Near-Duplicate Detection for Web-Forums

Klemens Muthmann
Technische Universität Dresden
Nöthnitzer Str. 46
D-01187 Dresden, Germany
Klemens.Muthmann@tu-dresden.de

Alexander Löser
Technische Universität Berlin
Einsteinufer 17
10587 Berlin, Germany
alexander.loeser@tu-berlin.de

Wojciech M. Barczyński
Falk Brauer
SAP AG
Chemnitzer Strasse 48
D-01187 Dresden, Germany
Firstname.LastName@sap.com

ABSTRACT

Unlike web search engines, current forum search engines lack the ability to identify threads with near-duplicate content and to group these threads in the search results. As a result, forum users are overloaded with duplicate search results. Furthermore, they often create additional similar postings without reviewing existing search results. To overcome that problem we have developed a new near-duplicate detection algorithm for forum-threads. The algorithm was implemented in a large case study using a real-world forum serving more than one million users. We compared our work with current algorithms, such as [3, 4], for detecting near-duplicates on machine generated web pages. Our preliminary results show, that we significantly outperform these algorithms and that we are able to group forum threads with a precision of 74%.

Keywords

Data Mining, Knowledge Management, Web-Forum Analysis

1. INTRODUCTION

More than 10 gigabytes of "user-generated content" is created in the Word Wide Web daily [11]. One particular example is a web-based question-answering forum, where humans pose questions and other humans provide answers. As the number of threads in a forum often grows drastically, a high precision search engine over forum content is necessary to grasp the mass of forum content and to spot relevant answers with high precision. However in practice, current forum search engines miss the ability to identify threads with near-duplicate content and are not able to group these threads within search results. E.g., we examined forum content for a software developer network (SDN) of a large software company. In this business-to-business forum more than one million users contribute each day on approximately 60 moderated sub-channels. We observed that users often are frustrated by the low precision of existing forum search engines. Hence most postings still depend on a human-generated answer. As a result, the average waiting time before receiving an answering post may range from a few minutes to even days. In fact, in our test-snippet of 219.000 postings we measured *one day* as average waiting time for a first answer (not necessarily a correct one). Even worse, we observed that *only 27% of posted questions received a response* by another community member.

To overcome this situation, we propose a new algorithm for grouping similar answers from existing threads in search results. As a result, users may more efficiently browse answers from the forum search engine and may spot existing answers. Furthermore, users willing to post answers can immediately oversee, which new threads have been answered already and can focus on unanswered threads. Common methods to detect duplicates on web-content are near-duplication detection approaches e.g., s [3, 4]. These methods are developed for automatically replicated web pages describing the same entity (e.g., a product) but present the entity by different agents (e.g., vendors). However, unlike replicated web-pages, forum threads focusing on a similar question often differ significantly in the syntax and language. In fact, in our initial experiments we could observe that these existing metrics result in only a decent precision of ca. 50%. Therefore we have developed a new near-duplicate detection algorithm for forum threads. To our best knowledge, we are not aware of any work on forum search engines to detect such near-duplicates with high precision. Our major contributions are:

- We identify common reasons leading to near-duplicate threads in web-forums.
- We propose a new mixed-model for measuring the similarity among web-forum-threads. We extend a hypergraph algorithm [10] to detect and group near-duplicate threads.
- We evaluate our method on a large web-forum. Prelimitary results show that we detect a similar number of near duplicates as state-of-the art algorithms for detecting near-duplicate web pages but achieve a 26% higher precision on forum content.

The paper is structured as follows: In Section 2 we unravel common reasons for near-duplicate threads in web-forums. In Section 3 we introduce our novel grouping and fingerprint algorithm. Section 4 presents and discusses our results and in Section 5 we review related work.

2. REASONS FOR NEAR-DUPLICATES

Motivated by our initial experiments with an existing forum search engine we carefully analyzed reasons for duplicate postings. Our main motivation is to identify common patterns during thread creation and answering process that may result in duplicate postings. The following analysis builds on a user study and on interviews with experienced forum administrators. For our user study we chose 50 threads between 2007 and 2008 in our data set from the "ABAP, General" forum, that contain the word "smartform". This term is specific to the domain of this channel and resulted in several duplicates from the data set. We analyzed posting structure, content, and occurrences of duplicates in these threads manually. Our initial research and discussion with the forum administrators unraveled four common reasons for duplicate content in threads:

R1. Low precision of forum search engine answers. Posting users often search before posting and hope to retrieve an immediate answer. However, current search engines produce just a list of often incorrect matching answers (usually based on the keywords).

R2. Lack of transparency in the forum structure. Often inexperienced users are overwhelmed by the forums size and do not know where to find or post their question. In our case, the site incorporates about 60 subforums, each with more than 1000's of threads.

