On-the-fly Entity Resolution from Distributed Social Media Sources for Mobile Search and Exploration

B. Opitz, T. Sztyler, M. Jess,
F. Knip, C. Bikar, B. Pfister
University of Mannheim
fknip, cbikar, bpfister, jopitz, tsztyler,
mijess@mail.uni-mannheim.de

A. Scherp
ZBW – Leibniz Information Centre for
Economics and Kiel University, Germany
a.scherp@zbw.eu

ABSTRACT

We present an approach and mobile application for the interactive exploration and search of geo-located social media entities from different, distributed data providers on the web. When querying the providers, the returned results typically have some overlap. In addition, one has no guarantee that the providers reply within a given time interval. Thus, in order to provide users with geo-located entities in their vicinity in a timely manner, we need to take the asynchronous nature of the data providers' replies into account. Our novel on-the-fly entity resolution engine starts the entity resolution once it retrieves the first responses. It incrementally extends the entity resolution model when more responses arrive. Entities are propagated to the client once the resolution engine has processed them for the first time. Resolution results produced at a later point in time are sent as updates to the client and improve earlier, incomplete results. Our experiments show a matching precision of 95% and scalability of the on-the-fly entity resolution w.r.t. the number of resources being simultaneously processed.

CCS Concepts

• **Information systems** → **Entity resolution;**

Author Keywords

Entity resolution; mobile exploration; social media

INTRODUCTION

There exist a multitude of providers for spatio-temporal data such as public places and organizations as well as events like DBpedia[1], Eventful[2], Qype[3], OpenPOI[4], and GeoNames[5]. The data that can be retrieved from these providers usually have overlaps amongst one another, while at the same time information may not always be complete. Thus, 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 [3, 5, 1]. This is further complicated by variations between retrieved entities such as different spellings (possibly mistakes) or missing information [1]. Another 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. This is further complicated by the rapid growth of the amount of data in the databases [3]. Thus, when querying databases like the providers mentioned above, one has to deal with unclean, incomplete, and duplicate results. 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, users will prefer a consolidated resource representing an entity instead of multiple resources describing the same entity. In addition, the users expect results in a matter of seconds as our earlier user evaluation shows [12].

The process of eliminating duplicates and merging them into one resource is called entity resolution. In this paper, we present a novel approach for entity resolution using on-the-fly matching, deduplication, and integration of entities such as locations and events from multiple data providers. We use techniques such as fuzzy matching and threading as well as precondition heuristics that reduce the number of comparisons to be carried out. The entity resolution process starts once first responses from the data providers are received. The entity resolution model is incrementally extended while more and more responses arrive. The key difference to existing work is that we do not need to have all records processed before sending out the first results to the mobile users. The entities are already propagated to the mobile client once they have been processed in the resolution engine for the first time. These initially incomplete results are improved later by continuously sending updates from the resolution engine

[1] http://dbpedia.org, accessed: 8/21/15
[2] http://event-ful.com, accessed: 8/21/15
[3] http://qype.com, accessed: 8/21/15
[4] http://open-pois.net, accessed: 8/21/15
[5] http://geonames.org, accessed: 8/21/15

MUM '15, November 30-December 02, 2015, Linz, Austria

ISBN 978-1-4503-3605-5/15/11…$15.00

DOI: http://dx.doi.org/10.1145/2836041.2836043

to the client. Thus, the results gradually become more complete. Our extensive experiments confirm the applicability of the approach: Already after one second of issuing a query already 20% of the entities are returned and 50% resolved. In addition, the resolution engine scales with the total number of resources being concurrently processed on the server. Finally, the quality of the matching is about 95%.

Below we first discuss related work and introduce mobEx[6] as showcase for an incremental matching of distributed social media resources. Subsequently, we provide a generalized introduction of the matching problem and describe in detail our incremental entity resolution approach. The novel entity resolution engine is evaluated along different criteria like quality and performance, before we conclude.

RELATED WORK

Different approaches for entity resolution have been investigated in the past such as semi-automatic and interactive matching [10, 4, 9], (fuzzy) logic systems [2, 13, 19], as well as machine learning approaches [28, 21, 7, 15]. Any non-automatic approach is infeasible for an on-the-fly entity resolution like it is required in our case where a-priori unknown user queries need to be instantly answered. On the contrary, the automated approaches for entity resolution and existing frameworks Silk [25] and LIMES [18] do not support sending out incremental updates of the entity resolution process to a consumer like in our case the mobile client. Thus, even efficient techniques for entity resolution are not prepared for a setting where the first, even still incomplete resolution results need to be send to the users in very short time where the data providers' responsiveness depends on the web and may be slow. In addition, LIMES requires that all resources to be available before starting the resolution. This makes it practically infeasible to apply such a framework as we would need to wait for a complete result from all providers before starting the resolution process and sending our results to the mobile client. In fact, we observed response times of up to 2.5 minutes by some providers.

In order to address this challenge, our incremental entity resolution engine already propagates matching results to the mobile mobEx application after the entities are processed the first time in the matching process. The initially incomplete results are continuously improved by sending out updates to the mobile client once they become available. In order to provide such an on-the-fly entity resolution, we make use of known techniques for entity matching such as forests [20] and pairwise matching and merging [1]. In addition, we use buckets where pairs of resources are only considered in the matching process, if they are of the same type, i. e., belong to the same buckets [21]. In our case, we use events, locations, persons, and organizations as buckets. While optimal buckets can also be automatically learned from the data [21], this is not applicable in our case as we never have a complete dataset at hand. In addition, the characteristics of the retrieved social media data may depend on the geospatial coverage of the data providers like countries and continents. Thus, buckets that perform well for certain locations may not elsewhere.

