

OLAP Cubes for Social Searches: Standing on the Shoulders of Giants?

Konstantinos Morfonios
Dept. of Informatics and Telecommunications
University of Athens, Greece
kmorfo@di.uoa.gr

Georgia Koutrika
Dept. of Computer Science
Stanford University, USA
koutrika@stanford.edu

ABSTRACT

The advent of social bookmarking has signaled a new era towards a more “pluralistic” web, where entities of multiple types (e.g., users, resources, tags) can coexist and get interconnected. In this paper, we propose going beyond classical searches for resources based on keywords to exploring social data starting from any type of entity, i.e., user, resource or annotation, and requesting aggregated views of related entities based on the relationships defined between entities. We map this type of social searching to OLAP query processing bridging the gap between OLAP and the social web and we study various ways to support on-the-fly aggregations of social data. We further describe how data cubes can be used for precomputing and materializing the results of all possible aggregate queries over a social dataset. Our experiments show that cubes can efficiently support searches over social data and they particularly fit searches that combine popular entities over large datasets.

Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—*Retrieval models, Search process*

General Terms

Algorithms, Experimentation, Performance

Keywords

Advanced Search, OLAP-Style Search, Social Network, Web

1. INTRODUCTION

Since its conception back in 1989, the web has been built around a single type of entities, i.e., resources, which may take several forms, such as images, videos, pdf documents, but in their majority comprise pages linked to each other. Searching in the web has been always understood as locating resources based on their attributes, which typically involve the keywords found in text documents but may also include other features, such as format, date or author. More sophisticated search schemes have been also proposed. For example, faceted search allows navigating in a resource space

through a precomputed set of dimensions, which represent significant features of the resources in the space [2]. Nevertheless, existing searches typically build on the same discipline: accessing one type of entities through their attributes.

The advent of social bookmarking has signaled a new era towards a more “pluralistic” web, where entities of multiple types can coexist and get interconnected. Social bookmarking made possible for users to interactively contribute resources and annotate these or resources provided by other people with descriptive strings, i.e., tags, building communities that share their expertise and resources. Users, resources and annotations are interweaved and become all first class citizens of the web as a result of the social bookmarking activity. This pluralism is a key characteristic of many emerging social systems, such as Wikipedia, del.icio.us, and Flickr, which have gone from being a small niche of the web to one of its most important components.

How could we search this new web? Users, resources, and annotations come with their own attributes, which could be the name of users, keywords for resources, and categories for annotations. In this sense, searching for example for resources using keywords is still very prominent. However, the existence of entities of multiple types and the relationships defined between them, such as user x ‘created’ resource z and user y ‘added a tag’ w on resource z , call for new search paradigms that go beyond the classical web search style.

One could conceive this social web as a graph, where nodes represent users, resources and annotations, and edges, possibly of various types, capture the different kinds of relationships between nodes. Then, we can explore this graph based on the relationships of the different entities that coexist in it. We could start from any node and request an aggregated view of its neighbors. Such questions could reveal useful insights for the questioned entity as well as the entities that are related to it. For example:

- “which resources are most often tagged with ‘XML’”? This question could provide interesting resources on this subject matter.
- “which users contribute the largest number of resources tagged with ‘C programming’”? This question would reveal active or expert users in this particular area.
- “which are the most frequent tags used by a particular group of users”? This question could show interesting patterns regarding the behavior of a group.

Following an iterative search strategy, one could explore the space of users, resources and annotations starting from a node, picking a neighbor as the next node to inspect and

requesting to see the neighbors of this one, moving in this way from one neighborhood to another and switching from nodes of one type to nodes of a different one. For example, starting with the previous question on users that have contributed the most resources on ‘C programming’, one could pick a user, e.g., the most active one, and request their top tags. From those, one could pick ‘programming tips’ and search for related resources to finally reach a resource on this topic saved by the most active user on C programming.

Most current social systems support searches limited to resources based on tags. A user can manually provide a tag or select one from a tag cloud, which is an aggregated view of the tags in the system [14]. In this paper, we propose *advanced social searches*, i.e., searches over the social web that allow *starting from any type of entity*, i.e., user, resource or annotation, and *requesting aggregated views of the entities related to it* based on the relationships defined between them. Clearly, such functionality imitates operations common in spreadsheets and in advanced applications associated with decision support, and makes strong use of aggregation in order to produce aggregated summaries of the entities. In this sense, advanced social searches can naturally build upon On-Line Analytical Processing (OLAP), since aggregation is of central importance in OLAP.

