers containing the term t'. Let ℓ denote the maximum of $\ell_{t'}$ over all terms $t' \in C(\overline{e}) \cap D$. Notice that among the blockers of $\overline{e}$, only those blockers having size at most ℓ may contribute to the score any term. A blocker b of size $s > \ell$ cannot contribute until $s - \ell$ terms are deleted. We process and update the collections $B'(\overline{e}, t', j)$, for $j \leq \ell + r$ and ignore the collections containing larger blockers. Extensions of the above idea lead to further improvements; we omit the details for the sake of brevity. Fast-BTop refers to an implementation of the BTop scoring function based on these ideas.

7. EXPERIMENTAL STUDY

In this section, we present a preliminary experimental study to evaluate the different algorithms presented in the paper.

Metrics. The evaluation involves two natural metrics: *quality*, measured in terms of the number of terms in the sanitized output document, and *efficiency*, measured in terms of the running time of the algorithm.

Purpose. The purpose of this evaluation is to show that:

- The BTop heuristic outperforms BFreq and BSize in solution quality; therefore, the extra effort involved in implementing BTop is useful. Further, BTop often produces solutions of size close to the optimal solution.

- For smaller sized documents, the BestFirst algorithm is efficient; whereas, for larger documents, BTop is the algorithm of choice.

- The optimizations proposed for implementing BTop (i.e., Fast-BTop) offers significant performance gains over the naive implementation (Naive-BTop).

- BTop scales well with increase in various parameters such as document size, number of protected entities and K.

Platform. The implementation was done in Java with J2SE v1.4.2 (approx. 1000 lines of code) and executed on a 2.4 GHz Machine with 3.62 GB of RAM running Windows XP SP1. The term-to-entity set index (ref. Section 5.2, Section 6.2) was built using Lucene.[8]

Entity Set. The dataset was generated synthetically. We used a universe of 200 terms to generate the entities. A collection C of 100 *base-sets* of size 50 each was generated at random. We take each base-set $s \in C$ and generate 30 different entities by appending 50 more random terms. Thus, the total number of entities is $|E| =$ 3000; each entity e has 100 terms in its context ($|C(e)| = 100$), of which a subset of 50 terms overlap with 29 other entities and the remaining set of 50 terms may have random partial overlaps. We see that the contexts of the entities overlap in complex ways. Thus, the above process provides us with a fairly challenging dataset to test the algorithms. We randomly selected 450 entities and declared them as protected.

[6]Some extra book-keeping is required, however, to keep account of this pruning to avoid terms involved to be considered as unsafe after the deletion

[7]This pruning step is effective for both BestFirst and Fast-BTop; our implementation of BestFirst also exploits this pruning.

[8]http://lucene.apache.org

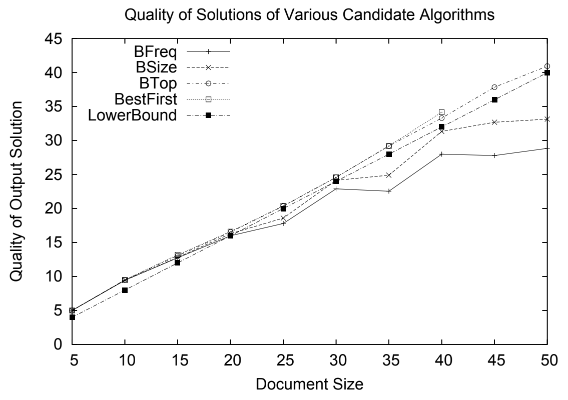


Figure 5: Quality: Effect of document size $|D|$

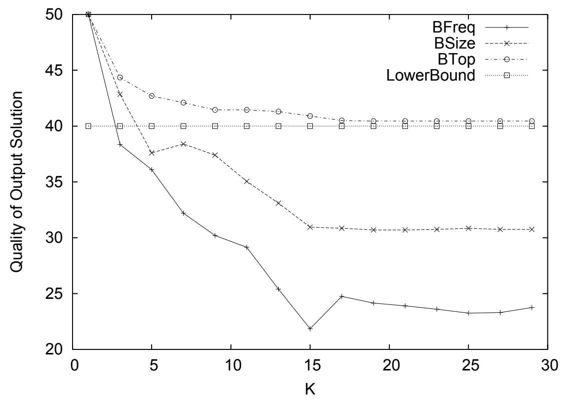


Figure 6: Quality: Effect of sanitization parameter K

Recall that our algorithms, in a pre-processing step, drastically prune the set of entities (see Section 6.2). In our setup, because of the way the entity set and documents are generated, every entity in the dataset is relevant for each document; as such, it is fair to consider the entity set in each experiment below as the set obtained *after* the pruning. Thus, while the number of entities per document appears small, it is probably more than what one may expect in real-life documents.

