Constructing Subsumption Hierarchies of Web Queries

Alexander Prohaska[1], Christos Tryfonopoulos[2],
Georgiana Ifrim[3]

[1] University of Kaiserslautern, Kaiserslautern, Germany,
prohaska@informatik.uni-kl.de
[2] Max-Planck Institute for Informatics, Saarbuecken, Germany
trifon@mpi-inf.mpg.de
[3] University College Dublin, Dublin, Ireland
georgiana.ifrim@ucd.ie

Abstract. In this work, we present an approach for automatically identifying subsumption relations between web queries, a difficult (due to feature sparseness and ambiguity), but extremely useful task for many applications, ranging from user profiling and semantic enhancement of query logs, to traffic minimisation in distributed search environments (e.g., federations of digital libraries or cloud-based systems). We start by matching each query to the topics of a comprehensive web directory, and use these topics to apply query expansion in an iterative fashion. Subsequently, all expanded queries are mapped onto the DMOZ hierarchy, and the resulting subsumption relations are directly inferred from the directory structure once conflicts in the hierarchy are resolved. We evaluate our technique on real-world queries, and show that our approach is effective under all settings.

1 Introduction

In recent years, human input, generated both implicitly and explicitly, has rapidly increased in terms of quantity and quality, offering added value to a number of applications ranging from social media and crowdsourcing systems to digital libraries. Input from query logs, blog comments, 'like' buttons, and click-throughs are nowadays analysed and exploited by a number of applications, in an attempt to leverage their services and offer users a context-aware, burden-free experience. Humans, on the other hand, in their online life use systems and tools to query, annotate, rank, and evaluate all kinds of content, expecting faster and more personalised services that relieve them from tedious tasks. Typical examples of such expectations that were transformed to working services are, among others, personalised news filtering, query auto-completion/auto-correction, and recommendation systems. Exploiting this abundance of user-provided information in a meaningful way and offering general solutions that can be utilised in a number of applications is of paramount importance.

Idea and Challenges In this paper, we present a new algorithm for creating *subsumption hierarchies* from a set of user queries. This work is, to the best of our knowledge, the first attempt in the literature to derive a subsumption hierarchy from a set of keyword queries. Our method infers hierarchies for an arbitrary number of keyword queries and outputs a single subsumption hierarchy. Notice that this is qualitatively different from previous approaches, as classification [4, 7, 14, 15, 28] would probably group in the same category queries like "basketball" and "slam dunk contest" or "too much calcium" and "hypercalcemia", while in our case the output would be that "basketball" is more general than "slam dunk contest", and "too much calcium" is more general than "hypercalcemia".

Creating a subsumption hierarchy from an arbitrary set of queries is a difficult task due to (i) feature sparseness, (ii) ambiguity and polysemy, and (iii) vocabulary size of web queries. Analysis of several hundreds of millions of web queries showed that the average length of web queries is 2.2 terms [5], while the average length of popular queries is only 1.7 terms. Additionally, queries like the term "cell" may refer to documents about living organisms in biology, the name of the Sony/Toshiba/IBM micro-processor, the movie where Jennifer Lopez starred, cell phones, fuel cells, and many more. Finally, query vocabulary is in a constant state of change [5], following the content dynamicity of web pages as each year approximately half the web's content is renewed [24].

Applications Identifying subsumption pairs and utilising hierarchies of web queries is a particularly useful task for many applications. *User profiling* applications could benefit from more sophisticated modelling of user interests, while *semantic enhancement* of query logs could help search engines present in-context advertisements. Additionally, construction of query hierarchies in distributed environments would *leverage local caching* mechanisms in a variety of applications like cloud-based information management systems, distributed publish/subscribe systems, or federations of digital libraries, and *reduce message traffic.*

Contribution In this work, we present algorithm QSub that may be used to create query subsumption hierarchies from an unrestricted domain of keyword queries. To do so, we utilise the data from a large web directory to *expand* each individual query in an iterative manner. We pay attention in keeping the expanded query close to the original one by resorting in a query-driven weighting strategy for the query features. Subsequently, we *map* the expanded query to its most similar topics of an offline web directory. These mappings are then used to create the query subsumption hierarchy, by (i) considering their parent-child relations in the directory's taxonomy and (ii) resolving logical errors and *conflicts.* We evaluate our technique on a small hand-crafted query set and three sets of *real-world queries* (sports-related, health-related and unfocused logs).

Our results show that the creation of a query subsumption hierarchy is possible by using only the web directory, without resorting to privacy-sensitive or proprietary data like click-throughs. We experimentally evaluate our algorithm and demonstrate that it is effective both on domain-restricted and domain-unrestricted queries.

In the light of the above, the contributions of this work are threefold.

