TREMA-UNH at TREC 2017: Complex Answer Retrieval

Matt Magnusson magnusson3@gmail.com
Ben Gamari ben@well-typed.com
Laura Dietz dietz@cs.unh.edu

TREMA lab, Department of Computer Science, University of New Hampshire, USA.

1 Introduction

Our long-term goals are to develop new IR algorithms that utilize knowledge graphs and semantic annotations for composing comprehensive essays on complex questions. We participated both in the passage and entity retrieval task. Our passage retrieval runs are based on a class experiment from the course on data science for knowledge graphs and text, taught in Spring 2017. The best performing methods, BM25 and a special form of query expansion, were submitted. Our entity retrieval runs are based on a novel method to develop graph walk algorithms for knowledge graphs that can reason in every step which edges to follow to stay on topic.

2 Task Description

In the this first year, we participated in both passage and entity tasks. These are defined as follows:

Passage Task: Given complex topic outline Q, retrieve for each of its sections H_i, a ranking of relevant passages S.

Entity Task: Given complex topic outline Q, retrieve for each of its sections H_i, a ranking of relevant entities E and with support passages S. These support passage should motivate the why the entity is relevant for the query.

The passage S is taken from the provided passage corpus. The entity E refers to an entry in the provided knowledge base. A passage or entity is defined as relevant if the passage content or entity is appropriate for the knowledge article on the complex topic.

An example outline for complex topic "Candy Making" is given in Figure 1.

```
Title: Candy Making
1. History
2. Safety
3. Hard candy
        3.1. Sugar stages
4. Soft candy
        4.1. Cotton Candy
        4.2. Marshmallows
5. Chocolatiering
6. Tools and machinery
```

Figure 1: Example outline for complex topic "Candy Making".

MUST be mentioned:
`Sugar glass` is made by dissolving `sugar` in `water` and heating it to at least the "`hard crack`" stage (approx. 150 C / 300 F) in the candy making process. `Glucose` or `corn syrup` is used to prevent the sugar from `recrystallizing`, by getting in the way of the `sugar molecules` forming crystals. `Cream of tartar` also helps by turning the sugar into `glucose` and `fructose`.

CAN be mentioned:
`Sinuklob` is a very sweet candy that is also used in making `Bukayo`. It is the cheaper version of `caramel` in the Philippines. Sinuklob is made from melted `brown sugar` hardened into a `chewy consistency`.

Roughly on TOPIC but non-relevant:
Most candies are made `commercially`. The industry relies significantly on `trade secret` protection, because candy recipes cannot be `copyrighted` or `patented` effectively, but are very difficult to duplicate exactly. Seemingly minor differences in the `machinery`, `temperature`, or `timing` of the candy-making process can cause noticeable differences in the final product.

Figure 2: Example passages and relevance with marked up entities that are relevant for "Candy Making / Hard Candy" (Query ID "Candy%20making/Hard%20candy").

3 Passage Ranking

3.1 Methods

Corpus. All paragraphs were indexed with Lucene/Solr. Terms were tokenized, punctuation removed, lower-cased, and Porter stemmed for both the index and the query.

Queries. A query is constructed from each section in the outlines as follows: The section heading, page title, and all headings of parent sections were concatenated, then tokenized into individual query terms Q. For example, the section `Green%20sea%20turtle/Habitat` would be turned into the query "Green sea turtle Habitat", then case normalization, stopword removal, and stemming performed. Any sections that were listed in Phase 4 of `http://trec-car.cs.unh.edu/process/dataselection.html` were filtered out.

Baseline ranking BM25. For each section, paragraphs were ranked using the section query Q using Lucene's default BM25 model ($k1 = 1.2$, $b = 0.75$).

Query Expansion. Two separate methods were used to generate query expansion terms: 1) TagMe, and 2) other sections with same name. Both methods were tested individually but neither had better performance than the two methods combined. Evaluation of the two different query expansion methods is not included in this report.

TagMe [6] provides a web-based API that annotates plain text with entity mentions and link resolution to Wikipedia pages. The TagMe expansion consisted of having the section queries tagged by TagMe with "include_abstract" option turned on. Top IDF terms (based on corpus statistics) were extracted from abstracts for entities returned by TagMe. The goal was to identify the terms that were more distinct (rare) to help distinguish terms with semantic meaning for the given article stub and headings.