We investigate various ways for performing and accelerating aggregations on the fly. Performing aggregation on the fly raises significant scalability concerns, since it involves scanning, sorting (or alternatively hashing), and applying the aggregate function over typically very large datasets in real time. On the other hand, data cubes have been found very powerful and scalable tools appropriate for improving the performance of OLAP aggregate queries. We study their potential for social searches. This is the first time, to the best of our knowledge, that cubes are brought in the web arena for supporting searches and the first attempt to search the bulk of social data in an OLAP-style fashion.

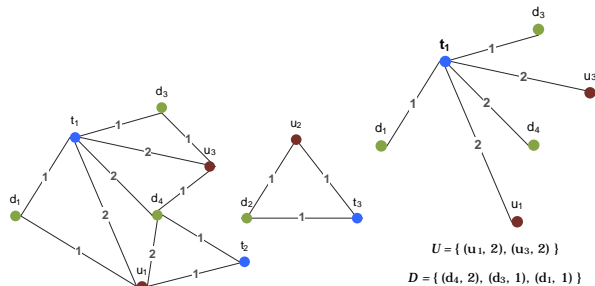
In brief, our contributions are the following:

- We propose social searches that start from any type of entity and return aggregated views of the related entities taking into account their relationships (Section 3).
- We map this type of social searches based on aggregations to OLAP query processing bringing OLAP closer to the social web (Section 4).
- We investigate various ways for performing and accelerating aggregations of social data on the fly. We further study the use of data cubes for precomputing and materializing the results of all possible aggregate queries for supporting advanced social searches. (Section 5).
- We describe experiments showing that cubes can be used to efficiently support searches over large datasets and for different workloads and particularly show their potential for searches involving popular entities (Section 6).

2. RELATED WORK

We provide a brief overview of the existing literature related to our work, namely social systems and OLAP cubes.

Social Systems. We are witnessing a growing number of social services on the web, which enable people to share and tag different kinds of resources, such as: photos [4], URLs [3], blogs [15], and so forth. The increasing popularity of social systems has attracted significant research interest and motivated a number of approaches on studying interesting



(a) A social graph (b) An answer

Figure 1: Example social search.

properties and phenomena of them [6, 16], such as tag evolution, and harvesting social knowledge. The approaches to the latter have been mainly focused on automatic suggestions of high quality tags for an object based on what other users use to tag this object [9, 16], characterizing and identifying users or communities based on their expertise and interests [7], building hierarchies of tags based on their use and correlations [12], aggregating and studying semantic aspects of tags [7, 16], and so forth. On a different track, social annotations have been tested on improving web search [1]. To the best of our knowledge, our approach is the first that focuses on social searches, and proposes enabling “full” searches over users, resources, and annotations, specifying any kind of node and requesting any type of related entities based on the entity relationships.

OLAP Cubes. OLAP performs aggregation over typically large volumes of multidimensional data and provides summaries useful for various applications, e.g., decision support. As shown in this paper, exploring social data taking into account the relationships between the different entities and generating useful aggregations can be mapped to OLAP-style processing. Data cubes have been proposed as the most efficient way to implement OLAP queries [5]. A data cube is a materialized structure that stores precomputed results of group-by aggregate queries on all possible combinations of the dimension-attributes over a fact table in a data warehouse. Cube construction methods can be divided into three main categories: ROLAP [11], MOLAP [17], and graph-based [13], which use materialized views, multidimensional arrays, and complex graph structures, respectively. We propose using data cubes for supporting advanced social searches. Note that our algorithms are general enough and can be combined with any cube implementation. We have selected CURE [11] for constructing cubes due to some properties that are important to the problem of concern (see Section 5.4 for details).

3. SOCIAL SEARCH

We consider that a social bookmarking system is made up of a set \mathbb{D} of *documents* (e.g., photos, web pages, etc), a set \mathbb{T} of *tags* associated with the documents, and a set \mathbb{U} of *users*, all of these associated through a set \mathbb{P} of postings. A *posting* $\langle u, d, t \rangle$ shows that user u assigned tag t to document d . Note that we use the term tag but we virtually mean any type of annotation since our search model does not depend on the content of the resources nor the content or form of annotations but exploits the associations between tags, re-

sources and users. Likewise, we use the terms resource and document interchangeably. Users, tags, resources and postings constitute a *social space* \mathcal{S}_S .