- Algorithm QSub is the first *fully automated* approach in the literature to create subsumption hierarchies from sparse datasets. Our approach is applicable to a number of datasets with the aforementioned characteristics like queries or tags.
- The proposed method is general, privacy-aware, and network-agnostic, as it does not utilise proprietary, private, or network-accessible resources. It is also incremental by design, as insertions/deletions/updates in the query set are accommodated in the constructed hierarchy by examining only the modified part of the dataset.
- Algorithm QSub is effective and accurate; the experimental evaluation demonstrates that it achieves as much as 80% precision in identifying subsumption pairs under different settings.

Paper Organization The rest of the paper is organised as follows. Section 2 discusses related work on query classification and query subsumption. Section 3 presents the QSub algorithm and its variations, while Section 4 presents its experimental evaluation. Finally, Section 5 outlines future directions.

2 Related Work

Our work fits into a rich body of semantic web approaches, and is closely related to a number of research areas:

Keyword-query classification [4, 7, 14, 15, 28], where the goal is to classify a set of user queries into pre-defined categories, by using, e.g., search engines, click-through data, query logs, web directories, data mining or computational linguistics. This is a very challenging problem, which led to the KDDCup in 2005 [22] focusing on this task. Nevertheless, establishing subsumption relations between user queries is even more challenging: rather than grouping queries into similar (flat) categories, we want to group them into an `is-a` hierarchy (from the most general to the most specialised query).

Search personalization through the use of hierarchical user-interest-profiles [19, 26, 30], where typically the user browsing activity is observed and the web pages collected over time or explicitly expressed interests are used to infer an hierarchical user profile. In particular, [19] have investigated the use of three different approaches to hierarchically organising user interests and have found that using DMOZ for inducing an hierarchical profile gave best results. Nevertheless, [19] simply added the user interests as leaves in the already existing DMOZ hierarchy, thus producing rather large profiles with DMOZ nodes into the final structure. Additionally, the context around user interests is typically much richer and less ambiguous than web queries, while tracking user activity or click-through logs may raise privacy concerns.

Folksonomies and crowdsourced taxonomies [10, 17, 20, 21, 31, 32], is a very popular research area bringing forward the importance of user-provided tags and the need for tag-hierarchies for easier browsing and searching of web data. Some of these approaches automatically derive similarity measures between tags and use agglomerative clustering for producing tag-hierarchies, but much of previous work assumes a static tag space, despite its dynamicity. Newer approaches [21], rely on users providing the actual subsumption relations between tags, but the experiments so far have used a rather controlled vocabulary, ignoring semantically rich web keyword queries. Semi-automatically extracting ontologies and web directories, e.g., [33, 6] by means of data mining and information extraction from semi-structured text is also related to our problem, although again, the context around the target concepts is typically much richer than that provided by sparse Web queries.

Inferring query subsumption hierarchies has been addressed before [1, 8, 13, 29]. Nevertheless, previous approaches have either restricted the query set, or used external resources such as search engines and click-through data for expanding the queries. For example, [1, 13] focused on relational queries and gave an algorithm for computing subsumptions for function-free relational queries, e.g., queries $q_1 = pilotsalary > 10000$ and $q_2 = pilotsalary > 20000$ executed on a database with objects of type *pilot* which have a property *salary*. Later [12] extended their technique for queries with function-free first-order predicates. The work in [8, 9] used search engines to expand queries by taking the titles and snippets of highly ranked search results, and then built subsumption trees through hierarchical clustering. The leaves of their hierarchy are similar queries, while the intermediate nodes are named using the most frequent co-occurring terms of the children nodes. Contrary, our goal is to automatically produce a subsumption hierarchy from a given web query set, without (i) relying on the heavy use of online resources (e.g., search engines, click-through data), and (ii) introducing additional "expanded" queries that unnecessarily mount and clutter the resulting hierarchy.

3 The QSub Algorithm

In this section, we present the QSub algorithm designed to identify subsumption relations in multi-keyword queries. The QSub algorithm resorts to a comprehensive web directory to identify the topics that are most relevant to each query and maps the queries on the directory nodes. The resulting subsumption relations are directly inferred from the directory structure once conflicts in the hierarchy are resolved. In the following, we present a brief overview of the algorithm and describe the individual phases of the hierarchy construction.