R3. Amount of daily created data. In our forum approximately 5000 postings (questions or answers) are created daily. To provide room for new threads, existing threads are shifted to the end of a channel. Even if they are not yet answered they vanish from the attention of the community.

R4. Slow response time. Additionally, the system may not cope with the data load. This leads to users clicking multiple times on the 'submit button' or becoming annoyed of the slow performance of the search engine.

Near-duplicate postings occur as symptoms of these reasons. While this set of observations is not exhaustive, it serves as initial thesis throughout the paper and is evaluated in Section 4.3. We identified the following symptoms during our initial research and our experiments:

S1. Aggressive posting user. Such users submit the same question to several threads. Their main motivation is to increase the visibility of their question to potential expert members and hence to increase the likelihood of a fast response. They usually suffer from the overload of the forum (reason R3). Such postings share similar titles, initial posting texts and a single user who created the thread. Usually there are some minutes or hours between such postings, but we found examples of postings that occurred only seconds after the other.

S2. Impatient submitting. Because of slow forum reaction (reason R4), a user may hit the submit button twice, hence the same posting is submitted twice in short succession.

S3. Correcting mistakes. Users conduct mistakes while typing a posting, e.g., choose an unsuitable thread title. Frequently, they correct these errors by posting the same text again but with a different title.

S4. Thread closed too early. Usually, threads receive little traffic after they have been closed. If the creator has set its thread to 'answered' and suddenly notices that the answer was not as helpful as he thought, he creates a new thread with the same question.

S5. "Copy and Paste user". A user's total forum points mark him as an expert, well known and respected by the community. However a minority of community members just copy and paste existing answers. Such a behavior often results in threads containing syntactically similar answers that often even share the same user.

S6. Impatient, inexperienced user. These near-duplicates result from a combination of all reasons. Users just want to retrieve an answer fast. Often they are too 'lazy' to browse the forum structure or work through a long list of search results. Thus, they just post their question where they think it is appropriate or might even post it multiple times to different channels to increase the attention from possible experts. Further, these users are not aware of and not interested in frequently asked questions. Rather, they rely on the expertise of forum users to point them to suitable answers. Resulting near duplicates often have a link to the correct answer or share many 'entities', but the syntactic representation of the problem in the posting is usually very different.

Chowdhury et al. states in [5] 'the definition of what constitutes a duplicate is unclear'. As result of our manual inspection of typical patterns leading to near-duplicate threads we will use the following intuition to capture a near-duplicate thread:

DEFINITION 1. *Near-duplicate thread (human perspective) Two threads are near-duplicates: (1) if they have the same intention in the questioning posting, (2) if their answer postings provide a similar solution to the problem (and optionally is marked as correct by the requesting user) or (3) if both threads consist only of the question and answer postings, which contain the same link to other thread. A near-duplicate is incorrect if both threads discuss mainly different questions. The remaining pairs are undecided.*

This fuzzy definition captures the human intention when detecting a duplicate and provides much room for interpreting 'topic', 'problem' and their similarity in two threads.In the following sections we will give details on how to formalize, capture and model relevant features from threads, how to compare feature representations and how to group threads.

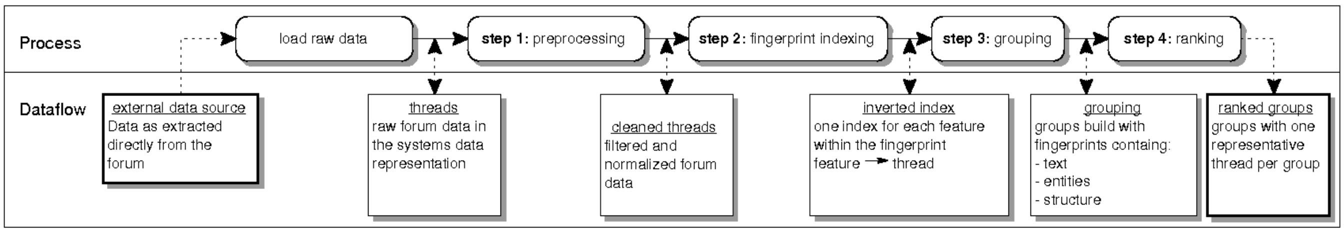
3. DETECTING NEAR-DUPLICATES

We present the process and give details for each step.