Based on ideas of earlier work [1, 14], section headings from the dataset unprocessed_train (formerly called half-wiki) were extracted. Top IDF terms (based on corpus statistics) from all sections with the same section heading as the query were extracted. For example if the outline was `Green%20sea%20turtle/Diet`, then all "Diet" sections would be extracted and top IDF terms (after stop words were removed) were added.

K was set at 25 for expansion terms after filtering out the the lowest 50 IDF terms in the passage corpus. Preference was to first include terms that occurred in both expansion techniques and then selecting approximately an even amount from each of the remaining expansion techniques. Terms that were not contained in both expansion methods were then selected based on highest IDF. Pseudocode for query expansion given in Algorithm 1.

A boosting factor (×4) was applied to the original query terms. TagMe requests and unprocessed_train section queries were cached to increase retrieval speed for any additional requests of the same entity or section.

Algorithm 1 Query Expansion.

```
expand_query(query):
        TagMe_expansion=expand_TagMe(query)
        sections_expansion=expand_sections(query)
        sect_terms=get_top_k(sections_expansion, k=25)
        TagMe_terms=get_top_k(TagMe_expansion, k=25)

        expand_terms =[]
        expand_terms.add(intersection(TagMe_terms, sect_terms))
        while expand_terms< k #k=25
                expand_terms.add_even_amounts(sect_terms,TagMe_terms)
        return query^boost_factor + k
return expand_terms
```

3.2 Evaluation on Test200 v1.4

Both approaches, BM25 baseline and expansion using TagMe and sections, were applied to the four folds in TREC CAR release-v1.4. In total, 341,688 sections from release-v1.4 were obtained and 100 rows for each section query Q using each method were obtained. Utilizing the fold's `train.hierarchical.qrel` files, we found that the average number of paragraphs were 2.34 with a standard deviation of 2.7 paragraphs per query and the distribution of relevant paragraphs exhibits a strong right skew. Analyzing the percentage of paragraphs that were relevant across the queries evaluated, 47.8% of queries had only 1 relevant paragraph, 20% had 2 associated paragraphs, 12.7% had 3 relevant paragraphs.

The baseline method retrieved 0.980 relevant paragraphs per section query on average with a standard deviation of 1.13. The average number of relevant paragraphs retrieved by the expansion method was 0.963 with a standard deviation of 1.12. BM25 returned the following percentiles of 36.3%, 43.8%, 12.1%, 4.60%, and 1.80% for 0 to 4 relevant documents retrieved respectively. In other words, BM25 returned only one relevant document for just over 40% of all queries and just under 40% of the time it did not retrieve any relevant documents. While the baseline method did retrieve slightly more relevant documents on average than expansion, they were very similar in retrieval characteristics. That they behave similar isn't surprising given that expansion featured a boosted version of the baseline query.

For the three different performance measures evaluated, R- Precision, Reciprocal Rank, and MAP, the expansion query technique outperformed the baseline BM25 ranking. Results are listed in Table 1. The standard error bar test was low for the different performance measures evaluated (ranged between 4e-4 to 5e-4) due to the large query size utilized for evaluation. The p values on the t-tests performed comparing performance measures between bm25 and expansion were effectively 0. Both of these measures support a high level of confidence that each method demonstrated a difference that was statistically significant.

The helps/hurts analysis shows that query expansion helped query performance between 1.7 to 1.8 times more than it hurt query performance. For example, query expansion helped queries in terms of map, 42,861 times and hurt them 25,532 times. This would mean also that query expansion did not hurt or help (in terms of map) 273,295 queries. Queries were divided into easy and hard by ordering queries based on BM25 performance measure on a baseline measure and then dividing into two partitions. Query expansion outperformed BM25 on both easy and difficult queries for all performance measures except R-precision for easy queries. This indicates that BM25 may be more effective for recalling relevant passages when the terms of the sections are also prevalent in the passage, but that query expansion is more effective for retrieving relevant passages when a passage has a semantic relationship as opposed to a term-based relationship.

Table 1: Passage results on Test200-v1.4.

Method	R-Precision	StdErr	MRR	StdErr	MAP	StdErr
BM25	0.149	0.0005	0.262	0.0006	0.193	0.0005
Expand	0.162	0.0005	0.279	0.0007	0.204	0.0005

(a) Performance.