3.1 Algorithm Overview

To build a query subsumption hierarchy we utilize the idea of *term subsumption* as a "generalisation/specialisation" (`is-a`) relation [27, 18] applied to keyword queries. A query A *subsumes* another query B, denoted as $A \succ B$, if B is a specialisation of A and A is a generalisation of B, with respect to the meaning of the queries (e.g., "NBA teams" $\succ$ "Dallas Mavericks"). In our setting, a tree built out of subsumption relations will be called a *subsumption hierarchy.* Since it is not straightforward to decide whether a pair of queries entails a valid subsumption, due to query ambiguity and feature sparseness [5], our idea is to utilise a multi-step process to expand user queries and subsequently map them to an existing hierarchy. This process involves the following steps, described in detail in each of the following sections:

Step 0. A database of feature vectors of DMOZ topics is created. This is a *one-time procedure*, independent of the query set at hand, that is done at the bootstrapping phase of the algorithm. (Section 3.2)

Step 1. A feature vector for each query is created in an iterative way. Four alternatives for the construction of the feature vector, ranging from simple keyword weighting to query expansion, are presented. (Section 3.3)

Step 2. Queries are mapped to the DMOZ hierarchy by means of cosine similarity to the DMOZ topics. (Section 3.4)

Step 3. The structure of the taxonomy is utilised to retrieve a mapping tree, conflicts in the tree are resolved and subsumption pairs are extracted from the mapping tree. (Section 3.5)

Step 4. Low-rated subsumption pairs are filtered out and the subsumption hierarchy is derived from the remaining ones. (Section 3.6)

3.2 Step 0: Creation of Topic Vectors

In this section, we describe how we construct the feature vectors of topics by resorting to the DMOZ directory; these topics will be utilised in the next step of the algorithm for query expansion and mapping. DMOZ is the largest and most comprehensive human-edited directory of web pages consisting of more than five million web sites categorised into more than one million topics. The topics are structured in a comprehensive hierarchy of fine granularity. Each web site has a short title and a comprehensive, user-created description to help users locate the page they are looking for, while topics contain entries of web sites that provide topic-specific. This high-quality user-created information can be utilised to create a feature vector that describes each topic. Notice that this step may be performed by using *any structured collection* that can provide us with a taxonomy of topic feature vectors, including the Yahoo! directory, DBpedia, or Wikipedia category graph.

The topic feature vectors are weighted using the *tf-idf* weighting scheme [3], and vectors are normalised by their Euclidean norm. In our approach, keywords do not contribute equally to topic feature vector: keywords located in the path from the root of the directory to a specific topic contribute with a weight of 1.0, keywords in topic titles contribute with a weight of 0.5, and keywords in topic descriptions contribute with a weight of 0.25. The creation of the topic feature vectors is a one-time procedure done at the bootstrapping phase of the algorithm, and the resulting topics are used for applying algorithm QSub to any query dataset. Additionally, when parsing the titles and descriptions of the web sites in the DMOZ topics, the YAGO [33] semantic knowledge base is utilised to identify entities. These entities are then used to enrich the keywords in the topic feature vectors and improve the quality of the result set.

3.3 Step 1: Creation of Query Vectors

After all topics are represented as feature vectors, the main part of the QSub algorithm begins. In this section, we describe four alternatives for the creation of query vectors, ranging from simple keyword weighting to weighted query expansion.

Keyword Weighting. The simplest variation of the QSub algorithm employs keyword weighting (QSub-W) to compute the query feature vector; it utilises the frequency of occurrence (*tf*) [3] of words in queries to construct the query feature vector and uses its Euclidean norm for normalisation. This is a simple approach designed to be stable in terms of query weighting to the insertion/deletion of queries (no *idf* is used). The idea behind this method is to minimise false-positive subsumptions, as in the next phase of the algorithm queries will match the few topics that contain one of the query keywords.

Averaging. Algorithm QSub-A is an extension of algorithm QSub-W as it employs the same machinery to create the initial feature vector for each query, but also introduces topic-based query expansion. To do so, it identifies *similarities* between the topic and the query vectors, and expands the queries with the features of the $\mathcal{K}$ most similar topics. In algorithm QSub-A, all participating topics in the query expansion process are *weighted equally* (hence the word averaging in the name of the algorithm). Considering only a fixed maximum of the most similar topics by using *k-nearest neighbors (kNN)* [11, 23] favours the features of the topics with the highest cosine similarity to a given query and avoids noise addition from low similarity topics. The above procedure is repeated for a fixed number of iterations $\mathcal{I}$. The choice of $\mathcal{I}$ influences the accuracy of query expansion as a high $\mathcal{I}$ value may lead to over-expansion of queries, introduction of noise in the feature vectors, and low running times. The number of features in an expanded query is also limited to $\mathcal{F}$. A low value for $\mathcal{F}$ will cancel out the benefits from query expansion, i.e., the discovery of related topics without direct query keyword matches, while a high value will make the expanded query lose its original meaning. Averaging in feature weighting allows the expanded query to gradually move away from the initial one in every iteration; this process will eventually move the expanded query towards the centroid of the $\mathcal{K}$ nearest neighbouring topics.