3.1 Overview

Figure 1 shows the grouping process for finding near-duplicates in a community forum. The process consists of the following steps:

Preliminary data loading. Raw data is loaded from an external source, e.g., a database, a file or a rss datastream. Raw data incorporates unstructured forum content and additional structured meta information, e.g., extracted



Process
load raw data
step 1: preprocessing
step 2: fingerprint indexing
step 3: grouping
step 4: ranking
Dataflow
external data source
Data as extracted directly from the forum
threads
raw forum data in the systems data representation
cleaned threads
filtered and normalized forum data
inverted index
one index for each feature within the fingerprint feature → thread
grouping
groups build with fingerprints containg:
- text
- entities
- structure
ranked groups
groups with one representative thread per group

Figure 1: Process for identifying Near-Duplicate Threads

entities, links, information about the author, etc. These features are classified in Section 3.2.1 and Section 3.2.2.

Step 1. Preprocessing cleanses raw content to prepare it for the fingerprint generation. According to [2] we remove stop words and eliminate punctuation. Further, we lower cased all words in natural language.

Step 2. Fingerprint generation produces a fingerprint for each thread (cf. Section 3.2). Fingerprints are created from text- and structure-based features. Mappings from features to threads are stored in an inverted index structure.

Step 3. Thread grouping identifies similar threads and combines them into a group. Each group has a fingerprint, which is the highest common denominator its members share. New threads are inserted into a group if the fingerprint of the group is similar enough with the fingerprint of the group. This step is discussed in detail in Section 3.3.

Step 4. Ranking and choosing group representant. From each group we identify a thread, that represents the entire group in the search results. Currently we take the longest thread from each group.

3.2 Fingerprint Generation

Figure 2 shows our generic model, which represents a thread in a forum. Each thread includes a title and one or several contributions. Each contribution is linked to an author and may have received award points from other users, which found it helpful. Furthermore, each contribution links to a textual body, which may include domain-specific entities or links. Our grouping approach incorporates text- and structure-based features, shown in Figure 3. In the following we will give details on how we obtain and model each feature.

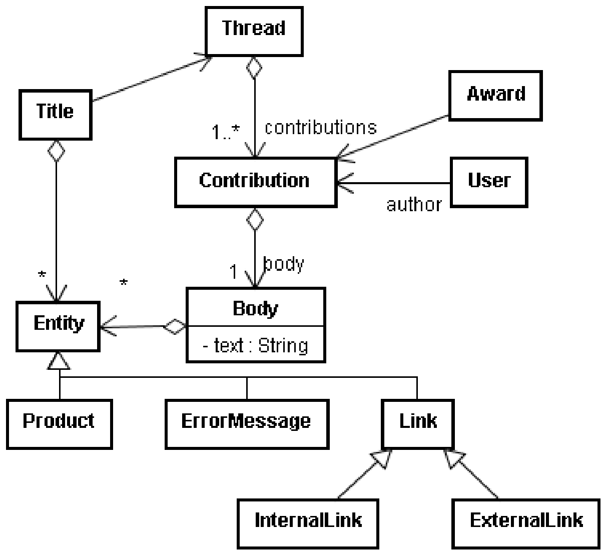


Figure 2: Thread Model

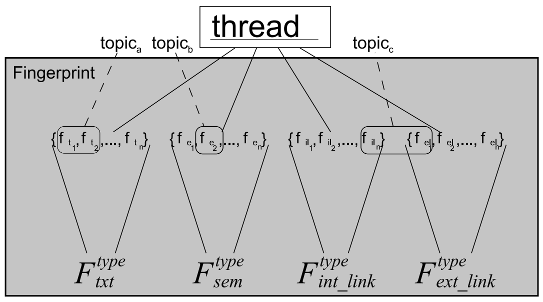


Figure 3: Conceptual Feature Model Wojciech: @Klemens: Please remove grey background.

3.2.1 *Text-based Features*

These features capture duplicate threads based on similar text passages, overlapping links, or overlapping extracted entities:

Domain-independent Text Features (F_{txt}^{type}). In B2B forums, such as in this paper, we assume that forum users utilize a common language, such as learned from software documentation[1] or fixed phrases, e.g., learned from application error messages. After stop word and punctation removal we represent the textual content of each thread as a set of its most frequent 1-grams.

Domain-independent External Links ($F_{extr\_link}^{type}$). Frequently answers in threads include links to external web-resources. Such resources are often files, blogs, home pages, wiki entries, or company specific help pages. Similar to anchor text, these links are complemented with short description, such as "Introduction to Netweaver see http://...". We extract each link and its corresponding description.

Domain-dependent 'Semantic' Features (F_{sem}^{type}). Often forum users refer to entities in the forum using different semantics. E.g. some users refer to software documentations as "SAP NOTE 934848" and some just use number "934848". To capture these entities, we use simple list- and rule-based extractors, such as [9], to complement text-based features. Our thread collection includes the following collection-specific-entities:

- SAP products, e.g., "Exchange Infrastructure"

[1]http://help.sap.com/content/additional/terminology

- Java exceptions, e.g., “java.lang.ClassNotFoundException”
- ABAP exceptions, e.g., “CX_SY_ZERODIVIDE”
- Referrals to software documentation, e.g., “SAP NOTE 934848”

We argue, that these lists of domain-specific vocabulary will be available for many B2B forums. They may be added with little costs and will complement the fuzziness of syntactic text-features.

