Quantitative Considerations about the Semantic Relationship of Entities in a Document Corpus

Andreas Schmidt
Karlsruhe University of Applied Sciences &
Karlsruhe Institute of Technology
andreas.schmidt@kit.edu

Steffen Scholz
Karlsruhe Institute of Technology
steffen.scholz@kit.edu

Abstract

Providing suggestions for internet-users is an important task nowadays. So for example, when we enter a search string into the Google interface, it suggests further terms, based on previously formulated queries from other users having used the search engine before. In the context of an entity based search engine, entity-suggestion is also a very important task, when specifying the entities by the user. Additionally, this feature can also be utilized to suggest further entities, which are somehow related to already specified entities. If the suggestions are eligible the user can very quickly formulate his search desire. If the suggestions are based on the search corpus itself, new and previously unknown relationships between entities can be discovered along the way.

The aim of this paper is a quantitative analysis of relationships between entities in a big document corpus under the aspect of providing suggestions for entities in real time.

1. Introduction

Entity Disambiguation [1] is a powerful technology to extract semantic information from text. Based on this technology, new search engines like STICS [2], which rather use concepts than words as search input, have emerged. One challenge for such systems is the specification of the entities to be searched by the user. Typically, this is solved by an autosuggestion function, which suggests possible entities based on a given prefix. Figure 1 gives an example for the suggestion of entities, specified by the given prefix "`unive`". After final selection of one of the suggested entities, further entities can be specified. If the suggestions are good, this can lead to a very effective way for formulating the search query. One critical point is the order in which the suggestions are presented. The goal is to present the most probable entities at the top, so that they can be selected quite fast. To rank the suggested entities, a global measure, like the publicity of an entity can be used. In the case of STICS, where the entities are extracted from Wikipedia, the publicity of an entity can be calculated based on the number of links an entity receives from other Wikipedia pages. This approach is called *insensitive* to a specific document corpus. A corpus *sensitive* approach on the other side can count the number of times a specific entity appears inside the corpus and use this value as a measure of popularity of an entity.

Figure 1: Auto-Suggestion based on given prefix

The disadvantage of this approach is that after the first entity has been chosen, the approach of presenting the most popular entity at the top of the suggestion list

URI: http://hdl.handle.net/10125/50003
ISBN: 978-0-9981331-1-9

is no longer appropriate. Choosing one of the top suggested entities often leads to empty result sets because considered isolated, the selected entities are most often very popular on their own but the combination of the entities doesn't make any sense in many cases. So in a worst case scenario, no documents contain both entities and an empty result-set is returned. Instead of the global probability, a conditional probability of an entity with respect to the previously specified entities has to be considered. So for example, consider the case, where we want information about the friendship between David Bowie and Iggy Pop. After *<David Bowie>* has been chosen as the first entity, the prefixes "po" or "pop" returns a number of former popes, which are globally seen more important as the good old friend of David Bowie. Indeed, in the whole collection we will not find one article mentioning David Bowie and a pope together. In contrast, in a context sensitive search engine, the musician *<Iggy Pop>* should be ranked very high in the context of *<David Bowie>*.

2. Problem Description

In the present case, we are collecting news-articles from over 500 news-feed all around the world since 2013. Using AIDA [3] as disambiguation engine, we identify the entities mentioned in the news-articles, as well as their position inside the article. Figure 2 shows the ER-model of the relationship between the news articles and the included entities. The attribute `description` of the entity "Entity" contains the label of the entity, as it is displayed for the auto-suggestion (see Figure 1). Typically, this is not the representation in the text, which differs in general (i.e. "University of California, Berkeley" vs. "UC Berkeley"). The appearance in the text is represented by the attribute `name` of the `n:m` relationship, together with the attribute `position`, which represents the position of the entity in the news text.

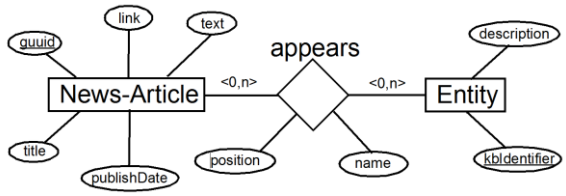

Figure 2: Relationship between news articles and included entities.