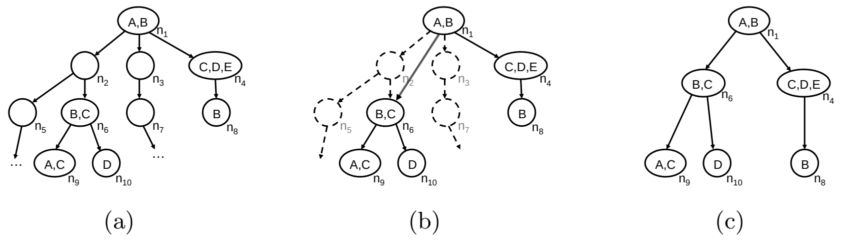
Centroid-Driven. The centroid-driven variation (QSub-C) is an iterative algorithm that utilises the same machinery as QSub-A, but now all features used to expand a given query are *weighted* according to the cosine similarity of their respective topic to the *expanded* query (resulting from the previous iterations of the algorithm). This process is repeated for $\mathcal{I}$ iterations and will eventually result in moving the expanded query towards the centroid of the topics that participated in the expansion (hence the term *centroid-driven* in the name of the algorithm). This approach favours keywords from topics that are closer to the expanded query and makes the expanded queries less sensitive to outliers. The disadvantage of this approach is that it may distort the meaning of the initial query.

Query-Driven. The query-driven variation (QSub-Q) uses the same machinery as the QSub-C algorithm, but now all features that are used to expand a given query are weighted according to the cosine similarity of their respective topic to the *initial* query [2, 23], leading to a *query-driven* approach. This process is repeated for $\mathcal{I}$ iterations. Penalising terms of topics with a low cosine similarity to the original query is an effective way to preserve the core meaning of the query. QSub-Q is the most sophisticated algorithm for weighting the query vectors, since it employs an iterative query-driven feature expansion process that avoids query distortion by resorting to by resorting to query-centred feature expansion.

3.4 Step 2: Mapping Queries to Topics

After completing query expansion, the expanded queries can be mapped to the DMOZ hierarchy to create a *mapping tree* of queries. In this step, we begin with an empty hierarchy and map each expanded query to the $\mathcal{T}$ most similar DMOZ topics. Different queries may be mapped to the same node, and each query may be mapped to at most $\mathcal{T}$ nodes in the mapping tree; see for example Figure 1(a) where queries A and B are both mapped to node n_1. A low value of $\mathcal{T}$ considers only the best matching DMOZ topics and is expected to return a small mapping tree of high quality. On the other hand, a high value of $\mathcal{T}$ takes into account the different meanings of ambiguous words as the expanded query is mapped in more positions in the mapping tree. This increases the probability to cover all different meanings of a query and will, in turn, increase the number of true-positive subsumption relations in the final subsumption hierarchy. However, this will also increase the size of the mapping tree and thus also the number of false positives, as many spurious subsumption relations may be added in the final hierarchy.

After mapping the expanded queries to the DMOZ topics, the resulting mapping tree will contain nodes *labelled* with the identifiers of the queries that were mapped to the specific topic and *unlabelled* nodes with no mapped queries. An example of such a tree is shown in Figure 1(a), where the queries A, B, C, D, E are mapped to nodes n_1, n_4, n_6, n_8, n_9, n_{10}, while nodes n_2, n_3, n_5, n_7 are



(a) (b) (c)

Fig. 1. (a) Initial mapping tree with unlabelled nodes, (b) restructuring of mapping tree after deletion of unlabelled nodes, and (c) final mapping tree.

unlabelled nodes. Subsequently, all unlabelled nodes are deleted, and each of their parent nodes is connected to all children nodes of the deleted node. Figure 1(b) shows the restructuring of the mapping tree after all unlabelled nodes are deleted. Notice that if the root is unlabelled and is thus deleted, all the children nodes are connected to a new empty root to maintain the tree structure. Figure 1(c) presents the final mapping tree; notice that this tree is not a subsumption hierarchy yet, as it may contain conflicts as it will be explained in the next section.

3.5 Step 3: Identifying Subsumption Pairs

In this section, we describe the procedure to extract subsumption pairs from the mapping tree, filter-out the low-rated ones, and create the final subsumption hierarchy. After constructing the mapping tree, we use depth-first search to collect all paths that start from the root of the tree and end at one of the leaves. This results in a list of *multi-labelled paths* (`ml-paths`); in our example, the mapping tree of Figure 1(c) contains three `ml-paths`, namely $(A, B) \rightarrow (B, C) \rightarrow (A, C)$, $(A, B) \rightarrow (B, C) \rightarrow D$, and $(A, B) \rightarrow (C, D, E) \rightarrow B$. Subsequently, `ml-paths` are transformed into *single-label paths* (`sl-paths`) by creating all path permutations with exactly one label (i.e., query) at each node. These `sl-paths` may contain logical errors, so all paths in which the same query appears in more than one positions are filtered-out, except when these positions are adjacent (i.e., parent-child nodes). For example the `sl-path` $A \rightarrow C \rightarrow C$ is not filtered out as it can be simplified to $A \rightarrow C$, while $A \rightarrow C \rightarrow A$ contains a contradiction and is thus filtered-out. The remaining `sl-paths` are then used to derive the query subsumption pairs.

