Multi-Attribute Similarity Search for Interactive Data Exploration

Kostas Patroumpas
Athena Research Center, Greece
kpatro@athenarc.gr

Alexandros Zeakis
Athena Research Center, Greece
azeakis@athenarc.gr

Dimitrios Skoutas
Athena Research Center, Greece
dskoutas@athenarc.gr

Roberto Santoro
SpazioDati S.r.L., Italy
santoro@spaziodati.eu

ABSTRACT

We have developed SimSearch, a tool that simplifies data exploration by enabling top-k similarity search over large collections of entities involving multiple heterogeneous attributes from different sources. We present the supported modes for data access, and the query mechanism orchestrating multi-attribute similarity search over diverse types of attributes, including textual, numerical and spatial. Users can specify their query parameters and preferences through a web interface, and visually inspect and compare the results through appropriate visualizations for the different types of attributes involved. We demonstrate SimSearch using a real-world, commercial dataset, highlighting its capabilities for interactive, user-friendly, and intuitive data exploration.

1 INTRODUCTION

Data scientists spend a large portion of their time on data exploration. *Similarity search* is a fundamental operation for exploring massive data collections, e.g., documents in web search engines, products in e-commerce marketplaces, photos in image hosting services, etc. Given a user query (e.g., a set of keywords or an image), the goal is to retrieve the entities most similar to it.

The problem becomes more challenging when the entities being explored involve multiple attributes of different types. For example, consider a collection of company profiles. Besides an identifier and name, these may include various attributes, such as a textual description, keywords characterizing the type of business, annual financial figures (revenue, turnover, profit, etc.), location(s) of the headquarters and possible branches, number of employees, important dates (founding, change of status, change of owner, etc.), etc. Hence, attributes can be of different types, including numerical, textual, and spatial. Furthermore, different attributes may reside in different sources. For instance, textual data (keywords, descriptions, etc.) may be stored in Elasticsearch, to take advantage of its full-text search capabilities; personnel data may be stored in semi-structured format in a JSON file to better reflect its hierarchical nature; financial data may be stored in a relational database to enable fast extraction of statistics.

Similarity search over such multi-attribute entities needs to combine similarity measures and scores over different types of attributes. In the above example, the user may wish to find medium-sized companies employing around 10 persons (*employees*), which deal with Software, Technology, or Telecommunications (*keywords*), are located in or near Milan (*location*), and have an annual *revenue* of around 300,000 euros. To evaluate such queries, the system needs to access the corresponding data source for each involved attribute, and query the underlying data using an appropriate index (e.g., R-tree for spatial, B-tree for numerical, and inverted index for textual data). It will then employ a suitable similarity measure (e.g., Jaccard coefficient for textual or Euclidean distance for spatial and numerical attributes) to rank the results according to each attribute. The individual rankings can then be weighted and aggregated to produce the global top-k results. After visually inspecting the results, the user may wish to modify the attributes and/or the weights in the query, and repeat the search to find more relevant information.

Currently, a data scientist needs to write a series of custom scripts to perform all these steps, ranging from data access and querying over multiple sources to similarity computations, aggregations and visualizations of results. This is time consuming, error-prone, and requires a sufficient level of data management and programming skills. Notice that multi-attribute similarity search is different from faceted search (e.g., [1]); instead of applying a conjunction of Boolean filters per attribute, we employ similarity measures and scores to rank the results. Multi-attribute similarity search is particularly useful for exploring and navigating the contents of a data lake; however, recent works like [2, 7] focus mostly on finding similar datasets or related tables.

To simplify this task, we have developed SimSearch, an open-source software for top-k similarity search over multi-attribute entity profiles possibly residing in different, heterogeneous data sources. SimSearch treats top-k multi-faceted similarity search as a *rank aggregation* problem [5]. More specifically, we evaluate top-k similarity search queries in two steps, as presented in [6]. First, an individual k-nearest neighbor (kNN) query is executed per attribute, fetching a ranked list of entities with respect to it. Then, rank aggregation is performed to obtain the global top-k results taking into consideration all attributes. Each kNN query is executed in parallel, reducing the overall response time. Moreover, it applies its own similarity measure and takes advantage of corresponding index and query execution algorithm.