As we query multiple data providers, it may not seem obvious why query-time entity resolution is chosen. Still, there is a clear motivation for it. First of all, we do not know the user's query beforehand. Thus, it is infeasible to pre-compute all results and keep aggregations for different selections of the data providers on our server. In addition, the information about locations and events may change on a daily basis. This puts another burden on pre-computing the results. The approach by Bhattacharya et al. [3] allows for a query-time entity resolution by computing relational similarity and collective resolution in addition to attribute similarity. However, this requires that certain links are available within the data like co-authorship relations. Malhotra et al. [14] show that references between records such as a driver's licence ID referred to in a passport record help in merging relational datasets coming from different, controlled sources. However, such links are not present in the closed data silos of the providers we consider. At least it cannot be assumed that such links are always available.

Welch et al. [26] propose an incremental entity resolution approach with the goal to populate and curate a knowledge base with information coming from different professional data sources as well as facts extracted from websites. Like in our work, the authors also explicitly consider the possibility of having spam in the data, i. e., cases where a business owner adds several variations of their information to have their listing shown multiple times. While Welch et al. also consider merging from sources of varying latencies, they do pre-compute all resolution results before adding them to the knowledge base and making them available to the users. As said above, in our case such a pre-computation is not feasible but a resolution at users' query time is needed. In addition, Welch et al. assume that there are no true negative records, i. e., for each record there has to be a match with some other record [26]. This assumption cannot be made in our case and a single record can constitute an entity of its own. Whang and Garcia-Molina [27] present an approach for adapting the matching logic when data updates arrive. They investigated how to decide when to base the entity resolution on a previous result versus starting it from scratch. In addition, Gruenheid et al. [8] compare two different graph-based clustering approaches for an incremental entity resolution over evolving data. Thus, the authors assume that the same entity may arrive at a later point in time (i. e., when a database is updated) containing different attribute values. These approach have in common that the entity resolution results are pre-computed before they are available to the users for search. Thus, they do not update the mobile (or web) client with results of the entity resolution process while data comes in from different sources in a highly

[6] Available from Google Play: https://play.google.com/store/apps/details?id=de.unima.mobex.client, accessed: 8/21/15

asynchronous manner. In addition, the approaches require keeping (and maintaining) expensive indices of the data on the server. In our case, the entity resolution results are thrown away after each user query in order to avoid expensive caching or storing of resolution results. Rather, we solely rely on the data that is returned at query time from the distributed data providers. Finally, it is noteworthy that typically precision and recall is only compared against some naive baselines [27, 26], while we did manually check the quality of the matching results.

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 [17, 16]. Since then, several matching methods using fuzzy string matching were developed [1, 5, 15, 6]. Some of them are specifically designed for attributes describing organizations or locations such as names, addresses, or phone numbers [1, 15, 6]. These works provide the foundation of our weight assignments. More distinctive features receive a higher weight, while less distinctive features are weighted such that they may tip the scale if necessary.

MOBILE SOCIAL MEDIA EXPLORATION

The mobEx application serves as showcase for our incremental entity resolution approach. It consists of a mobile client application for Android mobile phones and a server for carrying out the entity resolution process as shown in Figure 1. The mobile application mobEx sends a user request to the server and queries for events, organizations, persons, and places at a given location. The mobEx server receives the user query and maps it to the various data providers such as DBpedia, Eventful, Qype, OpenPOI, and GeoNames mentioned in the introduction as well as klickTel Open API[7], Google Places[8], LastFM[9], and Twitter[10]. At the server side, the returned social media records are compared and merged in real-time using our novel incremental entity resolution approach. Thus, the entity resolution of the different resources retrieved from the data providers entirely takes place on the mobEx server. A resource represents an entity that is either an event, organization, person, or place. When querying data providers, all retrieved records are mapped into such a resource. Please note, at the moment the alignment of the data providers' schemata to the internal schema of mobEx is hard coded. It may be extended by some automatic approaches like [24] in the future.

The entity resolution results the mobEx client receives from its server can be navigated through a facet structure as depicted in the screenshot in Figure 2 (left). The faceted navigation has been extended from earlier work [22, 11]. The screenshot next to it shows the map view of the app. Details of an entity such as a website and opening hours can be seen in the details view as shown in two the screenshots in Figure 2 (right). While the first screenshot provides opening hours, the second is a concert with the start time of the event.

[7] http://openapi.klicktel.de/, accessed: 8/21/15
[8] https://developers.google.com/places/, accessed: 8/21/15
[9] http://www.last.fm/, accessed: 8/21/15
[10] https://twitter.com/, accessed: 8/21/15

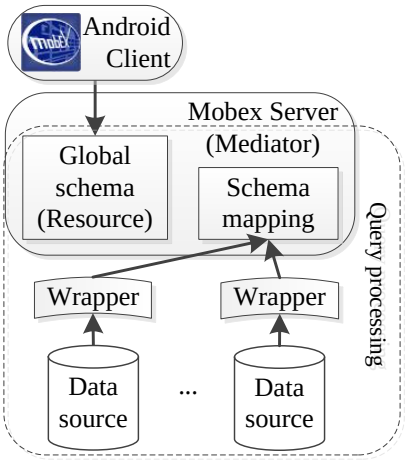

Figure 1: Client-server architecture of mobEx. The server acts as mediator between the mobile application and the distributed data providers.

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, providers may sometimes not answer at all, e. g., due to a temporary outage. These issues make it infeasible to wait for a complete result of all providers before beginning with the entity resolution. Thus, one problem our approach has to handle is rooted in the nature of asynchronous replies from the different data providers: The complete set of all resources—which are thus candidates for entities—is not known a-priori and resources arrive depending on the latency of the providers' APIs. Still, users will not be willing to wait a long time for an answer from the server [12]. Therefore, we face a trade-off of between run time and effectiveness of the entity resolution: 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 prioritize precision over recall.