Table 1(a) contains all `sl-paths` extracted from the `ml-path` $(A, B) \rightarrow (B, C) \rightarrow (A, C)$ of the previous example, together with their simplified versions and the subsumption pairs that are derived. Notice that we cannot simply combine all derived pairs, as they may also contain conflicts. In our example, pairs $A \succ B$ and $B \succ A$, and $A \succ C$ and $C \succ A$ are conflicting and are filtered out, thus leaving $B \succ C$ as the only one valid subsumption pair for this `ml-path`. All the valid subsumption pairs derived from the mapping tree of Figure 1(c), along with their corresponding `ml-paths` are shown in Table 1(b). This

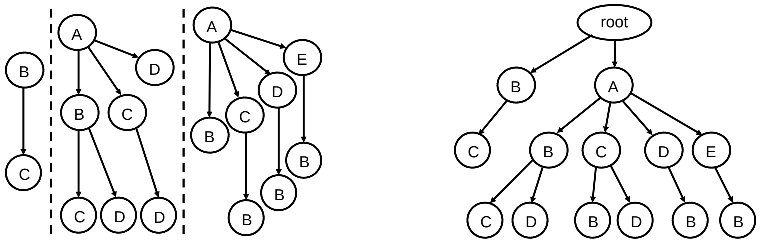
sl-path	simplified sl-path	subsumption pairs
A → B → A	-	-
A → B → C	A → B → C	A ≻ B, A ≻ C, B ≻ C
A → C → A	-	-
A → C → C	A → C	A ≻ C
B → B → A	B → A	B ≻ A
B → B → C	B → C	B ≻ C
B → C → A	B → C → A	B ≻ C, B ≻ A, C ≻ A
B → C → C	B → C	B ≻ C

(a)

ml-path	subsumption pairs
(A,B) → (B,C) → (A,C)	B ≻ C
(A,B) → (B,C) → D	A ≻ B, A ≻ C, A ≻ D, B ≻ C, B ≻ D, C ≻ D
(A,B) → (C,D,E) → B	A ≻ B, A ≻ C, A ≻ D, A ≻ E, C ≻ B, D ≻ B, E ≻ B

(b)

Table 1. Extraction of subsumption pairs from `ml-paths`.



(a) (b)

Fig. 2. (a) The subsumption forest and (b) the resulting subsumption hierarchy.

process is performed for each of the `ml-paths` derived from the mapping tree and the subsumption pairs created are examined for conflicts *separately* for each `ml-path`. Thus, conflicts in subsumption pairs belonging to *different* `ml-paths` will *not be filtered-out* since they correspond to *different contexts*.

3.6 Step 4: Building the Subsumption Hierarchy

In this section, we describe how to rate the derived subsumption pairs and utilise only the top-rated ones to create the query subsumption hierarchy. Rating for a subsumption pair can be computed by taking into account the similarity score of each expanded query to the topic it was mapped to. Low similarity scores for one or both queries means that the queries are not highly related to the topics they were mapped to, thus the subsumption relation is probably a false-positive. Similarly, high similarity scores for both queries indicate a correct mapping and thus a true-positive subsumption relation. Given a query subsumption $A \succ B$, where A is mapped to topic T_A and B is mapped to topic T_B, the rating of the pair is calculated as $rating(A, B) = cos(A, T_A) * cos(B, T_B)$, where $cos()$ is the cosine similarity between the two vectors.

After computing the rating of all subsumption pairs identified in the previous section, we remove all pairs with a rating below threshold ϑ. The remaining ones

```
for all queries do
  create feature vector $Q$ of query                              ▷ Step 1
  for $\mathcal{I}$ iterations do
    for all topics $t \in \{\mathcal{K}$ most similar topics to $Q\}$ do
      $Q \leftarrow$ all weighted terms $\in t$
    end for
    keep only the $\mathcal{F}$ highest-weighted terms of $Q$
  end for
  map $Q$ to the $\mathcal{T}$ most similar DMOZ topics           ▷ Step 2
end for
for all ml-paths $M$ do                                           ▷ Step 3
  for all sl-paths $S \in M$ do
    insert high-rated subsumption pairs from $S$ to hierarchy     ▷ Step 4
  end for
end for
```

Fig. 3. Pseudocode for the query expansion variants of algorithm QSub.

are then used to create a forest of subsumption trees. The number of trees in the forest equals the number of `ml-paths` identified in the final mapping tree. We use a simple recursive tree-construction algorithm [16] to construct the subsumption tree from the subsumption pairs of an `ml-path`, asserting that no path that does not occur in the mapping tree is created. The resulting trees are then merged [16], if possible, and combined under a (potentially) empty root to form a subsumption hierarchy. In our example, we assume for simplicity that all subsumption pairs of Table 1(b) have a rating higher than ϑ, thus no pair is filtered-out. Thus, all the identified pairs will be used to create a forest of three trees as shown in Figure 2(a), one for each `ml-path`. After applying the merging and combination of all trees under the root, the subsumption hierarchy of Figure 2(b) is produced. Notice that in this hierarchy $A \succ B \succ C$ and $A \succ C \succ B$ are both valid since they refer to a different context.

