# Bulk Sorted Access for Efficient Top-k Retrieval

Dustin Lange
Hasso Plattner Institute, Potsdam, Germany
dustin.lange@hpi.uni-potsdam.de

Felix Naumann
Hasso Plattner Institute, Potsdam, Germany
naumann@hpi.uni-potsdam.de

## ABSTRACT

Efficient top-$k$ retrieval of records from a database has been an active research field for many years. We approach the problem from a real-world application point of view, in which the order of records according to some similarity function on an attribute is not unique: Many records have same values in several attributes and thus their ranking in those attributes is arbitrary. For instance, in large person databases many individuals have the same first name, the same date of birth, or live in the same city. Existing algorithms, such as the Threshold Algorithm (TA), are ill-equipped to handle such cases efficiently.

We introduce a variation of TA, the Bulk Sorted Access Algorithm (BSA), which retrieves larger chunks of records from the sorted lists using fixed thresholds, and which focusses its efforts on records that are ranked high in more than one ordering and are thus more promising candidates. We experimentally show that our method outperforms TA and another previous method for top-$k$ retrieval in those very common cases.

## 1. TOP-K SEARCH IN LARGE DATABASES

Given a set of objects, a query object, and an overall similarity function (also called ranking, scoring, or aggregation function), the goal of top-$k$ retrieval is to determine the $k$ objects in the set that have highest overall similarity to the query object. The similarity function is often a monotonous function that combines a set of base similarity functions, each responsible for calculating the similarity of a specific aspect. For example, an overall similarity function for records in a person database may be the weighted sum of similarity functions for the first name, last name, and city attributes of a person record.

Fagin's Threshold Algorithm (TA) is among the most well-known algorithms for retrieving the top-$k$ objects from a database [3]. Its main idea is to retrieve similar objects from several sorted lists. The lists offer a view on all objects, each sorted by their similarity regarding *one* of the base similarity functions. Various improvements of the original algorithm have been proposed, as surveyed by Ilyas et al. [4]. TA and its variants use a simple round-robin approach to access the attribute lists [1, 3], or they consider how fast similarity values decrease in the sorted lists [6].

However, the proposed solutions cannot handle data with frequent attribute values well: For instance, consider a database of US citizens and a query for a person with the first name `Peter`, last name `Smith`, and city `New York`. TA now prepares a sorted list for each given query value. Because of the frequent query values, all sorted lists start with many records with a similarity value of 1.0 (representing exact matches). In these sorted lists, all records with the *same* similarity value are ordered arbitrarily. Such arbitrary order can also occur for similarity values below 1.0: for example, a similarity function for zip codes might only count the matching digits, so that the function produces only a small set of different similarity values.

For algorithms that rely on a round-robin approach for accessing the sorted lists, the probability is high that many records need to be evaluated before the overall most similar records can be found. In our example, there are many people with the last name `Smith` who do not live in `New York`, and many `New York` citizens with a last name different than `Smith`. All those irrelevant records may have higher positions in the sorted lists than a person that has similar values for all three query values. To avoid this problem, we should favor records that have *several* similar values. Finding relevant records earlier allows earlier pruning of irrelevant records as well as providing better results when only limited query time is available.

We exploit these observations in our Bulk Sorted Access Algorithm (BSA). Our idea is to first perform a bulk sorted access, i.e., to retrieve a bulk of records with a similarity value above a threshold from each sorted list, in particular *all* those with same similarity values. We have one or more attribute similarity values for each retrieved record, and there may be missing similarity values. We combine the retrieved information into a priority score where the priority represents the maximum achievable similarity (upper bound) of each record. We process the records according to their priority: With the available information, we perform comparisons with the most promising records first – a significant advantage if only limited query time is available.

Moreover, we can reduce the number of required attribute and overall similarity calculations as records with low maximum achievable similarity can be pruned earlier. We show in an evaluation on two large real-world data sets (10m and 2.2m records) that our method outperforms TA and another previous method [1] for top-$k$ retrieval in most cases.

