Scalable Similarity Search with Dynamic Similarity Measures

Martin Köppelmann[2], Dustin Lange[1], Claudia Lehmann[2], Marika Marszalkowski[2], Felix Naumann[1], Peter Retzlaff[2], Sebastian Stange[2], Lea Voget[2]
Hasso Plattner Institute, Potsdam, Germany
[1] firstname.lastname@hpi.uni-potsdam.de
[2] firstname.lastname@student.hpi.uni-potsdam.de

ABSTRACT

Similarity search on structured data assumes some similarity measure on the data – often a combination of individual measures per attribute. Users of a similarity search system may have different requirements on the similarity measure; the individual measures can be combined in many different ways, including a simple weighted sum of the similarities with varying weights, or, at the other end of the spectrum, much more complex machine learning techniques. Previous approaches to similarity search work only with static similarity measures or cannot exploit dynamic similarity measures.

In this paper, we present the DySim algorithm, a novel approach that answers similarity queries with query-specific configurations of similarity measures: For any query, users are allowed to define an arbitrary, including non-metric, overall similarity measure that is based on similarities of attribute values. This freedom allows to provide different search experiences to different user groups. Our approach creates a similarity index for each individual attribute. We then dynamically generate a query plan by using positive ad-hoc sample results and apply a filter-and-refine approach to quickly retrieve the list of results. We evaluated our approach on a large real-world data set. Compared to a baseline algorithm, the DySim algorithm needs about one third less comparisons to retrieve data at the cost of only a very minor decline in recall.

1. BEYOND STATIC SIMILARITY SEARCH

Similarity search aims to find all objects in a database sufficiently similar to a given query object [16]. This problem includes an overall similarity measure that defines the similarity between two objects. Many current similarity search engines work with a static and metric overall similarity measure. This implies that the overall measure, once defined, is the same for all queries and needs to fulfill the metric conditions. Hence, the search space can be reduced by using the triangular inequality [6, 16].

Such a setting does not allow to retrieve results that are dynamically tailored to the user's needs. Different users querying the same data with similar queries may have different intentions: Imagine a location database and a user searching for *Coffee House* and an address in *Berlin* with the aim to find an arbitrary café near to him. Hence, the search engine should focus more on the location than on the name to deliver results that meet the user's needs. Another user might query *Starbucks Coffee House* in combination with an address in *Berlin*, but this time choosing the name to be the most important attribute. Although both queries look similar to the system, they have different semantics. The same use case could be applied to only one user entering both queries at different points in time and each time the overall similarity measure should be adapted to the user's current needs. To achieve good results in both different cases, a weighting of the different attributes might be included in the query. As another use case, consider a credit rating agency that answers credit rating requests from different types of clients. While online shops pay much attention to a correct address to have a reliable contact address in case of payment problems, a bank might insist on a correct birth date for legal reasons. Thus, different users have different requirements for the similarity measure.

Previous work deals with query- or user-specific similarity measures for document retrieval. Van Bunningen et al. propose a document ranking model that incorporates the context of the user [15]. Chen et al. describe an approach for adapting ranking functions to user preferences (e.g., user clickthrough logs) [7]. These examples show that there is a need for ranking applications that allow query-specific similarity measures. In this paper, we address this need for similarity search in database records.

Our DySim algorithm solves the following problem: Given an arbitrary similarity measure $sim_{overall}$ for a specific query q and an overall similarity threshold $\theta_{overall}$, retrieve all records r with $sim_{overall}(r, q) \geq \theta_{overall}$.

DySim uses a filter-and-refine approach: In the filtering step, we use the similarity index proposed by Christen et al. in [8] and query plans suggested in our earlier work [11] to pre-filter the result set. In the refine step, we select all records with $sim_{overall}(r, q) \geq \theta_{overall}$ using the overall similarity measure for the respective query. The original idea of Christen et al. averages the similarities of the individual attributes to compute the overall similarity; a static overall similarity measure is used for each query. We do not make these restrictions; our overall similarity measure can be non-metric and personalized to various situations. We assume only that it is based on fixed similarity measures of the individual attributes.

Copyright 2012 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors.
DBRank'12, in conjunction with VLDB, August 27, 2012, Istanbul, Turkey.

To evaluate our approach, we use structured data of locations from Foursquare, Gowalla, and Facebook[1], all kindly provided by `uberblic.com`. Our data set consists of almost 1.5 million locations. We selected twelve attributes that all three sources have in common; but only a small fraction of records have values for all attributes.