3.2.2 Structure-based Features

These domain-independent features interconnect postings and threads in a forum. For this category we distinguish *internal links* and *forum structure features.*

Forum-internal Links ($F^{type}_{int\_link}$). Instead of retyping the answer, users frequently create a link to existing content in another thread when responding to a question. The link is complemented with some textual comment, such as: ”Please check this link” or ”For RFC Sender problems please refer to this thread”. We call such a posting a ***referral posting***, while the thread they refer to, we call the ***answering thread***. Similar to link graphs in the web, such referrals create a graph structure over interconnecting threads in a forum. Our intuition is that similar threads share the same links. Thus, for each posting we capture all forum-internal links and its corresponding descriptions.

Structural Patterns within Thread (F_{ea}). The relation between the thread containing a referral posting and the associated answering thread is even stronger if the first thread is only two postings long and contains only a question and the (often correct) referral posting. Therefore we capture the length of each thread and its correlation to an internal link.

Structural Patterns within Forum (F_{ch}). If two threads belong to the same channel (or sub-forum), we expect a higher correlation between near-duplicates candidates posted in the same channel. Therefore we mark for each thread the channel where it was posted.

3.3 Hyper-Graph-based Thread Grouping

Most duplicate detection approaches are faced with the “hyper graph problem” [10]: Every group is a set of sets where one subset may share objects with other subsets. Our solution is based on Manber’s work [10] and breaks the problem to creating a graph for each thread using an additional merging step to combine these graphs.

Our grouping algorithm starts from a set of normalized thread data and a complete fingerprint for each thread (cf. Figure 3). Algorithm 1 iterates over a given set of candidate threads H, which share at least one feature with an initially chosen seed thread. Therefore each group will share a set of features available in the seed thread. Next, the grouping algorithm compares each candidate thread $h \in H$ with the seed thread using the similarity threshold T. If the similarity of the current thread h is higher than a threshold T, h is inserted into each matching thread group. Depending on the similarity of the current thread h with g we distinguish two cases:

Subset Thread: If the fingerprint of the group g is a subset of h, then h covers the whole topic of all threads within the group and thus it is a valid member of g and is marked as sub-thread.

Algorithm 1 Algorithm grouping threads to seed threads

```
Require: seed Seed thread, T Similarity threshold, H Set of threads
  $G \leftarrow \emptyset$
  for all $h \in H$ do
    if $sim(h, seed) \geq T$ then {Disjoint Topic}
      $G \leftarrow G \cup (fn(h), val(G, fn(h) \cup \{h\}))$
      for all $g \in G$ do
        $O \leftarrow (fn(h) \cap fn(g))$
        if $|O| = |fn(g)|$ then {Subset Thread}
          $g \leftarrow (fn(g), val(G, fn(g) \cup \{h\}))$
        else if $sim(h, g) \geq T$, $|O| < fn(g)$ then {Overlapping Thread}
          $G \leftarrow G \cup (O, val(G, fn(g) \cup \{h\}))$
        end if
      end for
    end if
  end for
  return $merge(h, G)$
```

Overlapping Topic: Otherwise, if h is not a subset but g and h share a significant amount of features larger than T, h becomes am member of group g. Furthermore, we iterate overall remaining groups for the the current seed thread. If the overlap threshold permits, h is inserted into each of these groups.

Otherwise, h is considered as a *disjoint Thread* to *seed*. Once the algorithm iterated over all candidate threads in H, a new seed thread is chosen from H and the algorithm is started again. The algorithm terminates, if no further candidates for seed threads are available in H.

3.4 Mixed Model for Fingerprint Similarity

We distinguish between two similarity methods, one for content-based features and one for structural features, both are shown in Table 1. The equation for text and entities is

Text-based	$s_{t,e}(A,B) = \begin{cases} \frac{\lvert fn(A) \cap fn(B)\rvert}{\lvert fn(B)\rvert} & \lvert fn(B)\rvert \neq 0 \\ 0 & \text{otherwise} \end{cases}$
Structure-based	$s_l(A,B) = \begin{cases} 1 & \lvert fn(A) \cap fn(B)\rvert \geq 1 \\ 0 & \text{otherwise} \end{cases}$

