Near-Duplicate Detection for Web-Forums

Klemens Muthmann*
Technische Universität Dresden
Nöthnitzer Str. 46
D-01187 Dresden, Germany
klemens.muthmann at tu-dresden.de

Wojciech M. Barczyński
SAP AG, SAP Research
Chemnitzer Str. 48
D-01187 Dresden, Germany
wojciech.barczynski at sap.com

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

Falk Brauer
SAP AG, SAP Research
Chemnitzer Str. 48
D-01187 Dresden, Germany
falk.brauer at sap.com

ABSTRACT

Current forum search technologies 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 duplicated search results and prefer to create new threads without trying to find existing ones. In this paper we therefore identify common reasons leading to near-duplicates and develop a new near-duplicate detection algorithm for forum threads. The algorithm is implemented using a large case study of a real-world forum serving more than one million users. We compare this work with current algorithms, similar to [4, 5], 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%.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval

General Terms

Algorithms, Human Factors, Semantic Web, Knowledge Discovery, Data and Process mining, Databases for e-commerce

Keywords

Data Mining, Knowledge Management, Web-Forum Analysis

*Work done while at SAP Research CEC Dresden
†Work done while at SAP Research CEC Dresden

IDEAS 2009, September 16-18, Cetraro, Calabria [Italy]
Editor: Bipin C. DESAI

1. INTRODUCTION

More than 10 gigabytes of "user-generated content" is created in the World Wide Web daily [11]. One particular example is a web-based question-answering forum, where users pose questions and other users 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 (*SAP Developer Network* - SDN[1]) of a large software company. In this business-to-business forum more than one million users share more than 5000 contributions each day on approximately 60 moderated sub-channels. We observed that users are often frustrated by the low precision of existing forum search engines. Hence most contributions still depend on a human-generated answer. As a result, the average waiting time before receiving an answer may range from a few minutes to even days. In fact, in our test-snippet of 219.000 contributions 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 are able to browse threads provided by the forum search engine more efficiently and may spot existing answers faster. 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., see [4, 5]. 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 syntax

[1]http://www.sdn.sap.com

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 hybrid-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. Preliminary 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.

In this paper we concentrate on the description of the problem and a first solution providing good precision. For production systems efficiency in the form of processing speed, distribution and memory usage is also important. However we want to tackle such issues in further work.

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. Section 6 concludes our work.

2. REASONS FOR NEAR-DUPLICATES

Motivated by our initial experiments with an existing forum search engine we carefully analyzed different scenarios for duplicate postings. Our main motivation was to identify common patterns during the process of thread creation and answering, that may result in duplicates. The following analysis builds on a user study and on interviews with experienced forum administrators and a data set from the SAP Developer Network (SDN). For our user study we chose 50 threads between 2007 and 2008 from the "ABAP, General" channel, that contain the word "smartform". This term is specific to the domain of this forum 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 eight common scenarios for duplicate content in threads:

S1 Impatient aggressive posting user.
Over time old threads are shifted to the end of the forums overview page. To improve their questions visibility some users copy them to a new thread, that gets again presented at the top of the overview page.

S2 Inexperienced aggressive posting user.
Users that are unable to decide which area (i.e. subforum, channel) is the right place for their topic, just post it to several or every possible place in hope to catch the right one by chance.

S3 Impatient inexperienced posting user.
Sometimes people in need of a fast answer, are unable to find it using the forums search function and are too "lazy" to study the forums structure and existing content very deeply. So they create a new thread with their question, even if it was answered before.

S4 Impatient submitter
If the forum engine reacts a little slowly, some users just hit the submit button several times. Engines not able to handle such a case get overloaded with copy after copy of the same thread.

S5 Correcting mistakes.
Some users create new threads in a hurry or without thinking deeply about their writing. Afterwards they realize they made an error, like a badly chosen title. If the forum engine does not support the feature of editing existing threads or if a user is not aware of this feature, he just creates a corrected copy of his question.