The contributions of this paper are:

- Definition of the novel problem of *dynamic similarity search* with query-specific similarity measures
- Dynamic query plans to speed up retrieval without the need of training data
- Evaluation of the DySim algorithm on a large, real-world location data set of 1.5 million records

The remainder of this paper is structured as follows: Section 2 provides an overview of related research in the field of similarity search. Next, in Sec. 3, we give a formal definition of the problem of similarity search with dynamic similarity measures. The idea of dynamic query plans is explained in Sec. 4 and followed by the evaluation of DySim in Sec. 5. The paper concludes with an outlook in Sec. 6.

2. RELATED WORK

For similarity search there is a variety of approaches for specific cases of similarity measures and data. If the data can be transformed into a vector space, the search of similar objects can be reduced to the search of close vectors. The Euclidean distance is a popular similarity measure for vectors and various approaches use it [1]. For the well-explored metric space, the data can have an arbitrary form, while the similarity measure needs to be a metric [16]. There is also work on efficient top-k processing in databases for vector data [3] as well as for the overall similarity measures *min* and *max* [5].

Another approach to non-metric similarity measures are AL-Trees, which exploit inverted indexes for all attribute values [9]. As stated by the authors, the AL-Tree is suitable only for attributes with very small numbers of distinct values. In our setting with very many distinct values per attribute, this approach is infeasible.

Christen et al. suggest a combination of three indexes for similarity search [8]. Blocking is used to identify similar attribute values and all values belonging to one block are stored in a block index. Additionally, the similarity of each value in a block to each other value in the same block is computed and stored in a similarity index for faster retrieval. The concrete overall similarity function that Christen et al. used is the average of the available attribute similarities. In contrast, we allow more sophisticated overall similarity functions, which can be chosen by the user and may change with every query. In Sec. 5, we compare our approach with an adjusted version of the approach by Christen et al. that incorporates an arbitrary overall similarity function.

In our previous work we suggest using *query plans* as an approach to speed up the information retrieval process [11]. A query plan defines which thresholds to apply to the individual attributes in the data filtering step. We proposed the top neighborhood algorithm to determine a near-optimal query plan for a data set. While that algorithm is designed for a static overall similarity measure, the solution presented in this paper is designed to determine a query plan for query-specific overall similarity measures.

Chaudhuri et al. propose a method that generates an initial query for record matching, a problem very similar to similarity search with (dynamic) query plans [4]. They show that their performance is comparable to domain-specific solutions and at the time the best known machine learning approach. However, their solution relies on training data, which our approach does not.

Fagin's algorithm retrieves the top k records by accessing the list of records sorted by their similarity to the query object regarding different aspects, e.g., order all pictures by their similarity to the properties *blue* and *bright* [10]. In contrast to Fagin's approach, we employ a preparation step where we generate a query plan for retrieving a set of potentially relevant matches. While our approach may have incomplete results (Fagin always has complete results), we can execute queries much faster, because in our query plans we prefer records where several attributes match well (Fagin retrieves many irrelevant records where only one attribute matches well).

3. DYNAMIC SIMILARITY SEARCH

We define the problem of dynamic similarity search as follows: Given a set of n attributes with domains A_i, the universe $U = (A_1 \times \ldots \times A_n)$ of all possible records, a similarity query $(q, sim_{overall}, \theta_{overall})$ with a query $q \in U$, an overall similarity measure $sim_{overall} : (U \times U) \to [0, 1]$, and a similarity threshold $\theta_{overall} \in [0, 1]$, and given a dataset $R \subseteq U$, find all records $r \in R$ for which $sim_{overall}(r, q) \geq \theta_{overall}$.

In addition to the overall similarity measure $sim_{overall}$, we need to define n attribute-specific similarity measures $sim_1, \ldots, sim_n$ with $sim_i : (A_i \times A_i) \to [0, 1]$. These similarity measures form the basis of the overall similarity and are used to build the similarity index.

Both the overall similarity measure $sim_{overall}$ and the attribute similarity measures $sim_1, \ldots, sim_n$ can be arbitrarily chosen. However, we assume that $sim_{overall}$ is based on the individual attribute similarities and is monotonically increasing with respect to all sim_i. That is, if for any $r, r' \in U$ $sim_i(r, q) \geq sim_i(r', q)$ then $sim_{overall}(r, q) \geq sim_{overall}(r', q)$.