The data retrieved from the providers are called records. Each record is specific to its provider, i. e., the record structure and data labels are not consistent across providers. While one provider API may call an attribute *name*, other providers may call it *label* or *title*. However, looking at the APIs' documentation the semantics of these attributes are identical. We manually harmonized the schema information of the different data providers by mapping the records into so-called resources. A resource represents an object that is either an event, organization, person, or place. When querying data providers, all re-

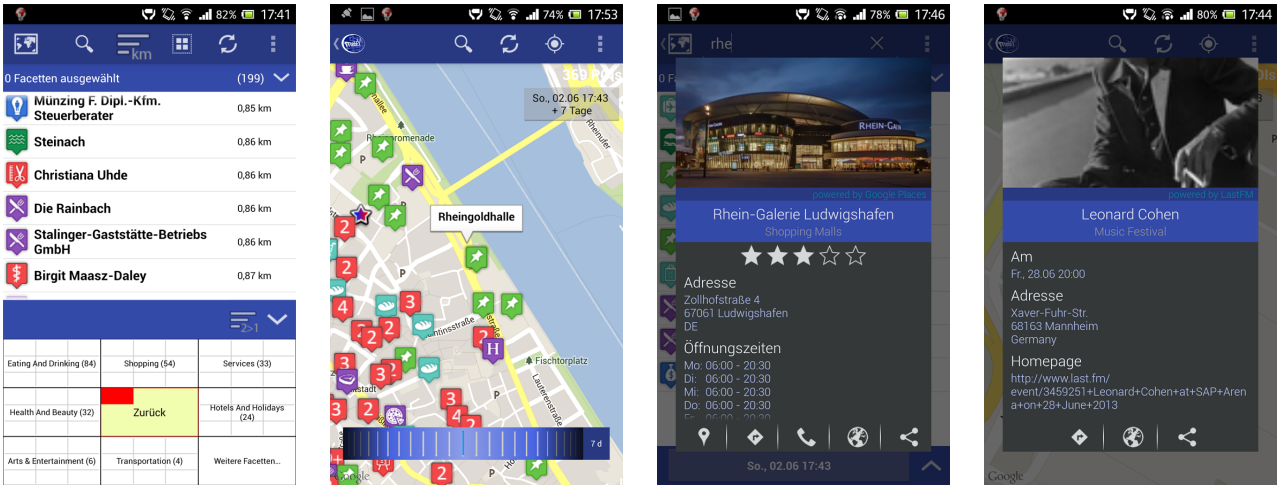

Figure 2: Screenshots of mobEx as it is available from Google Play. From left to right: Facet view, map view, details view with opening hours, details view with start time of an event.

trieved records can be mapped into such a resource for our mobEx application. This allows for an easy comparability of resources retrieved from different data providers. Thus, we ensure that the mapping respects the semantics of the properties, i. e., for any data source, the result of the mapping to the resource structure has the same meaning.

The set of resources retrieved from the different data providers for a given user query is denoted with $R = \{r_1, ..., r_n\}$. The set of entities that are obtained from these resources after applying the matching process is $E = \{e_1, ..., e_m\}$ with $m \leq n$. At the beginning of the matching process, we assume that each resource $r_i \in R$ resembles an entity $e_j \in E$ on its own. Thus, the entity e_j is represented by the values of r_i's attributes. In the course of the matching process, the resource r_i might be identified to be merged with a resource r_j, $i \neq j$. To this end, we merge the attribute values of r_j with r_i. Thus, after entity resolution some resources like r_i contain attribute values that previously belonged to one or even several other resources. In addition, we need to store the information which resources have been merged with each other. Here, we use a forest of merge trees [20], where each tree corresponds to a set of resources that have been merged in the matching process. Consequently, there is one tree for each entity. In addition, each resource appears only in exactly one merge tree. Details of the forest and how we use it in the matching process are described below.

MATCHING PROCESS

Our matching process for resolving the identity of the resources takes place in a highly parallel fashion. This is required as there is no guarantee that the data providers respond within a given time. Thus, one cannot assume to receive all resources at the same time. Furthermore, as our prior usability study shows [12], the users are not willing to wait a long time before receiving the first results on their mobile phones.

Below, we first present an (i) overview of our matching process. Subsequently, we present the (ii) details of the incremental entity resolution engine. This includes a discussion of the (iii) preconditions and (iv) weighting of attributes for comparing resources, (v) computing the similarity of resources, and (vi) merging of duplicate resources.

(i) Overview of the Matching Process

An overview of our matching process for an incremental entity resolution is depicted in Figure 3. As mentioned above, the entity resolution entirely takes place on the server. Thus, we start the description of the matching process with the server receiving a request from a mobile client and finish with the delivery of the entities that emerged from the retrieved resources to the mobile client. In the following description, the numbers correspond to the single steps of the matching process as shown in Figure 3.

Once the server obtains a user query from the client, there are two main steps in the matching process: First, querying for the records from the distributed data sources, which is shown in the upper half of Figure 3. Second, carrying out the incremental entity resolution itself, which is conducted by the steps depicted in the lower half of Figure 3. Both steps are executed in multiple threads, controlled by three central units, namely the Main-Thread, Entity-Manager-Thread, and Entity-Resolver-Thread. The Main-Thread is responsible for parallel querying the records from different data providers like Geonames and OpenPOI. The Entity-Resolver-Thread coordinates the parallel execution of the entity resolution. The Entity-Manager-Thread decouples these two steps such that they do not have to wait for each other.