Document Generation. The efficiency and effectiveness of any algorithm depends on how many terms must be deleted to make the document K-safe. For instance, in one extreme are the K-safe documents; if the input document D is already K-safe, we need not delete any terms and so, these are easy to handle. The situation becomes more challenging when the optimal K-safe subset of D is of smaller size, where we need to carefully choose the terms to delete. Thus, it is natural to evaluate the proposed algorithms on documents of varying inherent quality.

Towards this goal, our document generation procedure takes as input as an additional parameter called *goodness*, which roughly measures the size of the optimal solution of the generated document. Given a length n and a value $\alpha \leq 1$, the method generates a document D of size n and goodness α, as described below. Choose one of the base sets $s \in C$ at random. Then, randomly select αn many terms from s and add $(1-\alpha)n$ random terms not belonging to s. The resulting set D is output.

Notice that $D \cap s$ is contained in the context of at least 30 entities. And thus, for $K \leq 30$, $D \cap s$ is K-safe. We have $|D \cap s| = \alpha|D|$ and so, $\alpha|D|$ is a lowerbound on the size of the optimal solution.

Parameters. There are three parameters that need to be considered in out experimental study: (i) Document size $|D|$; (ii) Sanitization parameter K; (iii) Goodness used in document generation α. In any specific experiment, we shall fix two of these parameters and vary the remaining parameter and study its effect. When the parameters $|D|$, $|K|$, α are not varied, their value is fixed as follows: $D = 50$, $K = 10$, and $\alpha = 0.8$.

In the next section, we compare the quality of the solution produced by the various heuristics. An experimental study on efficiency of the algorithms is presented in the subsequent section.

7.1 Comparison of Solution Quality

In this section, we study the quality of solutions produced by the proposed heuristics against the optimal solution produced by the optimal BestFirst algorithm.

7.1.1 *Effect of Document Size* $|D|$

In this experiment, we studied the effect of varying the document size $|D|$ from 5 to 50, in a setup with $K = 10$ and $\alpha = 0.8$. For each $|D|$, we generated 20 documents and averaged the solution quality of each heuristic. The result is plotted in Figure 5, and contains a curve for each of the three heuristics and the BestFirst algorithm. The curve for BestFirst runs only upto $|D| = 40$; the algorithm was too slow for documents of a larger size. (We have also plotted $\alpha|D|$, termed *lowerbound*, which is a lowerbound on the optimal solution size, as pointed out earlier; this is for validation, because in the later sections (which are for $|D| = 50$, we will be using αn as an approximation for the optimal.)

For small document sizes, the heuristics perform close to the optimal; however, as the document size increases, there is a significant degradation in the quality of the solutions returned by BFreq and BSize. The little difference for small documents is expected because given the small number of terms, the difference in terms in the solution produced by the different algorithms is small as well. For larger documents, we see that BTop is very close to the optimal, while BSize performs about 15-20% worse and BFreq about 20-25% worse. This validates the decisions to differentiate between the blockers, and having the scoring function intelligently exploit the parameter K. The near-optimal performance of BTop further asserts that any further optimization of BTop is not crucial to sanitization quality.

7.1.2 *Effect of Sanitization Parameter* K

In this experiment, we studied the effect of varying the sanitization parameter K from 1 to 30, in a setup with $D = 50$ and $\alpha = 0.8$.

For each K, we generated 20 documents and averaged the solution quality of each heuristic. The result appears in Figure 6, and contains a curve for each of the three heuristics. Since running the optimal BestFirst algorithm was not feasible for $|D| = 50$, we use αn, which is actually a *lowerbound* instead (any solution below this watermark is certainly suboptimal).