SimSearch includes a graphical interface, allowing users to select desired attributes, specify preferred values and weights, and then visually inspect the results through appropriate visualizations, such as maps, histograms and word clouds. Results for several weight combinations can be retrieved with a single query and visualized side-by-side, providing valuable insight on the underlying data characteristics and enabling users to calibrate their preferences and progressively navigate and explore the data.

The remainder of the paper is structured as follows. Section 2 outlines the processing flow of SimSearch. Section 3 discusses the back-end mechanism. Section 4 outlines how multi-faceted similarity search is executed. Section 5 presents the user interface. Finally, Section 6 outlines a demonstration scenario.

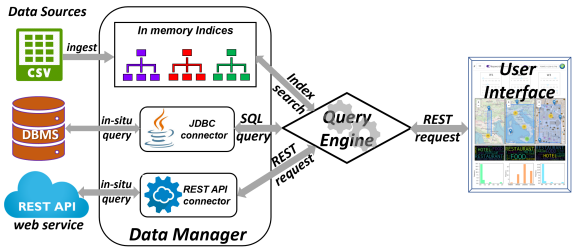

Figure 1: SimSearch architecture.

2 SYSTEM OVERVIEW

Given a query entity described by multiple attributes, SimSearch retrieves a ranked list of the top-k most similar entities. SimSearch[1] consists of three main components, as shown in Figure 1:

Data Manager. This component keeps track of the available data sources and the queryable attributes in each one, specified in a configuration file. Attribute values may be *ingested* or *queried in-situ*, if supported by the underlying data store. For ingested data, suitable indices are constructed in-memory.

Query Engine. Based on the attributes involved in the query and the user's preferences (values, weights, number of results), the Query Engine handles execution in two stages. First, a ranked list of candidate entities is retrieved separately for each attribute based on individual similarity queries at the respective data sources or internal indices. Then, these individual ranked lists are combined to compute the final top-k results and their aggregate scores.

User Interface. A web-based interface allows users to specify their query parameters, including the attributes to be used and the preferred value and weight per attribute. User requests are submitted to the server via a REST API. Query results can be viewed as lists, as well as through different types of visualizations per attribute (maps, word clouds, histograms) to offer better insight on their relevance. The REST API can also be used by administrators to define and configure new data sources or remove existing ones. The API also returns a catalogue of the configured sources and the types of their respective attributes.

Next, we describe each component in more detail.

3 DATA MANAGER

The *Data Manager* keeps track of the available data sources and their queryable attributes. It supports different types of attributes, including *numerical* (e.g., integers, floats, dates), *spatial* (e.g., coordinates, geometries) and *textual* (e.g., keywords, tags). Attribute values may come from heterogeneous and remote sources. Two access modes are supported:

- *Ingest mode.* Attribute values are ingested by the Data Manager, which builds a corresponding index (e.g., B+ tree for numerical, R-tree for spatial, inverted index for textual attributes). Such indices reside in memory to allow executing kNN queries with low latency. Thus, this mode is preferable for interactive exploration and visualization.
- *In-situ mode.* In this case, the Data Manager uses an external query endpoint, rather than an internal index, to retrieve results for the specific attribute. If the data resides already in a DBMS or a RESTful web service that can process kNN queries over the desired attribute, then the Data Manager can retrieve a ranked list of results by submitting a query to that endpoint instead of requiring to ingest and index the data internally. This mode avoids data replication and offers the ability to scale according to the scalability of the accessed DBMS or RESTful service.

The Data Manager is implemented in Java and employs suitable JDBC and REST API connectors to external data sources. It currently supports ingesting data from CSV files, but it is straightforward to extend it with importers for other data sources (e.g., JSON). For in-situ queries, it can connect to any DBMS (e.g., PostgreSQL) via a JDBC connection, thus taking advantage of their provided indices and query execution capabilities. Connectivity through REST APIs can be used to retrieve data from remote search engines, such as Elasticsearch.