When the Main-Thread receives a request from a mobile client (1), it starts and controls for each data provider a distinct Data-Provider-Thread (2). These Data-Provider-Threads retrieve the records such as restaurants, hospitals, and parks and parse them into the common schema, namely our resources. The Entity-Manager-Thread (3) administrates 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). As mentioned above, the Entity-Resolver-Thread handles the entity resolution. To this end, it tries to identify duplicate records such as "Cafe Vienna" and "Vienna Cafe". All resources newly arriving from the data providers to the Entity-Resolver-Thread are compared to the already received resources. This is conducted by the so-called Entity-Resolver-Worker threads (5), which actually carry out the comparisons of the records and merge the information from the duplicate records. The results of the Entity-Resolver-Worker threads are returned to the Entity-Manager-Thread (6), which in turn returns the resolved resources and the entities formed from these resources to the Main-Thread (7). In reply to the client request, the resolved entities are delivered by the Main-Thread to the mobile client (8) as soon as they become available, i. e., we do not wait until all records have arrived from all the different data providers. Therefore, an entity that has already been delivered to the mobile client is updated or even rectified in a later step of the incremental entity resolution process.

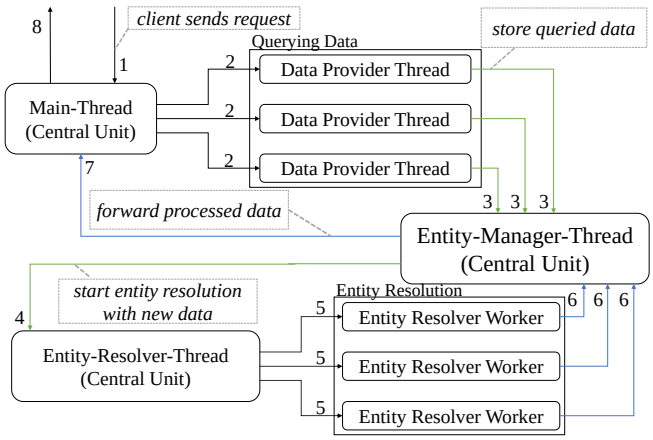

Figure 3: Data flow of processing a client request (1), incrementally matching the data (2-7), and sending the resulting entities back to the client (8). The numbers in brackets denote the execution order.

(ii) Incremental Entity Resolution Engine

After obtaining the data from the providers in the first step of the matching process, the actual entity resolution is conducted in the second step. As said above, the entity resolution is multi-threaded and thus carried out in a parallel fashion. Each time a data provider delivers a batch of resources, the Entity-Resolver-Thread starts a new Entity-Resolver-Worker thread. The Entity-Resolver-Worker threads actually carry out the entity resolution. Here, the newly arrived resources are mapped with each other as well as with all previously received resources (from other providers) for the same mobile client's query.

This multi-threaded data processing brings up several challenges: First, duplicate comparisons have to be avoided to safe computation time. Second, once a data provider delivers its batch of resources, their processing needs to be seamlessly integrated with the entity resolution over the already delivered and partially processed resources. In addition, given two resources r_1 and r_2, where r_1 is the older resource (i. e., it has been delivered to the entity resolution earlier) and r_2 is the newer resource. When r_1 and r_2 are identified to represent the same entity, then the attribute values of the newer resource r_2 are merged into those of r_1. Let us further assume that another resource r_3 occurs that should be merged with r_2. The problem now is that r_2 is no longer valid as merge target as it has previously been merged with r_1. Instead, r_3 now has to be merged into r_1. These challenges are addressed by keeping track of the merge process in a graph, more precisely a forest. A forest is a set of merge trees. Naturally, each merge tree represents an entity shown to the user of the mobile mobEx client. A tree is defined as directed acyclic graph $G = (V, E)$ where each node $v \in V$ represents a resource. Two resource nodes r_1 and r_2 are connected (i. e., it exists the edge $(r_1, r_2) \in E$), if the entity resolution process recognizes them as identical and thus merges them (see Figure 4, left). The problem of updates and deletions is handled in so far that the mobile client receives corrections to earlier results through updated edges in the forest. Such corrections are eventual updates or deletions of duplicate entities delivered in an earlier step of the resolution.

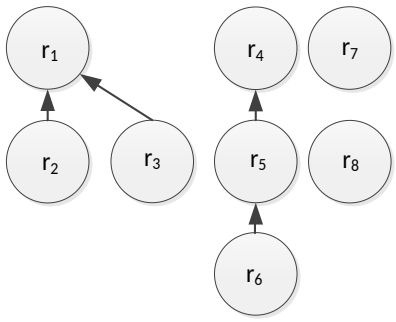

Figure 4: An example of a forest as result from matching over the resources r_1 to r_8. The current state of the forest is: The resources r_2 and r_3 have been merged into r_1. The resource r_6 has been merged into r_5, which in turn has been merged into r_4. The resources r_7 and r_8 are not merged (yet). Each tree in the forest represents an entity.

A further challenge 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 of the resources, this is easily ensured. In our threaded approach, we assure this by the Entity-Manager-Thread that receives the resources from providers as disjoint sets, i. e., batches of records. Thus, the Entity-Manager-Thread can start a separate Entity-Resolver-Worker thread on each set of records. As the Entity-Resolver-Worker threads share the forest in the main memory, they can indeed check all (newly arrived) resources against all other sets received earlier.

