An Incremental Approach to Entity Resolution

Bernd Opitz
University of Mannheim
Mannheim, Germany
jopitz@mail.uni-mannheim.de

Timo Sztyler
University of Mannheim
Mannheim, Germany
tsztyler@mail.uni-mannheim.de

Michael Jess
University of Mannheim
Mannheim, Germany
mijess@mail.uni-mannheim.de

Florian Knip
University of Mannheim
Mannheim, Germany
fknip@mail.uni-mannheim.de

Christian Bikar
University of Mannheim
Mannheim, Germany
cbikar@mail.uni-mannheim.de

Bernd Pfister
University of Mannheim
Mannheim, Germany
bpfister@mail.uni-mannheim.de

Ansgar Scherp
University of Mannheim
Mannheim, Germany
ansgar@informatik.uni-mannheim.de

ABSTRACT

We present a query-time entity resolution process that works in a highly parallel fashion. We use the application MobEx to showcase our process, which consists of a mobile client and a server, where the server takes the role of a mediator and carries out the resolution. Results are propagated to the client as early as possible. Resolution results that are produced later in the process are send as updates to the client and thus improve earlier results.

Keywords

geospatial entity resolution, fuzzy/approximate matching

1. INTRODUCTION

Due to the evolution of the Internet, today a multitude of providers for geospatial data such as public places and organizations as well as temporal data such as events exist. The data that can be retrieved from these providers usually have overlaps amongst one another – oftentimes even within the results from one provider – while at the same time, information may not always be correct and complete. A further problem is that the provision of data is heterogeneous concerning access methods and data structures. Entity resolution in these databases is often considered too expensive which is further complicated by the growth these data providers face [3]. Thus, when querying these databases, one has to deal with unclean, incomplete, and duplicate results. In our approach, we try to combine existing research as well as our own ideas into an approach that deals with these issues and provide on-the-fly matching, deduplication and integration of information from multiple sources.

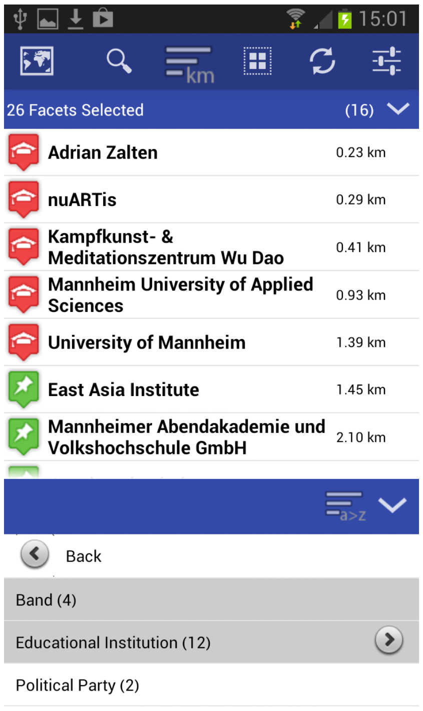
When querying multiple data providers for information about the same subject or location, it is quite common to find redundancy in the overall result. For example, multiple providers may have complementary information about the same entity or even (exact) duplicates [1, 3, 6]. This is further complicated by variations between the retrieved data such as different spellings (possibly mistakes) or missing information [1]. 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) or matching. As data and its structure are heterogeneous across data providers, preprocessing and refinement of the data is required, before resolution can take place. In this paper, we present an on-the-fly entity resolution approach, i.e., we carry out ER at query-time. The MobEx application will serve to showcase our approach. We will 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. The key difference between most existing approaches and our approach is that we do not have all records when the resolution process is started, i.e., we do not have a complete view of all (possibly) relevant data. Thus, the resolution process receives more resources as we go along and results gradually become more complete.

2. PROBLEM DESCRIPTION

When querying a data provider, one has no guarantee that it will reply within a given time. When querying multiple data providers, one cannot expect to receive all results at the same time. In addition, some providers may not answer at all. These issues make it infeasible to wait for a complete result before beginning entity resolution. Thus, one problem our approach has to handle, is rooted in the asynchronous nature of the replies from data providers: The complete set of all resources – which are thus candidates for entities – is not known a priori and resources arrive asynchronously. Still, users will not be willing to wait a long time for an answer from the server – they will most likely expect results in a matter of seconds. We are therefore faced with the tradeoff of run time and efficiency: While the process may not substantially increase the time until the client receives at least some results, false merges should be avoided at all costs (as they essentially render the data useless) while a few remaining duplicates may be acceptable. Thus, we slightly prioritize precision over recall.

2.1 Our showcase: MobEx

The MobEx application will serve to showcase our resolution approach, which should – with a few adaptations – be applicable in any similar scenario. MobEx consists of an (Android) client application (see Figure 1) and a server. The client queries the server for events, organizations, persons, and places (with sub-categories or "facets") for a given location. The results a client receives can be navigated in a facet structure or simply displayed on a map. A details view for a selected object, e.g., a restaurant also provides further options such as allowing to initiate a call to that restaurant if a phone number is available.