Figure 3 shows an extract from the relational table representing the relationship between `News`, `Article` and `Entity` from Figure 2, without the attribute `name`, which is not relevant for the calculation of relatedness. The data from Figure 3 is the main data-basis for our suggestion-system. For example having already specified the entities *<National_Security_Agency>* and *<Hong_Kong>*, a possible suggestion for the prefixes 'ed' or 'sn' would be the entity *<Eduard_Snowden>*, because there exists at least one news-article (with ID 1), which includes these three entities.

news_fk	entity_value_fk	position
1	<Edward_Snowden>	0
1	<The_Guardian>	97
1	<National_Security_Agency>	115
1	<National_Security_Agency>	203
1	<Edward_Snowden>	221
1	<Hong_Kong>	441
1	<Edward_Snowden>	452
1	<National_Security_Agency>	681
1	<Edward_Snowden>	885

Figure 3: Relation containing information about entities appearing in news articles

So, what we need is a function `ranked_entities = f(entities,prefix)` which performs the following task:

TASK: Given a number of previously chosen `entities` and a `prefix`, it will suggest related entities (`ranked_entities`), so that the result set containing news-articles is not empty. The suggested entities, should be ordered by decreasing relevance for the given entities. In the case of no prior specified entities, the returned entities satisfying the prefix condition are sorted by a global measure as discussed in Section 1.

In the rest of this paper, we will now discover how these suggestions can be calculated and stored accordingly to a number of constraints. The main contributions of this paper are:

- Providing a quantitative estimation how entities in a text corpus are related.
- Presentation of a new relatedness measure for entities based on the co-occurrence of entities in a single document within a specific range.
- The development of technologies how the relatedness information can be stored and

accessed accordingly, due to hard time constraints (t_{max} < 0.1 sec.).

3. Entity Relatedness Measure

Following our argumentation before, two entities can be considered somehow related, if they appear inside the same document. To fulfill our goal to avoid empty result-sets, a first approach can be to build an entity-document matrix, which allows the calculation of related entities. Figure 4, gives an example of such a matrix.

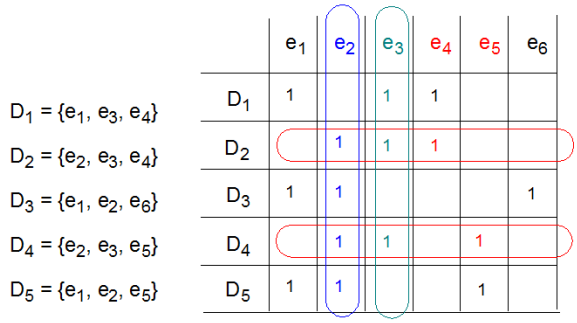

Figure 4: Entity-Document Matrix

First of all, on the left of the Figure, there are documents D_1, …, D_5, containing some of the entities e_1, …, e_6. The right side displays the corresponding entity-document-matrix. The matrix contains a row for every news-document and a column for every possible entity. A value of 1 inside the matrix indicates that the document contains the entity. By this every entity can be presented as a bitvector (the columns), with the n-th component set to 1, if the entity can be found in the n-th document, otherwise the component is set to 0. An inverted index, as it is known from Information Retrieval (IR), is typically built this way [4]. Additionally, we also have a bitvector for every document, indicating which entities can be found inside the document (row). Assuming, that entity e_2 (blue) and e_3 (green) are already given, we can calculate possible suggestions, by performing an AND-operation along the involved entities (the columns). The result (red) of this operation is a bitvector (bottom line of Figure 4) which indicates that all possible entities can be found in the documents D_2 and D_4. These documents now form the base for further possible entities, which can be found inside these documents (and only in these). So in our example, entity e_4 appears together with the given entities in document D_2, while entity e_5 appears in document D_4. The extraction of these entities can easily be done by an OR-operation with an additional XOR to remove the already selected entities from the final list.

This data-structure is appropriate, if only a small number of entities (not more then 5-10) qualifies for the suggestion of one or more given entities and for a prefix of at least one character. In this case, the entities can be presented as suggestions, without a special order. If we have cases where there are more than a handful of suggestions, we need a ranking model to select the most probable entities for suggestion. A simple extension of the previous model would be to not only indicate if an entity can be found inside a news-document, but also how often it appears there. Figure 5 gives an example of this approach. In contrast to the previous concept, every document vector (row) contains the information how often an entity appears in the document. The relatedness value r_x for each entity e_x can then be calculated on base of the cardinalities, as it is shown at the bottom of Figure 5 for the two entities e_4 and e_5. In this case, the value is simple calculated, by multiplying the cardinalities of the involved entities, divided by the number of entities involved. If a possible entity appears in multiple relevant documents, the relatedness value is simply cumulated. The selection of possible entities is based on the same concept as in Figure 4, where only documents, containing all of the previously given entities are considered for further suggestions.

