ECIR – a Lightweight Approach for Entity-centric Information Retrieval

Alexander Hold, Michael Leben, Benjamin Emde, Christoph Thiele, Felix Naumann
Hasso-Plattner-Institut
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

Wojciech Barczynski, Falk Brauer
SAP Research Center Dresden, SAP AG
Chemnitzer Str. 48, 01187 Dresden, Germany

Abstract— **This paper describes our system developed for the TREC 2010 Entity track. In particular we study the exploitation of advanced features of different Web search engines to achieve high quality answers for the 'related entity finding'-task.**

Our system preprocesses a user query using part-of-speech tagging and synonym dictionaries, and generates an enriched keyword query employing advanced features of particular Web search engines. After retrieving a corpus of documents, the system constructs rules to extract candidate entities. Potentially related entities are deduplicated and scored for each document with respect to the distance to the source entity that is defined in the query. Finally, these scores are aggregated across the corpus by incorporating the rank position of a document. For homepage retrieval we further employ advanced features of Web search engines, for instance to retrieve candidate URLs by queries such as 'entity in anchor'. Homepages are ranked by a weighted aggregation of feature vectors. The weight for each individual feature was determined in beforehand using a genetic learning algorithm.

We employed a commercial information extraction system as a basis and implemented our system for three different search engines. We discuss our experiments for the different web search engines and elaborate on the lessons learned.

I. Introduction

An Entity-Centric Information Retrieval system (ECIR) promises a huge potential for web- and business-users. We study in particular the application and value of entity-centric search algorithms for enterprise applications. Thus we motivate our approach by an entity-centric business task: Consider a business analyst of a large corporation producing industrial goods who needs to determine potential customers (leads). This is usually an extremely time consuming task. Having an ECIR-system he might for instance simply query for all competing companies and retrieve the corresponding customers for each of them. There are many other usage scenarios of an ECIR-system in an enterprise search setting.

However, a company is usually not able to index the entire Web. Thus, we focus on how to incorporate available services on the Web, especially search engines, and combine them with limited local hardware resources to provide a system that is able to provide high quality answers for the Entity Tracks' *"find related entities"-(REF)* task.

A task of the entity track as part of the TREC 2010 challenge is to provide a system that finds all entities with their homepages that are in a relation – described in natural language – with an input entity. In our work, we denote these entities as *target entities* and use the term *source entity* for referencing the input entity.

A. Input

The input of our system is a list of queries (*topics*) in an XML format. Listing 1 shows an example query that is used across the paper.

```
<query>
<num> 40 </num>
<entity_name> Costco </entity_name>
<entity_URL> cw09-en0006-60-20817 </entity_URL>
<target_entity> organization </target_entity>
<narrative> Find homepages of manufacturers of
LCD televisions sold by Costco. </narrative>
</query>
```

Listing 1: Topic 40 from Entity Track 2010

The element `num` describes the identifier of the query. The element `entity_name` contains the name of the *source entity* and `entity_URL` a clueweb ID – a reference to the homepage of this entity. The definition of `target_entity` determines the type of the *target entity*. The last element is the `narrative`, which qualifies the relation between the *source* and *target entity* in natural language.

B. *Output*

The designated output as shown in Listing 2 contains one line for each found *target entity* with the following attributes:

- topic number,
- the constant string 'Q0',
- the clueweb document id for the homepage,
- the rank position of the particular *target entity*,
- the computed score for the *target entity*,
- the unique run ID and
- the normalized entity name (optional).

```
40 Q0 cw-en0002-13-22651 1 2.65 G8 Panasonic
40 Q0 cw-en0010-32-35757 2 1.99 G8 Sony
```

Listing 2: Top 2 results for Topic 40

II. ECIR Architecture

The process of retrieving and ranking *source entities* is illustrated in Figure 1. Our system starts with a query expansion and rewriting step, which analyzes an input topic to create a keyword query for the configured Web search engine and to preprocess the query for the extraction task. The document retrieval component queries for each topic a specified search engine and stores the resulting web pages to the file system. The acquired document corpus and the preprocessed topic are the input for subsequent processing steps.

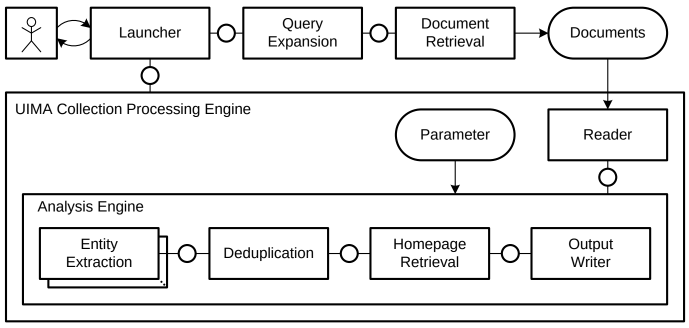

Fig. 1. System Overview

After the document retrieval has finished, the analysis process is started. It reads the files each containing the text of a single web page. For each of the documents a data container object is created, which is passed through the process steps.