15:01

26 Facets Selected (16)

Adrian Zalten	0.23 km
nuARTis	0.29 km
Kampfkunst- & Meditationszentrum Wu Dao	0.41 km
Mannheim University of Applied Sciences	0.93 km
University of Mannheim	1.39 km
East Asia Institute	1.45 km
Mannheimer Abendakademie und Volkshochschule GmbH	2.10 km

Back

Band (4)

Educational Institution (12)

Political Party (2)

Figure 1: A screenshot of the MobEx application showing the facet view with the facets *Band* and *Educational Institution* selected.

The entity resolution process entirely takes place on the server. When a client wants to retrieve information, he can choose from a predefined list of data providers on his phone. The server in turn queries the selected data providers and delivers the result to the client. Thus, it takes the role of a mediator (see Figure 2). In MobEx, entity resolution takes place on resources. The exact formalization valid throughout the remainder of this paper can be found in Section 2.2.

2.2 Formalization

When retrieving information from data providers, we will call the received data *records*, where each record represents what the provider deems to represent an entity. Each record is specific to its provider, i.e., the data labels are not consistent across providers. While one provider may call an

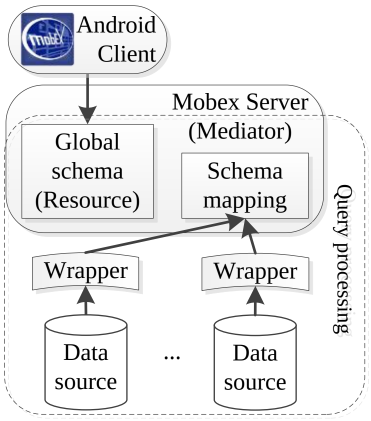


Figure 2: The data integration architecture of the MobEx showcase project. The server acts as a mediator between data sources and the client.

attribute *name*, other providers may call it *label* or *title* while the semantics of these attributes are (roughly) identical. We manually harmonized the schema information of the data providers by mapping the records into resources. 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 can be mapped into such a resource, allowing for easy comparability. We take care that the mapping respects the semantics of the properties, i.e. for any data source, the result of the mapping has the same meaning. We represent an entity e_i by using a resource which contains information taken from one or more resources. Given a set of resources $R = \{r_1, ..., r_n\}$, the assumption at the beginning of the resolution process is that each represents an entity of its own, which we denote by $\forall i \in \{1, \ldots, n\}\ e_i = \langle r_i \rangle$. During or after resolution, the state will have changed to $e_j = \langle r_{k_j}, r_{l_j}, \ldots \rangle$ for some $js \in \{1, ..., n\}$, while it remains the same for all others. This represents the resolution found that resources $r_{k_j}, r_{l_j}, \ldots$ represent the same entity, namely e_j. In other words: after entity resolution, some resources contain information that previously belonged to several other resources. No resource can represent two different entities, i.e. $\forall e_i, e_j\ e_i \neq e_j \nexists r_k : r_k \in e_i$ and $r_k \in e_j$. Thus, after resolution: if a resource r contains information from resources r, s and t, then r, s and t represent the same entity and none of them represents any entity. While the resolution process still knows which (source) resource was merged into which other (target) resource, the information that were not merged from the source to the target will no longer be visible to the client. Information that is overwritten in the target of a merge is no longer available as it is assumed that the new information is better.

In the remainder of this paper, we will discuss existing approaches and related work in Section 3. Our match and merge process will be presented in detail in Section 4. The evaluation of our approach and its results are discussed in Section 5, along with an analysis of potential problems. We will conclude with an outlook in Section 6.

3. RELATED WORK

Various approaches to entity resolution already exist, among them semi-automatic/interactive matching as presented in [4, 10, 11], (fuzzy) logic systems [2, 12, 17] or (machine) learning and clustering based approaches [13]. However, any non-automatic approach is infeasible for on-the-fly entity resolution. We therefore focus on automatic approaches. There are various methods of carrying out automatic entity resolution. The underlying conditions of a generic entity resolution process described in [1] are very similar to the conditions in our approach. These conditions are:

- Pairwise matching and merging: We only compare two resources with each other and decide whether to merge them or not. However, this does not imply that the results of a matching will not affect further matching. Indeed, previous matches and merges may influence the further process.
- No confidences: While there may be similarity computations involved in the matching process itself, a merge is always done with full confidence or not at all. That is, the matching may use confidence values, but a merge is an absolute decision after which the merged resource has no confidence value assigned.
- No relationships: If a concert (as an event) takes places at a stadium (as a place), there exists a (real world) relationship between these two entities. While such relationships do frequently exist in the real world and using these information thus might lead to better results in the resolution process, they are not considered as this makes resolution by far more complex.
- Consistent labels: This condition describes the assumption, that fields of a resource carry a fixed meaning that is the same for all resources. It is not required that there is only one value for every field in the resource. However, not all attributes in our resources can carry multiple values (see Table 1).

