An Approach for Incremental Entity Resolution at the Example of Social Media Data

B. Opitz[1], T. Sztyler[1], M. Jess[1], F. Knip[1], C. Bikar[1], B. Pfister[1], A. Scherp[1,2]

[1] University of Mannheim, Germany
[2] Kiel University and Leibniz Information Center for Economics, Kiel, Germany

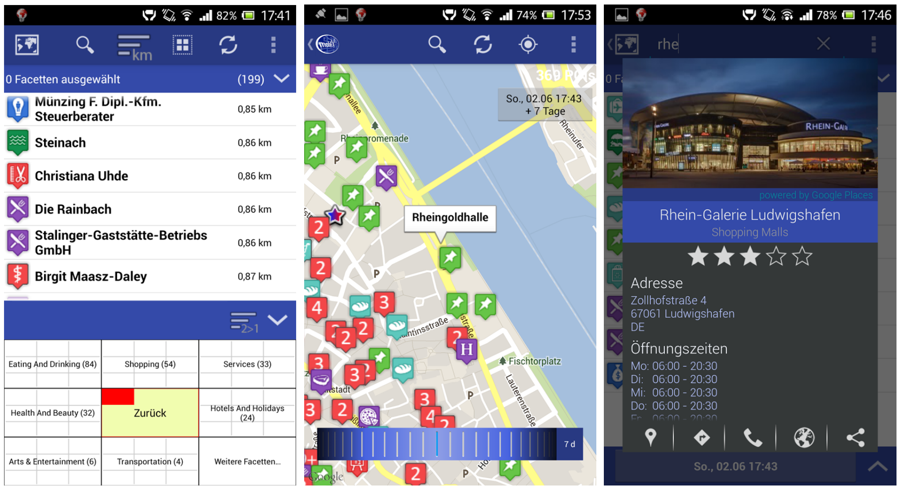
Abstract When querying data providers on the web, one has no guarantee that they will reply within a given time. Some providers may even not answer at all. This makes it infeasible to wait for a complete result before beginning with the entity resolution. In order to solve this problem, we present a query-time entity resolution approach that takes the asynchronous nature of the replies from data providers into account by starting the entity resolution as soon as first results are returned. Resolved entities are propagated from the entity resolution engine to the mobile client as early as possible. Resolution results that are produced later are send as updates to the client and thus improve earlier results.

1 Introduction

Where can I find the next bakery? Is it open on Saturdays? Is there any concert in the area or around my current location? What to do for the night without wasting much time browsing multiple websites? For all these questions, our mobile social media explorer mobEx provides the answer. It aggregates data from different Web sources to provide information to plan one's holiday, leisure time, or any other amusements. However, when querying multiple data providers for information about the same subject or location, it is quite common to find redundancy in the overall result. For example, multiple providers may have complementary information about the same entity or even (exact) duplicates [1,2,3]. This is further complicated by variations between the retrieved data such as different spellings (possibly mistakes) or missing information [3]. Assuming that we retrieve information from an arbitrary number of providers in the form of records, i. e., a composition of information about an entity, an end user will most likely prefer a consolidated resource representing an entity instead of multiple resources describing the same entity. The process of eliminating duplicates and merging them into one resource is called entity resolution (ER). In this paper, we present a novel query-time entity resolution approach, i.e., we carry out ER at query-time. The mobile application mobEx serves as showcase for our approach. We use techniques such as fuzzy matching and threading as well as precondition heuristics that reduce the number of comparisons that have to be carried out. Key difference to existing work is that we do not have all records readily at hand when the resolution process is started. Thus, the resolution process receives more resources as we go along and results gradually become more complete.

2 Showcase: Social Media Exploration with mobEx

The mobile application mobEx sends a user request to the mobEx server. The server queries the various data providers such as DBpedia (http://dbpedia.org), Eventful (http://eventful.com), Qype (http://qype.com), OpenPOI (http://openpois.net), and GeoNames (http://geonames.org) for events, organizations, persons, and places. The results the mobEx client receives can be navigated through a facet structure as shown in Figure 1 (left). The faceted navigation has been extended from Schneider et al. [4]. In the middle of Figure 1 one can see a screenshot of the app with the map view showing the results of a query in terms of locations and events. Finally, the details of an object such as a website and open hours can be viewed as as shown in Figure 1 (right). The entity resolution of the different resources retrieved from the data provider entirely takes place on the mobEx server. A resource represents an object that is either an event, organization, person or place. The relevant properties of a resource for the entity resolution process are outlined in Table 1. When querying data providers, all retrieved records are mapped into such a resource. The alignment of the data providers' schemata to the internal schema of mobEx is hard coded. It may be extended by some automatic approaches in the future like [5].