While the attribute similarity measures sim_i are assumed to be arbitrary but fixed and indexed with a similarity index, the overall similarity measure $sim_{overall}$ is a parameter of the query itself, and can thus be selected independently for each query. It could, for example, be defined as a weighted sum of all attribute similarities, the maximum of all attribute similarities, or by a machine learning approach, such as a decision tree. DySim uses the given overall similarity measure to dynamically generate a query plan for the query. The query plan is used to filter possible matches from the data set. We then refine the result set by applying the overall similarity measure to each of the remaining records and selecting only those records r for which $sim_{overall}(r, q) \geq \theta_{overall}$.

3.1 Indexing Data

We first create several indexes on the attributes with the attribute similarity measures. We follow the indexing idea for record linkage proposed by Christen et al., including their suggested optimizations [8].

[1]`foursquare.com`, `gowalla.com`, and `facebook.com`

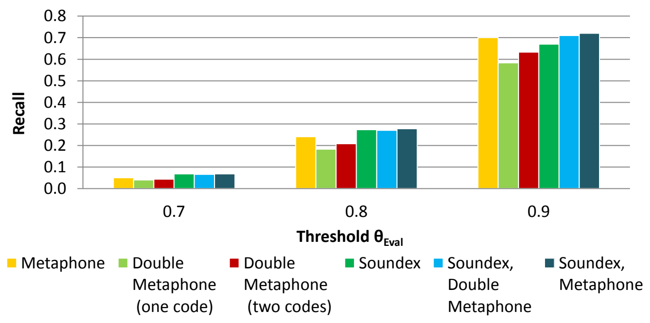

Figure 1: Recall of blocking algorithms for different values of θ_{eval}

We index the data by creating three indexes for each attribute. The *record index* is a regular inverted index. It points from the distinct attribute values to the respective record IDs. The *block index* groups similar attribute values together: For each distinct attribute value of the record index, a blocking key is generated by using some blocking algorithm. All attribute values with same blocking key belong to the same block, thus grouping roughly similar values into one block. The third index, the *similarity index*, is based on the block index. It stores the similarity between each pair of values of the same block. Therefore, the similarity index has the same key set as the record index.

For a given query value for an attribute, we use the similarity index to determine all similar values according to the given attribute similarity predicate. With the record index, we then determine all occurrences of the similar values, which is the result set of the query for this attribute. If the query value has not yet been indexed, we use the block index to first calculate the similar values.

3.2 Index Creation with Blocking

The block and similarity indexes are created using blocking: Only attribute values with the same blocking key are compared [12]. Because the selected blocking algorithm has impact on the overall search performance, we compare different blocking algorithms based on phonetic encodings (Soundex [2], Metaphone [13], Double Metaphone [14], and combinations of those) and different similarity thresholds for the attribute *name* in our dataset in Fig. 1. The figure shows that the best results are achieved with a combination of Soundex and Metaphone with $\theta_{eval} \geq 0.9$. Thus, we choose this blocking method for all later evaluations and considerations.

4. FILTERING WITH DYNAMIC QUERY PLANS

A query plan defines constraints in disjunctive normal form (DNF) that specify a similarity threshold for each attribute within the plan [11]. A record that meets one of these constraints is considered as probably similar to the query record. Consider this example of a query plan:

$$(attr1 \geq 0.7 \wedge attr2 \geq 0.9) \vee (attr1 \geq 0.9)$$

There are two possibilities to fulfill this query plan: Either the values for $attr1$ and $attr2$ of the candidate record compared to the query record achieve a similarity of 0.7 and 0.9 or higher, or the comparison of the candidate record and query record concerning $attr1$ results in a similarity of at least 0.9.

The purpose of a query plan is to efficiently filter the record set to find as many records as possible that are similar to the query record. Using a query plan, the number of overall similarity calculations can be reduced, as shown in Section 5. Overall similarity calculations can be quite expensive, depending on the provided function. Thus, query plans can significantly reduce costs. However, the shape of the query plan that is used for similarity search depends on the overall similarity measure. As we want to support dynamic similarity measures at query time, we cannot use a predefined query plan. Instead, the query plan has to be generated at query time, too. We now present DySim, an algorithm to dynamically generate query plans.