Please note, the (intermediate) results of the entity resolution become indeterministic, as there is no guarantee that resources will always be compared and merged in the same order. The order highly depends on the response time of the data provider APIs. As we do not store any entity resolution results, this indeterminism is a feature by design. It is more important that the mobile client receives (potentially) incomplete results as soon as possible rather than waiting for all data providers to answer and then merging the resources in a deterministic manner. This is best explained at an example of three resources r_1, r_2, and r_3 with the labels r_1.label ="Example 1", r_2.label ="Example Two", and r_3.label ="Example Three". The indices of the resources represent their 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 first matched (and merged) r_2 and r_3, the intermediate result would contain r_2 with label "Example Three". This example of indeterminism also shows that the delivered results only represent the data at a given point in time. Resolution results at a later point in time may result in updates of the attribute values of the corresponding entity. It is even possible that an entity that is already delivered to the mobile client has to be deleted from the list of entities. This is the case, e. g., when initially an entity consists of a single resource and that this resources is later merged with some other entity and thus no longer represents an entity of its own.

(iii) Preconditions for Comparing Resources

When querying the different data providers, we may very well receive a total of more than 1,000 records in major cities. Naive entity resolution, i. e., comparing all pairs of resources, is therefore out of the question since it would result in $\binom{n}{2}$ and thus $O(n^2)$ comparisons. This requires too much time for an on-the-fly matching. Instead, we cut down the number of comparisons by applying precondition heuristics which are based on the conditions documented in the literature (for a detailed discussion of the literature, we refer to the related work section). The general conditions of the work by Benjelloun [1] are very similar to the matching conditions in our approach. In detail, these conditions are:

- Pairwise matching and merging: We only compare two resources with each other and decide whether to merge them or not. However, in our case 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 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.

In addition to these general conditions, some more domain-specific conditions need to hold. In detail, we use the following further precondition heuristics:

- Transitivity: Given two resources, e. g., r_2 and r_3 as shown in 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. The common ancestor r_1 already represents that the resources r_2 and r_3 refer to the same entity.

- Type: Only resources of the same type are compared, i. e., we compare events with other events, but not with locations, persons, or organizations. This precondition is called "buckets" and is also motivated from Benjelloun et al. [1]. Buckets are created by selecting some attribute as the bucket label and binning the resources accordingly. Only pairs of resources that are in the same bucket are compared in the matching process.

- Physical location: The longer the physical distance between two resources, the less likely it is that they describe the same entity [18]. We account for that by calculating the distance between resources using the Haversine formula [23]. 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 geo-coordinates from a data provider). We define two postal addresses as similar, if the average of their Jaro-Winkler and Levenshtein similarity is larger than 0.75.

  The distance threshold of $500m$ was empirically determined (see Experiment 1 for details). 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 [23].

Type	Attribute	W	Description	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	latitude	*	latitude of the entity's location	49.48429
Double	longitude	*	longitude of the entity's location	8.46301
complex	address	2	addresses like birth place and 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/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	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 (* indicates that the weight is determined by a specific similarity function).

(iv) Attribute Weights for Comparing Resources

When comparing two resources for the purpose of merging, there are some attributes that are more important in determining whether a mapping should exist between two resources and there are less important attributes [15]. We account for these differences by assigning weights to the attributes as shown in Table 1. A higher weight represents that the feature is more distinctive or authoritative. The weights were assigned using information from the literature [1, 15, 6]. While the related work did not provide explicit numbers, there were hints which indicate that certain properties are more important or more distinctive. The actual weights used in our matching process were then empirically determined and assigned to the attributes. For example, it is legitimate to assign the URL the greatest weight, as URLs are by their nature a unique identifier. This means, if two resources of two different data sources have the same URL, then the likelihood that they are actually referring to the same entity is very high. A similar argument applies for the label (i. .e., name) and phone number of a person, organization, or place, which are designed to serve as an identifier. These rules are not without exception. For example, a postal address may not be distinctive, e. g., if there are multiple organizations located in the same office building. Details are discussed in our experiments below.

(v) Computing the Similarity of Resources

As stated above, we only compare pairs of resources for which the preconditions apply. In addition, certain attributes are more important than others when comparing resources. Thus, when actually computing the similarity of two resources (the so-called scoring), we account for that by using the weights of the different attributes as shown in Table 1. The scoring is then computed as follows: Let r_1 and r_2 be two resources, $C = \{x_1, \ldots, x_n\}$ the set of resource attributes with non-negative weights and that are present in both resources (i. e., they are not null or empty). Furthermore, let $compare(r_1.x, r_2.x)$ be a comparing function over the resource attribute x, except for the label and parts of postal addresses. Then, *compare* is simply defined as follows: $compare(r_1.x, r_2.x) = 1$ iff $r_1.x = r_2.x$ and 0 otherwise. For the label and postal address, 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). Based on this, we compute the similarity of two resources r_1 and r_2 as:

$$score(r_1, r_2) = \sum_{i=1}^{n} w_i \cdot compare(r_1.x_1, r_2.x_1)$$

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. In this work, we use $t = 0.7$ (this value showed a precision over 95% in our empirical studies). There are two exceptions to this procedure that are mentioned here because they further speed up the entity resolution process: First, there can be duplicate resources that a provider delivers (i. e., the provider's id for two resources are identical). Those resources are always considered to be identical. Second, resources (from different providers) with exactly the same label and description—given that the properties are non-empty in both resources—are also considered identical. This is determined by hashing and comparing the attribute values.

(vi) Merging Attributes of Duplicate Resources

Resources are merged if the score is above the given threshold. When merging two resources, in general the attribute values from the older resource take precedence over the newer resource. Thus, the older resource becomes the merge target, while the newer resource is the merge source (see also Figure 4). Exceptions to this are the label and the description, respectively. Here, we make the assumption that a longer text is better. A resource is considered old(er) if it has already been merged into some other resource. Thus, the oldest resource is the root of the merge tree. We refer to the merge target as r_{old} and the merge source as r_{new}. When merging two resources, only r_{old} will be changed. The attribute values of r_{new} remain untouched. An edge in the merge tree

pointing from r_{new} to r_{old} is added in order to indicate the direction of the merge.