In the entity extraction phase each document is scanned for possible target entities. These are stored in the document's data container object. In the next step the entities are subjected to deduplication and are scored respectively (Section III-C). For the ranked entities, which need to have a required minimum score, the homepages are determined (Section III-D). At a last step, we map the retrieved URLs to document ids of the clueweb document corpus.

We use Apache's implementation of the *Unstructured Information Management Architecture*-standard (UIMA, `http://uima.apache.org/`) to manage the analysis process, which is implemented as a UIMA collection processing engine (CPE). The advantage of using UIMA is that components can be exchanged and developed independently.

III. System Components in Detail

In the following we introduce each component that is illustrated in Figure 1 and discuss the most important concepts in detail.

A. *Query Expansion and Document Retrieval*

The task of the query expansion and document retrieval component is to provide text documents that may contain information about the relation between *source entity* and *target entity*. This relation is described by the *narrative*. The main challenge here is to estimate the number of documents to retrieve enough input for a good recall, but not too many to achieve acceptable processing times. Further, if we retrieve too many documents, the corpus may contain many less relevant documents and thus harm precision.

Due to obvious limitations in local hardware resources, we did not build our own search engine. Instead, we use commercial Web information retrieval engines, namely *Yahoo*, *Google*, and *Bing*. In order to retrieve the most likely relevant documents, we need to generate a suitable keyword query for a search engine based on the topic description and capabilities of a search engine.

In the first step of the query expansion and rewriting we extract part-of-speech elements from the *narrative*. We are interested in finding verbs and noun groups. Verbs are important, because they qualify the relation between *source entity* and *target entity*. Noun groups often qualify the *target entity* in more detail and can be interpreted as a kind of a selection criterion. For the example topic in Listing 1, we extract the following: `homepages`, `manufacturers`, `LCD televisions`, and `sold`.

The next step is to find alternative names for the *source entity*. Thus, we expand our query to allow for variations that are popular on the Web. For this purpose we exploit the Freebase (`http://www.freebase.`

`com/`) search service. As a query we send simply the *source entity* name and retrieve alternative names. For each alternative name we form an intermediate keyword query, built from an alternative name and the *source_entity* name to determine the number of hits returned by a commercial search engine. We sort all alternative names by this number in descending order and choose the most popular four. For example, for *Costco* we obtain `Costco travel`, `Costco Wholesale`, `Price Costco`, and `Costco Wholesale Corp`.

We concatenate the nouns and verbs extracted from the *narrative* and the *source_entity* together with its alternatives to a single query. Further, we make sure that the web search engine also allows for synonyms of the tokens derived from the *narrative*. For instance in the case of Google, we add a tilde as a token prefix. Thus, in case of our example and Google as Web search engine, we produce the keyword query:
`~sold ~homepages ~manufacturers ~LCD ~televisions ("Costco" OR "Costco travel" OR "Costco Wholesale" OR "Price Costco" OR "Costco Wholesale Corp")`.

We send the query to a search engine and download the first k websites in the result list. These websites are aggregated to a document corpus, which is used for further processing. Note that we run experiments with varying k and studied its influence (see Section IV).

B. Entity Extraction and Relation Finding

Similar to [1] our system employs a commercial fact extraction framework, namely the SAP BusinessObjects XI Text Analysis system. In order to extract possible relations, i.e., the occurrence of a *source entity* together with the predefined target entity type, we construct an extraction rule for each topic that is denoted as *CANDIDATE* rule in Listing 3. The rule shall extract the sentences where the *source entity* occurs, together with potential *target entities* in its context. The *CANDIDATE*-rule combines three other rules (see Listing 3):

Source: The *source entity* and the most popular alternative names are encoded in the *SOURCE*-rule to recognize the *source entity*. For topic 40 as shown in Listing 1, we include $\langle$Costco$\rangle$ (where $\langle\cdot\rangle$ denotes a token) and the alternative names of "Costco" such as $\langle$Costco$\rangle\langle$travel$\rangle$ and $\langle$Costco$\rangle\langle$Wholesale$\rangle\langle$Corp$\rangle$. The respective tokens of one (alternative) name may occur in different order in a sentence.