The pseudocode providing a high-level description of the query expansion variants (QSub-A, QSub-C, QSub-Q) is given in Figure 3.

4 Evaluation

In this section, we present a series of experiments that compare the variations of algorithm QSub in terms of effectiveness and efficiency.

Datasets. For our evaluation we used two query sets: (i) a small hand-crafted set of 70 queries which contained selected queries from a real query log and (ii) three sets of 1000 randomly chosen queries from a large query log. It should be noted that the major problem in the evaluation of this type of algorithms is the lack of a ground truth to compare against. For all query sets, the constructed subsumption pairs, and hence the created subsumption hierarchies, had to be manually evaluated.

Hand-crafted Query Set. This query set was used as ground truth and is a hand-crafted set of 70 queries; all queries within the set are related to sports,

	Subsumption pairs (hand-crafted query set)
Correct	basketball teams ≻ miami heat
	champions league ≻ chelsea
	nba playoffs ≻ nba scores
Incorrect	golf players ≻ illinois
	mclaren racing ≻ formula one
	uefa cup news ≻ real madrid

(a)

	Subsumption pairs (large query sets)
Correct	too much calcium ≻ hypercalcemia
	gallbladder ≻ gallstones
	philosophy ≻ deism
Incorrect	midwest motorsports ≻ elliot sadler
	nursing journals ≻ schoolnurse
	football ≻ kittens

(b)

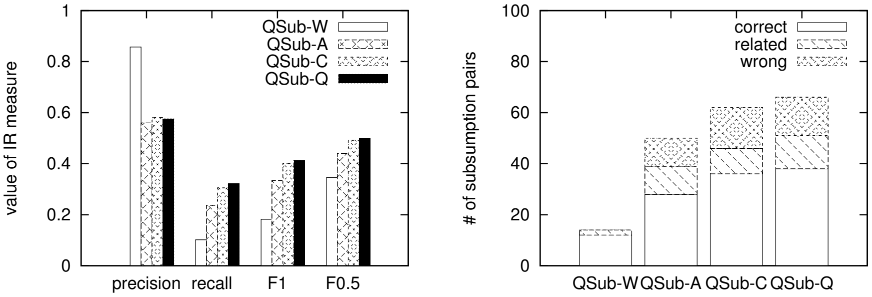
Table 2. Sample correct/incorrect subsumptions for (a) hand-crafted and (b) large query sets.

apart from 4 queries that are related to geography and act as noise in the query set. The query subsumptions in this set were manually identified to allow for comparison with the ones automatically generated by the variations of algorithm QSub; to do so we generated all possible combinations of query pairs, and a user evaluation study identified 118 correct subsumption pairs. Table 2(a) presents some of the queries and correct/incorrect subsumption relations between them, as identified by users. The query set along with all the user-identified correct subsumption pairs have been provided as supplementary material to this submission and are available to the reviewers through the submission system.

Large Query Sets. This collection contains three sets of 1000 randomly chosen queries from an AOL query log that contained random web queries of more than 600.000 users, collected in March, April and May 2006 [25]. The queries were categorised, two query sets were chosen to be thematically focused to the Sports and Health categories, while the third query set was thematically unfocused (named All in the graphs) and comprised of queries from all categories in the given query log. The Sports query set contains many names of colleges, sports teams, and leagues, the Health query set contains mainly queries about drugs, pharmaceutical companies, symptoms and treatments for various diseases, while the unfocused query set contains queries ranging from ancient philosophers to submarines and constitutes a stress test for the algorithm. Table 2(b) presents some of the queries and correct/incorrect subsumption relations between them, as identified by users.

Experimental Setup. All the algorithms were implemented in Java and executed in an off-the-shelf PC. The time shown in the graphs is wall-clock time and all algorithm measurements are derived by using the best-performing parameter setup from our evaluation ($\mathcal{I} = 3$, $\mathcal{F} = 50$, $\mathcal{T} = 3$, $\vartheta = 0.3$); the experiments needed to determine these values are omitted due to space considerations.

In our evaluation, we use standard information retrieval metrics to evaluate the quality of the derived subsumption pairs and hierarchy. These metrics involve (i) *precision (P)*, defined as the percentage of correctly identified subsumption pairs from the set of all returned pairs, (ii) *recall (R)* defined as the ratio of correctly returned subsumption pairs to the total number of all correct subsumption pairs for all queries, (iii) $F_i = (i^2 + 1)PR/(i^2P + R)$ measure,



(a) (b)

Fig. 4. (a) IR measures and (b) subsumption pairs for the hand-crafted query set.

and especially F_1 and $F_{0.5}$ as our emphasis is on precision [3], and (iv) number of returned subsumption pairs. These measures can only be computed for the hand-crafted query set since all correct subsumption pairs have to be known to calculate recall and F-measure. For this reason precision and number of returned subsumption pairs are used to assess the performance of the algorithms in the large query sets.