Thus, for merging the two resources r_1 and r_2, we have to check whether one (or both) of them have already been merged before. If r_2 has already been merged into another resource, we will find the root node 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 merge of r_4,r_5, and r_6 depicted in Figure 4 resulted from merging r_6 into r_5 and subsequently merging r_5 (or r_6) into r_4.

EXPERIMENTAL EVALUATION

To evaluate our matching approach for incremental entity resolution, we conduct a series of different experiments. Each experiment considers a different aspect of our matching process. Below, we first briefly describe the experiments, before we present the results and their interpretations in the following sections. Please note that we have also conducted a field study of mobEx [12], where we collected qualitative feedback on the application's usability over a period of three weeks. Here, we focus on evaluating the novel resolution engine.

First, we determine the effect of the preconditions on the matching performance. We hypothesize that the application of 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. Goal of this experiment is to identify a good value for the distance measure threshold.

Second, we evaluate the quality of the matchings. To this end, we collect data from the five largest cities by population in the United States and Germany. We run queries to nine data providers that are currently implemented in mobEx. The queries cover the area of the selected cities. Subsequently, we apply the resolution process and manually check the results for correct merges, i. e., the merged resources represent the same entity (true positive), and incorrect merges, i. e., if the merged resources represent different entities (false positive). In addition, we conduct a qualitative analysis of recall by looking at examples that should have been merged, i. e., pairs of resources that were not merged although they represent the same entity (false negatives). Based on the examples, we identify general cases where our scoring function and thus the matching did not recognize duplicates.

Third, mobile users will not be willing to wait a long time until they receive results to their queries [12]. Thus, we determine how long it takes to resolve a specific number of resources. In particular, we are interested in analyzing the relationship between the run time needed for processing a resource and the total number of resources being concurrently present in the entity resolution engine.

Fourth, we investigate how long it takes until entities arrive at the mobile client once the resolution process is started. Particularly, we are interested in the ratio between the percentage of delivered entities and resolved resources and its evolution over time.

For all experiments, we only regard the time needed by our incremental entity resolution process. The time required by the APIs of the data providers to return the records is ignored. The motivation for this procedure is: First, any application that uses one of the providers' APIs would face these latency times. Thus, it is not inherent to the computational time needed by our incremental entity resolution. Second, the response times of the APIs differ quite a lot depending 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, i. e., more than 2.5 minutes. Thus, a fair evaluation can only be conducted by controlling the factor of varying API response times.

Experiment 1: Determining the Distance Treshold

We queried all nine data providers implemented in the mobEx server in the five largest cities in the US and Germany. These 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 center of each city with a radius of 3.1km. Subsequently, we carry out our entity resolution on the received resources and iteratively apply the following distance thresholds: 300m, 500m, 1,000m, 1,500m, 2,000m, 2,500m, 3,000m, and infinite distance. Table 2 shows the results averaged over the then cities.

Distance	Avg. Matches	Avg. Savings (in %)
300	103.11	52.69
500	103.11	50.28
1,000	102.89	41.06
1,500	103.11	28.37
2,000	103.11	19.13
2,500	103.11	12.78
3,000	103.11	7.69
∞	103.22	0

Table 2: Avg. number of matches and avg. saved comparisons for different distance thresholds.

As the results show, the number of matches do not vary a lot while there is an increase in the number of comparisons that can be saved when considering a distance threshold of $500m$. 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 runtime, allowing for faster results at the cost of (possibly) remaining duplicates.

Experiment 2: Evaluating the Quality of the Matching
Regarding the assessment of correctly merging the records retrieved from the different data sources, we aggregate the records over all cities. Applying the entity resolution process, we obtain 9,834 entities. For computing those entities, 1,122 merges took place. Out of the 1,122 merges, we manually identified that 1,068 were correct, i. e. the merged resources actually described the same entity. In 54 cases, we found incorrect merges. This gives us an overall precision of 95.19% as shown in Table 3, which is very high. However, some merges were actually fairly easy as they were duplicates from a provider where resources share the same identifier (short: id). We consider a match "hard", if it is not an obvious duplicate. Table 3 shows the distribution and precision of our approach for three assumptions on what an "obvious" duplicate is and what a "hard" duplicate is. These three cases are:

a) There are no obvious duplicates, i. e., all matches are considered hard. Thus, all the matches that are based on identical id (column ID in Table 3) or label and description (column L&D) as well as all 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. These matches are always 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) On top of b), we consider duplicates as obvious 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.

	M.	TP	FP	ID	L&D	hard	P
a)	1,122	1,068	54			1,122	95.19
b)	566	512	54	556		566	90.46
c)	251	197	54	871		251	78.49

Table 3: Results of the matching process aggregated over ten major cities with in total 9,834 entities (after matching). Column labels: 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

Several potential challenges emerged in the course of evaluating our matching approach: First, there is a general problem when there is only little information available about the resources. If some of these sparse attribute values vary, then matching is close to impossible. However, as said in the introduction, we designed the incremental entity resolution to favour precision over recall. Thus, the remaining duplicate resources (i. e., resources that should be merged but were missed) may be more acceptable than potentially merging too many records (false positives).

Second, 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 identical, but only the address was different. This might be a problem as it causes relevant merges getting lost. Actually, the majority of 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 a few cases where this actually resulted in missing some duplicates (on average around 10 obvious duplicates per city). This is of course not desirable but probably acceptable. Nevertheless, this presents a starting point for future improvements of our approach.