The idea behind DySim's way of generating query plans is inspired by Fagin's algorithm [10] and based on the assumption that the overall similarity function is monotonically increasing. Because of this assumption, it is very likely that records with a high overall similarity also have high similarities regarding some of the attributes. Thus, if we sample the records with the highest similarities for each attribute, there is a high probability that the top hits are among the sample. That is why DySim starts with sampling records that have a high similarity regarding one of the attributes. We compute the overall similarity of these records and again choose some of them, namely those that are potentially good matches for our query. We then generate a query plan that retrieves at least these sample records, and hopefully all other sufficiently similar records.

The generation of query plans in the DySim algorithm comprises three steps: Sampling, query plan preparation, and query plan minimization. Below, these steps are described in detail, each followed by an example.

4.1 Sampling

During the sampling step, N records are retrieved that are similar to the query based on their attribute similarity. For each attribute value of the query record, the N records with the highest individual attribute similarity are collected in a set. To do so, we follow the approach of Christen et al. described in Section 3.1, but stop retrieval when N records have been retrieved.

As an example, consider a query for all locations with the name *Starbucks* in the city *Berlino*. The DySim algorithm first retrieves the IDs of the N records with names similar to *Starbucks*. Likewise, the N IDs of records with a city similar to *Berlino* are retrieved. For this example, possible results are shown in Table 1 assuming $N = 3$. In some cases, there are many entries with the same similarity. In our example, there are more than three entries in our database with the value *Berlin* for the attribute city. It is important to remember that in this step we do not aim to retrieve all records with a high similarity, but a data sample that includes some records with a possibly high similarity. Thus we can break similarity ties through random choice.

After retrieving those sample records, we apply the overall similarity measure and sort the records by descending similarity. In our example, we retrieve the records 1, 3, 4, 5, and 7. Table 2 presents the records including their name and city sorted by their overall similarity.

Query Attr. Value	Record ID	Record Attr. Value	Attr. Similarity
Starbucks	7	Starbucks	1.0
	4	Starbucks Cafe	0.93
	1	Starbucks Olivaer	0.91
Berlino	3	Berlin	0.9
	5	Berlin	0.9
	7	Berlin	0.9

Table 1: *top* 3 records per query attribute value

Record ID	Name	City	Overall Similarity
7	Starbucks	Berlin	0.95
5	Starband	Berlin	0.84
1	Starbucks Olivaer	Berent	0.82
4	Starbucks Cafe	Munich	0.79
3	Good Coffee	Berlin	0.73

Table 2: Retrieved records sorted by their overall similarity

There are two important parameters of our sampling algorithm. Since we want the query plan to cover at least all records r for the query q with $sim_{overall}(r,q) \geq \theta_{overall}$, all sampled records that fulfill that condition should be used for creating the query plan. At this point, it might be good not to use a similarity bound as strict as $\theta_{overall}$ Thus, we introduce ϕ, a factor adjusting the threshold. Hence, $\phi \cdot \theta_{overall}$ denotes the lower similarity bound for the top records. Furthermore, we want to ascertain that the query plan is always influenced by multiple sample records. for this reason, we introduce σ. σ denotes the fraction of the so-far sampled records that are definitely used for constructing the query plan. In our example, $\sigma = 1$ would mean, that all five records were included in the query plan construction, and $\sigma = 0.4$ would result in a query plan that is based at least on records 7 and 5 (or more, if more records have a similarity $sim_{overall}(r,q) \geq \phi \cdot \theta_{overall}$) .

Thus, the final sample comprises the top records of the original sample (as defined by the fraction σ) plus all sample records with an overall similarity greater than or equal to $\phi \cdot \theta_{overall}$. Regarding our example, there are five top records. Assuming that $\sigma = 0.2$, $\phi = 0.9$, and $\theta_{overall} = 0.89$, record 7 is used to create the query plan, because it is within the top 20 % of the sample. Additionally, all records with a similarity equal to or higher than $0.9 \cdot 0.89 = 0.801$ are considered. Thus, the sample consists of the records 7, 5, and 1.

4.2 Query Plan Preparation

The query plan preparation step creates a query plan that covers at least all the records in the previously determined sample. To do so, the query plan constraints are built depending on the sampled records one by one. For each attribute of a certain record, we build a so-called *threshold record*. It consists of the attribute name (not value) and a similarity threshold, defined by the corresponding attribute similarity and calculated for the attribute value at hand. As a result, we obtain a threshold record for all attributes that are set in the sample record and the query record. All resulting threshold records for a sample record are combined into a conjunction.