Table 1: Feature Types and Similarity Metrics

based on function fn, which returns a set of features of a fingerprinted object. We use the Jaccard Distance to compare both sets. The equation for structure-based features, such as links, models a binary decision: Two fingerprinted objects are near duplicates if there there is a link between them, otherwise not. Equation 1 calculates the similarity of two fingerprints based on its structure and text-base features and allows to weight each feature class.

$$sim(A,B) = \frac{\sum_{i=1}^{n}(w_i * s_i(A,B))}{\sum_{i=1}^{n}(w_i)} \quad (1)$$

It takes two fingerprints A and B and calculates their similarity as a weighted sum of the similarities of every feature type. This sum is normalized by the sum of all weights to get a value between 0 and 1, that denotes the similarity of threads A and B. The function s calculates the similarity of two fingerprints incorporating each feature type. To remove additional parameters in our pre-limitary experiments we start with an equal weight for both feature types.

3.5 Example

Consider an arbitrary seed thread *seed* as shown in Figure 4. Its fingerprint contains six features $f_1 - f_6$. Now the threads from H are grouped to *seed* with a threshold of th. The thread a having a fingerprint of $f_1, f_2, f_4, f_5, f_7, f_8$ has a similarity of 0.66 to *seed* and since this is greater than the threshold it is added to a new group. The fingerprint of this new group becomes the similarity of *seed* and a, which happens to be the same as a's fingerprint. In the next step b is

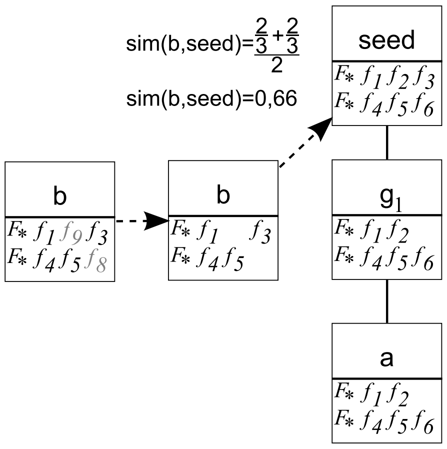


Figure 4: Group Creation for Single Seed Thread

grouped to *seed*. Its similarity is greater than the threshold too and so it is inserted to a group with b's fingerprint at first. Afterwards b's fingerprint is compared to all remaining groups which happens to be only g_1. Since b's fingerprint completely overlaps that of g_1, b is just added to this group.

The thread c is inserted during the last step. It shares features f_1, f_3, f_4 and f_5 with *seed*, which is enough to reach the threshold of $0,5$. At first c is inserted again to a group g_3 that has the same fingerprint as c itself. Afterwards it is compared to the remaining groups g_1 and g_2. It has the same similarity to both, so a new subgroup g_4 is created capturing that similarity and containing the threads a and b from the old groups together with the new one c. The final grouping as result of the algorithm is shown in Figure 5.

4. EXPERIMENTAL RESULTS

In this section we describe our data set, define our evaluation methodology, our experiments and evaluate different settings for grouping operations.

4.1 Data Set and Methodology

Our experiments are based on a sample of 54570 threads (219.000 postings) collected from the SDN forum and obtained between 2007 and 2008. From this set we chose 2587 threads (ca. 5%) randomly as Gold-Standard. Among these threads 437 only featured a question, while 2150 at least provided a question and some response (cf. Figure 4.3). As it is not possible to determine all near-duplicate pairs for several thousands of threads by hand, we cannot determine the recall of the different algorithms. Instead we compared

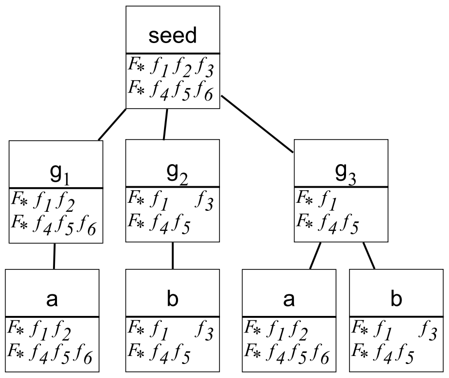


Figure 5: Final Grouping Seed Thread

the algorithms based on (1) precision and (2) the number of duplicate threads each method was able to identify. Metric (1) required human evaluation: From the result of each candidate algorithm we evaluated them by an expert, who labeled pairs of duplicate threads as correct or incorrect. We used the following dentition for a correct near-duplicate: Two threads are *correct near-duplicates* if (1) their text differ only by the following: Same questions where expressed using different words, (2) answers included the same links to existing threads or (3) the difference is a combination of the items listed in (1) and (2). **Alexander: Klemens: Bitte hier besser Deinen Eval Prozess beschreiben!** A near-duplicate thread pair is*incorrect* if the main intention of the two threads were different . For example, **Alexander: Klemens: Bitte hier besser beschreiben .**

4.2 Evaluation

In the following subsections we explain each experiment in detail. We compare our metrics for near-duplicate threads against a set of seven experiments. Table 2 provides an overview on our experiments. We start with experiments on exact matching threads (Baseline). Next, we investigate which text-based features (Text-75 and Text-80) and which structure based features (External Link) contribute to a high precision and recall. Finally, we conduct experiments using all features (ALL) on threads with different length (Thread-1 and Thread-2).