Based on this information, a ranking (relevance) value can be given for each possible suggestion entity. If the number of possible suggestions is high, only the first n entities with highest ranking values are displayed.

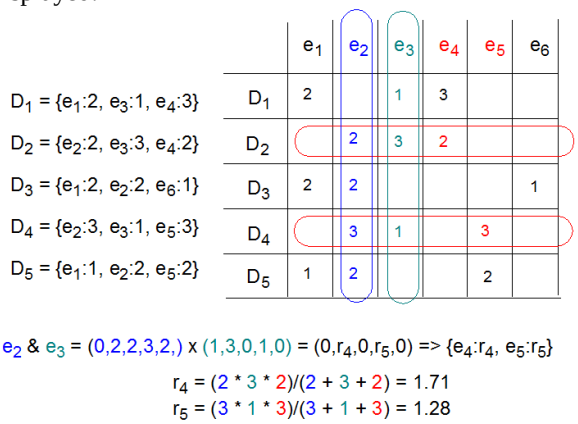

Figure 5: Quantitative Entity-Document Matrix

This approach could be further extended with the incorporation of the tf*idf value [5], as it is common in IR. A disadvantage of this approach is that it consumes far more memory than the previous model, which can i.e. operate on compressed bitmaps as described in [6].

For the implementation of our suggestion function we have the following possibilities, trading memory consumption vs. computational power:

- **Online calculation:** In this case, the suggestion entities for a (potential empty) set of given entities and a prefix are calculated online.

- **Offline calculation**: The calculation of the suggestions are done in advance at crawling time or in a batch processing step. As a result, a key-value store can be used. The key is represented by the composition of one or more entity-identifier and the prefix. This allows for a fast access to the pre-calculated entity-list, which is stored as the value.

- **Mixed calculation**: Parts of the calculation are done offline, while other parts are done online. This is often a compromise between the two previous solutions. In the present case for example, only the entities form the key and the selection of entities satisfying the prefix is done online.

To get an idea about the computational effort needed and the memory consumption, we performed a detailed quantitative examination of our dataset in the following sections.

4. Quantitative Aspects

YAGO [7], the knowledge base used by AIDA, has about 5 million different entities in its knowledge base. Up to now, 4.8 million news articles have been indexed so far, in which over 800.000 different entities appear. The total number of found entities is about 95 million.

4.1. Distributions

How these entities are distributed over the news articles is presented in Figure 6 and 7. Figure 6 shows how many different entities appear in a certain news article. The x-axis represents the individual news articles and the number of different entities inside the individual news-articles are displayed on the y-axis. The news articles on the x-axis are further sorted by the number of contained entities (in decreasing order). As we can see, about 1000 (from over 4.8 million) news-documents contain more than 100 different entities and about 20% contain more than 10 different entities (consider that the x-axis is logarithmic).

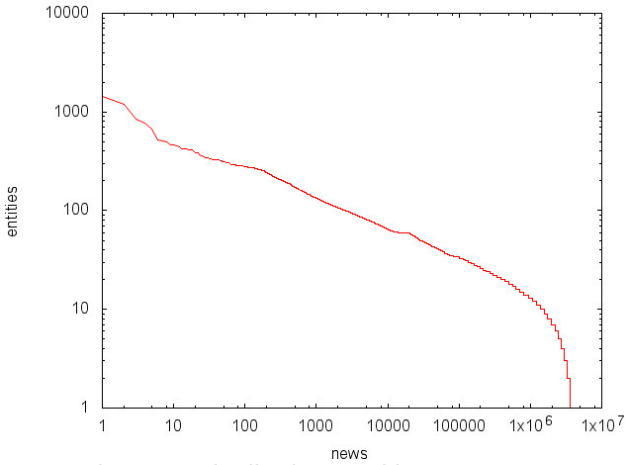

Figure 6: Distribution: Entities per News

The average number of entities is about 10 entities per article. The maximum of different entities found in a news article amounts up to 1450 entities.

