YumiInt – A Deep Web Integration System for Local Search Engines for Geo-referenced Objects

Eduard Dragut[#1], B. P. Beirne[*2], A. Neyestani[*3], B. Atassi[*4], Clement Yu[*5], Bhaskar DasGupta[*6], Meng Weiyi[$7]

[#] *Cyber Center, Purdue University*
[1]edragut@purdue.edu

[*] *Computer Science Department, University of Illinois at Chicago*
[2]bbeirn2@uic.edu, [3]aneyes2@uic.edu, [4]abader1@uic.edu, [5]yu@cs.uic.edu, [6]dasgupta@cs.uic.edu

[$] *Computer Science Department, Binghamton University*
[7]meng@cs.binghamton.edu

***Abstract*— We present YumiInt a deep Web integration system for local search engines for Geo-referenced objects. YumiInt consists of two systems: YumiDev and YumiMeta. YumiDev is an off-line integration system that builds the key components (e.g., query translation and entity resolution) of YumiMeta. YumiMeta is the Web application to which users post queries. It translates queries to multiple sources and gets back aggregated lists of results. We present the two systems in this paper.**

I. Introduction

In this paper, local search is the use of specialized search engines that allow posting geographically constrained queries against data bases of local business listings. It amounts to 20% of Google queries. Local search engines (LSE) contain erroneous, outdated, contradictory and incomplete data. For example, the restaurant The Black Sheep is listed among "Top Chicago restaurants" on Metromix.com, whereas it is listed as closed on Yelp.com. These issues can be alleviated if we aggregate the information about the same entities from multiple independent and uncooperative sources. We create a system, YumiInt, that connects to, gathers and integrates data from multiple LSEs such as Yahoo, YellowPages, and CitySearch.

Suppose the query Q = (`Cuisine` = "Ethiopian"; `Neighborhood` = "Uptown") is posted to Zagat.com search engine. Many relevant restaurants are not returned by Zagat, such as Demera. This restaurant however is returned and ranked 1^{th} by many other search engines for the same query. Hence, by combining the results from multiple search engines, we provide users with a more complete set of restaurants. Furthermore, the address of the restaurant Sea Shell Fish and Chicken is wrong in Yelp (i.e., 6124 S Ashland Ave.), but we can nonetheless identify the correct address (i.e., 6940 S Ashland Ave.) by considering the addresses of the restaurant in other search engines. We propose YumiInt, an integration system for LSEs for geo-referenced objects, which aims to provide more complete and correct data to users. YumiInt consists of two applications: YumiMeta and YumiDev. YumiMeta is the web site application, which the users interact with. YumiDev is the off-line application that is used to develop YumMeta. When YumiMeta receives a query submitted by a user, it forwards the query to the LSEs, which process the query and return the results to YumiMeta. YumiMeta merges and re-ranks the results into a unified list. A prototype version of YumiMeta is already deployed online at `www.yumi-meta.com`. YumiMeta is developed using ASP.NET and Microsoft C#. YumiDev is developed in C#.

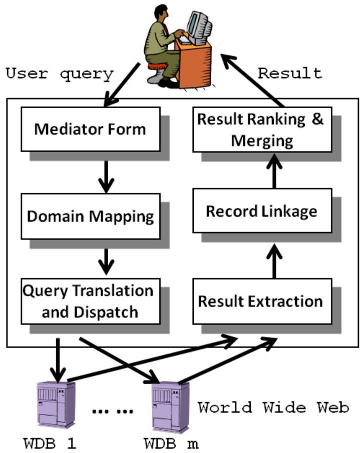


Fig. 1. The architecture of YumiMetaWeb.