If a record has set all attributes of this conjunction and the similarities are at least as high as the respective thresholds, the record is covered by this part of the query plan. As the corresponding record's attribute similarities fulfill the constraints of the conjunction, it also is covered by it. The resulting query plan in DNF is created as a disjunction of all the conjunctions built on the sample records. The conjunctions for our example are:

Conjunction for record 7
$name \geq 1.0 \wedge city \geq 0.9$

Conjunction for record 5
$name \geq 0.82 \wedge city \geq 0.9$

Conjunction for record 1
$name \geq 0.91 \wedge city \geq 0.73$

The resulting (still partially redundant) query plan is:

Query plan in DNF
$(name \geq 1.0 \wedge city \geq 0.9)$
$\vee (name \geq 0.82 \wedge city \geq 0.9)$
$\vee (name \geq 0.91 \wedge city \geq 0.73)$

This query plan covers all records of the sample. As an example, let us take a look at record 1. The names *Starbucks Olivaer* and *Starbucks* have a similarity of 0.91, while the cities *Berent* and *Berlino* have a similarity of 0.73. As the last conjunction is based on these values, this record is covered by the query plan. However, this query plan is not yet optimal and is improved in the next step.

4.3 Optimization

There are several points that can be optimized in the prepared query plan. First of all, threshold records can become too strict. For example, a threshold record with a threshold of 1.0 allows only attributes with the exact same value. This might be a problem, as we could lose result objects that have very similar but not exactly the same attribute values. To prevent this kind of strictness, we define the threshold θ_{plan}. If a similarity threshold within a threshold record is higher than θ_{plan}, it is lowered to θ_{plan}. With $\theta_{plan} = 0.9$ our example query plan is changed to:

Query plan with θ_{plan}
$(name \geq 0.9 \wedge city \geq 0.9)$
$\vee (name \geq 0.82 \wedge city \geq 0.9)$
$\vee (name \geq 0.9 \wedge city \geq 0.73)$

If, on the other hand, the fraction of records that surpass the lower similarity bound for a query is less than σ, the sample may include records with some low attribute similarities. A low attribute similarity probably does not increase the overall similarity, thus attributes with low similarities are probably not indicators for good matches. Additionally, the similarity index stores only attribute values with similarities higher than or equal to θ_{index}, so that it is sufficient to consider attributes with similarities that are at least as high as θ_{index}. Otherwise, it would be necessary to calculate all attribute value similarities within the corresponding block. In our example the last threshold record of the last conjunction has a threshold of 0.73. Assuming that θ_{index} is 0.8, this threshold can be dismissed:

Query plan with θ_{index}

$(name \geq 0.9 \wedge city \geq 0.9)$
$\vee (name \geq 0.82 \wedge city \geq 0.9)$
$\vee (name \geq 0.9)$

As mentioned above, some of the conjunctions are redundant in the sense that their result set is a subset of the result set of another conjunction. This is the case for two conjunctions A and B if B contains lower thresholds than A for the same attributes, or if B contains fewer attributes (with all other attribute thresholds being the same). We then say that A is dominated by B and we can dismiss conjunction A. In the example, the first conjunction is dominated by the second conjunction. Therefore, we can eliminate the first conjunction and the resulting final query plan is:

Minimized query plan

$(name \geq 0.82 \wedge city \geq 0.9) \vee (name \geq 0.9)$

4.4 Filtering

The resulting minimized query plan can now be used for filtering. The threshold records of each conjunction are applied to the similarity index: the similar attribute values for every attribute of the conjunction are fetched if their similarity is above the respective threshold. The corresponding record IDs are fetched if all threshold record constraints are met. We optimized the process of fetching similar attributes by determining the minimum thresholds for all attributes within the entire plan and retrieving the corresponding attributes. For the previously explained example considering the name attribute, there are the threshold records $name \geq 0.82$ and $name \geq 0.9$. As the first one has a smaller threshold, all name attributes with a similarity ≥ 0.82 are retrieved from the similarity index. Now the overall similarity function is applied to all retrieved records and the final results are sorted by their similarity.