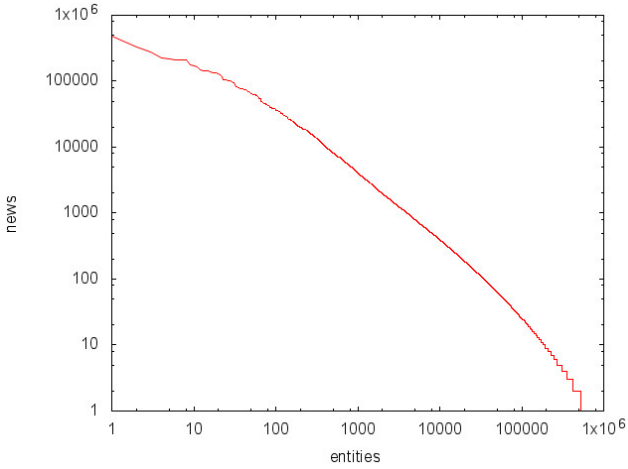

Figure 7: Distribution: News per Entity

In Figure 7, the distribution of entities in the news articles is presented. The x-axis represents the individual entities, sorted by the number of documents they appear in (y-axis). The values differ from over 900,000 news per entity for highly popular entities (i.e. `<United_States>`), up to 237,000 entities which appear in only a single news-article. From over 800,000 used entities, 672,000 appear in 20 or less articles. On average, an entity appears in 56 news articles.

The most interesting question is now how the entities are related to other entities, because this dominates the processing time for providing the suggestions. The distribution is presented in Figure 8 and 9.

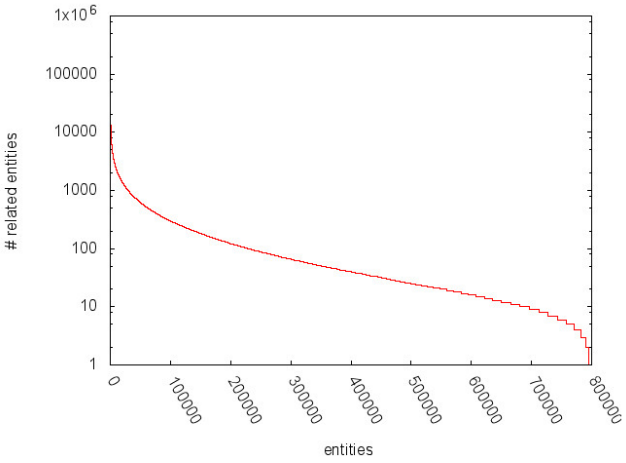

Figure 8: Distribution: Number of Related Entities for a given Entity (complete)

Whereas Figure 8 shows the complete distribution over all entities, Figure 9 only shows the first 1000 most popular entities. It becomes apparent that the number of possible related entities can be very high for a number of popular entities. So for example, the most popular entity is related to more than 412 thousand other entities. The average number of related entities is 217. Considering the average number of 217, this value is still too great to present them all as suggestion to the user. So, a ranking model is needed.

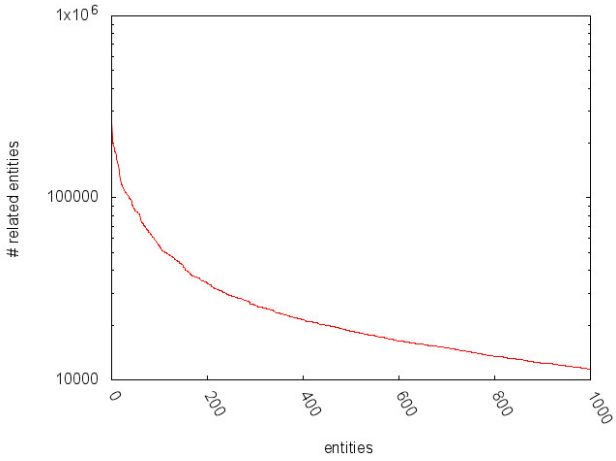

Figure 9: Distribution: Number of Related Entities for a given Entity (first 1000)

Since the suggestions are also based on a given prefix, it is also interesting to look at how the entities are distributed along the different prefixes. Figure 10 gives an overview on this distribution. We made experiments with different prefix lengths, starting from 1 up to 4 characters. All experiments were performed without a previously given context entity.

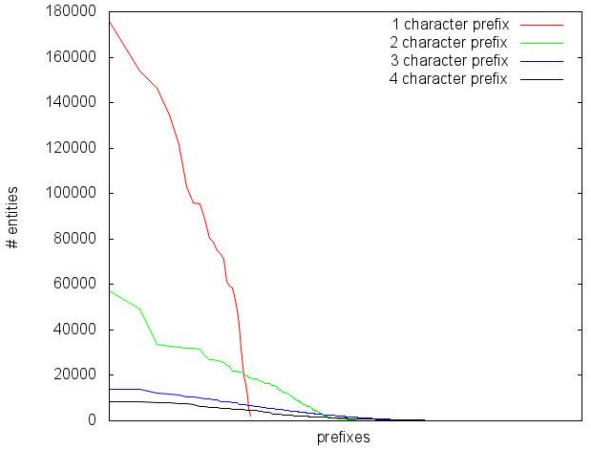

Figure 10: Distribution: Entities per Prefix

With a 1-character prefix, we got the red distribution, starting with about 175,000 entities for the character 's' up to only 2100 entities for the prefix 'x'. The 2-character prefix has 57,200 entities at most ('ma'), while the 3-character prefix starts with about 21700 entities.