4.2.1 Baseline: Exact Matches

For our first experiment we measured threads with exact matching postings. A thread is considered as an exact duplicate if both candidate threads have exactly the same initial posting. Using the baseline on the test data set we identified 18 groups containing 2 threads with exact initial postings. In total our baseline algorithm identified 36 near duplicates, which is about 1.4% of the data set. By a closer observation we found out that 18 of these duplicates were created by the same user. Only one of the 19 groups contained two duplicates posted by different users, possibly due to multi accounting or account sharing.

4.2.2 Text-Only

To compare our improvements to existing near-duplicate detection approaches we conducted two experiments using text-only features. We focused on question postings in this experiment. We expected similar results to algorithms, such

Experiment	Generated groups	Near duplicate threads	Precision	Group size		
				2	3 - 5	5+
Baseline	18	36	100%	18	0	0
External link	5	10	20%	5	0	0
Text-75	139	272	48,82%	96	29	14
Text-80	88	186	57,80%	76	9	3
All	126	194	74,33%	118	8	0
Thread-1	10	19	80%	10	0	0
Thread-2	5	10	80%	5	0	0

Table 2: Summary of Experiments

as [10] or [3], for detecting machine generated duplicates. We conducted an experiment with a similarity threshold T of 0.8 and 0.75 minimal similarity. These experiments are called "Text-80" and "Text-75". Most grouped threads contain short questions, with a small increase in longer questions for the lower threshold. Text-based similarity approaches do not result in a sufficient precision. Most errors where created because of the limited amount of text, especially in threads with only one posting. E.g., we observed an initial posting which only contained four words: "What is a smartform?". Several threads with more postings matched this informational question, but very few actually had the same intention, namely to locate a wiki-page or a thread explaining the term "smartform". Another major source for mistakes were "copy and paste users". These users often re-used old postings just changing a few words to make up for a new question. Here, text similarity identifies the template text and matches it against all other occurrences of the template. Furthermore, we observed that longer questions decrease the precision of the second experiment from 58% for Text-80 similarity to 48% for Text-75 similarity.

4.2.3 *External-Links*

For the second experiment our intention was to investigate the assumption that equal questions are solved by referring to the same external resources. We used a threshold that identified two threads as near-duplicates if they share at least two common links. As result of this experiment 1.4% of the threads from the test data set where grouped as near duplicates, while the precision was as low as 20%. Main reason for such as low precision was the frequent usage of hub-links, such as "https://forums.sdn.sap.com" referring to the entry page of the forum portal. Therefore, when applying link-based similarity, we suggest to restrict to links to authority pages only.

4.2.4 *Text-based and Structure-based*

Finally we conducted three experiments using all features together: Since most groups were focused on small threads we examined two subsets from the results of this experiment. The first experiment was called "Thread-1". It contains groups of threads with only one posting. Such groups cover threads with only the question posting, which have little value for the searcher. Next, we examined groups containing only threads with two postings "Thread-2". Such threads often include a question and an obvious answer or a link to an answer. These threads are frequent in the data set and of high value for the answer-seeking user. In our last experiment "All", we examined all threads independent of the length. For both experiments - "Thread-1" and "Thread-2" - we could observe a precision of 80%. However, we could only find 10 ("Thread-1") and 5 ("Thread-2") groups. The "All" experiment found 126 groups with a precision of $74,33\%$. Most errors in this category where similar to the ones observed during the text experiments. However the addition of semantic and forum structure significantly lowered these issues. This conclusion is also true for the two sub experiments.

4.3 Lessons Learned

We summarize our observation as follows:

- **Existing matching strategies are not sufficient.** Our experiments clearly show, that simple metrics, such as exact or text-only matches are insufficient and do only produce decent results.
- **Mixed approach increases precision drastically.** Adding information about the link structure within a forum or between threads of the same channel and of extracted entities increases precision while the amount of found groups nearly remains at the same level.
- **Algorithm works especially well for shorter threads.** Figure 6 shows the distribution of threads with different postings lengths in our data set. Short threads