For $K = 1$, all approaches perform equally well; this is because it turns out that no term in the document belongs to exactly one entity (this is an artifact of how the entity set and the documents are generated from the base sets, as discussed earlier), and so neither of the approaches removes any term from the document. The results for $K > 1$ are interesting – we see that BFreq and BSize degrade very drastically with increasing K, BFreq degrading even faster than BSize. BTop performs extremely well in comparison; it stays well above the lowerbound watermark throughout and it seems reasonable to assume that it is actually very close to the op-

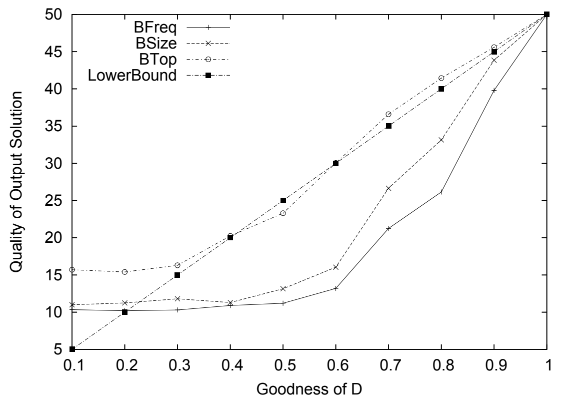


Figure 7: Quality: Effect of goodness parameter α

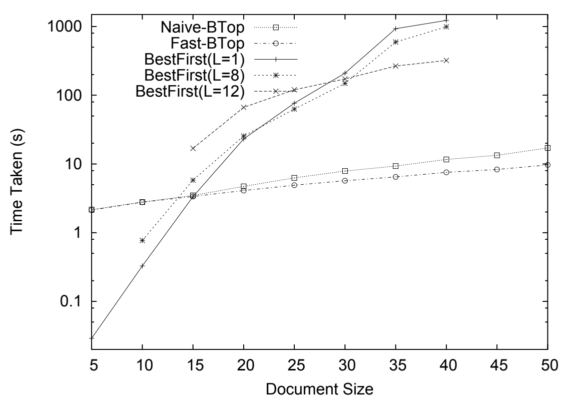


Figure 8: Efficiency: Effect of document size $|D|$

timal (extrapolating from the behaviour for smaller document sizes, demonstrated in the previous experiment). Overall, the substantial difference between BTop and the rest reaffirms the appropriateness of the BTop scoring function in the greedy heuristic.

7.1.3 Effect of Goodness Parameter α

In this experiment, we studied the effect of varying the document goodness parameter α from 0.1 to 1.0, in a setup with $D = 50$ and $K = 10$. For each α, we generated 20 documents and averaged the solution quality of each heuristic. The result appears in Figure 7. Again, as in the previous experiment, since it was not feasible to run the optimal algorithm BestFirst on this document size, we plotted the lowerbound on the optimal, $\alpha|D|$ as an approximation.

First, notice that all approaches perform equally well (optimally) for $\alpha = 1.0$. This is again an artifact of our implementation – the document generated for $\alpha = 1.0$ contains only terms from a given base set, and each term is present in at least 30 entities by design. For $K = 30$, therefore, the document generated is K-safe and no term needs to be deleted from the document. The result is interesting, however, for $\alpha < 1.0$, and we see that BTop, again, performs very close to the optimal throughout. Again, there is a vast difference between BTop and the other two heuristics, BSize and BFreq. It is interesting to note that the difference in performance of the three approaches widens significantly in the middle values of α; this is the region where the number of possible sanitization options increase, and the impact of the decisions in refining the scoring functions from BFreq to BSize to BTop shows very clearly.

7.2 Efficiency

The previous section compared the three different heuristic approaches, and showed that BTop significantly outperforms the other two. This was the motivation behind the effort to speed up the BTop algorithm; in Section 6.2, we discussed the naive approach (Naive-BTop) and an optimized approach (Fast-BTop). In this section, we compare Fast-BTop, Naive-BTop and BestFirst, and show that in contrast to the other two, Fast-BTop scales very reasonably with increasing document size, sanitization parameter as well as document complexity.

We also show that for a tuned value of the window size L (=12), BestFirst is more efficient than the heuristic approaches for small document sizes. For small documents (short email messages, etc.), this tuned version of BestFirst could actually be the approach of choice. To demonstrate the impact of tuning, we compare the tuned version of BestFirst with a naive version (L=1), which roughly corresponds to choosing UB_{naive} instead of UB_L (ref. Section 5.2).