Target: The *TARGET*-rule captures the predefined target entity type. We leverage a subset of the 35 predefined key entity types and map them to entity types that are addressed in the REF-task. In Listing 1 [TE ORGANIZATION]$\langle\rangle$+[/TE] (where the [TE]-tag describes the reference to a predefine entity type of the extraction framework) targets the entity type "organization'. The underlying general purpose rules, e.g, for ORGANIZATION, were developed by a team of linguists and shipped with the fact extraction framework.

Context: The most relevant terms from the narrative relation description are encoded in the *CONTEXT*-rule. Therefore, we consider each token that occurs within a noun or verb group in the *narrative*. To support a broader range of linguistic variations, our system embeds the stemmed tokens into stem-tags. Thus, the *CONTEXT*-rule fires if at least one token shares the same stem with a relevant token of the narrative query.

The *CANDIDATE*-rule combines the previously described rules in one statement. Thus, we extract each paragraph that contains at least a word group as defined in the *SOURCE*-rule, the target entity type (see *TARGET*-rule), and at least one token from the *CONTEXT*-rule.

For each document we extract the paragraphs (i.e., the facts consisting of *SOURCE*, *TARGET*, and *CONTEXT*) that might fulfill the selection criteria as described in the user query. In the following we denote the sentence in the paragraph that contains the source entity as c (candidate sentence). We further define a proximity threshold for target entities (distance in sentences to c) that are treated as potentially related. Other entities of the specified target type are ignored. For entities that occur in the left context of c we define a threshold of $t_{\text{pre}} = 1$ and for subsequent entities a threshold of $t_{\text{succ}} = 3$. In general we assume $t_{\text{pre}} < t_{\text{succ}}$, because it is more likely that a related entity occurs in the right context of c. For ranking entities, we keep the name of the target entities, the distances to a candidate sentence and the rank of the document in the corpus.

Note that the information extraction system also supports co-reference resolution. Thus, we may extract for instance the most likely person, that is referred for instance by "he" based on the linguistic structure of a text. The information extraction system returns in such cases the most likely alias in the context (e.g., the person name).

C. Deduplication and Ranking of Entities

Having extracted all possible candidate facts, our system merges duplicated target entities and scores duplicate groups. To identify potential duplicates, we combine the Jaccard and Jaro-Winkler similarity (see [2]). The Jaccard similarity is a token-based vector-space similarity,

```
#group CANDIDATE (scope="Paragraph"): [UL]%(Target),%(Source),(%(Context))[/UL]
#group SOURCE (scope="Sentence"): (<Costco>|<Costco><travel>|<Costco><Wholesale><Corp>|..)
#group TARGET (scope="Sentence"): [TE ORGANIZATION]<>+[/TE]
#group CONTEXT (scope="Sentence"): (<STEM:sold>|<STEM:LCD>|<STEM:manufacturer>|..)
```

Listing 3: Simplified Extraction Rule for Topic 40

while the Jaro-Winkler similarity is a character-based edit distance, which was especially designed for name matching tasks since it punishes errors at the ending of strings less than at the beginning. The combination was necessary, because we realized that, e.g., Jaccard performed well for organizations, but caused many false negatives for certain person names. In turn, Jaro-Winkler caused many false positives for entities of type organization but is suited well for persons.

In order to combine the advantages of both similarity metrics, we aggregate them as follows: Let $\mathrm{jac}(\varepsilon_1, \varepsilon_2)$ be the Jaccard similarity and $\mathrm{jar}(\varepsilon_1, \varepsilon_2)$ the Jaro-Winkler similarity between two potential target entities ε_1 and ε_2. Thus we compute their similarity $\mathrm{sim}(\varepsilon_1, \varepsilon_2)$ by their average as:

$$\mathrm{sim}(\varepsilon_1, \varepsilon_2) = \frac{\mathrm{jac}(\varepsilon_1, \varepsilon_2) + \mathrm{jar}(\varepsilon_1, \varepsilon_2)}{2} \qquad (1)$$

Our system starts with an arbitrary entity ε in the result list, creates a duplicate group E by testing the set of not yet processed entities. It recursively extends E to determine the transitive closure of ε with respect to $\mathrm{sim}(\varepsilon_1, \varepsilon_2)$. The procedure proceeds until every entity is assigned to a duplicate group. For each group E of entities we select the longest name from its assigned instances as its representative.