	R-Precision	MRR	MAP
Help	20,273	47,712	42,861
Hurt	11,253	28,282	25,532
Ratio	1.8	1.69	1.68

(b) Helps/Hurts of expansion method over BM25 baseline.

Method	T-stat	P values
RPrec	-17.3	1.0e-67
MRR	-18.5	5.2e-77
MAP	-15.1	8.4e-52

(c) T-Test between of Expand over BM25.

	Easy		Difficult	
Method	Mean	StdErr	Mean	StdErr
BM25	0.298	0.001	0.000	-
Expand	0.286	0.001	0.038	0.0004

(d) Difficulty of section queries for R-Precision.

	Easy		Difficult	
Method	Mean	StdErr	Mean	StdErr
BM25	0.38097	0.0008	0.004	0.00002
Expand	0.398481	0.0008	0.0094	0.00009

(e) Difficulty of section queries for MAP.

	Easy		Difficult	
Method	Mean	StdErr	Mean	StdErr
BM25	0.516	0.001	0.008	-
Expand	0.541	0.001	0.017	0.0001

(f) Difficulty of section queries for MRR.

4 Entity Ranking

The entity runs are based on graph walks on the knowledge graph generated by co-mentions of entities.

The advent of fast and accurate entity linking algorithms and growth of publicly available knowledge graphs such as DBpedia [2] and Freebase [3] have led to a range of approaches that use knowledge graphs and entity links for text-centric retrieval tasks. Many of these approaches use a component that retrieves entities from a knowledge graph in response to an information need Q. While displaying these entities to the user already serves a purpose on its own, entities have been shown in many ways to help derive a ranking function for text documents [13, 10, 16, 5, 9]: by harvesting expansion words, by avoiding disambiguation mistakes through entity links, and by exploiting ontological types.

In this work, we focus on improving the entity ranking subcomponent.

Task: Entity ranking with seeds. A knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{E} \times \mathcal{E})$ of entities $e_i \in \mathcal{E}$ and their relations (e_i, e_j) is given. For a given information need Q and given seed entities E_0, the task is to return a ranking of entities by relevance.

Especially for the task of addressing open-domain information needs, common entity ranking approaches use a combination of graph structure and text retrieval. Typically, an initial set of entities are retrieved based on query matches in attributes, knowledge article text, or entity link contexts in feedback runs. These entities are expanded or re-ranked using additional indicators based on the graph structure, where entities are represented as nodes, and relations between entities are represented as edges. Common features include measures of popularity (e.g., PageRank [12]) and in-betweenness (e.g., Shortest Paths between seed entities [11]). While this approach seems reasonable at first sight, empirical analyses [4, 15] show that features based on the graph structure only contribute disappointingly small improvements. We speculate that these are due to misleading edges in knowledge graphs, and with appropriate filtering and weighting of the graph much better performance can be achieved.

4.1 Approach

We obtain seed entities E_0 by retrieving entities as text using page title and lead section of each entity's article.

The key idea of our approach is to derive a knowledge graph where edges between entities are associated with textual content. Using the section query Q, a *candidate subgraph* $\mathcal{G}_Q$ is obtained by retrieving edges through their associated content. Using this graph, an entity ranking can be obtained through graph walk algorithms based on eigenvector centrality, degree centrality, or shortest paths. A baseline is a conventional graph approach, where starting with seed entities E_0, the graph of entities and edges that are reachable within h hops are selected as a subgraph $\mathcal{G}_h$. The difference is that the query-biased subgraph is better reflecting the structure that is relevant for answering the query, where the conventional graph may contain many different aspects of the seed entities, some of which are relevant for this topic, others are non-relevant.

To demonstrate that this is a fruitful direction, we use BM25 to retrieve and rank edges based on their associated text, leaving additional retrieval models to future work. We focus on assessing the effects on the performance of entity ranking methods depending on i) different ways of *filtering* the graph with text-based retrieval and ii) different ways of deriving an *edge weight* from the retrieval score.

Several well-known graph-based entity ranking methods based on eigenvector centrality or shortest paths are included in this study. These are PageRank [12], personalized PageRank [7], AttriRank [8], and shortest paths [11]. We additionally include the degree centrality, which derives a score for the entity by summing the weights of its incident edges (MargEdges). This method has the advantage that it is fast and simple to compute.