A social space can be naturally represented as a graph. We consider the *social graph* $g(V, E)$ mapping to a social space \mathcal{S}_S , which is an undirected, labeled graph with the following characteristics. Nodes in V map to users, resources, and tags in the space. Edges in E connect (a) a user node to a tag node, representing that the corresponding user has used this tag or (b) a user node to a resource node, showing that the user has tagged the corresponding resource or (c) a resource node to a tag node, showing that the corresponding tag has been assigned by some user to the respective resource. Edges are labeled with a measure that characterizes the number of associations of the connected nodes through the postings. An edge is denoted $e(v, s, a)$, where v, s are the connected nodes and a is the measure. Note that we can consider additional types of edges in order to capture finer relationships among users, resources and tags. For example, we could have edges that map to relationships of the type user u ‘created’ resource d and edges for relationships of the kind user y ‘tagged’ resource d . For simplicity in exposition, we will keep this simple representation of the graph that suffices for explaining the concept of social searches. Our approach is applicable to graphs of richer semantics too.

Example. An example social graph is shown in Figure 1(a). The labels represent the number of co-occurrences of the connected entities in the postings.

Advanced social searches. Given a social space comprising the sets \mathbb{D} , \mathbb{T} , \mathbb{U} and \mathbb{P} and the corresponding social graph $g(V, E)$, a query can be any entity q from $\mathbb{D} \cup \mathbb{T} \cup \mathbb{U}$. Given a query q , the answer comprises a distinct set of all entities of the same type on g that are related to q ordered on the basis of their relationships, as follows:

$$A = \{(s_i, a_i) | \forall e(q, s_i, a_i) \in E, \text{ with } s_i \text{ and } s_{i+1} \text{ being of the same entity type } \wedge a_i \geq a_{i+1}, i = 1 \dots n\}.$$

A top- k answer is the ordered subset of A :

$$A_k = \{(s_i, a_i) | a_i \geq a_{i+1}, i = 1 \dots k, k \leq n\}$$

Example (cont’d). Consider a query that maps to node t_1 on the social graph of Figure 1(a). There are two groups of related entities: users that have used t_1 and documents that are tagged with t_1 , which can be ranked based on the co-occurrence measure a (shown in Figure 1(b)):

$$U = \{(u_i, a_i) | \forall e(t_1, u_i, a_i) \in E \text{ with } u_i \text{ being a user } \wedge a_i \geq a_{i+1}, i = 1 \dots n\}.$$

$$D = \{(d_i, a_i) | \forall e(t_1, d_j, a_j) \in E \text{ with } d_j \text{ being a document } \wedge a_j \geq a_{j+1}, j = 1 \dots n\}.$$

Consequently, searching a social space is conceived as traversing the respective social graph starting from any node, i.e., a query, and finding related entities, i.e., the answer, by following the edges on the graph. In the following sections, we will see that we can support social searches in practice by mapping them to OLAP queries.

4. MAPPING TO OLAP

A social space consists of various types of entities, i.e., users, resources, and tags. In an OLAP environment, every such type can be mapped to a dimension table. Then, every entity of a given type can be stored as a tuple in the

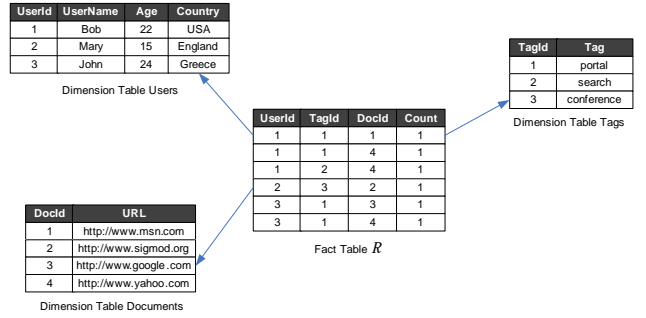


Figure 2: Example star schema.

corresponding dimension table. Furthermore, a fact table \mathcal{R} holds every posting as a tuple, representing the relationships among entities as points in a multidimensional space. Then, social searches can be translated into aggregate queries over the fact table. Showing all the results of such queries would be too detailed; hence, we assume that only the tuples with the k -largest aggregate values are returned. Hence, the expected workload consists of top- k aggregate queries.