Our scoring algorithm works as follows: Let $\mathrm{dist}(\varepsilon, c)$ be the distance in sentences between a target entity ε relative to a candidate sentence c. Having identified potential duplicates, the scoring algorithm computes weights for each duplicate group E with respect to a document d as:

$$s(E, d) = \max_{(\varepsilon \in d, E)} \left[\frac{\mathrm{t_{max\_dist}} - |\mathrm{dist}(\varepsilon, c)| + 1}{\mathrm{t_{max\_dist}} + 1} \right], \qquad (2)$$

where $t_{\mathrm{max\_dist}} = t_{\mathrm{pre}}$ for $\mathrm{dist}(\varepsilon, c) < 0$ (occurrences in the left context of c) and $t_{\mathrm{max\_dist}} = t_{\mathrm{succ}}$ for $\mathrm{dist}(\varepsilon, c) \geq 0$ (occurrences in the right context). As result of the assumption $t_{\mathrm{pre}} < t_{\mathrm{succ}}$, we obtain lower scores for preceding entities than for succeeding ones.

To aggregate entity scores across the document corpus D, we consider the rank position k_{d} of a document d and compute:

$$s(E, D) = \sum_{d \in D} s(E, d) \cdot \log(|D| - k_{\mathrm{d}} + 1), \qquad (3)$$

where $\log(|D| - k + 1)$ ensures that potential target entities that occur in documents with a lower k are preferred over entities occurring at a higher k, because they are more likely to be relevant.

D. Homepage retrieval

The REF task also requires to output the primary homepages of the encountered entities: "A primary homepage is devoted to and in control of the entity" [3]. Our approach to primary homepage retrieval first queries one of the employed commercial search engines for a set of homepage candidates by extracting the items from the search engine's result list. Those candidates are ranked by matching their URLs, titles, and snippets[1] against the information about the entity. The ranking function works by extracting 17 features from the candidate homepage into a *feature vector* $f = (f_1, .., f_{17})$ and then applying configurable weights to the features. We designed a genetic algorithm [4] in order to find the best *weight vector* $w = (w_1, .., w_{17})$ for each search engine.

Homepage candidate retrieval: The initial candidate set is retrieved by simply querying for the *entity name*. The set is then expanded by issuing more queries that are different for each search engine. To address those differences we use a *source-specific ranking*: If a candidate has been retrieved in another way than a simple query against the main search engine, it is marked with a flag f_n that is later being evaluated by the ranking. Flagging a candidate means setting the respective feature to $f_n = 1$.

Set expansion for all search engines: For each candidate that is a *Wikipedia* page, we load the *Wikipedia* page and extract all outgoing links that do not point to

[1]Snippets are the short summaries that search engines output along with the URLs of their search results.

Wikipedia. Those outgoing links are added as candidates as well and are flagged as having *Wikipedia* as their origin (f_1). The feature “is a Wikipedia page” is discussed later on. Such a page should score less because Wikipedia pages should never be classified primary homepages.

After applying the following search engine specific set enlargement operations, for each candidate URL we add the shorter forms of the URLs to the candidates by repeatedly removing subpaths from the end of the URL.

Set expansion for Google was done by using special search operators. Each page related[2] to a *Wikipedia* candidate page was included in the set, being flagged with f_2. Furthermore, we queried for:

- `allintitle` operator with *entity name* (feature f_3)
- `allinanchor` operator with *entity name* `OR` (*entity name + narrative*) (f_4)
- `feature:homepage` operator with *entity name* (f_5).

Set expansion specific to Yahoo was implemented by additionally querying for

- `in-title` operator with *entity name* `OR` (*entity name + narrative*) (f_3)
- `feature:homepage` operator and *entity name* (f_5).

Bing also offers a set of special search operators, but we did not make use of them due to time constraints.

We deduplicated the candidate set by hashing the normalized URLs and concatenating all values (homepage title and snippet) and features to a single candidate.

Ranking the homepage candidate set: For each homepage candidate a set of 17 features is extracted into a *feature vector*. A vector of the same size contains the weights for each of these features. The *feature vector* is made up of the following items:

- f_1 to f_5: Source-specific flags (see above)
- f_6: length of URL subpath
- f_7: snippet or title contain the keywords ”official site”, ”company website” or ”official website”
- f_8: rank from source search engine result list
- f_9: candidate is *Wikipedia* site, which decreases the score
- f_{10} *entity name* contained in URL
- f_{11} *entity name* contained in title
- f_{12}: Jaccard index of trigrams between entity (*entity name*, *narrative*, expanded query[3]) and homepage (title, hostname, snippet)
- f_{13}: reciprocal Levenshtein distance between *entity name* and homepage hostname
- f_{14}: Jaccard index of bigrams between *entity name* and homepage hostname
- f_{15}: Jaccard index of trigrams between entity (*entity name*, *narrative*) and homepage (title, snippet)
- f_{16}: Jaccard index of trigrams between *entity name* and the entire homepage URL
- f_{17}: Jaccard index of terms between entity (*entity name*, *narrative*, expanded query) and homepage (title, host, snippet)

Once a *feature vector* for a candidate homepage has been determined, each feature value is multiplied with the corresponding weight of the *weight vector* of the selected search engine, to calculate the final score $s(p)$ for candidate page p:

$$s(p) = \sum_{i=1}^{17} (f_i \cdot w_i) \tag{4}$$

The candidate page p with the highest score $s(p)$ is selected to be the primary homepage.