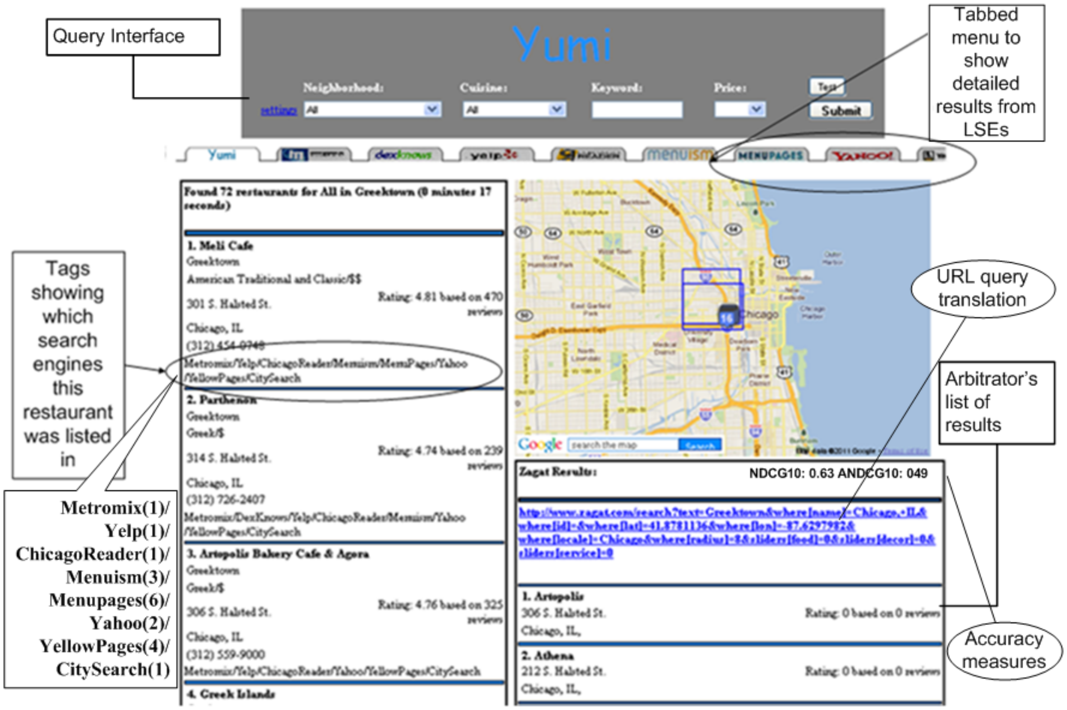
II. YumiMeta

YumiMeta currently connects to 9 search engines: ChicagoReader.com, CitySearch.com, DexKnows.com, MenuIsm.com, Yelp.com, MenuPages.com, Metromix.com, local.Yahoo.com, YellowPages.com.

A. Architecture

YumiMeta requires the implementation of several key components (Fig. 1). The first component is a *mediator form* [2], which provides a uniform access to the data sources in a given domain. A user formulates the desired query on it and the query is automatically translated conforming to the query interfaces of the underlying sources by the *query translation and dispatch* component. The *result extraction* component extracts the data returned by individual sources [6]. The *record linkage* component identifies the records across lists referring to the same entity [4]. This information is used to merge the

978-1-4673-4910-9/13/$31.00 © 2013 IEEE



Found 72 restaurants for All in Greektown (0 minutes 17 seconds)

1. Meli Cafe
Greektown
American Traditional and Classic/$$
301 S. Halsted St. Rating: 4.81 based on 470 reviews
Chicago, IL
(312) 454-0748
Metromix/Yelp/ChicagoReader/Menuism/MenuPages/Yahoo/YellowPages/CitySearch

2. Parthenon
Greektown
Greek/$
314 S. Halsted St. Rating: 4.74 based on 239 reviews
Chicago, IL
(312) 726-2407
Metromix/DexKnows/Yelp/ChicagoReader/Menuism/Yahoo/YellowPages/CitySearch

3. Artopolis Bakery Cafe & Agora
Greektown
Greek/$
306 S. Halsted St. Rating: 4.76 based on 325 reviews
Chicago, IL
(312) 559-9000
Metromix/Yelp/ChicagoReader/Yahoo/YellowPages/CitySearch

4. Greek Islands

Zagat Results: NDCG10: 0.63 ANDCG10: 049

http://www.zagat.com/search?text=Greektown&where[name]=Chicago,+IL&where[id]=&where[lat]=41.878113&where[lon]=-87.629798&where[locale]=Chicago&where[radius]=8&sliders[food]=0&sliders[decor]=0&sliders[service]=0

1. Artopolis
306 S. Halsted St. Rating: 0 based on 0 reviews
Chicago, IL,

2. Athena
212 S. Halsted St. Rating: 0 based on 0 reviews
Chicago, IL,

Fig. 2. YumiMeta Front End, i.e., the query interface and the result page.

lists of results into a single merged list in descending order of each result's desirability to the user.

In this demo we focus on the mediator form, query translation, record linkage and result ranking components, with particular emphasis on their use in the local services domain.