## 2. RELATED WORK

Ilyas et al. provide a recent survey on top-$k$ retrieval algorithms in relational database systems [4]. Our work is based on Fagin's Threshold Algorithm (TA) [3]. We also retrieve a set of records with highest similarity for each attribute. In contrast to TA, we first perform a bulk sorted access and then aggregate available information. Because TA is among the most popular top-$k$ retrieval algorithms and shares many ideas with subsequent work, we perform empirical comparisons with TA in Section 5. In another work, Fagin et al. define novel metrics for rankings with many ties [2], while our work focuses on arbitrary monotonous overall metrics.

A related approach is proposed by Bruno et al. [1]. In the same way as TA, their approach "Upper" first retrieves records from sorted lists. Then it builds a priority queue with the retrieved information, an idea that we adopt in our approach. Their approach then schedules the next sorted and random accesses for the different lists. In contrast to this fine-grained approach, we perform a bulk sorted access only at the beginning and then process the retrieved record lists as efficiently as possible. Similar to TA, Upper performs sorted accesses in a round-robin style and has thus similar drawbacks regarding records with several frequent values, as our comparative evaluation in Section 5 confirms.

Some researchers have analyzed algorithms that have only a limited amount of time (or limited number of sorted and random accesses). In our own previous work on similarity range queries, we suggest to determine a query plan for accessing the sorted lists with high recall and cost below the specified cost threshold [5]. A different approach, by Shmueli-Scheuer et al., first retrieves an initial set of records from all available sorted lists and then tries to guess which sorted list to access next to achieve highest recall given the limited amount of sorted and random accesses [6]. Their approach explicitly exploits a given budget for the amount of allowed accesses. In contrast, our approach is, in their terms, budget-oblivious. However, due to the priority queue used, it can be useful in scenarios with limited query time.

## 3. PROBLEM SETTING

We follow the common notion of defining individual similarity measures for different attributes and attribute types. These individual similarities are subsequently combined to define the global similarity of two records (in this case: the query record and the database record).

We define a set of **base similarity measures** $sim_a(q, r)$, each responsible for calculating the similarity of the values from the specific attribute $a$ of the compared records $q$ and $r$ from a universe $U$ of possible records. Record $q$ is typically the query record and record $r$ a record in the data set. In our person data use case, we have the base similarity measures $sim_{\mathsf{FirstName}}$, $sim_{\mathsf{LastName}}$, $sim_{\mathsf{BirthDate}}$, $sim_{\mathsf{City}}$, and $sim_{\mathsf{Zip}}$, which can be chosen independently. For example, we could use Jaro-Winkler distance for $sim_{\mathsf{FirstName}}$, the relative distance between dates for $sim_{\mathsf{BirthDate}}$, and the numeric difference for $sim_{\mathsf{Zip}}$. We assume the domain of the similarity measures to be between 0 and 1, with 1 representing identity and 0 dissimilarity of the compared record parts: $sim_a : (U \times U) \rightarrow [0, 1] \subset \mathbb{R}$. A **composed similarity measure** $sim_{Overall}$ (also known as ranking or aggregation function) uses the base similarity measures to derive an overall similarity of the two compared records.

With a query record $q \in U$ and a record set $R \subseteq U$, a **top-$k$ query** retrieves a set $S$ of $k$ records from $R$ where the following condition holds (following [7]): $\forall r_k \in S : \forall r_n \in R \setminus S : sim_{Overall}(q, r_n) \leq sim_{Overall}(q, r_k)$. Our goal is to answer top-$k$ queries with a low number of overall similarity calculations, especially in the presence of many attribute values with same similarity.

## 4. BULK SORTED ACCESS ALGORITHM

One of the most popular top-$k$ retrieval algorithms is Fagin's Threshold Algorithm [3], which is the basis of our approach. As we described in Section 1, TA performs poorly in cases of many frequent attribute values. If there are many records with the same similarity (for instance, if they have the same attribute value), there is no use of the sortation for those records. TA then depends on the (random) position of the matching record in the sorted list. Also, TA does not recognize records that have high similarities for multiple attribute values. In a situation where many records have a high similarity in only one attribute ("similarity outliers"), TA spends much effort on processing probably irrelevant records.