Thanks to its own REST API, a SimSearch instance can also employ another SimSearch instance as an external query endpoint. This important feature provides an additional means for SimSearch to scale horizontally. Recall that in the in-situ mode, SimSearch relies on external engines (DBMS, REST API) to satisfy scalability requirements, whereas in the ingest mode its scalability is limited, since the indices for the ingested properties need to be kept in memory. However, using this feature, it is possible to use different SimSearch instances in several machines to ingest and index different attributes, and then query them from a "master" instance to answer requests combining all attributes.

4 QUERY ENGINE

The *Query Engine* is also developed in Java and is loosely coupled with the Data Manager. It receives a top-k similarity search request through the SimSearch REST API, and identifies the attributes involved. It then consults the Data Manager to find their corresponding data sources and prepares a suitable kNN query specification per attribute. For instance, if an attribute concerns ingested data (e.g., keywords), a search will be executed against the respective in-memory index (e.g., an inverted index). If the attribute data resides in a DBMS, an SQL query will be constructed and executed over a JDBC connection to the database. If attribute data is accessible through a RESTful service (e.g., Elasticsearch or another instance of SimSearch), an HTTP request will be sent to fetch candidate entities most similar to the given query value.

In SimSearch, we apply a uniform approach that allows ranking of entities by their similarity to a given query independently and in parallel for each attribute. Thus, a separate ranked list of candidate entities is fetched by each individual query. Entities in each list are sorted by their respective scores as computed by a respective scoring function. As detailed in [6], similarity measures may differ per attribute, hence an *exponential decay* function is employed for normalizing the scores per attribute. Nonetheless, each list should provide a sufficient number M of candidates (typically, $M \quad k$) to ensure that the top-k results with the highest aggregate scores are finally returned to the user.

Combining the various ranked lists produces the final results with aggregate rankings according to user-specified weights per attribute. This *rank aggregation* process is agnostic of whether the underlying data sources are ingested or queried in-situ by the Data Manager, as long as identifiers of entities in each ranked list can be matched. However, top-k rank aggregation against multiple lists may employ different access paradigms [5]. The

[1]Source code publicly available at https://github.com/smartdatalake/simsearch

main difference is whether *random access* to attribute values is allowed or not. SimSearch can be configured with three alternative ranking algorithms for the aggregation stage. Each of them iterates in parallel over the ranked lists to find matching entities and compute aggregate scores. We briefly outline them below:

- *Threshold Algorithm* (TA) [3]. This method employs unrestricted random access to the full contents of the underlying attribute values. It maintains a threshold T that represents an upper bound for the aggregate score of unseen entities. Once the aggregate score of an entity exceeds the current T value, this entity is issued as the next result. However, random access may be costly (e.g., numerous SQL queries to a remote DBMS) or even prohibited (a RESTful service may be limiting for API calls to mitigate denial-of-service attacks).
- *No Random Access* (NRA) [3]. In contrast to TA, this algorithm iterates strictly over the sorted items in each ranked list. No exact scores can be issued for missing values, so NRA maintains a lower and an upper bound for the aggregate score of each seen entity. This may incur significant overhead, especially for ranked lists of considerable size (i.e., larger M values).
- *Partial Random Access* (PRA) [6]. This hybrid variant of TA employs random access not to the full attribute data, but only to items already retrieved in the much smaller ranked lists. However, exact aggregate scores may not always be available (unless an entity appears in all ranked lists), so PRA employs bounds on the aggregate scores per entity as in NRA.

We are currently extending the Query Engine with a pivot-based multi-metric indexing mechanism [4], which can significantly speed up query execution. However, this is feasible only when all involved attributes are ingested and indexed internally.