Results for the Hand-crafted Query Set. Initially, we consider the performance of the algorithms in terms of the quality of the derived subsumption hierarchy for the hand-crafted query set. As we observe in Figure 4(a), QSub-W presents the highest precision and the lowest recall value among all variants of the QSub algorithm, resulting in the lowest F_1 and $F_{0.5}$ values. This demonstrates the necessity of query expansion in the quality of the identified subsumption pairs. All expansion techniques have comparable performance in precision, with QSub-Q performing slightly better than its competitors, and QSub-A presenting the worst performance of all three query expansion techniques. This was expected as QSub-A does not consider the query-topic similarity in weighting the feature vector of the expanded queries. As also shown in Figure 4(b) more correct subsumption pairs are discovered by algorithm QSub-Q, since the meaning of the initial query is not distorted by the query expansion process. Additionally, Figure 4(b) shows that QSub-W returns the lowest number of correct and incorrect subsumption pairs due to the lack of query expansion that matches queries only with highly related topics. Finally, algorithm QSub-Q returns the highest number of correct pairs, while the number of incorrect pairs is comparable to the rest of the query expansion variants.

Results for the Large Query Sets. In this set of experiments, we measure only precision and number of identified subsumption pairs, as recall, F_1, and $F_{0.5}$ measures cannot be computed for these query sets due to the large number of subsumption pairs that need to be evaluated manually (i.e., consider all possible

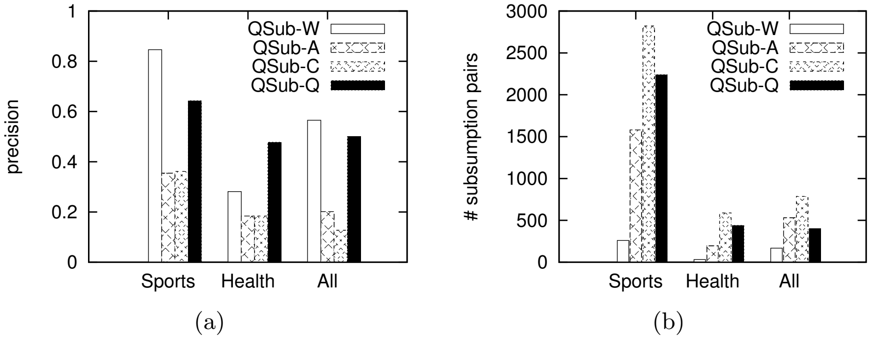
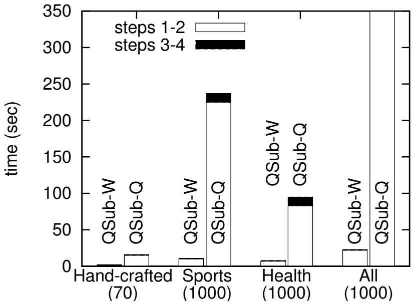


Fig. 5. (a) Precision and (b) number of subsumption pairs for the large query sets.

subsumption pairs among 1000 queries, and manually identify the correct ones). In Figure 5(a), we observe that QSub-W presents the highest precision value in two out of three query sets, followed by QSub-Q that outperforms the rest of the query expansion algorithms. The performance of the QSub-Q algorithm in the Health query set is notable, and is due to the type of subsumption relations that may be found in this set: drugs produced by companies, and symptoms/illnesses that are favoured by query expansion.

Figure 5(b) shows the number of returned pairs per algorithm; algorithm QSub-C returns the highest number of subsumption pairs, followed by QSub-Q. When observing Figures 5(a) and 5(b) together we conclude that algorithm QSub-Q is again the best choice as it manages to combine high-precision values with a high number of identified subsumption pairs. Contrary, algorithm QSub-W presents high precision but low number of identified pairs, while QSub-C achieves a high number of identified pairs but low precision.