B. Front End

The two main components of the front-end of a search engine are the query interface and the result page. The query interface is used to post queries and the result page shows the outcome of queries. The front-end of YumiMeta is developed so that the reasoning behind a returned list of results for a query is transparent to a user. Fig. 2 shows YumiMeta's front-end. The top part is the query interface. The query interface has the fields Neighborhood, Cuisine, Price and Keyword. The values of the field `cuisine` are often drawn from an ontology of culinary arts (e.g., Italian, Middle Eastern). Many search engines organize the values of the field `neighborhoods` in a hierarchy. For example, Metromix.com specifies the Chicago area as a composition of the following regions: Downtown, North, Near North, etc. and within each region, there are several neighborhoods. The attribute `Prices` is on a scale from 1 to 5 (usually denoted by "$" sign). Once the desired query keywords are filled in, a user presses the submit button. The button Test will be explained shortly. YumiMeta returns a ranked list of results. It is difficult for users to assess the performance of a search engine for a query in general if all they are presented with is a list of results. If the results have obvious flaws they know some error has occurred, but they would have a hard time to pinpoint the component of the search engine the erroneous data came from. To help both an ordinary user and a developer the result page is organized in four areas: the LSEs area (top), the ranked list of results area (left), the map area (right) and the *arbitrator* area (right lower corner). We describe them in turn.

Local Search Engines Area: The results returned by an LSE for a query can be accessed via a tabbed menu.

Ranked List of Results Area: This area displays the aggregated list of results for a query. The records in the list have information such as the name, address, cuisine, average user rating and total number of reviews. For each record, we display the set of LSEs that returned the record along with the position of the record in the returned list of each of the LSEs (Fig. 2, lower left corner). This is an important piece of information because, in conjunction with the tabbed menu, it helps to quickly locate records in the lists of results returned by the LSEs. This eases the debugging process for ranking, record matching and fusion. For instance, in ranking we can browse the results and see why a record is ranked higher than another. Errors in record matching and fusion can also be easily investigated.

Map Area: Google map is included in the front-end, so users can browse the result on the map. The map view can be changed into an advanced view that shows how the location specified in a query is translated to the LSEs by YumiMeta. Fig. 3 illustrates the advanced view. This shows the estimated bounded box for the neighborhood query and the set of neighborhoods it intersects in the LSEs. We give additional details about query translation in Section III-A.

Arbitrator Area: YumiMeta sends each query to 9 LSEs and to a 10th LSE, called *arbitrator*. The arbitrator area shows the query result of the arbitrator. The results of the arbitrator are kept separate as gold standard for evaluation purposes. We will explain how they are used in Section II-C.

Local Search Engine View: When the user selects one of the tabs in the tabbed menu, the LSE view is displayed.

This view shows a table on the right hand side containing the hyperlinked queries generated for the selected LSE along with the raw results for the engine. The results are shown on the left hand side after YumiMeta has filtered and sorted them. A map is also included. It shows the location of the top-20 restaurants. In the advanced view, the boundaries of the neighborhoods used in the queries are drawn. They are shown to assess how well the neighborhood query processing component is matching neighborhoods across LSEs (Section II-C).

At the top of the result list from an LSE, we include the URLs generated for a query. For example, in Fig. 2 we show the URL generated to translate the query "Neighborhood=Greektown" to the search engine Zagat. This is a useful piece of information because it shows (1) how YumiMeta communicates with an LSE and (2) the way the query is translated to the LSEs. A developer can thus easily notice whether an error has occurred in the query translation.

C. Evaluation Component

We employ two methods to evaluate YumiMeta: user-based and arbitrator-based. The latter ensures that our evaluation is unbiased. We only discuss the latter because it can be customized and automated. The former requires extensive human effort. For each search engine under evaluation, the top-k (k = 5, 10) retrieved results are considered.

The *arbitrator-based evaluation* method is motivated by the lack of large scale gold standards. The arbitrator is a third party, which is generally accepted as an authority in the domain of discourse. For example, Zagat is regarded as an authority in the restaurant rating industry in U.S. The assumption is that a highly ranked restaurant by Zagat is very likely to be a good restaurant. The arbitrator is used to estimate the relevance of the records in a list returned by a search algorithm. Let L_a and L_{se} be the top-k lists of results returned by the arbitrator and by an LSE, respectively, for query Q. We compare L_{se} against L_a using the *normalized discounted cumulative gain* (NDCG) [5], a popular measure for search performance evaluation. We compute the NDCG for L_{se} using L_a. Hence, when comparing two lists of results for a query, the one with the larger NDCG$_k$ is considered to perform better at rank k. If lists containing different objects are returned, NDCG may not be adequate for evaluating search results. Thus, we propose an alternative definition to NDCG, called ANDCG (arbitrator NDCG). Its definition is omitted due space constraints. Both measures are shown (see Figure 2). For this evaluation we use a batch of 1000 queries. Each query is submitted to YumiMeta, the arbitrator and each of the 9 search engines. We collect the results and compute the average NDCG and ANDCG for YumiMeta and each of the 9 search engines.