Our Bulk Sorted Access Algorithm (BSA) addresses TA's drawbacks. The main idea of BSA is to first retrieve high similarity records for all attributes, and then combine the results into a priority queue. With BSA, the most promising records are considered first, and the search can often be stopped even earlier than with TA. BSA sequentially performs the following steps:

First, we perform a **bulk sorted access** for every available attribute $a \in A$ and its given threshold $\theta_a$. We retrieve all records $r \in R$ with $sim_a(q, r) \geq \theta_a$. We later discuss the significance of appropriate selection of the **retrieval threshold $\theta_a$**. For every retrieved record, we now know the similarity regarding one (and sometimes more) attributes. We store the available information for each record in a table.

The next step is to **aggregate the retrieved attribute information**. We observe that the combinations of attribute similarities are often not unique. For instance, there may be many records where we have retrieved only the attribute similarity $sim_{Name}(q, r) = 1$. To save any further similarity calculations, we group all records according to the retrieved attribute similarities. A group of records consists of all records with the same values for all attribute similarities (including missing similarities).

We then build a **priority queue** to determine the order in which the retrieved record groups are processed. Our goal is to have the most promising record groups at the top, i.e., the records with probably highest similarity should have the highest priority. For any attribute $a$ of a record

$r$ from a record group, we either have retrieved the exact attribute similarity $sim_a(q, r)$ or we know that $\theta_a$ is the highest possible similarity (because otherwise we would have retrieved the exact similarity). This allows an upper bound estimation for any retrieved record group: We determine the highest possible overall similarity using $sim_{Overall}$ on the attribute similarities (either retrieved or with the threshold $\theta_a$) and use the result value as the priority.

Finally, we **process the record groups** according to the determined priority. The processing order of records within any group is irrelevant as any such record has the same available attribute similarities. For each record, all missing attribute similarities are determined using random accesses, and the overall similarity is calculated. At any point, we keep a list of the $k$ records with highest overall similarity seen so far. Ties are broken arbitrarily. After processing a group, we decide whether we need to evaluate any further group. We use the smallest similarity of the current $k$ highest observed similarities as the **threshold** $\phi$. If the highest possible similarity of the next group (i.e., its priority value) is lower than $\phi$, the next group as well as all other groups are discarded. The result consists of the $k$ most similar records.

## 5. EVALUATION

In this section, we discuss experimental results from two real-world data sets. We analyze the performance of BSA regarding different values for the retrieval threshold. In addition, we show comparative results of BSA with TA and Upper, another state-of-the-art approach for top-$k$ retrieval [1]. Upper retrieves record IDs via sorted access in a round-robin style and schedules further sorted and random accesses according to the expected similarity values and their upper bounds for the retrieved records.

### 5.1 Data Sets and Evaluation Settings

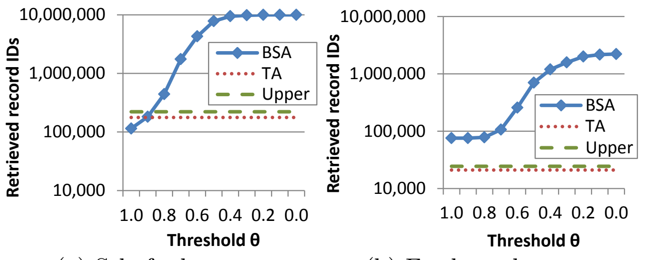
**Schufa data set.** Our first data set stems from Schufa, a German credit-rating agency, and consists of two parts: a person data set (66m records) and a query data set (2m queries). We randomly selected 10m records from the person data set, so that the data fits into main memory of our evaluation machine. The relevant fields for our search problem are name, birth date, and address data (street, city, zip). We randomly selected 1,000 queries (where each query has a corresponding match in the selected record subset).

**Freebase data set.** Freebase is an online knowledge base, managed by community experts. For our experiments, we have used the complete set of 2.2m person records available in Freebase (where at least a name is given). We use the most commonly filled attributes name, birth date, birth place, nationality, and profession. We randomly selected 1,000 records from the record set as queries.

For both data sets, we use the weighted average of the attribute similarities as overall similarity measure. We performed all tests on a workstation PC, running Windows XP on an Intel Core2 Quad 2.5 GHz CPU and 8 GB RAM.