The remaining question is how to obtain such a knowledge graph where edges are associated with text, so a retrieval engine could identify them. To this end we are constructing an alternative to DBpedia: Every paragraph is inspected for entity links, any set of co-mentioned entities is given a hyper edge, which is associated with the paragraph. For details see Algorithm 2.

Algorithm 2 Knowledge graph construction.

```
for all (sourceEntity, article) in collection do
    for all paragraph in article do
        targetEntities = { TARGET(link) | ∀ link ∈ paragraph }
        text = TEXTONLY(paragraph)
        EMIT HYPEREDGE(sourceEntity, targetEntities, text)
    end for
end for
```

4.2 Submitted Runs

Ranks entities by degree centrality on a sub-graph that is extracted using the following steps, where `$weightingScheme` and `$centrality` are varied between the methods.

1. Edges in KG are associated with paragraph-long text
2. A BM25 model is used to retrieve edges in response to the query;
3. Edges are weighted according to `$weightingScheme`
4. `$centrality` on this weighted graph is used to rank entities
5. Support paragraphs are taken from the the paragraph associated with the entity's highest ranking edge.

We contributed the following combinations of our framework as entity run submissions:

top100-c-pr-bm25: `$weightingScheme`: Edges are weighted according to their frequency;
`$centrality`: PageRank on this weighted graph is used to rank entities

top100-rr-marg-bm25: `$weightingScheme`: Edges are weighted according to their reciprocal rank;
`$centrality`: DegreeCentrality on this weighted graph is used to rank entities

top100-sc-ppr-bm25: `$weightingScheme`: Edges are weighted according to their frequency;
`$centrality`: Seed nodes E_0 are retrieved from an entity index of unprocessedtraing using Bm25; PersonalizedPageRank on this weighted graph with seed nodes is used to rank entities

Unfortunately, the software we used to produce the submitted TREC CAR runs had a major bug that randomized the ranking and therefore made the results unusable.

Acknowledgement

We acknowledge the efforts of all students who participated in the course on data science for knowledge graph and text and contributed with many experiments and ideas. These are Bahram Behzadian, Shilpa Dhagat, Colin Etzel, Matthew Magnusson, Tucker Owens, Gaurav Patil, Reazul Hasan Russel, and Hang Zhang.

References

[1] Siddhartha Banerjee and Prasenjit Mitra. Wikikreator: Improving wikipedia stubs automatically. In *ACL (1)*, pages 867–877, 2015.

[2] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia – A Crystallization Point for the Web of Data. *Journal of Web Semantics*, 7(3), 2009.

[3] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[4] Christopher Boston, Hui Fang, Sandra Carberry, Hao Wu, and Xitong Liu. Wikimantic: Toward effective disambiguation and expansion of queries. *Data & Knowledge Engineering*, 90:22–37, 2014.

[5] Jeffrey Dalton, Laura Dietz, and James Allan. Entity query feature expansion using knowledge base links. In *SIGIR*, 2014.

[6] Paolo Ferragina and Ugo Scaiella. Tagme: on-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1625–1628. ACM, 2010.

[7] Taher H Haveliwala. Topic-sensitive pagerank. In *WWW*, pages 517–526, 2002.

[8] Chin-Chi Hsu, Yi-An Lai, Wen-Hao Chen, Ming-Han Feng, and Shou-De Lin. Unsupervised ranking using graph structures and node attributes. In *WSDM*, pages 771–779, 2017.

[9] Rianne Kaptein and Jaap Kamps. Exploiting the category structure of wikipedia for entity ranking. *Artificial Intelligence*, 194:111–129, 2013.

[10] Xitong Liu and Hui Fang. Latent entity space: A novel retrieval approach for entity-bearing queries. *Information Retrieval Journal*, 18(6):473–503, 2015.

[11] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[12] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford, 1999.

[13] Hadas Raviv, Oren Kurland, and David Carmel. Document retrieval using entity-based language models. In *SIGIR*, 2016.

[14] Ridho Reinanda, Edgar Meij, and Maarten de Rijke. Mining, ranking and recommending entity aspects. In *SIGIR*, 2015.

[15] Michael Schuhmacher, Laura Dietz, and Simone Paolo Ponzetto. Ranking Entities for Web Queries through Text and Knowledge. In *CIKM*, 2015.

[16] Chenyan Xiong and Jamie Callan. Esdrank: Connecting query and documents through external semi-structured data. In *CIKM*, 2015.