All of these assumptions are commonly used in the commercial world and are also true for the approach used by MobEx. Aside from these conditions, we use another technique – namely "buckets". Buckets work such that some attribute is selected as the bucket label. As "record matching is inherently expensive" [1], we use the highest level facet, i.e., type or category (event, organization, person or place) of a resource as a bucket label. In the matching process, only pairs of resources that are in the same bucket are compared (for details see Section 4.4 below). Given the four aforementioned conditions, this significantly cuts down the number of comparisons as long as resources are distributed among the categories.

Multiple systems for entity resolution already exist that we could have used instead of developing our own approach. One very-well known system is Silk [20] another system that would partially suit our needs is LIMES [16]. We did not use Silk, because it is designed to access "data sources that should be interlinked via the SPARQL protocol", i.e. data is accessed using SPARQL and then "explicit RDF links between data items" [9] are set. However, most data providers that we access do not offer a SPARQL endpoint, which would provide an open and standardized access method to their data. For details on SPARQL please see [8, 18]. Instead, most of the providers we use are commercial data providers and thus offer a proprietary API Data access is regulated at the providers discretion and there is no standardized method to access multiple providers. If we wanted to use Silk, we would have to add a further step of indirection to our matching process, in particular we would have to apply some sort of transformation of the data to RDF. This indirection would reduce the performance of our approach.

One problem with LIMES is that it requires all resources to be available before starting resolution. Given that some data providers we query answer only after more than 2.5 minutes, there are two possibilities to handle this issue. We could send out results without any resolution as soon as we get them and send updates as soon as all resources have arrived and undergone resolution. The other possibility would be to wait for a complete result from the data providers and send the client only the complete resolved result. In both cases, it is questionable whether the updated/resolved results are still relevant to the user when he receives them. Our analysis of user behavior suggests that most users will not use the app at one specific location for a long enough time within a session: the average session time was around three minutes and only around 23% of all sessions were active for that long or longer. As most users were not even active for more than three minutes, it is safe to assume that most users will not be willing to wait that long until they receive results. Aside from the already mentioned problems, LIMES is also designed to work on SPARQL endpoints. This adds the same indirection that we would have faced with Silk, thus the same argument as above applies.

As we query multiple data providers, it may not seem intuitive why query-time entity resolution is chosen. Still, there is a clear motivation for it: Firstly, we do not know the client's query beforehand and it is infeasible to keep aggregations for different selections of data providers in stock. Secondly, we do not have complete knowledge of the data the providers return ahead of the query. Taking into account query and answer times, we thus have to carry out resolution on the fly as users will not be willing to wait a long time. Thirdly, information about events may change on a daily basis. These three criteria are further motivated by the approach described in [3] which deals with ER in a similar scenario, but in a completely different way. The approach described therein uses relational similarity and collective resolution in addition to attribute similarity. This requires certain links within the data, e.g. when resolving author names, looking at co-authors can be helpful as an author may regularly work together with the same co-authors, which would then help in identifying both as the same individuals. However, such links are not present in the closed silos of data providers. While the methods differ, the goal is the same, namely that a user querying one (or as in our case multiple databases) should receive relevant results which are resolved with respect to the particular query they posed.

Another aspect of the matching process is how comparisons of resources take place. Fuzzy matching, uses string similarity metrics such as edit distances and was proposed more than 15 years ago [14, 15]. Since then, several matching methods using fuzzy/inexact string matching were developed [1, 6, 7, 13], some of them specifically on the attributes of organizations or locations such as the names, addresses or phone numbers [1, 7, 13]. Their research also provides the foundation of our weight assignments. 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 formatted in an inconsistent/heterogeneous way (e.g. [13]), our approach makes use of string similarity metrics as well.

4. MATCHING PROCESS

4.1 Overview

As we do not have all resources at hand when the resolution process is started, the resolution process receives more information as we go along and results gradually become more complete. Our entity resolution approach takes place in a highly parallel fashion as can be seen in Figure 3. The numbers in the following description of our 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 3) as well as the entity resolution (lower half of Figure 3). 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) (green lines). The Entity-Resolver-Thread handles, as already mentioned, the entity resolution, i.e. it tries to find and handle duplicates such as ‘Cafe Vienna’ and ‘Vienna Cafe’. 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) (blue lines). 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, a delivered result may have to be rectified or updated in a later step of the resolution.