III. YumiDev - under the hood

In this section we give a brief overview of the algorithmic issues in the query processing, record matching and ranking components. They all are formally laid out elsewhere [3].

A. Neighborhood Query Processing

Suppose that a user posts the query (`Price` = "$$$"; `Neighborhood` = "Warehouse District"). YumiMeta needs to forward the query to the search engines Zagat, Yelp and Metromix. While Metromix understands `Neighborhood` = "Warehouse District", Zagat and Yelp do not, because their Neighborhood fields do not have a value directly corresponding to "Warehouse District". Thus, while we expect neighborhood divisions of cities to follow some standards, this is not true in the real world—different search engines have different neighborhood hierarchies and sets of neighborhoods for a (large) city.

The processing of a neighborhood query has the following steps. First, from the set of all neighborhood hierarchies of the LSEs we select the tallest one. This becomes the neighborhood hierarchy used in YumiMeta, called *target hierarchy*, and it is shown in the field Neighborhoods. Then, for each neighborhood h in the target hierarchy we determine the set of neighborhoods in the hierarchy of each LSE that "best" matches h. This problem is NP-hard and we developed an approximation algorithm. To compute the mapping, the area of each neighborhood is approximated by a bounding box. Since h is approximately translated (e.g., a neighborhood in YumiMeta may correspond, but not exactly equivalent, to a set of neighborhoods in an LSE), the LSEs may return results that are not in h. Thus, the set of local services returned by each LSE must be tested for inclusion in h. We compute a *variable lookup grid* over the target hierarchy and use it to quickly decide if a local service lies inside h.

B. Local Service Resolution (Record Linkage)

To accurately merge and rank the lists of results returned by LSEs for a query, the records referring to the same business need to be recognized across business listings. Our local services matching algorithm is as follows. We adopt a supervised learning solution. A business entity has several attributes: name, address, zip code, phone, user rating, user review, cuisine, etc. First, we determine the attributes that impact the judgment that two records are linked (e.g., phone is used; cuisine is not). Second, we define a similarity measure per attribute (e.g., string edit distance for name). Third, we obtain a training sample by posting a number of random queries via YumiMeta to the LSEs. We manually label the pairs of matching records and learn a *decision tree*. Finally, IF-THEN rules are extracted from the decision tree.

C. Ranking

Ranking/merging is performed after record linkage is completed. We implemented four ranking algorithms: (1) a Borda-based algorithm, (2) an algorithm that employs users' ratings and reviews in addition to Borda, called BordaR, (3) an algorithm that uses Borda's algorithm together with users' ratings and users' query criteria, called BordaUCR (UCR - user criteria and rating) and (4) a weighted version of the BordaUCR. The first is an adaptation of the Borda-fuse method [1], while the other three are our proposed algorithms.

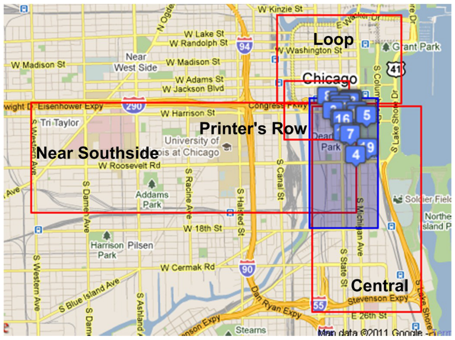


Fig. 3. The set of neighborhoods in Yelp intersecting with South Loop.