Third, in some cases, it seems that obvious duplicates exist. At a closer look, there are often slight variations in the address and the phone number. This raises the question whether the phone number is a good attribute in the sense that it serves as a good identifier for a resource. This question cannot easily be answered as larger businesses or organizations often have many phone numbers while small businesses such as restaurants often have one phone number. Keeping in mind that the approach should be generic, one must ask whether the introduction of special cases will decrease the performance of the entity resolution or even hinders its universal applicability.

Please note, we do not evaluate for recall on a quantitative basis. This would include missing merges as well as correct non-merges, i. e., resources that were not merged and should not be merged as they represent different entities (true negatives). In order to do so, one would have to compare every resource with all other resources which resulted in $\binom{n}{2}$ comparisons for n resources. This becomes practically infeasible, even for small datasets.

Experiment 3: Runtime of the Entity Resolution
We carried out a run time analysis of our incremental matching approach by querying all of our nine 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. Figure 5 shows the time required for processing a resource in the entity resolution engine in relation to the number of resources that existed in the process at that time. It can be seen that the processing time for each resource is higher when more resources are present in the entity resolution process on the server. Still, the resolution process is fairly quick for the amounts of data we faced. At an average of 746.13 resources being present in the resolution process, it took on average $0.046s$ to resolve one resource against all other resources. This corresponds to a processing speed of 204.67 resources per second. Please note, all results have been computed by averaging ten different runs.

The correlation coefficient $r = 0.72$ shows that there is a strong correlation between the run time and the number of resources. Around 72% of the increase in run time can

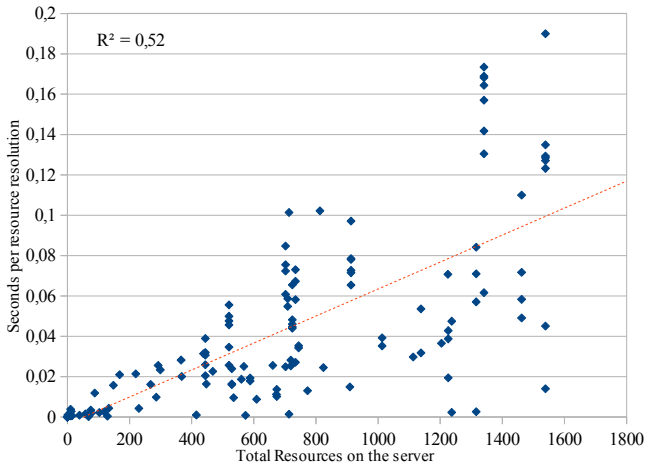

Figure 5: Run time per resource on the server. The red dotted line shows a linear regression curve with a correlation coefficient $r = 0.7208$ (coefficient of determination $r^2 = 0.52$).

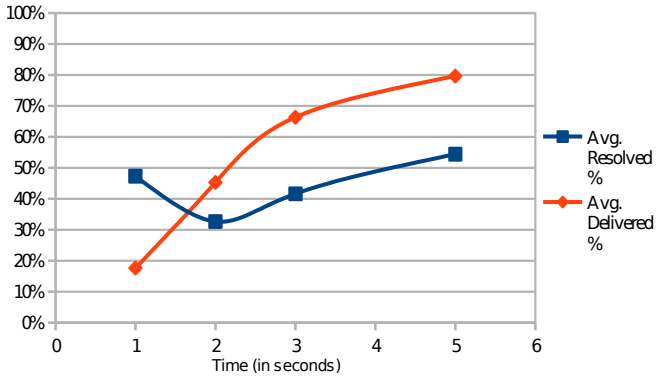

Figure 6: Percentage of resources received (red) vs. percentage of completely resolved resources among the resources delivered to the client (blue). The point in time when the client received 100% of the resources in a resolved state is not shown as it depends too strongly on the time when the last data provider answers the query.

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. In addition, the coefficient of determination r^2 of the linear regression in Figure 5 shows that 52% of the variations in the run time of our resolution approach can be explained by the number of resources.

Experiment 4: # of Resources Delivered to Mobile Client

In this experiment, we report a measure that we call *resolved ratio* (rr). The rr represents the number of resources that the matching server has received and that have undergone the resolution process in proportion to the total number of resources the mobile client has already received. This number should not be seen as exact figure, but as lower-bound estimate. Furthermore, as said above, we do not include the response times from the providers in our calculations. They indirectly affect the resolution process as resources that arrive at the server sooner than others are started early to be processed by the entity resolution. But there is no guarantee that threads finish in the order they were started. It may very well happen that a thread that is started later than another thread finishes sooner as the batch of resources delivered from the provider is smaller.

Analyzing the actual amount of resolved resources that the client has received at a given time is 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 the client has received (out of all resources delivered to the matching server for a specific user query) and the percentage of the received resources that have undergone resolution so far. In these tests, we retrieved on average a total of 959.2 resources from the nine data providers (computed over the five largest cities in the US and Germany). As can be seen from Figure 6, after one second already almost 20% of the resources could be delivered to the mobile client and almost 50% of the resources are resolved. Subsequently, the percentage of resolved resources goes down as more resources from the data providers arrive at the entity resolution server. But also more resources can be delivered to the mobile client. Within 5s almost 80% of all resources are delivered to the client and thus are at least partially resolved. At the same time, around 54% of the received resources have been fully resolved.

CONCLUSION

We have presented the design and evaluation of an entity resolution approach that is very precise while at the same time does not impose a long response time to the mobile users. To this end, our incremental entity resolution engine processes data records as soon as the first data provider respond to a user's query. As we do not have all resources at hand when the resolution process is started but users expect results in a timely manner, we start propagating partially resolved resources to the mobile client and sending updates later on. Thus, while the resolution process receives more and more data from the providers as we go along, the entities shown on the user's mobile client gradually become more complete. The weights of the scoring function are based on empirical observations and results found in the literature. Automated techniques may be used to learn an optimal weighting and replace the current solution in the future.