Example (cont’d). The social graph of Figure 1(a) can be stored as the star schema of Figure 2. Then, the social search for tag t_1 (this is the tag ‘portal’ with $TagId = 1$ in the star schema) is translated to two OLAP queries (in this example, we assume that $k = 5$):

Q_1 : `SELECT TOP 5 DocId, COUNT(*)`
`FROM \mathcal{R} WHERE TagId = 1`
`GROUP BY DocId`
`ORDER BY COUNT(*) DESC;`

Q_2 : `SELECT TOP 5 UserId, COUNT(*)`
`FROM \mathcal{R} WHERE TagId = 1`
`GROUP BY UserId`
`ORDER BY COUNT(*) DESC;`

For example Q_1 locates the resources that have been tagged with the tag ‘portal’, groups them according to their $DocId$, and counts the number of tuples in each group, which coincides with the number of times a given resource has been tagged as ‘portal’. The resources with the 5-largest counts comprise the query answer.

On the basis of these mappings, social searches can be built on top of any existing DBMS and the associated operations can be accelerated using OLAP techniques.

5. OLAP IMPLEMENTATION

In this section, we provide various algorithms for answering top- k aggregate queries of the form presented above: a straightforward solution that calculates aggregates on the fly, an improved version that takes into account the particular form of the expected workload, and a method that accesses pre-aggregated data stored in a cube.

5.1 On-the-Fly Aggregation

In this subsection, we provide a general sketch of an algorithm that performs on-the-fly aggregation (Algorithm *OTFA*) in order to answer a top- k aggregate query Q of the form described in Section 4. *OTFA* takes as input the following parameters: the fact table \mathcal{R} , the condition *Cond* as described by the *WHERE* clause of Q (expressed in SQL, as in the example Q_1), and the set *GBS* of dimensions that participate in the *GROUP BY* clause of Q . Note that this

Algorithm OTFA(\mathcal{R} , $Cond$, GBS)

1. Fetch from \mathcal{R} the set of tuples \mathcal{T} that satisfy $Cond$
 2. Sort \mathcal{T} according to the dimensions in GBS
 3. $t_{prev} = \text{FirstTuple}(\mathcal{T})$
 4. $\mathcal{S} = \{t_{prev}\}$
 5. $TopK = \emptyset$
 6. **for each** subsequent tuple $t \in \mathcal{T}$ do
 7. **if** $\text{Project}(t, GBS) = \text{Project}(t_{prev}, GBS)$
 8. $\mathcal{S} = \mathcal{S} \cup \{t\}$
 9. **else**
 10. $t_{new} = \text{Aggregate}(\mathcal{S})$
 11. **if** the aggregate of $t_{new} > \text{min. aggregate in } TopK$
 12. $\text{Update}(TopK, t_{new})$
 13. **end if**
 14. $\mathcal{S} = \{t\}$
 15. **end if**
 16. $t_{prev} = t$
 17. **end for**
 18. $t_{new} = \text{Aggregate}(\mathcal{S})$
 19. **if** the aggregate of $t_{new} > \text{min. aggregate in } TopK$
 20. $\text{Update}(TopK, t_{new})$
 21. **end if**
 22. Write $TopK$ to the output
-

is a general sketch of an algorithm and not necessarily the algorithm of choice in a real DBMS, since the optimizer of such a system may generate a different, albeit equivalent, execution plan, according to the particular statistics it holds in its catalogs. For example, it could perform projection before selection or it could pipeline selection and sorting. The description of such optimizations goes beyond the scope of this paper and can be found elsewhere [8]. Our purpose here is to describe the main steps of such an execution plan.