Münzing F. Dipl.-Kfm. Steuerberater	0,85 km
Steinach	0,86 km
Christiana Uhde	0,86 km
Die Rainbach	0,86 km
Stalinger-Gaststätte-Betriebs GmbH	0,86 km
Birgit Maasz-Daley	0,87 km

Eating And Drinking (84)	Shopping (54)	Services (33)
Health And Beauty (32)	Zurück	Hotels And Holidays (24)
Arts & Entertainment (6)	Transportation (4)	Weitere Facetten...

Rhein-Galerie Ludwigshafen
Shopping Malls

Adresse
Zollhofstraße 4
67061 Ludwigshafen
DE

Öffnungszeiten
Mo: 06:00 - 20:30
Di: 06:00 - 20:30
Mi: 06:00 - 20:30
Do: 06:00 - 20:30

Figure 1. Facet view (left), map view (middle), and details view (right) of mobEx.

3 An Approach for Incremental Entity Resolution

Our incremental entity resolution approach takes place in a highly parallel fashion as can be seen in Figure 2. The numbers in the following description of our

Data type	**Attribute**	**W.**	**Represented information**	**Example**
String	uuid	*	id used on the server	
String	type	*	type of resource	event, organization, person, place
complex	source	*	set of data provider names	{lastfm, eventful}
Double	longitude	*	latitude of the entity's location	49.48429
Double	latitude	*	longitude of the entity's location	8.46301
complex	postalAddress	2	possibly multiple addresses (birth place, death place) split into country, city, postal code, street and street number	Bismarckstr. 1, 68161 Mannheim (Germany)
String	label	3	name of the entity	University of Mannheim
String	description	*	short description of the entity	
Schedule	schedule	2	start date, end date, birthdays, opening hours	Mon, Tue, Fri: 9:00-18:00; 11.03.1952, ...
URL	url	4	website with further information	www.example.org
URL	imageUrl	1	url of a thumbnail/picture	www.example.org/image.jpg
String	phone	3	a phone number	phone number of a ticket hotline

Table 1. Properties of a resource for the entity resolution process. Complex types have an internal structure which is not relevant for the resolution process. The column W. shows the weights used in the resolution process. A * indicates that the property is not part of the scoring (but still relevant for the resolution).

process correspond to the numbers in the figure. In the given scenario where a client queries our server (1), it is important that the records are processed as fast as possible. Thus, all processes work in parallel. There are two main stages which are the querying of the records (upper half of Figure 2) as well as the entity resolution (lower half of Figure 2). Both stages are divided in multiple sub-threads, controlled by three central units, namely Main-Thread, Entity-Manager-Thread and Entity-Resolver-Thread. These components enable the parallel execution of all available tasks whereas the Main-Thread handles the querying of the records from different data-provider like Geonames or OpenPOI and the Entity-Resolver-Thread the entity resolution. The Entity-Manager-Thread decouples these two stages so that they do not have to wait for each other. Thus, when the Main-Thread receives a request (1) it starts and controls the data-provider querying threads (2) which retrieve the desired information such as restaurants, hospitals and parks and parses the retrieved records into the local/target schema, namely our resource model. When a data-provider thread delivers results, they are passed to the Entity-Manager-Thread (3). The Entity-Manager-Thread administers a container for the queried and processed resources. It listens to all data-provider threads and forwards their results to the Entity-Resolver-Thread (4). The Entity-Resolver-Thread tries to find and handle duplicates such as 'Cafe Vienna' and 'Vienna Cafe', i. e., it actually carries out the entity resolution. All arriving resources are compared to the already received resources by worker threads (5). The result is returned to the Entity-Manager-Thread (6) which in turn returns it to the Main-Thread (7). The processed resources are delivered in reply to the request (8) as soon as they are available, i.e. we do not wait until all records have arrived from the data-providers and undergone resolution. Therefore, earlier results may be updated in a later step of the resolution.