7.2.1 Effect of Document Size $|D|$

In this experiment, we studied the effect of varying the document size $|D|$ from 5 to 50, in a setup with $K = 10$ and $\alpha = 0.8$. For each $|D|$, we generated 20 documents and averaged the time taken. The result is plotted in Figure 8, and contains a curve for each of Naive-BTop, Fast-BTop, BestFirst ($L = 1$), BestFirst ($L = 8$) and BestFirst ($L = 12$).

The results verify that BestFirst has low pre-processing overheads, but does not scale with increasing document size; the heuristics (Naive-BTop and Fast-BTop) have a much higher setup overhead due to the preprocessing involved in computing the blockers, but scale very well. We see that for small document sizes ($|D| < 14$), even the untuned BestFirst ($L = 1$) turns out to be more efficient than the heuristics; the difference increases dramatically as the document size is decreased. The experiment also reveals that the value of L must be chosen carefully. A low value for L such as $L = 8$ does not significantly alter the performance of the BestFirst algorithm. The difference is more pronounced between BestFirst ($L = 1$) and BestFirst ($L = 12$). Recall that the value of L involves a tradeoff: larger the value of L, the upperbounds are better leading to a better search-space pruning in the BestFirst algorithm; on the other hand, it takes more time to compute these upperbounds (see Section 5.2.1). The results confirm the above analytical conclusion. When the document size increases beyond a threshold, BestFirst($L = 12$) outperforms BestFirst($L = 1$), since the overhead involved in computing the upperbounds compensates by the way proving tighter upperbounds. Also, as expected, Fast-BTop is more efficient than Naive-BTop.

The experiments suggest that the BestFirst is the algorithm of choice, if the document size is smaller and Fast-BTop is the better choice for larger documents. Of course, when running time is not a criterion and optimal solution is desired, BestFirst algorithm should be used. While using BestFirst algorithm, the parameter L should be tuned suitably. The best choice of L will mainly depend on the document size, but other parameters such as K and the nature of the data-set may also play a role. A more detailed experimental evaluation is required to arrive at thumb rules for setting the value of L. This is deferred as future work.

7.2.2 Effect of Sanitization Parameter K

In this experiment, we studied the effect of varying the sanitization parameter K from 1 to 30, in a setup with $|D| = 50$ and $\alpha = 0.8$. Since BestFirst does not scale up to our choice of $|D| = 50$, we consider only Naive-BTop and Fast-BTop in this experiment.