S6 Thread closed too early.
Some forums support the feature, that threads are marked as answered by the creator of the thread. Such threads are of little interest for further discussion and get little to no traffic. Sometimes the threads creator closes it and afterwards realizes that the wannabe answer is not as helpful as he thought. If the forum engine does not support the feature of reopening the thread, its creator can only reopen a new one containing the same question again.

S7 "Copy and paste" user.
Some forums identify the most active users with a ranking system, that is based on points. The more points one acquires the better ranked he is. This leads to users that answer duplicates by just copying existing answers, getting more points with almost no effort.

S8 Annoyed experts.
People who are very active in a forum community and know it inside out are experts for existing threads. They notice duplicates and get annoyed by answering the same questions over and over again. To shorten their work, they create links to existing answers instead of answering a question twice.

These scenarios reveal four shortcomings of existing forum engines. We state these shortcomings as reasons and symptoms for the existence of duplicate content in forums and are going to address them in our work.

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). By building groups of duplicates these lists can be improved to show one topic per list entry.

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 channels (sub-forums), each with thousands of threads. The more a forum grows the more confusing it gets the more duplicates are created. Duplicate detection breaks this vicious circle.

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. Duplicate detection can show a user existing threads for his question before he creates a duplicate and thereby reduce the amount of daily created data.

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. By removing or preventing duplicates the load on the forum can be reduced and the infrastructure in the form of hardware, software and the forum operators can concentrate on unique content.

While this set of observations is not exhaustive, it serves as initial thesis throughout the paper and is evaluated in Section 4. ***Chowdhury et al.*** states in [6] "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 question post, meaning that an answer solving the first question would also help in solving the second one (2) if their answer post provides a similar solution to the problem (and optionally is marked as correct by the requesting user) or (3) if both threads are linked to each other. A near-duplicate is incorrect if both threads discuss mainly different questions. For example if the discussed problems arise under different circumstances and the answer to one problem is not helpful in solving the other one. The remaining pairs are undecided in terms of duplicates.*

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

In this section we present the process for near duplicate detection. We provide a model for forum threads and classify the features used by our duplicate detection algorithm. Finally we present our grouping algorithm.

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 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.

Step 2. Fingerprint generation produces a fingerprint for each thread (cf. Section 3.2). Fingerprints are created from text-, extracted entity-, and structure-based features. Semantic features are entities, like product names or abbreviations extracted from the text. 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 to the fingerprint of the thread. 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, because we believe it contains the most information. This measure has to be improved in future work.

3.2 Fingerprint Generation

Figure 2 shows the model of a thread in a forum, used in our work. 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-, entity-, and structure-based features, shown in Figure 3. These feature types are the basic information needed for fingerprint creation. A fingerprint is a vector of feature type sets - one set per feature type. So a fingerprint for the structure shown in Figure 3 is a four dimensional vector, where each dimension corresponds to exactly one of the four presented feature types. In the following we give details on how we obtain and model each feature.