Next, we examined, how the number of used entities and used entity-tuples increased over time. This represents the situation when one resp. two entities are already given in the search context. Here, we cumulated the number of distinct entities resp. entity-tuples found in the news articles. Figure 11 shows the case for a single entity.

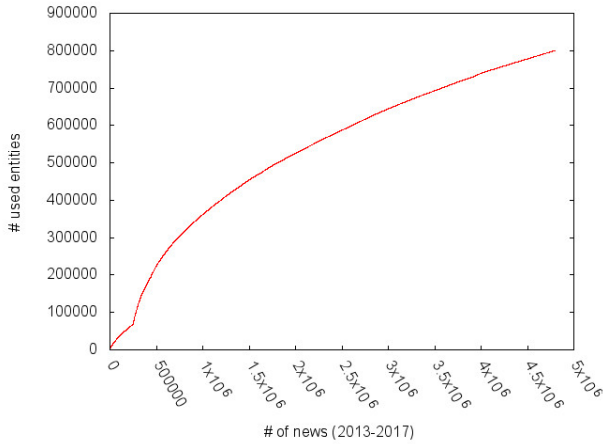

Figure 11: Increase of number of used entities over time

With increasing numbers of processed news the number of entities also increase which was no surprise for us. However, we could detect a slight saturation. A natural limit are the about 5 million entities from the YAGO knowledge base. Actually only about 800,000 entities are found inside the news corpus. In contrast, this saturation behavior does not appear in the case of

the examined entity-tuples (Figure 12). But the main problematic point still remains:

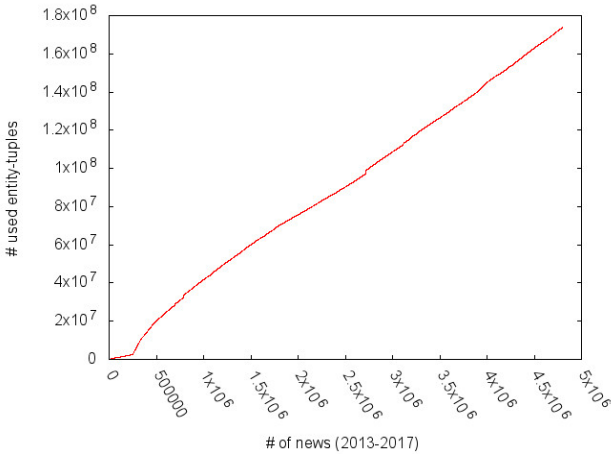

Figure 12: Increase of number of used entity-tuples over time

Overall, 848,943,610 entity-tuples were found in our corpus, from which 173,829,762 were different. The representation of this information in our MySQL database consumes (including an index) already about 20 GB of data. The storage of entity-tuples, triples, quadruples, etc. ((e_x, $e_{y,\ ...}$) -> (entity-list)) and their related entities would need exponentially more memory. The reason for this can be explained with a simple example: Consider a single news article with 100 different entities. It is a rather large article but by far not the largest in our corpus (see Figure 6). Even for this single news-article we have the following related entities as displayed in Table 1:

Table 1: Number of related entities in a single news article with 100 entities

	# of related entities
Single entity	100 * 99
Entity Tupel	100 * 99 * 98
Entity Tripel	100 * 99 * 98 * 97
Entity Quadtrupel	100 * 99 * 98 * 97 * 96
Entity Quintuple	100 * 99 * 98 * 97 * 96 * 95

Exploiting the symmetric characteristic of the relationships, we still have in total 1,283,975,715 relationships in this single article. In contrast, a news article with 10 different entities (which represents the average) is not a problem. In that case, we only have about 36,000 possible relationships. The problem remains that we can't get rid of these longer articles.

4.2. Consequences

After these experiments, we can postulate the following findings:

1. The number of possible related entities for a given entity is in many cases too large to be presented as suggestions. For this reason, a ranking function is essential to display only the most related entities to the user.
2. The real-time calculation of entity-suggestions for entity-tuples, triples, etc. is too expensive if entities are involved that are related to many other entities.
3. The huge amount of existing relationships between entities, entity pairs, triples, quadrupels, etc. and their associated lists of suggestions prohibits the possibility to store the pre-calculated results.
4. The problem with the huge amount of possible results increases with the number of found entities in a document. So typically, longer documents worsen the situation even more.