Tuning the homepage ranking with a genetic algorithm: The individual importance of the features above is not obvious. For this reason, a genetic algorithm was employed to find the best *weight vector* for each search engine. The genetic algorithm is initialized with a set of randomly created *weight vectors* that are constantly modified by mutation and recombination and evaluated by a fitness function. The poorest *weight vectors* are removed at the end of each iteration (selection).

Mutation is used to slightly modify weight vectors in order to climb up a (potentially local) maximum. Weight vectors are modified by slightly increasing or decreasing all their values by different random values.

Recombination means to combine good solutions in order to merge them into a better one, taking advantage of the “good parts” of both. As the best recombination method is unknown, we randomly selected one of the following three methods, to derive a new *weight vector* w from two given vectors a and b:

$$\text{crossing: } a \times_9 b = (a_1, .., a_8, b_9, .., b_{17}) \tag{5}$$

The cutting index 9 was intuitively selected and might not be the best choice.

$$\text{interweaving: } a \diamond b = (a_1, b_2, .., b_{16}, a_{17}) \tag{6}$$

$$\text{averaging: } a \circ b = \left(\frac{a_1 + b_1}{2}, .., \frac{a_{17} + b_{17}}{2}\right) \tag{7}$$

[2]*Google* query: ”related:http://en.wikipedia.org/wiki/*Articlename*”
[3]expanded query for document retrieval (defined in Section III-A)

To determine the quality of a weight vector w the genetic algorithm uses as *fitness function* the mean reciprocal rank (MRR) that was derived using a training data set. Let $k(w, \varepsilon, p)$ be the rank position of a Web page p within a list of ranked pages, obtained using the *target entity*, denoted as ε, and a weight vector w. For a training set $T = \{(\varepsilon, h)\}$ containing a set of pairs of *target entitiy* ε and correct homepage h, the MRR is given by

$$MRR(w, T) = \sum_{(\varepsilon,h)\in T} \left(\frac{1}{k(w, \varepsilon, h)} \right) \cdot \frac{1}{|T|} \qquad (8)$$

As T, we used a handcrafted test set of 30 *target entities* and their "correct results". We ran the genetic algorithm until it could not find a better *weight vector* w within certain time constraints.

Learned *weight vectors*: The genetic algorithm yielded weight vectors with the following MRRs: *Google* 0.87; *Yahoo*: 0.80; *Bing*: 0.82. Note that for *Bing*, no vector better than the *Google* vector was discovered after a processing time of 24 hours. A longer time might have lead to a better weight vector. Figure 2 shows the differences between the best *Yahoo* and the *Google* vector.

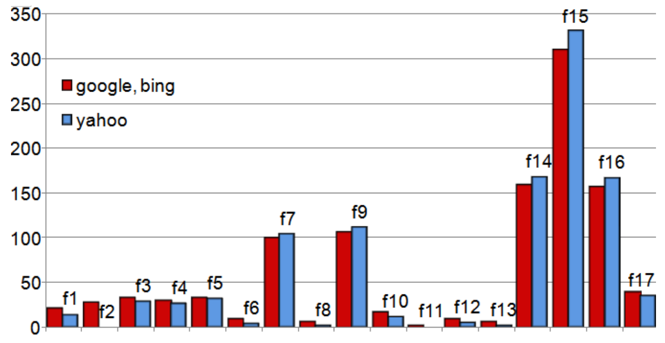

Fig. 2. Differences between best *weight vectors* for different search engines. The *Google weight vector* relies on operators (f_1 to f_5), while the *Yahoo* setting makes more use of text similarity (f_{12} to f_{17}). The prominent feature f_{15} is the trigram Jaccard index (see [2]) of homepage snippet, title and *entity name*, *narrative* concatenated.

IV. Evaluation

During the development of our system we used topics from 2009 to determine which parameters settings could perform the best for TREC 2010 queries. In this section we will focus on the final results of our system in TREC 2010.