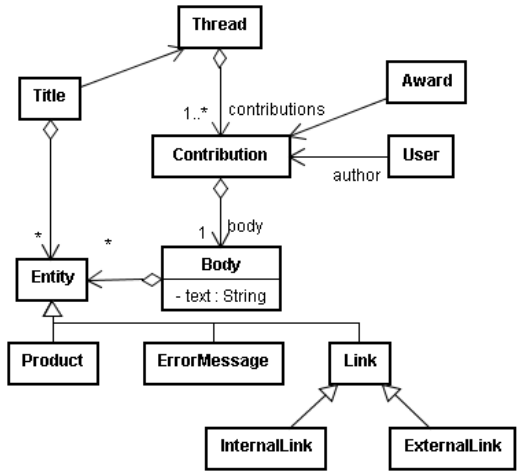

Figure 2: Thread model

3.2.1 Text-based Features

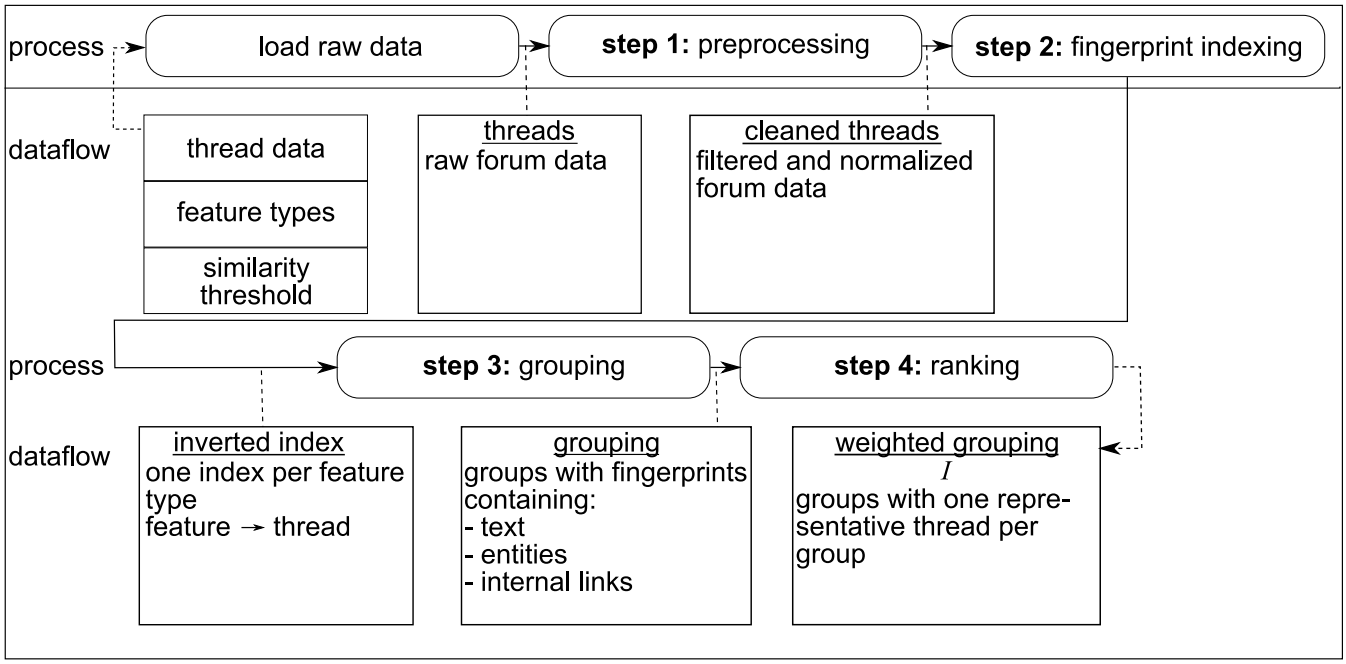

Figure 1: Process for identifying Near-Duplicate Threads

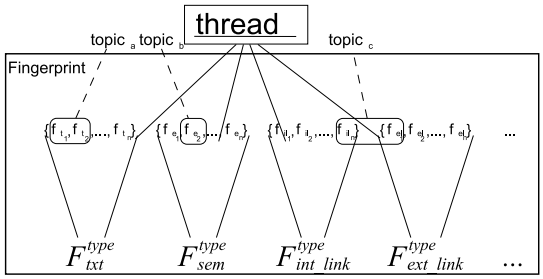

Figure 3: Conceptual feature model

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

Domain-independent Text Features (F^{type}_{txt}). In B2B forums, such as in this paper, we assume that forum users utilize a common language, such as learned from software documentation[2] or fixed phrases, e.g., learned from application error messages. After stop word and punctuation removal we represent the textual content of each thread as a set of its most frequent 1-grams.