The Borda-fuse method is a voting based technique [1], where LSEs act as voters and retrieved results are treated as candidates in an election. We briefly describe the main ideas of our ranking algorithms. They are detailed in [3]. The algorithm BordaR first reorders the lists of the results of LSEs by user ratings. If two restaurants happen to have the same ratings then it uses the number of reviews to break the tie. Then it merges the reordered lists using the Borda-fuse method. The BordaUCR uses the query conditions to further refine the ranking. Given a query Q, the list of results of an LSE is classified into three classes: (1) Q_{all} is the set of objects that satisfy Q, (2) Q_{part} is the set of objects that satisfy at least one of the conditions of Q, but not all of them, and (3) Q_{not} is the set of objects that do not satisfy any of the conditions of Q. Then, the objects returned by an LSE are re-ranked such that Q_{all} is ahead of Q_{part}, which is ahead of Q_{not}. In each of Q_{all}, Q_{part} and Q_{not} BordaR is used. In the weighted BordaUCR, each search engine is associated with a weight that reflects the overall quality or performance of the search engine. We devise two methods to obtain the weights. In one solution the weight of LSE is computed based on its average ANDCG scores obtained from a batch of queries (see Section II-C). In the other we use a supervised learning technique to estimate the weights using the *method of least squares*.

IV. Demo Plan

Our demo plan is divided into two parts: one for illustrating the construction of the key components of YumiMeta via YumiDev and the other demonstrating YumiMeta online.

YumiDev Demo Plan: For those visitors who are interested in grasping the methodology of building a functional deep Web integration system, we provide the following scenario. We walk the visitors through the following tasks: (1) the creation of the neighborhood mapping, (2) the development of the entity resolution function and (3) the estimation of the weights for the weighted BordaUCR ranking algorithm. We then illustrate the seamless inclusion of the outcomes of these tasks in YumiMeta to be used online. In (1), we use the neighborhood hierarchies of the 10 LSEs YumiMeta connects to. We will use visual means to illustrate how neighborhoods in the target hierarchy are mapped in the hierarchies of the LSEs. For example, in Figure 3 the target neighborhood South Loop intersects four neighborhoods in the search engine Yelp: Near South Side, Loop, Printer's Row and Central. But out of these neighborhoods in Yelp, Loop and Central are the desired matching for South Loop. The mapping is plugged into YumiMeta. In (2), a number of queries are posted via YumiMeta and the records from each of the LSEs are saved in a local database. Then, a friendly user interface allows the user to manually map the records from two LSEs. The manually created gold standard is used to train the decision tree, which is then automatically turned into IF-THEN rules used by YumiMeta. In (3), we show how we obtain the weights for the ranking algorithm, which are then plugged into YumiMeta's ranking algorithm.

YumiMeta Demo Plan: We envision two demo plans here based on the audience's interest. If the visitor has already been introduced to YumiDev, then we can show her how the components we developed with YumiDev are seamlessly plugged into YumiMeta. We will also show how each component affects the functionality of YumiMeta. If the visitor is only interested in YumiMeta then will start with the default settings and illustrate how YumiMeta runs online (Fig. 2). We will present the steps YumiMeta undergoes to execute a query, i.e., the query translation, local services resolution and the merging/ranking. For instance, we will use the advanced map view to illustrate how a neighborhood query is mapped from the target hierarchy to an LSE. There are multiple other options that the visitor could explore: e.g., changing the arbitrator or the list of LSEs. In the former, for instance, we will show the visitor the performance of the LSEs and YumiMeta according to the selected arbitrator. We will prepare a set of queries that will show the improvement of YumiMeta in data quality over the component search engines. We will identify records that are known to be wrong in some LSEs, but which are corrected via the record linkage and fusion by YumiMeta. We will also use another set of queries to do a comparison between YumiMeta's ranking and those of the LSEs. We will encourage an interactive and inquisitive demo session for each visitor.

To the best of our knowledge, there is no other published demo that shows the same capabilities as those of our demo: e.g., metasearching over local search engines for geo-referenced objects, searching by specific geographic areas such as neighborhoods, integrating local services resolution (i.e., record linkage) and ranking. In addition, in this demonstration we do not only show the end result (i.e., YumiMeta metasearch engine), but also demonstrate how to build such a system in practice (i.e., YumiDev).

References

[1] J. A. Aslam and M. Montague. Models for metasearch. In *SIGIR*, pages 276–284, 2001.

[2] E. Dragut, W. Wu, P. Sistla, C. Yu, and W. Meng. Merging source query interfaces on web databases. In *ICDE*, 2006.

[3] E. C. Dragut, B. P. Beirne, B. DasGupta, A. Neyestani, B. Atassi, C. Yu, and W. Meng. Integrating web query results from local search engines for geo-referenced objects. In *under review*, 2012.

[4] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1), 2007.

[5] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *TOIS*, 20:422–446, 2002.

[6] W. Liu, X. Meng, and W. Meng. ViDE: A Vision-Based Approach for Deep Web Data Extraction. *TKDE*, 22, 2010.