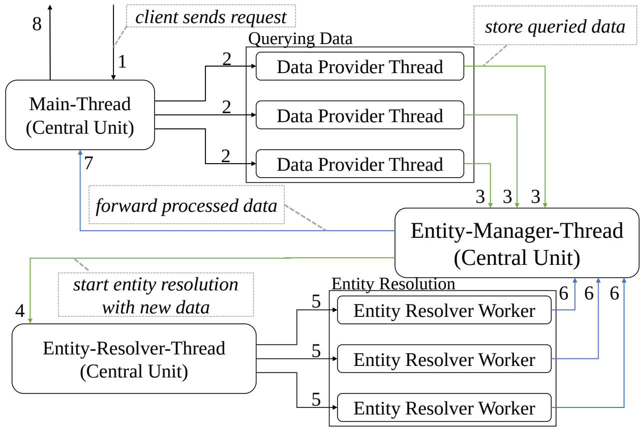


Figure 3: Data flow of processing a client request and incrementally matching the data. The numbers denote the order of the processes.

4.2 Facet Tree

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. 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 like DBpedia, Eventful, Qype, OpenPOI and GeoNames. 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 in this project.

4.3 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” [13]. 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 assigned using the information from [1, 7, 13]. While none of them gives explicit numbers, there are hints which properties are important or more distinctive. The actual numbers were then assigned using empirical results. For example, it is legitimate to assign the URL the greatest weight, as URLs are – by their nature – a unique identifier. A similar argument applies for the label and phone number which are both designed to serve as an identifier for, e.g., a person, organization or place. These rules are not without exception as can be seen in our problem analysis in Section 5.1. For example, a postal address may not be distinctive, e.g. if there are multiple organizations located in the same office building.

The entity resolution process in the MobEx server is a multi-threaded and thus parallel processing of the data. 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,r_2 where r_1 is the older object (and thus merge target) and r_2 have already been merged, the further merge process is affected as follows: r_2 as a merge target no longer exists. Thus, a resource r_3 that would be merged into r_2 instead has to be merged into r_1.

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”. This example of indeterminism also goes to show that the delivered results only represent the truth to the best of our knowledge at a given point in time. Resolution results at a later point in time may cause updates and possibly even deletions of resources that were already delivered earlier, because e.g. these resources no longer represent an entity of their own.

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 4). The problem of updates and deletions is handled in so far as that the client receives corrections to earlier results and eventually updates or deletes duplicates delivered in an earlier step of the resolution. The details of the matching process in terms of threading were explained in Section 4.1.

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.

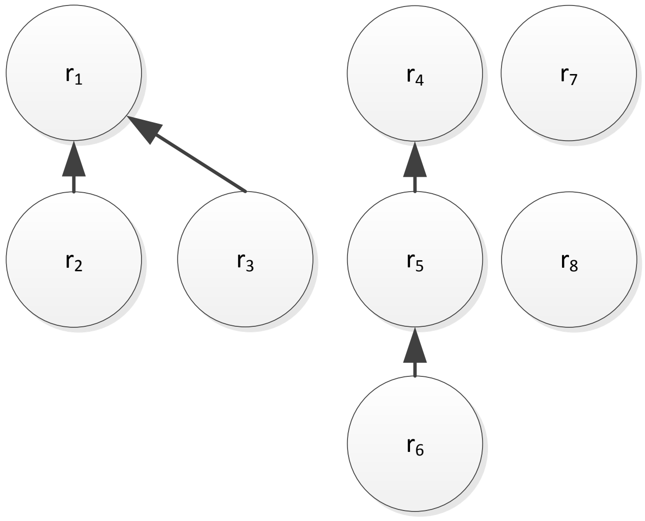


Figure 4: A forest of resources r_1 through r_8. State: r_2 and r_3 have been merged into r_1; r_6 has been merged into r_5 which in turn has been merged into r_4; r_7 and r_8 are not merged (yet). Entities thus are: $e_1 = \{r_1, r_2, r_3\}; e_2 = \{r_4, r_5, r_6\}; e_3 = \{r_7\}; e_4 = \{r_8\}$.

4.4 Preconditions

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 described in Section 3. The precondition heuristics we use are:

- **Transitivity:** Given two resources, e.g. r_2, r_3 from Figure 4. If they already have a common ancestor r_1 in the merge tree, we will not carry out a comparison of r_2 with r_3. This common ancestor represents that a merge between r_2 and r_3 or resources either of them has been merged into, have already taken place.

- **Type:** Only resources of the same type are compared, i.e. we compare events with other events, but not with locations, persons or organizations and so on. This is consistent with the literature like [1].