OTFA fetches from \mathcal{R} the set \mathcal{T} of tuples that satisfy $Cond$ (ln: 1), e.g., all tuples in \mathcal{R} tagged with ‘portal’ for \mathcal{Q}_1 . If \mathcal{R} is indexed, tuple selection can be accelerated by exploiting the proper index. Then, *OTFA* re-orders the tuples of \mathcal{T} according to their values along the dimensions that belong to GBS (ln: 2). It does so in order to bring in adjacent memory positions tuples that match, i.e., tuples with equal values along the dimensions in GBS , since these tuples must be aggregated together. (An alternative method would use hashing instead of sorting, but this would not affect the general sketch of the algorithm.) Subsequently, *OTFA* initializes some proper data structures (ln: 3-5): t_{prev} is a variable that keeps a tuple in order to compare it with the subsequent ones in the loop that follows (ln: 6-17), \mathcal{S} is a set of matching tuples, and $TopK$ is an array of k positions storing the aggregated tuples with the k -largest aggregates.

After the initialization of these variables, *OTFA* scans through all the tuples in \mathcal{T} (ln: 6) and for every such tuple t , it checks whether t matches with the previous tuple t_{prev} (ln: 7). If the condition holds, *OTFA* simply appends t in \mathcal{S} (ln: 8). Otherwise, it aggregates the matching tuples in \mathcal{S} in order to generate tuple t_{new} (ln: 10), which will be potentially written in the output, if its aggregate value is large enough, so that t_{new} enters and survives in $TopK$ (ln: 11-13). Then, *OTFA* clears \mathcal{S} and re-initializes it with the current tuple t (ln: 14). After the loop, *OTFA* processes the final set of matching tuples (ln: 18-21), as also described above, and finally, it writes to the output the tuples that have survived in $TopK$, which consist the results for the query \mathcal{Q} .

5.2 Accelerating On-the-Fly Aggregation

The nature of the advanced social searches we explore in this paper induces that the queries necessary for discovering

Userid	TagId	DocId	Count
1	1	1	1
1	1	4	1
1	2	4	1
2	3	2	1
3	1	3	1
3	1	4	1

Userid	TagId	Count
1	1	2
1	2	1
2	3	1
3	1	2

Userid	DocId	Count
1	1	1
1	4	2
2	2	1
3	3	1
3	4	1

TagId	DocId	Count
1	1	1
1	4	2
2	4	1
3	2	1
3	3	1

Userid	Count
1	3
2	1
3	2

DocId	Count
1	1
2	1
3	1
4	3

TagId	Count
1	4
2	1
3	1

Count
6

Figure 3: The data cube of \mathcal{R} .

Algorithm OLA(\mathcal{R} , $Cond$, GBS)

1. $\mathcal{N} = \text{Dimensions in } Cond \cup GBS$
 2. Fetch from $\mathcal{N} \in \text{Cube}(\mathcal{R})$ the set of tuples \mathcal{T} that satisfy $Cond$
 3. $TopK = \emptyset$
 4. **for each** tuple $t \in \mathcal{T}$ do
 5. **if** the aggregate of $t > \text{min. aggregate in } TopK$
 6. $\text{Update}(TopK, t_{new})$
 7. **end if**
 8. **end for**
 9. Write $TopK$ to the output
-

information associated to a given entity can be answered by accessing the same set of tuples in the fact table; in other words, they share the same *WHERE* clause. Returning to our running example, the social search for the tag ‘portal’ is mapped to the aggregate queries \mathcal{Q}_1 and \mathcal{Q}_2 , which both have the same *WHERE* clause. This remark enables us propose a rather straightforward, but significant improvement for *OTFA* in order to accelerate the execution overall. Fetching the tuples of \mathcal{R} that satisfy $Cond$ (ln: 1) is necessary only for the first query. Subsequent queries that refer to the same entity can reuse the same set of tuples \mathcal{T} produced by the first one; hence, their execution can start from ln: 2 of *OTFA*, by resorting \mathcal{T} according to the dimensions that participate in their own *GROUP BY* clause.

Note that fetching from \mathcal{R} the set of tuples \mathcal{T} that satisfy a given condition is not always a trivial operation. For example, if \mathcal{R} is very large and the condition involves popular entities, a situation common in real-world applications, then the number of tuples in \mathcal{T} can be very large. Accessing a large number of tuples, most probably using random access due to the use of indices, can be very costly in terms of execution time. Saving this considerable cost expectedly offers a significant improvement as we see in the next sections.