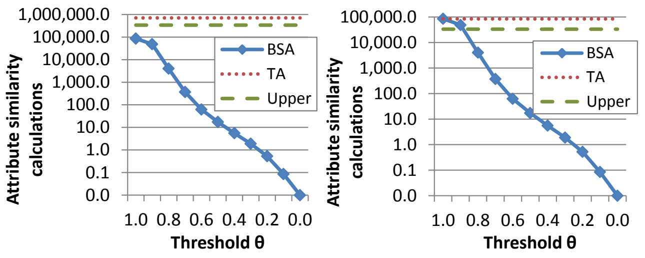
### 5.2 Experiments

We now experimentally examine the efficiency and recall of BSA in comparison to TA and Upper. We evaluated the performance of BSA for different values of $\theta_a$. For conciseness of the results, we used the same value $\theta$ for all attributes. Figures 1 – 3 show results for both data sets with $k = 5$. For BSA we report experimental results for different values of $\theta$, while for TA and Upper the results are independent of $\theta$.



(a) Schufa data set (b) Freebase data set

Figure 1: Number of retrieved record IDs



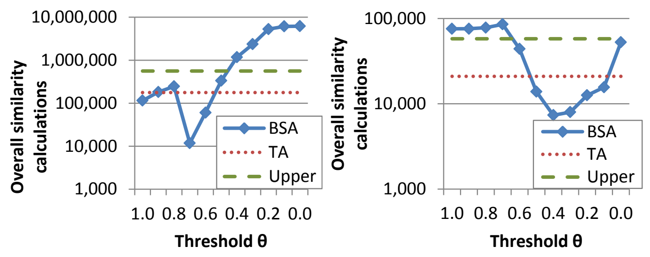
(a) Schufa data set (b) Freebase data set

Figure 2: Number of attribute similarity calculations

All compared algorithms first retrieve information about the records via sorted access. Figure 1 shows the overall number of retrieved record IDs via sorted access. For BSA, this value increases with smaller $\theta$. For the Schufa data set, almost all record IDs are retrieved with $\theta \leq 0.3$ (for Freebase with $\theta \leq 0.2$). In comparison to TA and Upper, BSA retrieves a larger number of record IDs for $\theta \leq 0.8$ for Schufa (and for any $\theta$ for Freebase). This disadvantage is mitigated in the next steps of the algorithm.

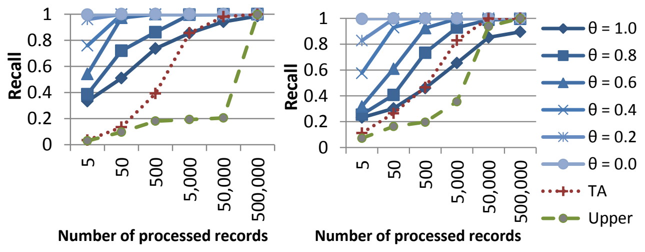
After processing the retrieved information into a priority queue, BSA calculates missing attribute similarities for a number of records. In Figure 2 the performed similarity calculations are shown. With smaller values of $\theta$, fewer attribute similarities are calculated. In these cases, more similarity values have already been retrieved from the sorted lists in the previous step (Figure 1). Another reason is that for lower thresholds more record groups can be discarded.

We show in Figure 3 the number of overall similarity calculations. The number of overall similarity calculations first increases until $\theta = 0.8$ for the Schufa data set. Up to this point, all record groups are processed. With $\theta = 0.7$, some record groups can be discarded, and the number of groups is still relatively low. With smaller $\theta$, more groups are discarded, but there is a much larger number of record groups, for which the priority needs to be calculated. In these cases, the dominant part of the overall similarity calculations is now calculating the group priorities. Both TA and Upper perform more comparisons than the best case



(a) Schufa data set (b) Freebase data set

Figure 3: Number of overall similarity calculations



(a) Schufa data set (b) Freebase data set

Figure 4: Recall with a limited number of comparisons

for BSA ($\theta = 0.7$) and fewer than the worst case for BSA ($\theta = 0.0$). Upper performs more comparisons than TA, because it recalculates a record's priority after each attribute similarity calculation. The measurements for the Freebase data set similarly show a slight increase until $\theta = 0.7$, then have a minimum value with $\theta = 0.4$ and then increase for the remaining values, when most similarity calculations are required for the larger number of record groups. Again, TA and Upper perform worse than the best case of BSA ($\theta = 0.4$) and better than the worst case of BSA ($\theta = 0.7$).