5. Sliding Window Approach

The main problem is the amount of found entities in a document. Having `n` distinct entities in an article means that there exists n choose k relations for `k` given entities, which can't be handled for even small numbers of k, if n is in the size of 100 or larger. A solution for this problem is the reduction of related entities. So far, we considered entities to be related if they appear in the same document. This is the approach as it is known from IR, but in our case, a more reduced approach would probably also be adequate. In a news article, which typically is short and thematically focused on one or a small number of topics, it might be a realistic approach to consider entities, which co-occur in a news article as related. But even here one can argue that with increasing distance in the text between two found entities the relationship shrinks. On the other hand, if two entities appear a number of times quite close together in a text, they can be considered as strongly related. As a first improvement of our measure, we can therefore consider the distance in words as a factor for the calculation of relatedness, which makes our measure more accurately So for example in [8], the relatedness between two entities is calculated as $rel(e_x, e_y) = log(1/d)$, where `d` is the distance in words between the two entities. And because only a small number of suggestions should be displayed, weaker relationships can probably be neglected. So, in a second step, we define a threshold for the maximum distance in words we want to consider. This can be seen as a sliding window over the text. The size of the window can differ with respect to

the n-tuple we are searching. So for example, the used windows size for an n-tuple can be calculated as:

```
WINDOW_SIZE = 10 * n
```

This means two entities must appear inside 10 words, while quintuples must appear inside 50 words. The sliding window technology is also the key for a huge reduction of entities, which have to be considered as related. Even more, the efficiency of the sliding window technology is independent from the size of a document. It reduces the number of possible related entities, independent of the size of a document.

In the following section, the sliding window approach will be applied on our complete news dataset.

5.1. Quantitative Aspects

We experimented with different window-sizes, and finally found a window size of `30 * sqrt(n-1)` words for `n`-tuples to be an good compromise between recall and storage requirements. In Table 2, you can see the number of n-tuples, with respect to different window sizes.

Table 2: Number of datasets

Window Size	Tuple	3-Tuple (Triple)	4-Tuple	5-Tuple
15	1,4 Mio	0,4 Mio	0,14 Mio	71 T
30	3,8 Mio	3,8 Mio	3,0 Mio	3,4 Mio
45	5,6 Mio	8,9 Mio	11 Mio	17,8 Mio
60	7,1 Mio	15,6 Mio	25 Mio	53,3 Mio

Figure 13 and 14 show the distribution of related entities for a given single entity. Because of the logarithmic scale on the y-axis, Figure 14 covers for better visibility only the first most critical 1000 entities.

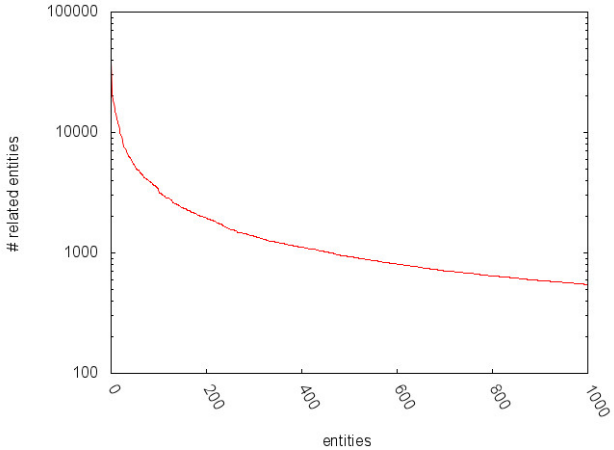

Figure 14: Distribution: Number of Related Entities for a given Entity (first 1000)

The first 20 entities have values between 53331 and 9782 related entities. The average number of related entities is 12. This is a reduction of a factor of eight for the problematic entities, compared to our prior approach from Figure 8 and 9. This makes it possible to simply store the list of related entities and filter the most relevant according to a given prefix at runtime.

Finally, Figure 15 also shows the 1000 entity-tuples, which have the most numbers of relationships to other entities. In the extreme case, this is about 4500 entities.