5.3 Off-Line Aggregation

A popular trend in OLAP used as an alternative to on-the-fly aggregation involves building data cubes in order to precompute aggregates in an off-line fashion. Figure 3 shows the data cube that corresponds to the fact table of Figure 2. Every node in the cube represents a group-by query and is labeled with its grouping attributes, which consist of the subset of dimensions that participate in the group-by clause of the corresponding query. Following the same trend, we present a general sketch of an algorithm (*Algorithm OLA*) that uses data stored in a data cube in order to answer a top- k aggregate query \mathcal{Q} of the form described in Section 4

for advanced social searches. *OLA* takes the same input parameters as *OTFA* plus the cube $Cube(\mathcal{R})$ that corresponds to the fact table \mathcal{R} and has been built off-line.

It can be proven that the most specialized node \mathcal{N} of the cube that *OLA* must access for answering \mathcal{Q} is the one with grouping attributes the union of the dimensions that participate in the condition *Cond* with the dimensions that participate in the *GROUP BY* clause *GBS* (ln: 1). The proof is straightforward and based on the properties of the queries in our query model.

Example (cont'd). The query \mathcal{Q}_1 is equivalent to \mathcal{Q}'_1 , where the set of dimensions in *Cond* is $\{TagId\}$ and *GBS* = $\{DocId\}$; hence, $\mathcal{N} = \{TagId\} \cup \{DocId\} = \{TagId, DocId\}$:

```

 $\mathcal{Q}'_1$ : SELECT TOP 5 DocId, COUNT(*)
FROM  $\mathcal{R}$  WHERE TagId = 1
GROUP BY TagId, DocId
ORDER BY COUNT(*) DESC;

```

On the basis of this equivalence, *OLA* does not perform any aggregation on the fly. Instead, it fetches from node \mathcal{N} of the cube of \mathcal{R} the set \mathcal{T} of tuples that satisfies *Cond* (ln: 2). Again, as also explained in the case of *OTFA*, *OLA* can accelerate the aforementioned selection by exploiting the existence of indices. Subsequently, *OLA* initializes the structure *TopK* (ln: 3), which is again an array of k positions necessary for storing the aggregated tuples with the k -largest aggregates, and then, it scans through every tuple in \mathcal{T} one-by-one (ln: 4). For each t , *OLA* checks whether its aggregate value is large enough, so that t enters in *TopK* (ln: 5-7). Finally, *OLA* writes to the output the tuples that have survived in *TopK* (ln: 9), which consist the answer for the original query \mathcal{Q} .

5.4 Selection of Cube Implementation

Algorithm *OLA* is general in that it can be combined with any data-cube implementation. In our implementation, we have chosen CURE [11], due to several properties that make it attractive for social searches, as explained below.

- It has been found efficient even when dealing with large volumes of data in various scenarios and generates a fully materialized, yet very compact cube, removing all types of redundancy from the final result. These properties are important for social searches, since the datasets in many real-world applications are extremely large.
- It has been found efficient on incremental maintenance as well, attributing its performance to a lazy approach, which updates the cubes on the fly during query processing without harming the performance of queries [10]. Efficiency during incremental maintenance is important for social searches, since social datasets are typically dynamic in nature as new postings arrive continuously.
- It supports very fast iceberg aggregate queries, i.e., queries that return only tuples with aggregate value greater than a minimum support threshold. Since our expected workload involves top- k aggregate queries, we can conclude that cube tuples with small aggregate values rarely survive in the result set, as described above. Hence, avoiding the access of such tuples is a significant optimization.

6. EXPERIMENTAL EVALUATION

In order to evaluate the cube-based technique for advanced social search against the on-the-fly aggregation, we have im-

plemented algorithm *OLA* (Section 5.3) and the two versions of algorithm *OTFA*, the naïve (Section 5.1) and the optimized one (Section 5.2). In our tests, we have used a dataset consisting of a large-scale crawl of del.icio.us performed in September 2006. The crawl was breadth first from the tag ‘web’ (details can be found elsewhere [6]). This dataset consists of 211,401,624 postings, 514,056 users, 915,989 different tags, and 1,271,845 unique URLs. We have run our experiments on a Pentium 4 (2.8 GHz) PC with 512 MB memory under Windows XP. In this paper, we focus on query response times and not on cube or index construction, since the latter have been studied elsewhere [11].