Acknowledgement

We thank Florian Rang and Katja Beer of the telegate Media AG for their valuable input and support. This work 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. 2009. Swoosh: a generic approach to entity resolution. *VLDB J.* 18, 1 (2009), 255–276.
2. Indrajit Bhattacharya and Lise Getoor. 2005. A latent dirichlet model for unsupervised entity resolution. In *Int. Conf. on Data Mining*. IEEE.
3. Indrajit Bhattacharya, Lise Getoor, and Louis Licamele. 2006. Query-time entity resolution. In *SIGKDD*. ACM, 529–534.
4. Mustafa Bilgic, Louis Licamele, Lise Getoor, and Ben Shneiderman. 2006. D-dupe: An interactive tool for entity resolution in social networks. In *Symposium on Visual Analytics Science And Technology*. IEEE, 43–50.
5. Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. 2003. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*. ACM, 313–324.
6. William W Cohen. 2000. Data integration using similarity joins and a word-based information representation language. *TOIS* 18, 3 (2000), 288–321.
7. Nilesh N. Dalvi, Marian Olteanu, Manish Raghavan, and Philip Bohannon. 2014. Deduplicating a places database. In *Int. World Wide Web Conf.* ACM, 409–418.
8. Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. 2014. Incremental Record Linkage. *PVLDB* 7, 9 (2014), 697–708.
9. Hyunmo Kang, Lise Getoor, Ben Shneiderman, Mustafa Bilgic, and Louis Licamele. 2008. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *Visualization and Computer Graphics* 14, 5 (2008), 999–1014.
10. Hyunmo Kang, Vivek Sehgal, and Lise Getoor. 2007. GeoDDupe: a novel interface for interactive entity resolution in geospatial data. In *Information Visualization*. IEEE.
11. Alexander Kleinen, Ansgar Scherp, and Steffen Staab. 2014. Interactive faceted search and exploration of open social media data on a touchscreen mobile phone. *Multimedia Tools Appl.* 71, 1 (2014), 39–60.
12. Florian Knip, Christian Bikar, Bernd Pfister, Bernd Opitz, Timo Sztyler, Michael Jess, and Ansgar Scherp. 2014. A field study on the usability of a nearby search app for finding and exploring places and events. In *Mobile and Ubiquitous Multimedia*. ACM, 123–132.
13. Xin Li, Paul Morie, and Dan Roth. 2004. Identification and tracing of ambiguous names: Discriminative and generative approaches. In *National Conf. on Artificial Intelligence*. AAAI/MIT, 419–424.
14. Pankaj Malhotra, Puneet Agarwal, and Gautam Shroff. 2014. Incremental entity fusion from linked documents. In *17th International Conference on Information Fusion, FUSION 2014, Salamanca, Spain, July 7-10, 2014*. IEEE, 1–8.
15. Martin Michalowski, Jose Luis Ambite, Snehal Thakkar, Rattapoom Tuchinda, Craig A Knoblock, and Steve Minton. 2004. Retrieving and semantically integrating heterogeneous data from the web. *Intelligent Systems, IEEE* 19, 3 (2004), 72–79.
16. Alvaro E. Monge and Charles Elkan. 1997. An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records. In *DMKD*. 0.
17. Alvaro E Monge, Charles Elkan, and others. 1996. The Field Matching Problem: Algorithms and Applications.. In *SIGKDD*. ACM, 267–270.
18. Axel-Cyrille Ngonga Ngomo and Sören Auer. 2011. LIMES: a time-efficient approach for large-scale link discovery on the web of data. In *AAAI*. 2312–2317.
19. Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart Russell, and Ilya Shpitser. 2002. Identity uncertainty and citation matching. In *Advances in neural information processing systems*. MIT Press, 1401–1408.
20. Banda Ramadan and Peter Christen. 2014. Forest-Based Dynamic Sorted Neighborhood Indexing for Real-Time Entity Resolution. In *Int. Conf. on Information and Knowledge Management*. ACM, 1787–1790.
21. Banda Ramadan and Peter Christen. 2015. Unsupervised Blocking Key Selection for Real-Time Entity Resolution. In *Advances in Knowledge Discovery and Data Mining*. Springer, 574–585.
22. Mark Schneider, Ansgar Scherp, and Jochen Hunz. 2013. A comparative user study of faceted search in large data hierarchies on mobile devices. In *Mobile and Ubiquitous Multimedia*. ACM.
23. BP Shumaker and RW Sinnott. 1984. Astronomical computing: 1. Computing under the open sky. 2. Virtues of the haversine. *Sky and telescope* 68 (1984), 158–159.
24. Mohsen Taheriyan, Craig A. Knoblock, Pedro A. Szekely, and José Luis Ambite. 2013. A Graph-Based Approach to Learn Semantic Descriptions of Data Sources. In *Int. Semantic Web Conf.* Springer, 607–623.
25. Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. 2009. Silk-A Link Discovery Framework for the Web of Data. In *Linked Data on the Web*. CEUR.

26. Michael J. Welch, Aamod Sane, and Chris Drome. 2012. Fast and Accurate Incremental Entity Resolution Relative to an Entity Knowledge Base. In *CIKM*. ACM, 2667–2670.

27. Steven Euijong Whang and Hector Garcia-Molina. 2014. Incremental Entity Resolution on Rules and Data. *The VLDB Journal* 23, 1 (Feb. 2014), 77–102.

28. Yang Yang, Yizhou Sun, Jie Tang, Bo Ma, and Juanzi Li. 2015. Entity Matching Across Heterogeneous Sources. In *SIGKDD*. ACM, 1395–1404.