The runtime of the compared algorithms depends primarily on the number of performed similarity calculations and retrieved records, but also on the time required to perform these operations. In our setting, we expect to retrieve record IDs (sorted accesses) fast (e. g., using an index), while any additional similarity calculations happen at query time and can be expensive. BSA requires only few attribute similarity calculations compared to the number of overall similarity calculations, so that the number of overall similarity calculations is indeed the dominant factor for runtime. The runtime of BSA is always better than that of TA and Upper, for its best case ($\theta = 0.7$, $t = 5s$) only a fraction of their runtime for the Schufa data set (TA: $t = 50s$, Upper: $t = 59s$). All time measurements include an average time of $2.7s$ for preparing the lists of similar values. For Freebase, BSA performs only a few seconds better than both TA ($t = 24s$) and Upper ($t = 23s$) even in its best case ($\theta = 0.5$, $t = 17s$). However, in this case on average a time of $9.5s$ is required for preparing the lists of similar values, which shows that the gain of BSA is in fact larger than the raw numbers reveal.

To examine how the compared algorithms perform under resource constraints, we analyzed the recall after performing only a limited number of comparisons (also for $k = 5$). Both BSA and TA keep a list of the $k$ seen records with highest similarity, which we use for measuring recall after the limited number of comparisons. Because Upper does not keep a temporary result list, we considered a record as found only if is has been returned it as a result, which may happen late in the query answering process. Because of this difference, Upper performs worse than both BSA and TA regarding recall for both data sets. Figure 4 shows that with lower $\theta$, we achieve high recall earlier. The reason is that a low $\theta$ results in more available information per record, so that the estimation of the overall similarity becomes more reliable, and the priority of the record groups represents a better processing ordering of the records. For the Schufa data set, BSA performs better than TA for any value of $\theta$. Already with $\theta = 1.0$ and after comparing only with the top 5 records from the priority queue, BSA outperforms TA. With lower $\theta$, the gain of BSA is even larger, while for $\theta = 0.0$, BSA has a recall of 1.0 (because all record IDs and similarity values are already known). After performing more comparisons, recall increases also for higher $\theta$, and is in most cases higher than the recall of TA. For the Freebase data set, BSA outperforms TA for $\theta \leq 0.8$. For small numbers of processed records, any value of $\theta$ results in a higher recall for BSA, albeit the gain of BSA is large only for small $\theta$.

## 6. CONCLUSION AND OUTLOOK

The Bulk Sorted Access Algorithm (BSA) for top-$k$ retrieval extends the well-known Threshold Algorithm (TA). BSA is optimized for the frequent use case of queries and data sets with many same similarity values in the lists of similar attribute values, where TA and its variants perform many unnecessary comparisons. Experimental results on real-world data sets show that BSA outperforms TA and Upper in many cases, but comes with the need for appropriately configured similarity thresholds; too high thresholds can cause relevant records to be not found. We are planning to handle this drawback by an extended algorithm, which is guaranteed to find all similar records at potentially higher costs.

## 7. REFERENCES

[1] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems (TODS)*, 27:153–187, 2002.

[2] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and aggregating rankings with ties. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, 2004.

[3] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, 2001.

[4] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40:11:1–11:58, 2008.

[5] D. Lange and F. Naumann. Efficient similarity search: Arbitrary similarity measures, arbitrary composition. In *Proc. of the Intl. Conf. on Information and Knowledge Management (CIKM)*, 2011.

[6] M. Shmueli-Scheuer, C. Li, Y. Mass, H. Roitman, R. Schenkel, and G. Weikum. Best-effort top-k query processing under budgetary constraints. In *Proc. of the Intl. Conf. on Data Engineering (ICDE)*, 2009.

[7] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search - The Metric Space Approach*. Springer, 2006.