Domain-independent External Links ($F^{type}_{extr\_link}$). 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 a short description, such as "Introduction to Netweaver see http://...". We extract each link and its corresponding description.

Domain-dependent 'Semantic' Features (F^{type}_{sem}). Often forum users refer to entities in the forum using different semantics. E.g. some users refer to software documentation as "SAP NOTE 934848" and some just use number "934848". To capture these entities, we use simple list- and rule-based extractors using a framework presented in [3], to complement text-based features. Our thread collection includes the following collection-specific-entities:

- SAP products, e.g., "Exchange Infrastructure"
- 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 ($F^{type}_{only\_ref}$). The relation between the thread containing a referral posting and

[2]http://help.sap.com/content/additional/terminology
http://msdn.microsoft.com/en-us/goglobal/bb688105.aspx

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}^{type}). If two threads belong to the same forum area (e.g. channel or subforum), we expect a higher correlation between near-duplicates candidates. 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 grouping is a set of overlapping subsets, meaning that one subset may share objects with other subsets. The complexity for creating such a hypergraph is exponential and thus we try to reduce it based on Manber's work [10], breaking the problem to creating a graph for each thread by using an additional merging step to combine graphs. The merging step is necessary since graphs are created on a per thread basis, thus every group gets created once for every of its members. The merging steps removes these double groups.

Our grouping algorithm starts from a set of normalized thread data and a complete fingerprint for each thread (cf. Figure 3). It iterates over all threads from this set and calls Algorithm 1 for each one. This one thread is called *seed* in the further explanations. Algorithm 1 iterates over a set of candidate threads H obtained from an inverted index structure. Each member of this set shares at least one feature with *seed*. Therefore each group will share a set of features available in the seed thread. The set of possible groups for *seed* is empty at the beginning. During the run of the algorithm it gets filled with tuples containing a fingerprint and a set of all threads that share that fingerprint. So one tuple in *grps* represents a group. The function *sim* returns the similarity of two objects with fingerprints. Section 3.4 explains how this is achieved. Additionally *getThreads* takes a set of group tuples and a fingerprint as input and returns the set of threads that share this fingerprint, while *getFingerpr* returns the fingerprint of a group or a thread. At first, the grouping algorithm compares each candidate thread $h \in H$ with the seed thread using the similarity threshold th. If the similarity of the current thread h is higher than the threshold th, h is inserted into each matching thread group. Depending on the similarity of the current thread h with g we distinguish two cases:

Algorithm 1 Algorithm grouping threads to seed threads

Input: *seed* Seed thread, *th* Similarity threshold, H Set of possible duplicates

```
grps ← ∅
for all h ∈ H do
 // Test for non disjoint threads
 if sim(h, seed) ≥ th then
  // Add h to group with same fingerprint or create it
  currentCont ← getThreads(grps, getFingerpr(h))
  grps ← grps ∪{(getFingerpr(h), currentCont ∪{h})}
  for all g ∈ grps do
   intersect ← (getFingerpr(h) ∩ getFingerpr(g))
   // Test if h is subset thread of g
   if | intersect | = | getFingerpr(g)| then
    // Add thread h to group g
    currentCont ← getThreads(grps, getFingerpr(g))
    g ← (getFingerpr(g), currentCont ∪{h})
    // Test if h has over threshold intersection with g
   else if sim(h, g) ≥ th, | intersect | < getFingerpr(g)
   then
    // Add content of g and h to a new group
    f ← getFingerpr(g)
    grps ← grps ∪(intersect, getThreads(grps, f) ∪ {h})
   end if
  end for
 end if
end for
return merge(h, grps)
```

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.

Overlapping Topic: Otherwise, if h is not a subset but g and h share a significant amount of features larger than th, h becomes a member of group g. Furthermore, we iterate over all 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 and the algorithm is started again. The whole algorithm terminates, if no further seed threads are available.