5 USER INTERFACE

SimSearch provides a Web interface, which is developed with Python's Dash open-source framework[2]. The user interface is illustrated in Figure 2 and allows specification of query preferences and values, interactive modification of weights per attribute, and visualization of the returned results.

More specifically, clicking on button *Settings*, the user can connect to a SimSearch service deployed over some data sources, inspect the queryable attributes, and optionally specify parameters controlling the rank aggregation process (ranking method, decay factor, etc.) Once connected, the user can specify top-k similarity search queries involving any combination of the available attributes. The user can specify her preferred value on any attribute to include it in the search (e.g., a comma-separated list of keywords, a numerical value, a date, etc.). Locations can be specified either directly with the Well-Known Text representation of the query point or by clicking on the globe button to display an interactive map and pick the desired coordinates. In case of data regarding named entities (e.g., company names), the user can instead search for an entity and readily fill in the search preferences with its respective attribute values.

Weights per queried attribute to be used in rank aggregation can be chosen using the sliders. Multiple combinations of weights can be configured (e.g., four combinations shown in Figure 2) with values specified by the user. For each such combination, a separate list of top-k results will be returned. Weights calibrate the importance of each attribute in the final rankings, hence results may differ among those lists not only in terms of their constituent entities, but also in their rankings and scores. For instance, if the user wishes to mostly find similar entities nearby the query location, the weight on this attribute should be set very close to 1, with reduced weights on the rest. In another combination, keywords may be considered more important, increasing the weight on this textual attribute while diminishing the rest.

When the *Search* button is pressed, a top-k similarity search request is sent to the server with the given parameters to calculate the results. We stress that this is not a Boolean match query typically supported in a DBMS; users actually wish to find entities that are mostly similar to their specified preferences, even if result values deviate in some attribute(s). Once the response is received, the listings of most similar results (one listing per weight combination) are displayed. Most importantly, comparative *visualizations* of these lists are offered, each involving a particular attribute in its results (Figure 2). As SimSearch currently supports queries on spatial, textual and numerical attributes, the front-end provides different views of results, respectively for each attribute type, as described below.

If a spatial attribute is involved in the query, *map views* of the locations of the results are plotted side-by-side. Such alternative views include heatmaps, clustered markers, and cluster polygons, so that the user can inspect how each specified weight has influenced the geographical extent (captured with the Minimum Bounding Rectangle) and spatial distribution of the results. Even results with locations far away from the query point can be identified in some cases. These results may qualify for a particular list, if their aggregate score is high enough, due to having strong similarity to the query in other attributes.

For results with a textual attribute involving keywords, a *word cloud* or *histogram* can be depicted per list. Keywords are shown in varying sizes in word clouds depending on their frequency in each respective list. Again, this provides insight on how weights affect the returned results, allowing the user to identify at a glance whether the most frequent keywords match her preferences in the query. Alternatively, histograms can provide more details regarding the top-10 keyword frequency in the results.

For numerical attributes, *histograms* show the distribution of their values in results per weight combination. Also, *box plots* enable a user to instantly observe if results are concentrated around her query value or deviate significantly from it.

Overall, if multiple combinations of weights are employed, the user can identify which results are closer to her intentions. She can then modify some weights (or add another combination) using the respective sliders and issue a new request. Any changes in the results are directly reflected on the visualizations per weight combination. Furthermore, SimSearch allows users to examine *correlations* in results. First, a similarity matrix captures the pairwise similarity between entities in the same list obtained for a given weight combination (*intra-list correlation*). In addition, *inter-list correlations* between the rankings of two lists can be calculated. For all pairs of result listings, the application provides three types of rank correlation coefficients: Spearman's ρ, Pearson's r, and Kendall's τ. These coefficients can quantify the statistical dependence of rankings between the lists, i.e., indicate the similarity of their respective results.

6 DEMONSTRATION SCENARIO

We will demonstrate the current functionality of SimSearch against the ATOKA knowledge base[3] owned by SpazioDati. This real-world data concerns companies in Italy and includes many

[2] https://dash.plotly.com/

[3] https://atoka.io/