- **Physical location:** The greater the physical distance between two resources, the less likely it is that they describe the same entity [16]. We account for that by calculating the distance between resources using the Haversine formula [19]. We will only consider pairs of resources for resolution if

  a) they have a distance of at most 500m from one another. We call this value the *distance threshold*.

  or

  b) their postal addresses are similar (to account for wrong coordinates from a data provider). We define two postal addresses as similar, if the average

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	*	a 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	the name of the entity	University of Mannheim
String	description	*	a 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	a 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: The relevant 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 is still required (explanations below).

of their Jaro-Winkler and Levenshtein similarity is greater than 0.75.

The value of 500m that we use for the distance threshold was determined empirically. When looking at a visualization of results from data providers on a map, we found that the last (obvious) duplicates were less than 500m apart. The Haversine formula is more appropriate than other distance measures such as the Euclidian distance since it is easy and fast to calculate and works well for small distances where other formulas show rounding errors [5, 19].

4.5 Matching (Scoring & Comparisons)

In the scoring process, 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. The scoring works as follows: Let r_1, r_2 be two instances, $C = \{c_1, \ldots, c_n\}$ the set of resource attributes with a non-zero weight (see Table 1) that are present on both resources (i.e. they are not null or empty), n=|C|, $w_1, \ldots, w_n$ the weights assigned to each attribute and $compare(r_1.x, r_2.x)$ a comparing function, defined as follows:

$$compare(r_1.x, r_2.x) = \begin{cases} 1 & r_1.x = r_2.x \\ 0 & \text{otherwise} \end{cases}$$

for all properties except for the label and parts of postal addresses for which we define $compare(r_1.x, r_2.x)$ as the average of the Jaro-Winkler and Levenshtein similarity of $r_1.x$ and $r_2.x$. Thus, $compare(x, y)$ is bounded by 0 (signifying dissimilarity) and 1 (signifying identity). With the score calculated as

$$score(r_1, r_2) = \sum_{i=1}^{n} w_i \cdot compare(r_1.c_1, r_2.c_1)$$

we consider $r_1 = r_2$, i.e. the two resources represent the same entity, if $score \geq t$, where we call t the comparison threshold (we use $t = 0.7$). There are two exceptions to this process to speed it up:

- Duplicates that a provider delivers (i.e. the source and the provider's id for two instances are identical) are always considered to be identical.
- Resources with the exact same label and description – given that both properties are present, i.e. non-empty, in both resources – are always considered identical.

4.6 Merging

Resources are merged if the scoring determines to do so. 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 respectively the description, where 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 4, if we were to merge r_7 into r_6, it would be merged into r_4 instead. If we were to merge r_6 into r_2, we would instead merge r_4 into r_1. Note that the depicted merging state of r_4,r_5 and r_6 thus came to pass by merging r_6 into r_5 and then merging r_5 (or r_6) into r_4.

5. EVALUATION

To evaluate our matching approach, we conduct the following experiments:

1. We want to determine the effect of the preconditions described in Section 4.4. The first hypothesis is that the application of the preconditions, especially the distance condition, massively reduces the number of comparisons that have to be carried out. We carry out matching on identical data with different parameter settings and evaluate the influence on the results.
2. We run queries for multiple data providers in the 5 largest cities by population in the United States and Germany respectively. The cities are: New York (NY), Los Angeles (CA), Chicago (IL), Houston (TX) and Philadelphia (PA) for the Unites States and Berlin, Hamburg, Munich, Cologne and Frankfurt am Main for Germany. We query all data providers in the exact center of each city with a radius of 3.1km and carry out entity resolution on the received data. We manually check the results for correct merges, i.e. the merged resources represent the same entity (true positive) and incorrect merges, i.e. the merged resources represent different entities (false positive). Missing merges, i.e. pairs of resources that were not merged although they represent the same entity (false negatives) and correct non-merges, i.e. resources that were not merged, that represent different entities (true negatives) will not be evaluated on a quantitative basis. In order to do so, we would have to compare every resource with all other resources which would result in $\binom{n}{2}$ comparisons for n resources. Instead, we will conduct a qualitative analysis by looking at examples that should have been merged, but were not. Based on the examples, we try to identify flaws in our scoring function and thus the matching. Based on the outcome of these evaluations, we will come up with potential improvements for our approach that should be addressed in future work on our approach.
3. Our entity resolution process is run at query time. As clients will not be willing to wait a long time until they receive at least some results, we want to determine how long it takes until *a)* the first items arrive *b)* 50% of all results have arrived *c)* the results are complete.

For all experiments, we will only regard the time added by the resolution process. Waiting time added by a provider's API is ignored, as any application running against an API would face these.

5.1 Results

In the following, we present our findings and analyze our approach as well as (potential) problems.

1. We queried all nine data providers implemented in the MobEx server in the ten cities named above and carried out entity resolution on the result of these queries for the following different distance thresholds: 300m, 500m, 1000m, 1500m, 2000m, 2500m, 3000m and infinite distance. The meaning of these threshold values is explained in Section 4.4. The averaged results can be seen in Table 2.