The two basic configuration parameters for tuning our system are the selection of the Web search engine – used for retrieving a document corpus and homepage candidates – and the number of retrieved documents, which determines the size of the corpus. The search engines we used were *Google* (G), *Bing* (B), and *Yahoo* (Y). We varied the number of retrieved documents from 8 to 128. We obtained the best results for *Google* with 16 documents (denoted as *G16*). For the remaining search engines a configuration with 64 documents turned out to be more precise. Thus, we also submitted runs for *Y64* and *B64*, and for comparison *G64*.

RUN	nDCG ALL	P10 ALL	MAP	Rprec
G16	0.0745	0.0702	0.0357	0.0539
G64	0.0625	0.0574	0.0252	0.0500
Y64	0.0222	0.0149	0.0055	0.0223
B64	0.0178	0.0128	0.0044	0.0122

TABLE I
OFFICIAL AGGREGATED RESULTS FOR TREC 2010

The main performance measures in TREC 2010 are normalized discounted cumulative gain (*nDCG*) and precision at 10 (*P10*). In addition, the evaluation script provided numbers for the mean average precision (*MAP*) and R-precision (*Rprec*). The average results of the submitted runs are shown in Table I. The results show that *Google* provides more accurate results in the first items of the result list. Thus *G16* outperforms *G64*, because the latter returned more non-relevant documents, which harm precision. For the other search engines a higher number of documents is required to achieve an acceptable recall. However, even compared to *G64* a significantly higher amount of less relevant documents is contained in their result sets.

In Figure 3 we show the results of the different configurations for each topic from 2010 (*topic 21* to *topic 70*). In general, we observed that our system performs better for finding related persons than for other entity types. The reason is that our system does not use domain dictionaries, e.g., generated from Wikipedia. Thus, for instance domain specific location names are often missed, while more general but less relevant ones, such as "South Pacific", are extracted and harm precision.

Another problem, that was detected after official submission, was a bug in keyword query generation. This bug dropped some relevant key words from the narrative query. Therefore, we show the results of the fixed *G16*-configuration in comparison to the the submitted *G16*-results in Figure 4. Furthermore, Figure 4 depicts the best and median results of the competing systems in 2010. It shows that the bugfix led to a significant improvement for

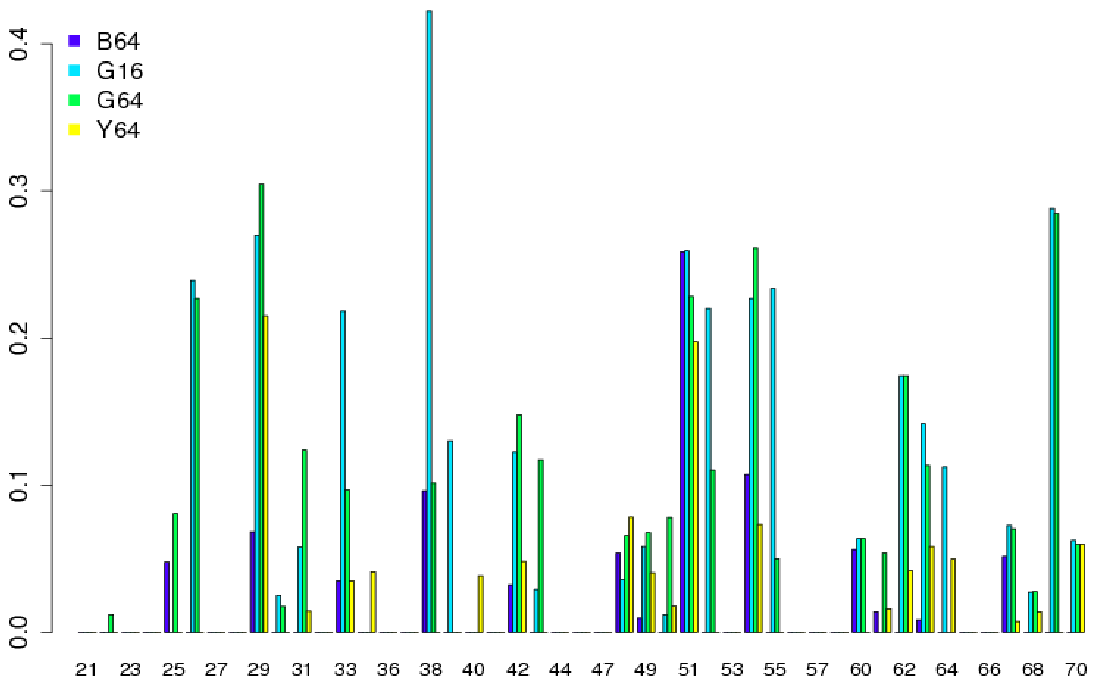

Fig. 3. Comparison of the official results from TREC 2010 for different configurations per topic ($nDCG$)

topics with extremely long narrative queries. The results for the other configurations improved in a similar way, but showed – compared to each other – a similar behavior as shown in Figure 3.