5. EVALUATION

Similarity search should achieve good results for both precision and recall while using as little resources as possible. Since the similarity function $sim_{overall}$ and the threshold $\theta_{overall}$ are part of the query, we know exactly which objects are relevant, thus *precision* is 1.0 by definition. The basis of our algorithm is the index proposed by Christen et al. We cannot achieve a *recall* higher than the recall we would obtain by comparing the query object with *all* indexed objects. Besides recall, our second metric is the *number of comparisons*, which counts the number of performed calculations of the function $sim_{overall}$. The number of comparisons highly correlates with the runtime and is a good indicator for performance.

We evaluated DySim against an extended version of the search algorithm by Christen with the possibility to incorporate an arbitrary overall similarity measure $sim_{overall}$. To achieve this, we replaced the accumulation of attribute similarities by retrieving all records that contain a similar attribute value and applying the overall similarity measure on all records in this set. The amount of retrieved records is not affected, but the quality of the result list (the overall similarity measure is used as a ranking function in this context). In the following, this approach is referred to as baseline. We compared the two algorithms to the naïve similarity search without index structure. This search iterates over all objects in the database. All results from the non-index-based algorithm that have a $sim_{overall}$ greater than or equal to $\theta_{overall}$ are used as our "ground truth" for recall computations.

We show that DySim needs only about 66% of the comparisons required by the baseline approach of Christen et al., while keeping recall nearly the same.

5.1 Prerequisites

To evaluate DySim we used a random sample of 100 spots from our data set and selected a random subset of attribute values from each of those spots as query values. Thus, it is guaranteed that all attribute values from the queries are stored in our index. We could also consider generated query objects containing attribute values that are not stored in the index but regarding our two evaluation metrics the number of comparisons would not change, and the recall would be dependent on the used attribute similarity measures. Since we abstract from the used attribute similarity measures, we decided not to use such query objects. Of course, the query times would increase, as instead of fast index lookups the attribute similarities would have to be calculated.

5.2 Parameters

The similarity measure $sim_{overall}$ used for the evaluation is the average of the individual attribute similarities. The parameters introduced in Sec. 4 were assigned static values for the evaluation.

We set the overall similarity threshold $\theta_{overall} = 0.89$. This value is empirically determined, nevertheless, it is a free parameter of the query. The threshold for the similarity index θ_{index} is set to 0.8 as this was an acceptable trade-off between time consumption for similarity calculations and memory needs. The maximal similarity threshold θ_{plan} within a threshold record of a query plan is set to 0.9 and could be subject of further analysis as it should lead to a higher recall if lowered. On the other hand, the number of comparisons might also increase. Finally, we set the percentage of records that should definitely be used in the query plan σ to 20%. The parameters ϕ and N are evaluated in detail in the following section.

5.3 Results

If the similarity threshold ϕ for the sample records is lowered, both recall and the number of comparisons increase as shown in Figure 2. This is the case because a low value of ϕ leads to more and potentially less similar samples being used to generate the query plan. If the used samples are less similar to the query object, the generated query plan is less strict. Thus, the similarity search performs more comparisons, but in turn finds more results. Here, we use 50 sample records to generate the query plans.

Figure 3 presents the recall and the number of comparisons for different values of N. The figure shows that an increasing N leads to an increase in recall as well as in the number of comparisons. A higher value of N means that more initial sample records are fetched, which leads to more comparisons being performed before the actual query plan is generated. Also, the number of samples that are used to generate the query plan increases, which may result in a query plan that is less strict and thus adds to the number of additional comparisons. On the other hand, a larger N increases the chances of sampling some good matches for the query object, which may also lead to an increased recall. We

Algorithm	Recall	Number of comparisons
No index	1	1.5 m
Baseline	1	20.9 k
DySim	0.979	13.8 k