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 based on function *getFingerpr*, which returns a set of features of given type. The first metric in Table 1 applies for text- and entity- based features, while the second is suitable for structured-based ones. We use them based on *Jaccard Similarity* to compare two sets containing features of the same feature type (i.e. elements of a fingerprint). The formula is not exactly the same as Jaccard Similarity to capture near-duplicates were one thread of a duplicate pair is overloaded with additional information but nevertheless contains many or all features of the other one. The equation for structure-based features, such as links is a binary decision: Two fingerprinted objects are near duplicates if 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 type.

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

The function *sim* takes two fingerprints A and B and calculates their similarity as a weighted sum of the similarities for n feature types i. 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. To remove additional parameters in our preliminary experiments we start with an equal weight for all feature types (i.e. w_i is always 1).

3.5 Complexity

Text-, Entity- based	$s_{t,e}(A,B) = \begin{cases} \frac{\mid getFingerpr(A) \cap getFingerpr(B) \mid}{\mid getFingerpr(B) \mid} & \mid getFingerpr(B) \mid \neq 0 \\ 0 & \text{otherwise} \end{cases}$
Structure-based	$s_s(A,B) = \begin{cases} 1 & \mid getFingerpr(A) \cap getFingerpr(B) \mid \geq 1 \\ 0 & \text{otherwise} \end{cases}$

Table 1: Feature Types and Similarity Metrics

Algorithm 1 in the worst case compares every group to every candidate thread from H for every seed thread *seed*. In this case it as an exponential complexity of $O(n^n)$, which is not better then the naive approach of comparing every thread to every other. However there are two conditions that must be true for this to happen. At first the candidate set for every seed thread has to be the whole input set, which is true if every thread has at least one feature in common with every other thread. Second the fingerprints of all threads need to be different so that a new group is created for every combination of two threads. While the second condition might occur quite easily the first is very unlikely and depends very much on which features-types are considered. Therefore feature-types of features occuring very seldom (for example links) are better for performance than ones that have many often occuring instances (for example text). In the second case it might happen that one word that is very common to the forum slows performance quite drastically, while in the first case the whole algorithm runs nearly linearly, because the size of the candidate set is close to zero.

3.6 Example

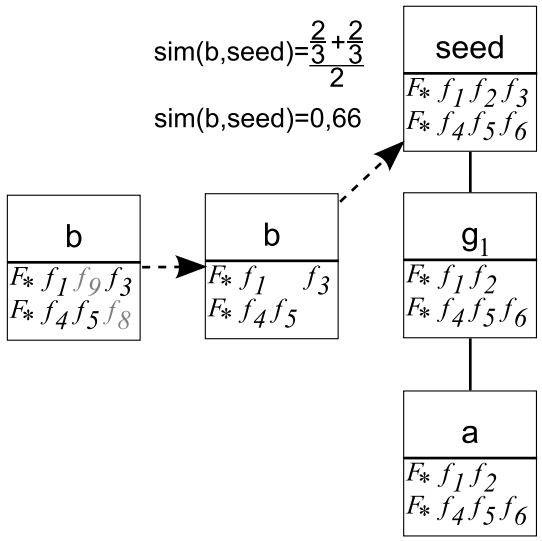

Figure 4: Group Creation for Single Seed Thread

Consider a thread b that gets grouped with a threshold of $th = 0.5$ to an arbitrary seed thread *seed* as shown in Figure 4. The fingerprint of *seed* contains six features $f_1 - f_6$ and it already forms a group g_1 with a thread a. The thread b having a fingerprint of $f_1, f_9, f_3, f_4, f_5, f_8$ has a similarity of 0.66 to *seed* and since this is greater than the threshold it is added to a new group g_2. The fingerprint of this new group equals the similarity of *seed* and b.

Afterwards b's fingerprint is compared to all remaining groups which happens to be only g_1. Since the similarity between b's and g_1's fingerprint is equal to the threshold th b is added to a new group g_3 together with a. This new group's fingerprint gets all features common between b and g_1, which are f_1, f_4 and f_5. The final grouping as result of the algorithm is shown in Figure 5.