The performance was not the focus of our system. However, it has to be mentioned that the runtime processing of documents leads to huge overhead (as opposed to annotating the whole clueweb corpus in advance). Thus the time to process a single topic with a configured corpus size of 16 was about 2 minutes (without any parallelization). The system spends most of the time in the deduplication phase and the mapping from real URLs to clueweb ids.

V. Discussion and Research Directions

In the following we discuss observations with respect to our system when running the 2010 topics and elaborate on potential extensions of our approach.

Query processing and document retrieval. The first crucial part of our system is the query preprocessing step. In general, our query preprocessing seems to perform well to formulate a keyword query that returns relevant documents.

We realized in particular that some Web search engines seem to take into account the position of a token in the keyword query to determine the relevance of documents. Thus changing a token at the last position of the keyword query has little influence on the candidate documents, while changing the first token has a huge impact. Therefore, at least a filtering of the narrative query is crucial to narrow the set of tokens. Reordering tokens might further improve the relevance of retrieved documents, e.g., by a metric to determine semantic relatedness [5] to the *source entity*.

We propose two more potential extensions for our query processing component. First, we realized that some topics, e.g., topics 37 and 44 contained a selection predicate for time ("in December 2008", "during June 2008"). A special treatment of those predicates may further improve performance, e.g., by generating alternative representations. Next, some topics contained semantic type information or entities, such as topic 21 ("art galleries in Bethesda, Maryland"), topic 22 ("countries"), topic 30 ("U.S. states and Canadian provinces") or topic 35 ("companies"). Parsing them, e.g., by using the semantic categories defined in *Wikipedia*, and reusing this information in the extraction phase promises a higher accuracy.

Entity extraction. Our system performs particularly well for queries that aim to find related persons. The reason is that there is common understanding of the semantic type "person". For locations, organizations, and products the treatment of the semantic type is more subjective. For instance, topic 21 asks for locations that

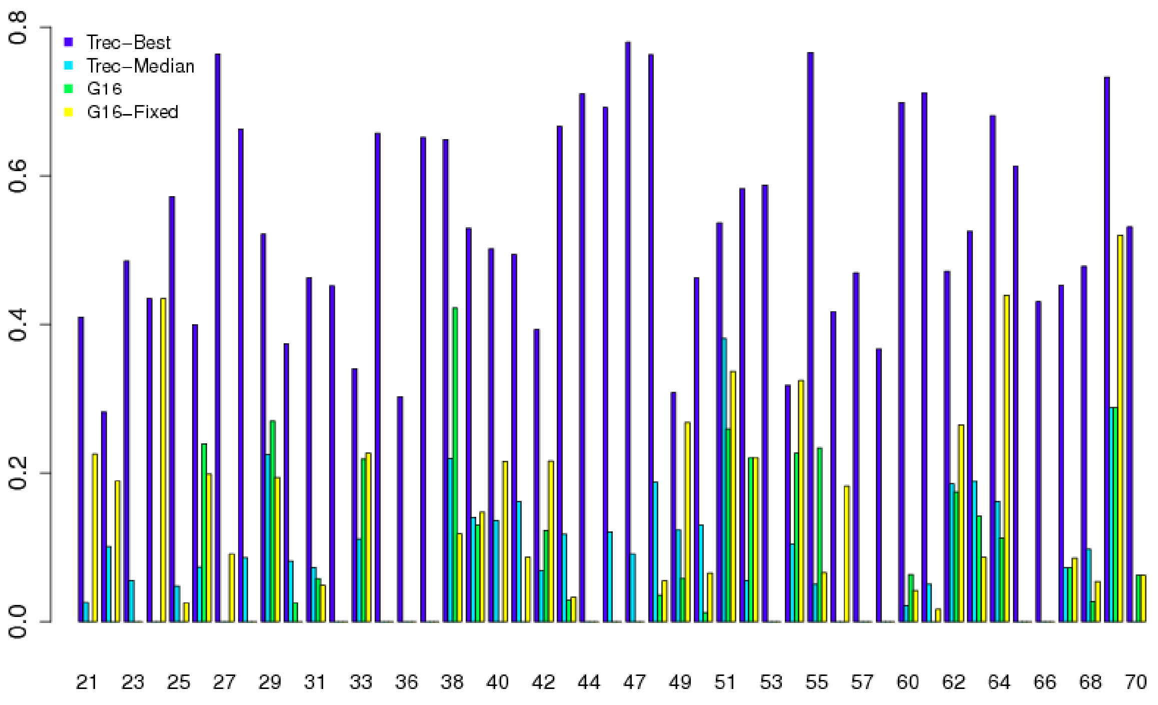

Fig. 4. $nDCG$ comparison of official G16-run / bugfixed G16-run to competing systems in TREC 2010