6.1 Methodology

A main factor affecting the performance of the algorithms presented is the number of tuples in the relation queried that satisfy the condition *Cond* (defined in Section 5). A large number of tuples forces a large number of disk seeks, which usually follow a random-access pattern induced by the use of indices, and further generates greater costs for sorting (or hashing) and aggregation; hence, expectedly, the average query response times will degrade, particularly in the case of algorithms that aggregate data on the fly. Two parameters mainly decide the number of tuples that satisfy *Cond*: (a) the size of the dataset and (b) the ‘‘popularity’’ of the entities that participate in *Cond*. Searches over larger datasets for information related to popular (i.e., frequently used) entities will expectedly generate heavier workloads. In order to evaluate the impact of these parameters on the performance of OLAP-style social searches, we have experimented with various combinations of their values.

Starting from the original dataset obtained from del.icio.us, we generated three datasets: a small, a medium, and a large one, by keeping 1%, 10%, and 100% of its tuples, respectively. The sizes of these datasets in a binary format are approximately 32 MB, 322 MB, and 3.14 GB, respectively. Secondly, we generated various query workloads using as parameter the percentage of the queries in the workload that are associated with popular entities. We have assumed that an entity is popular, if it belongs to the top-500 entities ranked based on their frequency in a given dataset. We generated different workloads consisting of 250 random top- k aggregate queries for every dataset, varying the percentage of popular entities between 0% and 100% with an increment of 25%. These workloads correspond to different usage scenarios. A small percentage indicates that the majority of searches involve rather infrequent entities. A large one shows that most searches are for frequent entities.

6.2 Results

Figure 4 shows the average query response times (AQRT) generated by the three algorithms for answering queries of various workloads over the large dataset. Figures 5 and 6 do the same for the medium and the small datasets, respectively. The x-axis of all graphs indicates the percentage of queries in each workload that are associated to popular entities. The y-axis is logarithmic. We observe that the optimized version of *OTFA* always behaves better than the naïve one due to its ability to exploit intermediate data calculated by previous queries, as explained in Section 5.2. The most significant trend exhibited by all graphs is that the cube-based algorithm (*OLA*) outperforms the other two, which perform aggregation on the fly. This is attributed to this al-

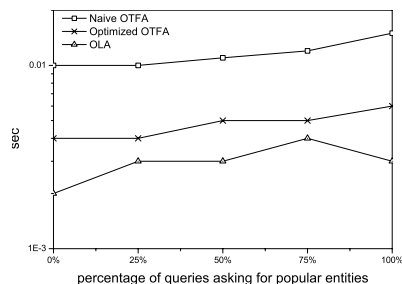
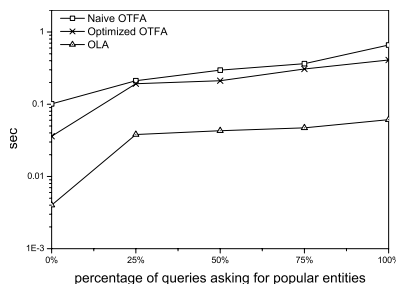
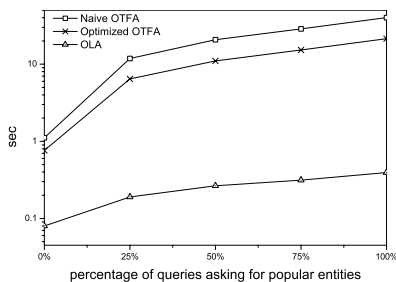


Figure 4: AQRT, large dataset. Figure 5: AQRT, medium dataset. Figure 6: AQRT, small dataset.

gorithm’s simplicity with respect to both versions of *OTFA*: *OLA* accesses a smaller relation (any cube node \mathcal{N} is at most as big as the fact table \mathcal{R} and most probably much smaller, since it stores aggregated data), it performs no sorting operation on the grouping attributes, and it does not aggregate anything during query processing.

The difference in execution times is considerably in favor of *OLA*, especially for the most realistic workloads, which involve queries over the largest dataset (Figure 4). For such workloads, *OLA* is approximately two orders of magnitude faster than its counterparts. This observation is important because it shows that while on-the-fly aggregation would seem to be the only solution of choice given the dynamic nature of social data, in practice it is not a good solution because social datasets are typically very large. On the other hand, the improvement gained with *OLA* shows that using cubes is a viable solution.