Distance	**Avg. Matches**	**Avg. Savings (in %)**
300	103.11	52.69
500	103.11	50.28
1000	102.89	41.06
1500	103.11	28.37
2000	103.11	19.13
2500	103.11	12.78
3000	103.11	7.69
∞	103.22	0

Table 2: Avg. number of matches and avg. saved comparisons for different distance threshold values on a total of 980 resources.

   As the results show, the number of matches does not vary significantly while there is a steep increase in the number of comparisons that can be saved up to the distance threshold of 500m. This supports our decision for the used threshold value. If one wanted to speed up the matching process even more, one could decrease the distance threshold to values such as 100m or even less and evaluate the impact on the outcome. One would expect that at some point, the number of matches will decrease. Thus the distance threshold provides a means to trade-off the number of matches for running time – allowing for faster results at the cost of (possibly) remaining duplicates.

2. Regarding the merging, we first of all aggregate over all cities. This gives us a total of 9834 entities which are the result of a match an merge process in which 1122 merges took places. Out of these 1068 were correct merges, i.e. the merged resources described the same entity and 54 were incorrect merges. This gives us an overall precision of 95.19%, which is very high. However, it has to be noted that some of these merges were actually very easy as they were actually duplicates from a provider which shared the same id. We consider a match "hard" if it is not an obvious duplicate. Table 3 shows the actual distribution and the precision of our approach for three assumptions of what an obvious duplicate is:

   a) There are no obvious duplicates, i.e. all matches are considered hard. Thus, matches that are based on identical id (column ID) or label and description (column L&D) as well as other matches are all summed up as *hard*.

   b) A duplicate is obvious, if two resources come from the same provider and share the same id. In that case, the match will always be considered correct and will not contribute to the precision. Thus, we sum up label and description (L&D) matches and non-ID matches as *hard*.

   c) In addition to b) we also consider duplicates easy if they share the same (non-empty) label and description. Thus, only matches where neither the id nor labels and descriptions were identical are considered hard.

   Several potential problems emerged during the evaluation of our approach:

	M.	TP	FP	ID	L&D	“hard”	P
	1122	1068	54	556	315	251	
a	1122	1068	54			1122	95.19
b	566	512	54	556		566	90.46
c	251	197	54	871		251	78.49

Table 3: Result statistics of the match and merge process aggregated over 10 major cities with a total of 9834 entities (after matching)
M.=matches, TP=true positives, FP=false positives, ID=same provider and id, L&D=identical label and description, hard=matches considered “hard” for the given assumption, P=precision in %

- A very general problem is that if only little information is available and some of these information vary, then matching is close to impossible. We try to achieve high precision as remaining duplicates may be more acceptable than potentially merging too many records (false positives).
- If the URL was present, it was often weighed too highly. While an identical URL may be a strong indicator of identity, this can also cause problems. A common problem was that resources of supermarkets were merged, because the URL, label and phone number were all identical but the address differed. This is a problem as it causes perfectly good and potentially important information getting lost. Actually, many of the false positives (around 35) are caused by this error in the matching. We also found duplicate resources in the results which were not merged because of a different URL, often in addition to (slight) variations in another attribute. However, there were only few cases where this actually caused remaining duplicates (on average around 10 obvious duplicates per city), which is of course not desirable but probably acceptable. Nevertheless, this presents a starting point for improvements to our approach.
- If only little information is available and those vary between two resources, the matching will most likely fail. There is not much that can be done about such cases as adjustments might as well cause more false matches.
- In some cases, it seems that obvious duplicates exist. On a closer look, there are often slight variations in the address, often in addition to a different phone number. This raises the question whether the phone number is a good key attribute in the sense that it serves as a good identifier for a resource. This question cannot be easily answered as larger businesses or organizations will often have many phone numbers while small businesses such as restaurants often have one phone number. Keeping in mind that the approach should be somewhat generic, one must ask whether the introduction of special cases for examples such as this will decrease the performance or hinder universal usability of the approach.

3. We carried out a run time analysis by querying all of our data providers in the ten cities listed above and averaging the results. The analysis was carried out on a Debian Linux 64 bit machine with a 2.8GHz i5-2300 processor and 6GB of RAM out of which at most 5 were available to the resolution server. It can be seen that the more resources are present on the server, the longer the resolution takes. Still, the resolution process is fairly quick for the amounts of data that we faced. At an average of 746.13 resources being present in the resolution process, it took 0.046s to resolve one resource against all other resources, which corresponds to a processing speed of 204.670 resources per second. Figure 5 shows the behavior of the required time per resource resolution in relation to the number of resources in the process. It can be seen that the processing time for each resource increases as more resources are present on the server, which is caused by more comparisons that have to be carried out. The correlation coefficient $r = 0.7208$ show that around 72% of the increase in run time can be explained by the increase in the number of resources. Other contributing factors may be that the resolution process runs in parallel threads. Especially those threads that resolve larger batches of resources may run longer and thus have to share the processor with other threads.