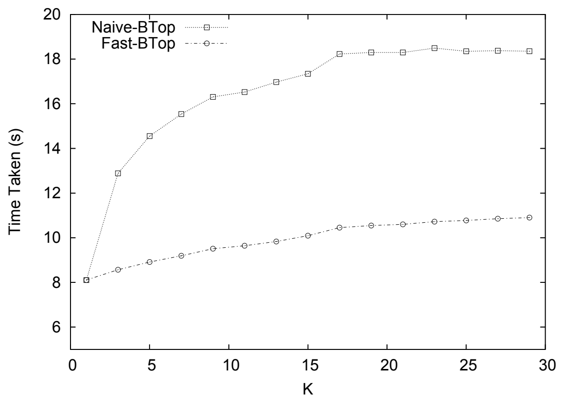


Figure 9: Efficiency: Effect of sanitization parameter K

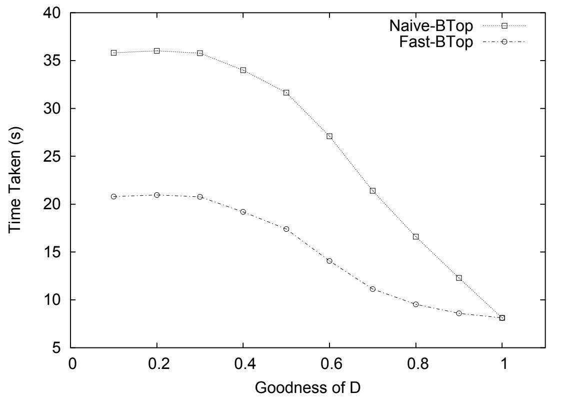


Figure 10: Efficiency: Effect of goodness parameter α

For each K, we generated 20 documents and averaged the time taken. The result appears in Figure 9, and contains a curve for each of Naive-BTop and Fast-BTop. We see that Fast-BTop significantly outperforms Naive-BTop for all values of K. When K is small, the optimal solution is large and hence, only a few terms need to be deleted to make the document K-safe. As a result, the difference in the performance of the two implementations is smaller for smaller values of K. With the increase in the value of K, the optimal solution size decreases and so more terms need to be deleted in this case. Thus, we see that gap in the performance increases. Clearly, the extra effort involved in the Fast-BTop implementation makes it more efficient than Naive-BTop.

7.2.3 *Effect of Goodness Parameter* α

In this experiment, we studied the effect of varying the document goodness parameter α from 0.1 to 1.0, in a setup with $D = 50$ and $K = 10$. Again, since BTop does not scale up to our choice of $|D| = 50$, we consider only Naive-BTop and Fast-BTop in this experiment.

For each α, we generated 20 documents and averaged the time taken by each approach. The result appears in Figure 10, and contains a curve for each of Naive-BTop and Fast-BTop.

The graph shows that Fast-BTop is faster than the Naive-BTop by a factor of 1.5 for documents with low goodness. As the goodness α increases, the difference between the two implementation decreases. Nevertheless, Fast-BTop outperforms Naive-BTop for all values of goodness. The reason for the decrease in the difference for larger α is similar to the one explained in Section 7.2.2. Namely, when α increases, the size of the optimal solution is large and so, only a few terms need to be deleted. As a result, the number of iterations decreases and hence, the difference in the running time also decreases.

8. CONCLUSION

In this paper, we presented ERASE, a framework for performing sanitization of unstructured text documents automatically. We presented a approach to address the document sanitization problem, and devised effective algorithms to perform the sanitization automatically. To the best of our knowledge, this is the first work that addresses the document sanitization problem in a formal manner and present principled approaches to solve the same. This work is an attempt to bring automated anonymization techniques, which have been so far studied and exploited extensively for structured data, to the domain of unstructured free-text documents.

9. REFERENCES

[1] E. Agichtein, L. Gravano, J. Pavel, V. Sokolova, and A. Voskoboynik. Snowball: A prototype system for extracting relations from large text collections. In *SIGMOD*, 2001.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

[3] A. Ceglar and J. F. Roddick. Association mining. *ACM Comput. Surv.*, 38(2), 2006.

[4] V. Chakaravarthy, H. Gupta, P. Roy, and M. Mohania. Efficiently linking text documents with relevant structured information. In *VLDB*, 2006.

[5] A. Chandel, P. Nagesh, and S. Sarawagi. Efficient batch top-k search for dictionary-based entity recognition. In *ICDE*, 2006.

[6] M. Douglass, G. Clifford, A. Reisner, W. Long, G. Moody, and R.G.Mark. De-identification algorithm for free-text nursing notes. In *Computers in Cardiology*, 2005.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[8] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Incognito: Efficient full domain k-anonymity. In *SIGMOD*, 2005.

[9] A. McCallum. Information extraction: distilling structured data from unstructured text. *ACM Queue*, 3(9):48–57, 2005.

[10] PARC. Xerox unveils technology that blocks access to sensitive data in documents to prevent security leaks, 2007. http://www.parc.com/about/pressroom/news/2007-10-15-redaction.html.

[11] Y. Saygin, D. Hakkani-Tur, and G. Tur. Sanitization and anonymization of document repositories. In *Web and Information Security*, 2005.

[12] L. Sweeney. Replacing personally-identifying information in medical records, the srub system. In *Journal of the Americal Medical Informatics Association*, 1996.

[13] L. Sweeney. K-anonymity: A model for protecting privacy. *Intl Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5), 2002.

[14] A. Tveit. Anonymization of general practitioner medical records. In *HelsIT'04, Trondheim, Norway*, 2004.

[15] U.S. Department of Energy. Department of energy researches use of advanced computing for document declassification.

[16] Wikipedia. Sanitization (classified information) — wikipedia, the free encyclopedia, 2008.

[17] D. Zuckerman. Linear degree extractors and the inapproximability of max-clique and chromatic number. In *STOC*, 2006.