For the medium dataset the difference is one order of magnitude in favor of *OLA*, whereas for the small dataset, *OLA* is approximately two times faster. Note that the differences become less marked as the size of the dataset decreases due to the effect of caching, which decreases the number of disk seeks that are necessary for answering a query, since data can be found resident in main memory with great probability. Clearly, caching is more effective as the percentage of the fact-table tuples that fit in main memory increases and hence, as the size of the dataset decreases.

Finally, it is evident that searching for information associated to popular entities requires more time on average, at least for the large and the medium datasets because popular entities participate in a large number of postings. Therefore, answering queries related to them involves accessing a larger number of tuples. This trend is less marked in the small dataset due to the effect of caching.

7. CONCLUSIONS AND FUTURE WORK

Social bookmarking makes the web more “pluralistic”: entities of different types (e.g., users, resources, tags) coexist and get interconnected. In this paper, we have proposed that search in the social web (social search) needs to allow starting from any type of entity and requesting aggregated views of the entities related to it taking into account entity relationships. We mapped this type of social searching to OLAP query processing. We studied various ways to support on-the-fly aggregations of social data for social searches. We further considered data cubes for precomputing and materializing the results of all possible aggregate queries over a social dataset. Our experiments have shown the potential of cubes for supporting social searches. Interestingly, they behave smoothly even for searches that combine popular en-

tities over large datasets making them possibly the “giants to hold OLAP-style social searches on their shoulders”. Many interesting research issues arise. For example, building and optimizing sophisticated interaction schemes on the basis of the social search paradigm presented, such as iteratively refined social searches, designing appropriate user interfaces, and so forth.

8. REFERENCES

- [1] S. Bao, G.-R. Xue, X. Wu, Y. Yu, B. Fei, and Z. Su. Optimizing web search using social annotations. In *Proc. of the 16th WWW Conf.*, pages 501–510, 2007.
- [2] O. Ben-Yitzhak, N. Golbandi, N. Har’El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev. Beyond basic faceted search. In *Proc. of 1st Int’l Conf. on Web Search and Data Mining (WSDM)*, pages 33–44, 2008.
- [3] Del.icio.us. url: <http://del.icio.us/>.
- [4] Flickr. url: <http://www.flickr.com/>.
- [5] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proc. of Int’l Conf. on Data Engineering (ICDE)*, pages 152–159, 1996.
- [6] P. Heymann, G. Koutrika, and H. Garcia-Molina. Can social bookmarking improve web search?. In *Proc. of the 1st ACM Int’l Conf. on Web Search and Data Mining (WSDM)*, 2008.
- [7] A. John and D. Seligmann. Collaborative tagging and expertise in the enterprise. In *Proc. of the CWTW Workshop in conj. with the 15th WWW Conf.*, 2006.
- [8] C. Li, K. C.-C. Chang, and I. F. Ilyas. Supporting ad-hoc ranking aggregates. In *Proc. of ACM SIGMOD*, pages 61–72, 2006.
- [9] G. Mishne. Autotag: collaborative approach to automated tag assignment for weblog posts. In *Proc. of the 15th WWW Conf.*, 2006.
- [10] K. Morfonios. Cube-lifecycle management and applications. *PhD Dissertation, University of Athens*, 2007.
- [11] K. Morfonios and Y. Ioannidis. CURE for cubes: Cubing using a ROLAP engine. In *Proc. of Very Large Data Bases (VLDB)*, pages 379–390, 2006.
- [12] P. Schmitz. Inducing ontology from flickr tags. In *Proc. of the Collab. Web Tagging Workshop in conj. with the 15th WWW Conf.*, 2006.
- [13] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis. Dwarf: shrinking the petacube. In *Proc. of ACM SIGMOD*, pages 464–475, 2002.
- [14] Tag cloud. http://en.wikipedia.org/wiki/tag_cloud.
- [15] Technorati. url: <http://www.technorati.com/>.
- [16] Z. Xu, Y. Fu, J. Mao, and D. Su. Towards the semantic web: Collaborative tag suggestions. In *Proc. of the CWTW Workshop in conj. with the 15th WWW Conf.*, 2006.
- [17] Y. Zhao, P. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. of ACM SIGMOD*, pages 159–170, 1997.