are specified as “art galleries in Bethesda, Maryland”. While our system retrieves documents that may answer the query, it misses all relevant entities, because our system classified occuring art galleries as “organization”, but the $TARGET$ rule (see Section III-B) is configured with the entity types for locations.

The above example is only one case where our mapping from the location entity type to the predefined entity types of the employed extraction system lowered the performance of the overall system. For instance, we mapped the “location” entity type also to the entity type “regions” of the employed extraction system (e.g., “South Pacific”). Regions often outperformed cities or countries in the ranking phase. A more careful mapping and a relevance scoring for entity sub-types promises better accuracy.

Considering entity sub-types, we may also leverage entity types that can be extracted from narrative queries. Having extracted from the query that the expected sub-type of “organization” is “university” or “company”, the extraction can be narrowed to those types as they are also available in the extraction framework.

Deduplication and ranking. The deduplication and ranking mechanisms perform quite well. However, our ranking scheme considers only statistics from the narrowed document corpus, which was acquired via a keyword query. Thus, it captures for each candidate the proximity of the source and potential target entity, and implicitly the number of occurrences and the relevance of a document. Another potential, quite simple (but time consuming) solution would be to query a Web search engine with source and target entity together with the narrative query to also include global statistics. In order to achieve reasonable processing times, we did not apply such statistics.

Homepage retrieval. We made extensive use of the *Google* operators, so the *Google weight vector* especially relies on them. If they are not available, the importance of textual similarity measures increases, with Jaccard index being clearly preferred over Levenshtein distance. The use of *Bing*’s special search operators was not evaluated at all. For further improvement, data like site structure, page content, or inclusion within certain directories (chamber of commerce, phone books) could serve for feature extraction as well. Especially the recombination operator could be adapted to better address this specific problem, for example grouping features using an evolution strategy [6].

VI. Conclusion

Our entity-centric information retrieval (ECIR) system preprocesses a given topic using part-of-speech tagging and synonym lists on the web in order to create a keyword query. The keyword query is sent to one of three evaluated Web search engines. The documents in the result list are aggregated in a document corpus, which in turn is used for further processing. Based on the preprocessed query an extraction rule is constructed to identify possible *target entities*. For each document we score each potential *target entity* by the distance to the *source entity*. Potential *target entities* are subjected to a deduplication, resulting in a set of duplicate groups. For each duplicate group, we compute a aggregated score with respect to the document corpus.

Our homepage retrieval and weighting scheme considers a set of 17 different features for ranking. It also considers special operators of the Web search engines. Interestingly, we observed that a genetic learning algorithm, trained on a small set of examples, weighted textual features, such as the Jaccard similarity between the entities and title, URL, and snippet higher than the source-specific features and special operators. However, we observed that these are extremely valuable to determine candidate pages.

We evaluated our system using three different Web search engines and varied the number of documents that were acquired. As a result, we find that our system performs best when it uses Google to retrieve 16 top pages. In this setting our system outperforms the average results that have been achieved by competing systems in TREC 2010. For some topics such as topic 24 and 54 our system outperforms the average results by far and achieves the same *nDCG* as the best performing system in TREC 2010.

References

[1] M. Kowalkiewicz and K. Juenemann, "Improving information quality in email exchanges by identifying entities and related objects," in *Proceedings of the CIKM workshop on Information credibility on the web (WICOW)*, 2008, pp. 81–84.

[2] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *Proceedings of the IJCAI Workshop on Information Integration on the Web (IIWeb)*, 2003, pp. 73–78.

[3] "Trec entity track guidelines - searching for entities and properties of entities," http://ilps.science.uva.nl/trec-entity/files/trec2010/TREC_Entity_2010_guidelines.pdf.

[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 2007.

[5] R. Gligorov, W. ten Kate, Z. Aleksovski, and F. van Harmelen, "Using Google distance to weight approximate ontology matches," in *Proceedings of the International Conference on World Wide Web (WWW)*, 2007, pp. 767–776.

[6] I. Rechenberg, *Evolutionsstrategie '94*. Frommann Holzboog, 1994.