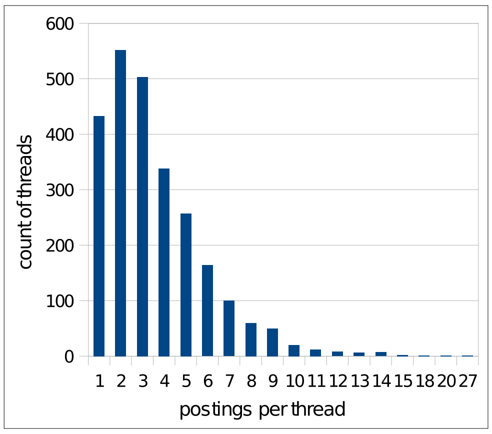


Figure 6: Postings per Thread

are more common: The average length of a thread is between 4 and 5 postings; however most common are threads with a length of two, three and one posting. Our text- and structure-based metrics work especially well for identifying near-duplicates on these shorter threads (see Figure 4.3). We assume, that these threads have a clear intention in the question, while longer threads are often disturbed by contributions not directly fitting the topic of the thread.

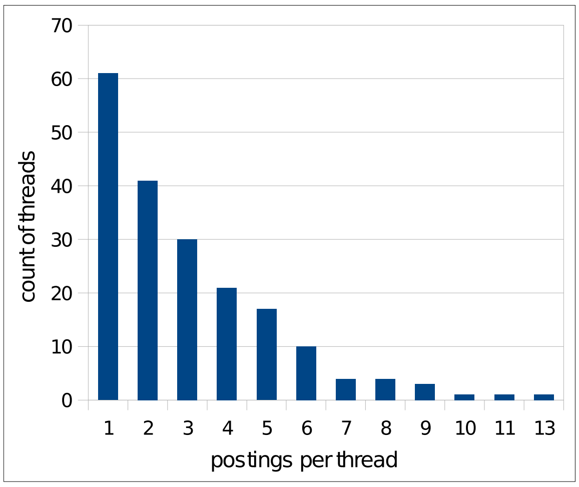


Figure 7: Postings per Thread for "ALL"

- **Few near-duplicates, very few exact duplicates.** Overall, 194 near-duplicate threads were spotted in our sample of 2587 threads using the *all-algorithm*. The number of exact duplicates is very low, in total 36 threads have an exact duplicate. A possible reason lays in the procedure to create content: "Traditional" web pages, e.g., in in shopping portals or news related web pages, are "created and replicated" using different CSS styles and HTML templates, while forum content is created by human beings and incorporates a much higher variety of different expressions for the same question or answer.

5. RELATED WORK

Our work is related to fingerprint creation and text abstraction algorithms, work about comparing fingerprints and work about web-based clustering techniques. In this section we compare our approach against these approaches.

Fingerprint generation. This area covers approaches for creation fingerprints. E.g., [4], [14] and [7] create fingerprints independent of the extracted features. Other approaches are sensitive to the chosen feature types, which are more similar to our work. Broder [3] concentrates on text-based features of a document. He creates multiple fingerprints from extracted shingles, which are lists of tokens from a document. In his work tokens might be letters, words or sentences. To create the final fingerprints, a random subset from all shingles of a document is selected and a hash function is applied. Manber [10] and Theobald et al. [12] propose methods to choose shingles based on anchor words, like important domain specific words or stop words. We extended these ideas on other specific feature types, like extracted entities and forum links. The I-Match algorithm proposed by Chowdhury et al. in [5] also uses document text to create fingerprints. They generate no hash fingerprint but keep the collection of words from every document and apply different filtering techniques based on the inverted document frequency value of each document. Since forum content is unstructured content and since we base our approach not just on text similarity, we incorporate information extraction techniques as well. For our extraction approach we used a rule based extraction system similar to the AVATAR proposed by [13]. Forum content can range in its quality from very low to very high. The authors of [1] present an approach for identifying high quality content based on various metrics but don't incorporate grouping techniques.

Measuring similarity. The second question is how to compare generated fingerprints. The k-Bit fingerprints from [4], [14] and [7] are comparing fingerprints using the Hamming Distance: If the fingerprints of two documents differ in at most f-Bit positions they are considered to be similar. Manber [10], Broder [3] and Chowdhury [5] use an approach based on set similarity, which is similar to our approach. They count the number of common elements and weight this intersection by the sum of available elements from both fingerprints. This is essentially the notion of the Jaccard distance.

Clustering techniques. Since it is not sufficient to know the similarity of two documents but necessary to group multiple documents together, all duplicate detection approaches are faced with the "hyper graph problem" [10]: Every group is a set of sets where one subset may share objects with other subsets. Some approaches [3] decide to simplify the problem and propose that a duplicate can only occur in one group. Our solution, based on Manbers work [10], breaks the problem to creating a graph for each thread and using an additional merging step to combine these graphs.