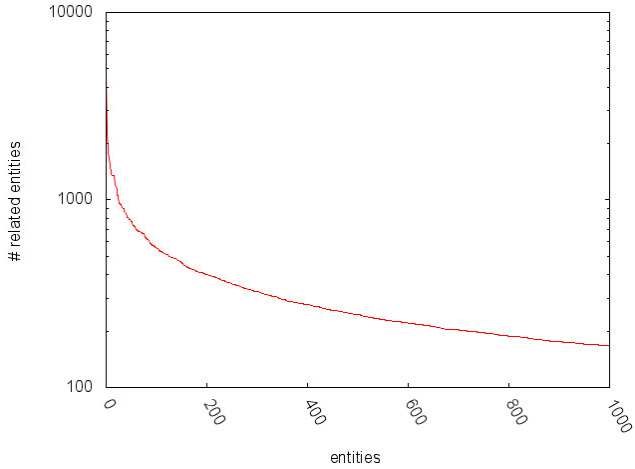

Figure 15: Distribution: Number of Related Entities for a given 2-Tuple of Entites (first 1000)

5.2. Consequences

In comparison to the first approach from Section 4, where all distinct entities in a news article are considered related resulting in an unmanageable amount of possible suggestion-entries in our database, our new approach uses a sliding window technology, which dramatically reduces the number of possible suggestions. This allows us to store preprocessed lists of suggestions for single entities, tuples, triples, quadruples and quintuples. Due to the fact that we only provide suggestions for further search entities, the focus on most probable entities is tolerable. Nevertheless, we also implemented a fallback mode, in case the given combination of entities and the prefix doesn't deliver any suggestions. If some boundary conditions about the expected cardinalities are fulfilled, a pure online search is performed. This is explained in the next section in more detail.

6. Implementation Aspects

Due to the reduction of related entities, a precomputation of the ranked entity-lists can be done. The offline computation of the relatedness values are

performed with a MySQL database. Starting point is the table `doc_entity` from Figure 3, which contains all the necessary information about the news articles, the found entities and their positions in the text. As an example, Figure 16 shows the SQL-script, which computes the relatedness between entity-2-tuples and their related entities, together with a relatedness value. At the beginning of the script, the sliding window size is calculated. The core of the statement is a multi-self-join over the table `doc_entity`. The join condition is the id (news_fk) of the incorporated news-documents and in the where condition we check the uniqueness of the entities in a dataset as well as the maximum distance (window-size) the entities are allowed to have. Since a tripel, consisting of an entity-tuple and a related entity, can appear multiple times with different relatedness values, we merged these tripels by cumulating the relatedness value (sum, group by). The `greatest`()-function calculates the actual distance defined by the position of the entities. Here we simply look for the greatest difference of two involved entities. The relatedness measure is then calculated by applying the `log()`-function on the inverse distance value. At the end, additional indexes are generated. The first index is responsible for the runtime queries returning the suggestions based on previously given entities. The incorporation of the weight attribute is twofold: First of all, it allows to quickly return the related entities by relevance and secondly, it allows so called index-only queries. Because all needed information is encapsulated in the index, the database does not have to access the table at all. This can speed up the response to a great extend provided that the index fits into the memory, but the table resides only on disk.

As mentioned in the previous Section 5.2, it is possible to specify entities which are not suggested be our suggestion function `f(entities, prefix)`. In this case, the already specified entities are examined with respect to the number of possible related entities. This can be performed quite fast, because it can be easily calculated offline from the table in Figure 3. The entries have the following format:

```
(Entity-ID, # of related entities)
```

The table has an index on Entity-ID, so estimations about the cardinality can be provide very fast. If the returned values fulfill some conditions, a slight variant of the query from Figure 16 can be performed online. Because the id-values for the n given entities can be provided, the computation is very fast.

```
set @BASE_DISTANCE = 30
set @max_dist = $(BASE_DISTANCE)*sqrt(2);

drop table if exists cooc3;
create table cooc3
engine=$(ENGINE)
    as
SELECT a.entity_fk as e1,
       b.entity_fk as e2,
       c.entity_fk as related,
       sum(1/log2(greatest(
                    abs(a.position-b.position),
                    abs(a.position-c.position),
                    abs(b.position-c.position))))
         as weight,
       count(*) count
  FROM doc_entity a
  JOIN doc_entity b
    ON a.news_fk=b.news_fk
  JOIN doc_entity c
    ON b.news_fk=c.news_fk
 WHERE a.entity_fk < b.entity_fk
   AND a.entity_fk != c.entity_fk
   AND b.entity_fk != c.entity_fk
   AND abs(a.position- b.position )<=@max_dist
   AND abs(b.position- c.position )<=@max_dist
   AND abs(a.position- c.position )<=@max_dist
 GROUP by a.entity_fk, b.entity_fk, c.entity_fk
 ORDER by 1,2,3,-1;

create index cooc3_e1_e2_weight_idx
          on cooc3(e1, e2, weight);

create index cooc3_e1_e2_e3_idx
          on cooc3(e1, e2, related);

create index cooc3_e3_idx
          on cooc3(related);
```