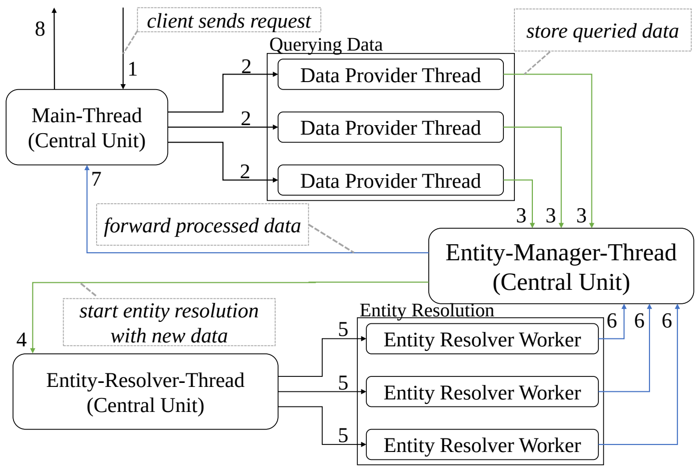


Figure 2. Data flow of processing a client request and incrementally matching the data. The numbers denote the order of the processes.

3.1 Facet Tree for Data Representation

Once a resource object is created it is assigned to a facet. A facet is a category and part of a large tree. This structure is called the facet tree and represents categories hierarchically [4]. The tree is static at run time and part of the resource object management. The core of the structure is based on data from DBpedia, i.e. it originates from the structure of Wikipedia. The facet tree consists of four subtrees that have the root nodes event, organization, person, and place. They originate from different integrated (social media) sources as mentioned in Section 2. Each resource object is assigned to one of these root nodes and to at least one child. Thus, the 'Cafe Vienna' is a 'place' and a 'Coffee Shop' (which is a sub-facet of place). The assignment of a facet to a resource is determined by the category information delivered by the data provider, e.g. 'Cafe'. Thus, the mapping process uses predefined rules to determine a correct match in the facet tree but if this fails, the process tries to find a match based on string similarity. However, only the root node (event, organization, person, or place) assigned to a resource is considered by the entity resolution. The hierarchical classification of each resource object to categories is not considered because the run time of the entity resolution would rise dramatically and speed has a high priority.

3.2 Entity Resolution Process

As already pointed out, entity resolution takes place on resource objects. Each attribute of such a resource contains a certain piece of information about an entity (see examples in Table 1) and is thus more or less representative of the actual entity. Consequently, “some attributes are more important in determining whether a mapping should exist between two objects” [6]. We account for that by assigning weights to the attributes where a higher weight represents that the feature is more distinctive or authoritative (see Table 1). The weights were taken from [3,6,7]. While none of them gives explicit numbers, there are hints which properties are important or more distinctive. The actual numbers were assigned using empirical results. For example, it is legitimate to assign the URL the greatest weight, as they are unique. A similar argument applies for the label and phone number which are both designed to serve as identifier for, e.g., a person, organization, or place.

The entity resolution process is multi-threaded. We start a new thread each time we receive a batch of records from a provider. These records are then mapped into resources which in turn are resolved against each other as well as all previously received resources that were part of the same client query. However, this processing brings up several new challenges:

1. Duplicate comparisons have to be avoided.
2. Given two resources r_1 and r_2. As r_1 is the older object it becomes the merge target. As r_2 has already been merged earlier, the further merge process is affected as follows: r_2 as a merge target no longer exists. Another resource r_3 that would be merged into r_2, will be merged into r_1 instead.
3. The (intermediate) results become indeterministic as there is no guarantee that resources will always be compared and merged in the same order. Let us consider the (simplified) example of three resources r_1, r_2, r_3 with labels r_1.label =‘Example 1’, r_2.label =‘Example Two’, r_3.label =‘Example Three’, the indexes represent the age, i. e., r_1 is the oldest resource. If we merge r_1 and r_2, the client will receive an intermediate result with r_1’s label being “Example Two” as we keep the longer label when merging. If instead, we matched (and merged) r_2 and r_3 first, the intermediate result would contain r_2 with label “Example Three”.