Duplicate query type. We can detect duplicates in document sets either online or off line. Manber proposes solutions for both approaches in [10] and the authors of [15] created a hybrid solution based on query logs of a large search engine. The current implementation of our approach is only able to do offline grouping. The algorithm however should be fast enough to do online queries, if only search results are grouped and some preliminary steps that are currently calculated on each run are pre-computed.

Performance Improvements. Another group of related work focuses on existing algorithms trying to improve performance issues. E.g., Henzinger in [8] proposes a solution to combine [3] and [4] to achieve better precision and remove some of the drawbacks of both algorithms. From the area of cluster creation there their work improves performance of clusters based on cluster ensemble collections. Fern and Lin in [6] present a method to create clusters by merging pre-selected subsets of clusters from a cluster collection. Our solution is similar, since we create multiple graphs, one for each feature type and keep only relations that have a high combined threshold from all graphs. In contrast to our approach these existing techniques identify mostly near-duplicates that share the same ancestor document. Thus, they are able to identify if one document is a changed version of another one. That is in most cases not true for threads; rather in threads two users create a new thread for the same question independently.

6. CONCLUSION AND FURTHER WORK

To the best of our knowledge, no other work has tackled and solved the problem of identifying near-duplicates in web-forums so far. In this work we have shown that identifying near-duplicates in web-based forums is possible. In an intensive user study, we could identify technical factors, such as the low search engine precision, but also human factors, such as aggressive or lazy users, as major reasons for the existence of duplicate postings. For spotting near-duplicate threads we used a fingerprint matching approach. We developed a feature model incorporating text- and structure-based features and developed methods to compute and compare fingerprints derived from these features. Our preliminary experiments on several thousand of threads of a real-world forum has shown that only a combination of all feature categories results in a sufficient high precision. In our future work we foresee three directions:

- **Larger Evaluation Data Set:** To understand better, how to detect duplicate forum postings we will run our experiments on a larger "Gold Standard". Furthermore we will evaluate our algorithm on different question and answering forums. E.g., we like to prove our findings for B2B forums in other forum types, e.g. B2C forums such as reviews for consumer products or C2C forums, such as Yahoo! Answer.
- **Trainable Parameter Model:** To weight similarity features, we use a linear interpolation approach. To avoid manual weight tuning, we like to investigate how to obtain correct weights from the forum data itself. E.g, some forums encourage users to mark threads as answered. It might be useful, to train a parameter model based on a corpus on answered threads and test the model on threads which have been potentially answered but are still unmarked.
- **Richer Feature Set to select "Best" Thread in a Group:** Our current heuristic to select the "best" thread within a near-duplicate group bases solely in the length of the threads. While we found this simple metric already useful, we will explore a richer feature models. E.g., morphological information, and information about grammar correctness might help to determine the quality of a thread in a near-duplicate group. Another direction will incorporate existing information about thread authors and their performance in the past. E.g., one might prefer to show threads, which have been written by users which often provide answers, or users which often write questions which have been received respond from many other users.

7. REFERENCES

[1] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne. Finding high quality content in social media, with an application to community-based question answering. In *Proceedings of ACM WSDM*, pages 183–194, Stanford, CA, USA, February 2008. ACM Press.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, May 1999.

[3] A. Z. Broder. Identifying and filtering near-duplicate documents. In *COM 0́0: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pages 1–10, London, UK, 2000. Springer-Verlag.

[4] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC 02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, New York, NY, USA, 2002. ACM.

[5] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe. Collection statistics for fast duplicate document detection. *ACM Trans. Inf. Syst.*, 20(2):171–191, 2002.

[6] X. Z. Fern and W. Lin. Cluster ensemble selection. In *SDM*, pages 787–797. SIAM, 2008.

[7] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[8] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 284–291, New York, NY, USA, 2006. ACM.

[9] T. S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases*, 29, 2006.

[10] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, San Fransisco, CA, USA, JanuaryJuly–FebruaryJanuary 1994.

[11] R. Ramakrishnan and A. Tomkins. Toward a peopleweb. *IEEE Computer*, 40(8):63–72, 2007.

[12] M. Theobald, J. Siddharth, and A. Paepcke. Spotsigs: robust and efficient near duplicate detection in large web collections. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 563–570, New York, NY, USA, 2008. ACM.

[13] T.S.Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. In *IEEE Data Engineering Bulletin*, May 2006.

[14] W. Xi, E. A. Fox, W. Fan, B. Zhang, Z. Chen, J. Yan, and D. Zhuang. Simfusion: measuring similarity using unified relationship matrix. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 130–137, New York, NY, USA, 2005. ACM.

[15] S. Ye, R. Song, J.-R. Wen, and W.-Y. Ma. A query-dependent duplicate detection approach for large scale search engines. In *APWeb*, pages 48–58, 2004.