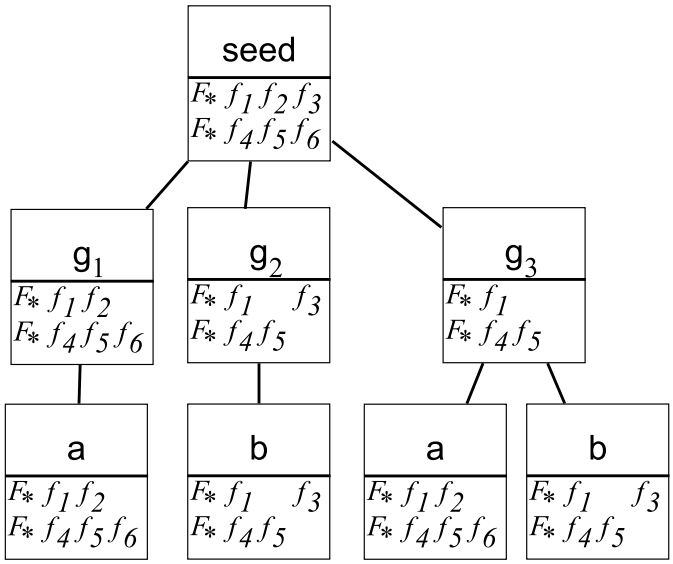

Figure 5: Final Grouping Seed Thread

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%), that were created in the timeframe of one week, as Gold-Standard. Among these threads 437 only featured a question, while 2150 at least provided a question and some response (cf. Figure 6).

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 the algorithms based on (1) precision and (2) the number of duplicate threads each method was able to identify. Metric (1) required human evaluation: At first we ran our process several times on the Gold Standard data set, varying the values for the parameters: feature type and similarity

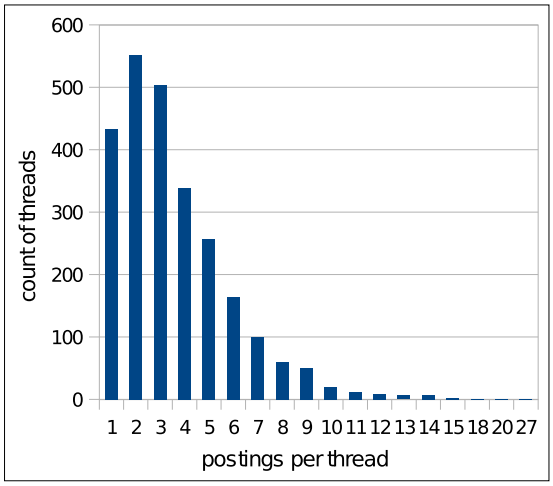

Figure 6: Postings per Thread in the evaluation dataset

threshold thereby. Each of these experiments generated a set of duplicate groups. The sets were evaluated by an expert, who labeled grouped threads as correct or incorrect members of their group. For simplicity we marked both, undecided and incorrect members as incorrect. Correct near-duplicates were chosen using Algorithm 1 from Section 2. In addition we calculated metric (2) for every experiment by summing up the size of all groups created by that experiment. With the precisions and the numbers of found duplicates for every experiment we are able to compare them to each other.

4.2 Evaluation

In the following subsections we explain each of our seven experiments in detail and compare our metrics for near-duplicate threads for all of them. Table 2 provides an overview of the results. We start with experiments on exact matching threads (Baseline) and continue to 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 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 as [4] or [10], for detecting machine generated duplicates. We conducted an experiment with a similarity threshold *th* of 0.8 and 0.75 minimal similarity. These experiments are called "Text-80" and "Text-75". We observed that precision is decreased significantly from 58% for Text-80 to 48% for Text-75. This is due to long questions that are matched very bad using only text as feature type. For the high threshold experiment 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.