The first two problems can be addressed by keeping track of the merge process in a graph, more precisely a forest. We define a directed graph $G = (V, E)$ where each node $v \in V$ represents a resource. Two resource nodes r_1, r_2 are connected (i.e. the edge $(r_1, r_2) \in E$), if the entity resolution process recognized them as identical and thus merged them (see Figure 3). The problem of updates and deletions is handled in so far as the client receives corrections to earlier results and eventually updates or deletes duplicates delivered to the client in an earlier step of the resolution.

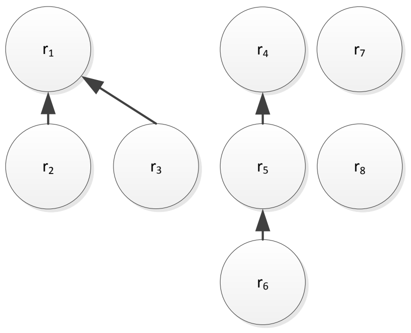


Figure 3. A forest of resources r_1 through r_8. State: r_2 and r_3 have been merged into r_1; r_6 has been merged into r_5 which in turn has been merged into r_4; r_7 and r_8 are not merged (yet). Entities thus are: $e_1 = \{r_1, r_2, r_3\}; e_2 = \{r_4, r_5, r_6\}; e_3 = \{r_7\}; e_4 = \{r_8\}$.

Preconditions for Comparing Resources A further issue that has to be addressed is that all resources should actually have the chance to be compared with one another, unless one of the preconditions rules out a match between them. Without parallel processing, this is easily ensured. In our threaded approach, we assure this by having an entity manager that receives the resources from providers in disjoint sets and thus can start a resolution thread on each such set against all others. When querying data providers, we may very well receive a total of more than 1000 records in major cities. Naive entity resolution, i.e. comparing all pairs of resources, is therefore out of the question, since that would result in $\binom{n}{2}$ and thus $O(n^2)$ comparisons which requires too much time for on-the-fly matching. Instead, we cut down the number of comparisons by applying precondition heuristics which are based on the conditions. The precondition heuristics we use are: a) Transitivity: If two resources have a common ancestor r_1 in the merge tree, we will not carry out a comparison of r_2 with r_3. b) Type: Only resources of the same type are compared, i. e., we build buckets and compare events with other events, but not with locations, persons or organizations and so on [3]. c) Physical location: We calculate the distance between resources using the Haversine formula [8] and only consider pairs of resources if their distance is less then 500m or their postal addresses are similar (to account for wrong coordinates from a data provider).

Matching and Merging We only compare pairs of instances for which the preconditions apply. When comparing instances, certain properties are more important than others. We account for that by assigning weights to the different properties as shown in Table 1, which are used in the scoring process. When actually comparing two resources, we apply fuzzy matching and inexact string matching [3,6,7]. Features that are more distinctive receive a higher weight, while less distinctive features are weighted such that they may tip the scales if necessary. As it is well known that exact text matching can be difficult on text

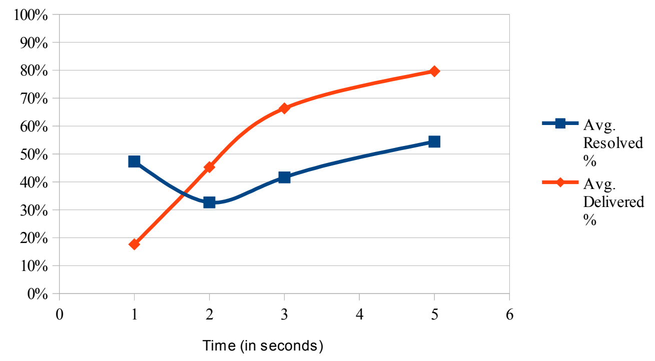


Figure 4. Delivered resources and percentage of resolved resources over time.

formatted in an inconsistent/heterogeneous way (e.g. [6]), our approach makes use of string similarity metrics as well.