Figure 16: SQL-Code for the Computation of Relatedness values (2-tuple -> entity-list)

6.1 Maximum Number of Possible Search Entities

According to statistica.com [9], in February 2017, the percentage of one word key-phrases was about 35% off all queries, followed by 25% for two word key-phrases and 18% for three word phrases. Considering queries up to 6 words, we have a coverage for nearly 97% of all queries formulated. According to these numbers, we decided to support queries up to six entities. We further argue that using disambiguated entities as input for a search engine is semantically more powerful and precise, compared to a keyword search with the same number of words, so the value of six entities at most seems sufficiently enough.

7. Further Improvements

Figure 1 gives an example for the suggestion of entities, specified by the prefix ‘unive’. It emerged clearly that in a number of cases the specification of an

entity by only one prefix is not constructive since a lot of entities have the same prefix (like in the case of the universities all over the world). An improvement allows now for the specification of multiple prefixes for the desired entity. Figure 17 shows an example with three prefixes. In this case, suggestions containing the word "`east`", as well as two additional prefixes ("`ind`", "`c`") are displayed. The last entered term is always considered as a prefix, while prior written words have to be specified with an asterisk-sign (like with `ind*`) to be handled as prefix.

Figure 17: Improved Auto-Suggestion based on multiple words and prefixes

8. Summary and Outlook

We presented a comprehensive analysis of entity-relatedness in a big news article corpus. The aim was to get an understanding of the quantitative relationships between multiple entities. Our research was driven by the requirement of providing a mechanism for a context-sensitive auto-suggestion and completion system for an entity-based search engine. Starting from the traditional approach of IR, where two words, resp. entities are considered related if they appear in the same document, we demonstrated that this approach is not feasible for context-sensitive entity-suggestion, due to the immense solution space required when multiple entities are present. As an improvement, we do no longer consider a whole document (or article) as the container for the determination of relatedness, but rather the smaller unit of a sliding window, which is shifted over the text. Following this approach, we were able to build a performant context-sensitive auto-suggestion system. The system is a hybrid approach, based on some offline batch-processing and real-time computation at query-time.

Actually, a MySQL database was used to deliver the suggestions. The runtime behavior is sufficient enough ($t_{max} < 0.1$ sec.) and the suggestions seem quite adequate. In a productive environment with massive multiple requests, the database can become a bottleneck. Due to the query-characteristics, a key-value store would be an adequate replacement. Redis [10], a main memory key-value store, provides a special datatype called "sorted set", which can perfectly handle the entities to suggest as well as the relatedness value. One important point is that Redis, like many other NoSQL-databases, is designed to scale very well horizontally, which means that this approach can even be used for highly frequented search engines.

Another interesting research direction would be the determination of the "optimal window size" or at least to prove the robustness of the approach for a range of different window sizes. While the second approach is actually under progress and looks quite promising for our news-corpus (different corpora will be examined in the future), the second approach is still future work and also very dependent on the results of our actual robustness experiments.

9. References

[1] Silviu Cucerzan, Large-Scale Named Entity Disambiguation Based on Wikipedia Data, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 708–716, Prague, June 2007.

[2] J. Hoffart, D. Milchevski, and G. Weikum, STICS: Searching with Strings, Things, and Cats. Demo at SIGIR 2014, Gold Coast, Australia, 2014.

[3] Yosef, M. A., Hoffart, J., Bordino, I., Spaniol, M. & Weikum, G., AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables. PVLDB, 4, 1450-1453, 2011

[4] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. 1999. Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[5] Ramos, Juan, Using TF-IDF to Determine Word 6elevance in Document Queries. 2003.

[6] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. 2006. Optimizing bitmap indices with efficient compression. ACM Trans. Database Syst. 31, 1. 2006.

[7] Fabian M. Suchanek, Gjergji Kasneci and Gerhard Weikum, Yago - A Core of Semantic Knowledge, 16th international World Wide Web conference (WWW), 2007

[8] Andreas Schmidt, Johannes Hoffart, Dragan Milchevski, and Gerhard Weikum, Context-Sensitive Auto-Completion for Searching with Entities and Categories, In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval (SIGIR '16), 2016

[9] U.S. online search query size 2017 | Statistic. https://www.statista.com/statistics/269740/number-of-search-terms-in-internet-research-in-the-us/, last accessed: 10.6.2017

[10] Josiah L. Carlson, Redis in Action, Manning Publications Co., Greenwich, CT, USA, 2013