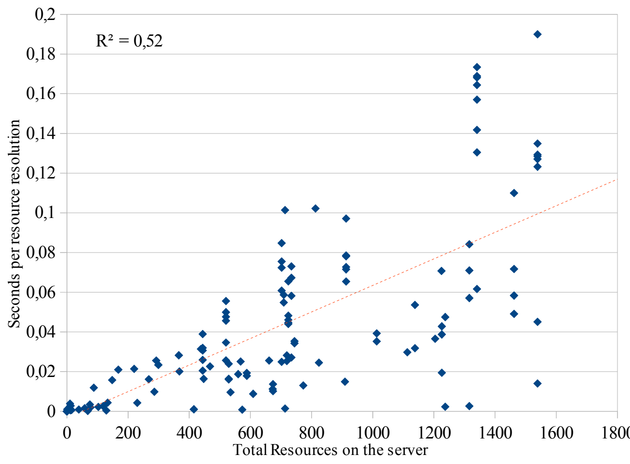


Figure 5: Run time per resource on the server. The red dotted line shows a linear regression curve with a correlation coefficient $r = 0.7208$ (coefficient of determination $r^2 = 0.52$).

The coefficient of determination r^2 of the linear regression in Figure 5 show that 52% of the variations in the run time of our resolution approach can be explained by the number of resources. The correlation coefficient r of 0.7208 shows that there is a strong correlation between the number of resources and the run time.

4. First of all, it is difficult to give an exact number of resolved resources that a client will have received at a given time. As our approach is threaded, it may very well happen that a thread that was started later than another will finish sooner. We will thus provide a number that we call *resolved ratio* (rr) which represents the number of resources that the client has received that have undergone the resolution process in proportion to the total number of resources the client has received. If we were to divide by zero, rr is undefined. This number should not be seen as an exact figure, but a lower-bound estimate. Furthermore, we do not directly include answer times from

providers in our calculations. They indirectly affect resolution as records that arrive at the server sooner than others, we start the resolution thread sooner. As already said, there is no guarantee that threads finish in the order they were started. Table 4 shows an example: First, 286 resources were added and resolved, then 13 more (second and third row). Now four batches of 100 resources and one with 25 resources arrived and started resolution. The batch of 25 resources arrived last, so it was resolved against 724 $(286 + 13 + 4 \cdot 100 + 25)$ resources. It finished before the four resolution threads that were running on the batches of 100 resources (fourth row). Thus, at that point in time 724 resources were forwarded to the client, but only 324 resources had been resolved.

acc. Add	Time	Res.	Matches	RR
0	0	0	0	-
286	2816	286	2	1
299	304	299	2	0,45
324	1637	724	19	0,59
424	4468	724	22	0,72
524	4608	724	24	0,86
624	4400	724	30	1
724	4816	724	31	1
824	2452	824	34	1
909	1266	909	36	1

Table 4: Example table of the resolution results. Each row in the table corresponds to one worker thread that has finished resolution on a bulk of resources. acc. Add=accumulated added resources, Time=Resolution time in ms, Res=Total resources at that time, Matches=Total number of matches, RR=resolved ratio

Analyzing the actual amount of resolved resources that the client has received at a given time proved difficult due to our threaded approach. We can provide an estimate though, which can be seen in Figure 6. It shows the average percentage of resources that a client as received (out of all resources to their request) and the percentage of the received resources that have undergone resolution so far. In these tests, we retrieved an average of 959.2 resources in total. As can be seen from Figure 6, almost 80% of all resources can be delivered within 5 seconds of being retrieved, i.e. they are at least partially resolved at that time. At the same time, around 54% of the received resources have been fully resolved.

6. CONCLUSION

Our goal was to build an entity resolution approach that is very precise while not imposing a long waiting time. When querying external data providers, there is already some waiting time involved which depends on various factors such as the time of the day and the query complexity. For our data providers, we found answering times as short as one second but also up to 157 seconds. As users are not willing to wait for a long time, we chose to design our approach for speed. Furthermore, we wanted to avoid information getting lost, i.e. we endeavored a precise match and merge approach. This includes forwarding partially resolved resources to the client and sending updates later on. In general, the weights assigned to the different attributes in the