Resources are merged if the score is above a certain threshold. When merging resources, some attributes require special treatment, while others may not need to be touched at all. In the merging process, the older/oldest resource takes precedence in so far, as its properties will mostly be assumed to be correct unless they were empty. Exceptions to this are the label and the description, respectively. Here, we assume that longer text is better. A resource is considered old(er) if it has already been merged into. Thus, the oldest resource is the root of its merge tree. We call the oldest resource in a merging process r_{old} or merge target, while the other resource is r_{new} or the merge source. When merging, only r_{old} will be changed. Thus, given that we want to merge two resources r_1, r_2, the merging process may or may not actually merge these two specific resources. If r_2 has already been merged into another resource, we will find the root of r_2's merge tree and set it as r_{new}. Respectively, if r_1 has already been merged into another resource, r_{old} will be set to the root of r_1's merge tree. In the situation depicted in Figure 3, if we merged r_7 into r_6, it would be merged into r_4 instead. If we merged r_6 into r_2, we would instead merge r_4 into r_1.

3.3 Experiment: Resolved Entities over Time

Analyzing the amount of resolved resources the client received at a given point in time is difficult due to our threaded approach. It may very well happen that a thread that was started later than another will finish sooner. We can provide an estimate though, which can be seen in Figure 4. It shows the average percentage of resources that a client has received (out of all resources to their request) and the percentage of resources that have undergone the resolution process. We ran queries for the 5 largest cities by population in the US and Germany, respectively. On average, we retrieved 959.2 resources per query. Almost 80% of all resources are delivered within $5s$ from the request. At the same time, around 54% of the received resources have been fully resolved. More details on our experiments like determining the true positive rates of our engine can be found in our TR [9].

4 Related Work

There exist various approaches to entity resolution such as [10,11,6]. The conditions underlying to our approach are very similar to [3]. Other systems for entity resolution we might have used are, e. g., Silk [12] or LIMES [13]. We did not use Silk, because it is designed to access data via the SPARQL protocol. In addition, Silk does not provide for a shared-memory model where the entity resolution is executed by in principle arbitrary many parallel entity resolution threads. One problem with LIMES is that it requires all resources to be available before starting the resolution process. Given that some data providers we query answer only after more than 2.5 minutes, we could not use LIMES. A more extensive discussion of related work can be found in our TR [9].

The mobEx app with the incremental entity resolution is available at Google Play: https://play.google.com/store/apps/details?id=de.unima.mobex.client

References

1. Bhattacharya, I., Getoor, L., Licamele, L.: Query-time entity resolution. In: SIGKDD, ACM (2006) 529–534
2. Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R.: Robust and efficient fuzzy match for online data cleaning. In: SIGMOD, ACM (2003) 313–324
3. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E., Widom, J.: Swoosh: a generic approach to entity resolution. VLDB J. **18**(1) (2009) 255–276
4. Schneider, M., Scherp, A., Hunz, J.: A comparative user study of faceted search in large data hierarchies on mobile devices. In: MUM. (2013)
5. Taheriyan, M., Knoblock, C.A., Szekely, P.A., Ambite, J.L.: A graph-based approach to learn semantic descriptions of data sources. In: ISWC. (2013) 607–623
6. Michalowski, M., Ambite, J.L., Thakkar, S., Tuchinda, R., Knoblock, C.A., Minton, S.: Retrieving and semantically integrating heterogeneous data from the web. Intelligent Systems, IEEE **19**(3) (2004) 72–79
7. Cohen, W.W.: Data integration using similarity joins and a word-based information representation language. TOIS **18**(3) (2000) 288–321
8. Shumaker, B., Sinnott, R.: Astronomical computing: 1. computing under the open sky. 2. virtues of the haversine. Sky and telescope **68** (1984) 158–159
9. Opitz, B., Sztyler, T., Jess, M., Knip, F., Bikar, C., Pfister, B., Scherp, A.: An incremental approach to entity resolution (2013) URN: urn:nbn:de:bsz:180-madoc-347579, URL: https://ub-madoc.bib.uni-mannheim.de/34757.
10. Kang, H., Sehgal, V., Getoor, L.: Geoddupe: a novel interface for interactive entity resolution in geospatial data. In: Information Visualization, IEEE (2007)
11. Bhattacharya, I., Getoor, L.: A latent dirichlet model for unsupervised entity resolution. In: Int. Conference on Data Mining. (2005)
12. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Silk-a link discovery framework for the web of data. In: Linked Data on the Web, Citeseer (2009)
13. Ngomo, A.C.N., Auer, S.: Limes: a time-efficient approach for large-scale link discovery on the web of data. In: AAAI. (2011) 2312–2317