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 a 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 links to authority pages only. Such pages do not contain links to other web resources but contain the required information directly.

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:

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

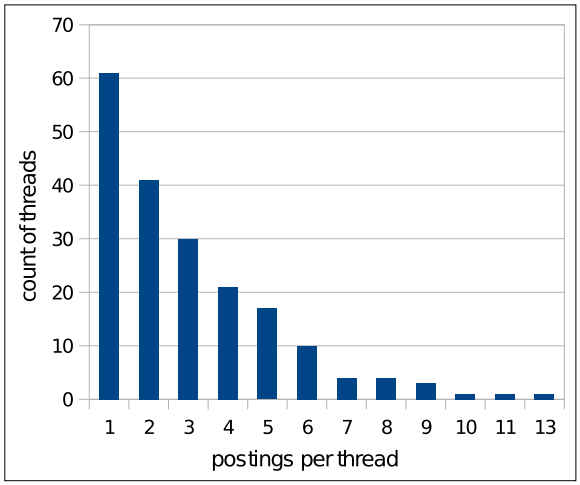

Figure 7: Postings per Thread in groups created by "ALL"

- **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 is especially fitting for short threads.** Figure 6 shows the distribution of threads with different postings lengths in our data set. Short threads 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 7). 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.
- **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 is due to the procedure to create content: "Traditional" web pages, e.g., 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.
- **High thresholds produce fewer results with better precision.** As expected increasing the threshold parameter also increases the precision of found duplicates, but also decreases the count of found duplicates.

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., [5, 14, 8] create fingerprints independent of the extracted features. Other approaches are sensitive to the chosen feature types, which are more similar to our work. Broder [4] 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 extend these ideas on other specific feature types, such as extracted entities and forum links. The I-Match algorithm proposed by Chowdhury et al. in [6] also uses document text to create fingerprints. They inspect the collection of words from every document and apply different filtering techniques based on the inverted document frequency value of each document. Using these filtered bags of words they create a hash fingerprint for each document and insert these fingerprints to a hash map. All documents with equal hash values are considered near-duplicates. A contribution in a forum however usually contains very few content. Therefore we don't select a subset of features based on TF/IDF [6] or shingle metrics [10, 12, 4], but rather use all features available[3]. Since forum content is unstructured 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

[3]After stop word removal of course.

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. In [5, 14] and [8] k-Bit fingerprints are compared 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 [4] and Chowdhury [6] 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 [4] 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 on-line 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 off-line grouping. The algorithm however should be fast enough to do on-line 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 [9] proposes a solution to combine [4] and [5] to achieve better precision and remove some of the drawbacks of both algorithms. Work from the area of cluster creation improves performance of clusters based on cluster ensemble collections. Fern and Lin in [7] 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-based features, features based on extracted entities for products, and structure-based features and developed methods to compute and compare fingerprints derived from these features. Our preliminary experiments on several thousand 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 four 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! Answers.

- **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 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 received responses from many other users.

- **Improved efficency:** The current version of our duplicate detection and grouping algorithm has a very high complexity and runs quite slow on large datasets. In future work we will concentrate on improving this behaviour. This can be achieved for example by distributing our algorithm on a cluster of computers, trying to improve the candidate selection phase and focus research on duplicate detection measures that might be more efficient than jaccard similarity.

7. ACKNOWLEDGMENTS

This work is supported by the FP7 EU Large-scale Integrating Project "OKKAM - Enabling a Web of Entities"[4] (contract no. ICT-215032).

8. 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.

[4] `http://www.okkam.org`

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

[3] W. M. Barczynski, F. Brauer, A. Loeser, and A. Mocan. Algebraic information extraction of enterprise data: Methodology and operators. In *IK-KR Workshop at IJCAI 2009 (to be published)*, 2009.

[4] A. Z. Broder. Identifying and filtering near-duplicate documents. In *COM ́00: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pages 1–10, London, UK, 2000. Springer-Verlag.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[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.