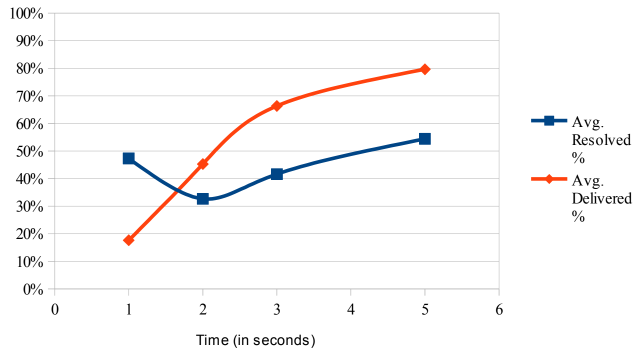


Figure 6: Resources the client receives and percentage of completely resolved resources among them over time. The point where the client has received 100% of the resources in a resolved state is not shown, because it depends too strongly on the time when the last provider answers and would thus distort the graphic.

scoring process and the scoring threshold may have to be adjusted, as they were assigned based on empirical observations and roughly corresponded to experiences found in other research. Furthermore, while similarity or even identity between two resources' attributes may indicate that they represent the same entity, dissimilarity does not necessarily indicate the contrary. We tried to account for that by weighted scoring, which fails in case of only little information being available.

The approach works fairly well on the data we faced. However, some problems are visible that can be traced back to the design of the approach, especially the weights that were assigned to the different attributes and the reaction of the approach to only little information being available. For example, when only little information is available for a resource, but one of these attributes is the (highly weighted) URL, it can happen that it is merged with another resource with the same URL although they represent different entities. This can happen with resources representing subsidiaries of large companies where the URL points to the website of the company. This is one aspect where future work would have to improve.

Acknowledgement. We thank Florian Rang and Katja Beer of the telegate Media AG for their valuable input and support for this work. This research is partially supported by the klickTel Award 2013.

References

[1] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal – The International Journal on Very Large Data Bases*, 18(1):255–276, 2009.

[2] Indrajit Bhattacharya and Lise Getoor. A latent dirichlet model for unsupervised entity resolution. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, 2005.

[3] Indrajit Bhattacharya, Lise Getoor, and Louis Licamele. Query-time entity resolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 529–534. ACM, 2006.

[4] Mustafa Bilgic, Louis Licamele, Lise Getoor, and Ben Shneiderman. D-dupe: An interactive tool for entity resolution in social networks. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pages 43–50. IEEE, 2006.

[5] U. S. Census Bureau. Geographic information systems FAQ, 1997. URL `http://www.movable-type.co.uk/scripts/gis-faq-5.1.html`. [Online; accessed 16 September 2013].

[6] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 313–324. ACM, 2003.

[7] William W Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems (TOIS)*, 18(3):288–321, 2000.

[8] W3C SPARQL Working Group. SPARQL 1.1 Overview, 2013. URL `http://www.w3.org/TR/sparql11-overview/`. [Online; accessed 19 September 2013].

[9] Robert Isele, Anja Jentzsch, Christian Bizer, and Julius Volz. Silk - a link discovery framework for the web of data, 2011. URL `http://wifo5-03.informatik.uni-mannheim.de/bizer/silk/#about`. [Online; accessed 19 September 2013].

[10] Hyunmo Kang, Vivek Sehgal, and Lise Getoor. Geoddupe: a novel interface for interactive entity resolution in geospatial data. In *Information Visualization, 2007. IV'07. 11th International Conference*, pages 489–496. IEEE, 2007.

[11] Hyunmo Kang, Lise Getoor, Ben Shneiderman, Mustafa Bilgic, and Louis Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(5):999–1014, 2008.

[12] Xin Li, Paul Morie, and Dan Roth. Identification and tracing of ambiguous names: Discriminative and generative approaches. In *Proceedings of the National Conference on Artificial Intelligence*, pages 419–424. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004.

[13] Martin Michalowski, Jose Luis Ambite, Snehal Thakkar, Rattapoom Tuchinda, Craig A Knoblock, and Steve Minton. Retrieving and semantically integrating heterogeneous data from the web. *Intelligent Systems, IEEE*, 19(3):72–79, 2004.

[14] Alvaro E Monge and Charles P Elkan. Efficient domain-independent detection of approximately duplicate database records. In *Proc. of the ACM-SIGMOD Workshop on Research Issues in on Knowledge Discovery and Data Mining*, 1997.

[15] Alvaro E Monge, Charles Elkan, et al. The field matching problem: Algorithms and applications. In *KDD*, pages 267–270, 1996.

[16] Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes: a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2312–2317. AAAI Press, 2011.

[17] Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart Russell, and Ilya Shpitser. Identity uncertainty and citation matching. In *Advances in neural information processing systems*, pages 1401–1408, 2002.

[18] Eric Prud'Hommeaux, Andy Seaborne, et al. SPARQL query language for RDF. *W3C recommendation*, 15, 2008.

[19] BP Shumaker and RW Sinnott. Astronomical computing: 1. computing under the open sky. 2. virtues of the haversine. *Sky and telescope*, 68:158–159, 1984.

[20] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk-a link discovery framework for the web of data. In *Linked Data on the Web*. Citeseer, 2009.