Table 3: Performance comparison of DySim and the baseline algorithm

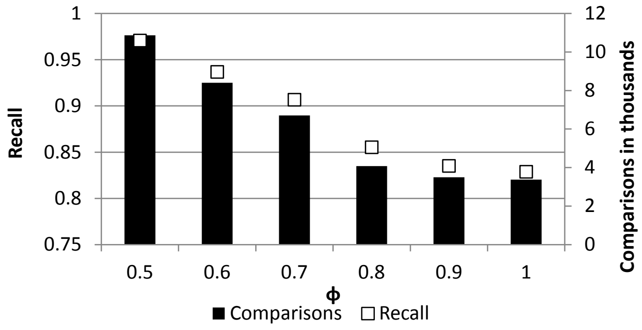

Figure 2: Recall and number of comparisons for different values of ϕ

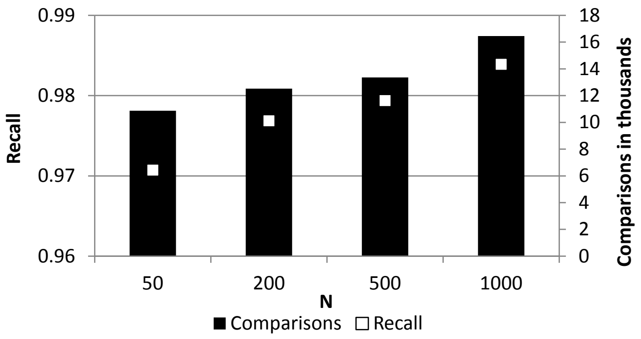

Figure 3: Recall and number of comparisons for different values of N

also observed that for query records with many set attribute values a low value of N is sufficient to achieve a high recall.

Finally, we compare the baseline algorithm against the DySim algorithm with $\phi = 0.5$ and $N = 500$. Table 3 shows the results in terms of recall and number of comparisons. With the sample used for evaluating the parameters, the baseline algorithm achieves a recall of 1.0 while performing about 20.9 thousand comparisons for all queries in the sample. In contrast, the DySim algorithm delivers a recall of 0.979, but makes only 13.8 thousand comparisons. This shows that compared to the baseline DySim has a minor decline in recall but has to execute only two-thirds of the comparisons.

6. CONCLUSION

In this paper, we provided a method for similarity search with arbitrary similarity measures. The only constraint for such a measure is that it has to be monotonically increasing and based on the individual attribute similarities, e.g., as is the case for weighted similarities. The presented DySim algorithm is based on the indexing methods of Christen et al. and a modified version of the according similarity search algorithm that allows arbitrary overall similarity measures. We added a mechanism for filtering to allow a faster search with fewer computations of the overall similarity measure. The demonstrated filtering mechanisms, the dynamic query plans, make use of an adaptive selection of sample records during filtering, resulting in adjusted query plans for arbitrary overall similarity measures. We showed that we can achieve a significant reduction of overall similarity computations at the cost of only a very minor decline in recall.

7. REFERENCES

[1] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33:322–373, 2001.

[2] C. P. Bourne and D. F. Ford. A study of methods for systematically abbreviating english words and names. *Journal of the ACM*, 8:538–552, 1961.

[3] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Trans. on Database Systems*, 27(2):153–187, 2002.

[4] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. Example-driven design of efficient record matching queries. In *Proc. of the Intl. Conf. on Very Large Databases (VLDB)*, pages 327–338, 2007.

[5] S. Chaudhuri, L. Gravano, and A. Marian. Optimizing top-k selection queries over multimedia repositories. *IEEE Trans. on Knowl. and Data Eng.*, 16(8):992–1009, Aug. 2004.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33:273–321, 2001.

[7] K. Chen, Y. Zhang, Z. Zheng, H. Zha, and G. Sun. Adapting ranking functions to user preference. In *Proc. of the Intl. Workshop on Ranking in Databases (DBRank)*, pages 580–587, 2008.

[8] P. Christen, R. Gayler, and D. Hawking. Similarity-aware indexing for real-time entity resolution. In *Proc. of the Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 1565–1568, 2009.

[9] P. M. Deshpande, D. P, and K. Kummamuru. Efficient online top-k retrieval with arbitrary similarity measures. In *Proc. of the Intl. Conf. on Extending Database Technology (EDBT)*, pages 356–367, 2008.

[10] R. Fagin. Fuzzy queries in multimedia database systems. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 1–10, 1998.

[11] 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)*, pages 1679–1688, 2011.

[12] H. B. Newcombe. Record linkage: the design of efficient systems for linking records into individual and family histories. *American Journal of Human Genetics*, 19(3):335–359, 1967.

[13] L. Philips. Hanging on the Metaphone. *Computer Language*, 7(12):39–44, 1990.

[14] L. Philips. The double metaphone search algorithm. *C/C++ Users J.*, 18:38–43, 2000.

[15] A. H. van Bunningen, M. M. Fokkinga, P. M. G. Apers, and L. Feng. Ranking query results using context-aware preferences. In *Proc. of the Intl. Workshop on Ranking in Databases (DBRank)*, pages 269–276, 2007.

[16] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach.* Advances in Database Systems. Springer, 2006.