Time and Space measurements. Figure 6(a) presents the running time for all query sets and for algorithms QSub-W and QSub-Q; the other competitors are left out to avoid cluttering the graph as they have exactly the same performance as QSub-Q since they just differ in the way of computing the expanded features for the query vectors. Additionally, the time measurement of the All query set is truncated to better illustrate the differences among the rest of the algorithms; the actual time measurement for the All query set was 2055 seconds. It is worth noting that the construction of the query feature vector (step 1) and the mapping of the queries to the DMOZ hierarchy (step 2) are the most time consuming tasks. Additionally, although all query sets (except the hand-crafted one) contain 1000 queries, the execution time of the algorithms varies. This can be explained by the number of topics the query vectors have to be matched against; the lower the number of DMOZ topics related to a category, the fewer the comparisons that have to be performed between each query and the category topics. Finally, memory requirements for all algorithms and query sets are shown in Figure 6(b);



(a)

Query set	Memory required
Hand-crafted	125 MB
Sports	125 MB
Health	80 MB
All	850 MB

(b)

Fig. 6. (a) Time and (b) space measurements for the different query sets.

notice that all algorithms have similar memory requirements as memory usage is heavily dominated by the size of the topic vectors.

5 Future Work

We are currently working on offering an online tool to identify subsumption relations on query logs and tags. We are also planning to deploy our methods in a distributed environment to quantify the gain in message traffic from caching due to subsumption.

References

1. M. Al-Qasem and S. M. Deen. Query subsumption. In *FQAS*, 1998.
2. C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning. *AI Review*, 11, 1997.
3. R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press / Addison-Wesley, 1999.
4. D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *KDD*, 2000.
5. S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder. Hourly analysis of a very large topically categorized web query log. In *SIGIR*, 2004.
6. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 2009.
7. A. Broder. A taxonomy of web search. *SIGIR Forum*, 2002.
8. S.-L. Chuang and L.-F. Chien. Towards automatic generation of query taxonomy: A hierarchical query clustering approach. In *ICDM*, 2002.
9. S.-L. Chuang and L.-F. Chien. Enriching web taxonomies through subject categorization of query terms from search engine logs. *DSS*, 35(1), 2003.
10. C. Van Damme, M. Hepp, and K. Siorpaes. Folksontology: An integrated approach for turning folksonomies into ontologies. In *ESWC.* Springer, 2007.

11. B. V. Dasarathy. *Nearest neighbor pattern classification techniques.* IEEE, 1991.
12. S. M. Deen and M. Al-Qasem. A query subsumption technique. In *DEXA*, 1999.
13. S.M. Deen. An architectural framework for ckbs applications. *TKDE*, 8(4), 1996.
14. L. Gravano, V. Hatzivassiloglou, and R. Lichtenstein. Categorizing web queries according to geographical locality. In *CIKM*. ACM, 2003.
15. M. Grobelnik and D. Mladenić. Simple classification into large topic ontology of web documents. *CIT*, 13(4), 2005.
16. J. L. Gross and J. Yellen. *Handbook of Graph Theory.* CRC Press, 2004.
17. M. Gupta, R. Li, Z. Yin, and J. Han. Survey on social tagging techniques. *SIGKDD Explor. Newsl.*, 2010.
18. P. Gallinari H. N. Fotzo. Learning generalization/specialization relations between concepts - application for automatically building thematic document hierarchies. In *RIAO*, 2004.
19. M. Haridas and D. Caragea. Exploring wikipedia and dmoz as knowledge bases for engineering a user interests hierarchy for social network applications. In *OTM*. 2009.
20. P. Heymann and H. Garcia-Molina. Collaborative creation of communal hierarchical taxonomies in social tagging systems. Technical Report 2006-10, Stanford InfoLab, April 2006.
21. D. Karampinas and P. Triantafillou. Crowdsourcing taxonomies. In *ESWC*. Springer-Verlag, 2012.
22. Y. Li, Z. Zheng, and H. K. Dai. Kdd cup-2005 report: facing a great challenge. *SIGKDD Explorations Newsletter*, 7(2), 2005.
23. T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.
24. A. Ntoulas, J. Cho, and C. Olston. What's new on the web?: the evolution of the web from a search engine perspective. In *WWW*, 2004.
25. G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *InfoScale*, 2006.
26. K. Ramanathan and K. Kapoor. Creating user profiles using wikipedia. In *ER*, 2009.
27. M. Sanderson and B. Croft. Deriving concept hierarchies from text. In *SIGIR*, 1999.
28. D. Shen, R. Pan, J.-T. Sun, J. J. Pan, K. Wu, J. Yin, and Q. Yang. Query enrichment for web-query classification. *TOIS*, 24(3), 2006.
29. D. Shen, M. Qin, W. Chen, Q. Yang, and Z. Chen. Mining web query hierarchies from clickthrough data. In *AAAI*, 2007.
30. A. Sieg, B. Mobasher, and R. Burke. Web search personalization with ontological user profiles. In *CIKM*, 2007.
31. Y. Song, B. Qiu, and U. Farooq. Hierarchical tag visualization and application for tag recommendations. In *CIKM*. ACM, 2011.
32. M. Strohmaier, D. Helic, D. Benz, C. Körner, and R. Kern. Evaluation of folksonomy induction algorithms. *ACM ACM TIST*, 2012.
33. F.M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A Large Ontology from Wikipedia and WordNet. *Journal of Web Semantics*, 2008.