Merging Query Results From Local Search Engines for Georeferenced Objects

EDUARD C. DRAGUT, Temple University
BHASKAR DASGUPTA, BRIAN P. BEIRNE, ALI NEYESTANI, BADR ATASSI, and CLEMENT YU, University of Illinois at Chicago
WEIYI MENG, Binghamton University

The emergence of numerous online sources about local services presents a need for more automatic yet accurate data integration techniques. Local services are georeferenced objects and can be queried by their locations on a map, for instance, neighborhoods. Typical local service queries (e.g., “French Restaurant in The Loop”) include not only information about “what” (“French Restaurant”) a user is searching for (such as cuisine) but also “where” information, such as neighborhood (“The Loop”). In this article, we address three key problems: query translation, result merging and ranking. Most local search engines provide a (hierarchical) organization of (large) cities into neighborhoods. A neighborhood in one local search engine may correspond to sets of neighborhoods in other local search engines. These make the query translation challenging. To provide an integrated access to the query results returned by the local search engines, we need to combine the results into a single list of results.

Our contributions include: (1) An integration algorithm for neighborhoods. (2) A very effective business listing resolution algorithm. (3) A ranking algorithm that takes into consideration the user criteria, user ratings and rankings. We have created a prototype system, Yumi, over local search engines in the restaurant domain. The restaurant domain is a representative case study for the local services. We conducted a comprehensive experimental study to evaluate Yumi. A prototype version of Yumi is available online.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models, Search process*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Web database integration, geospatial query processing, ranking

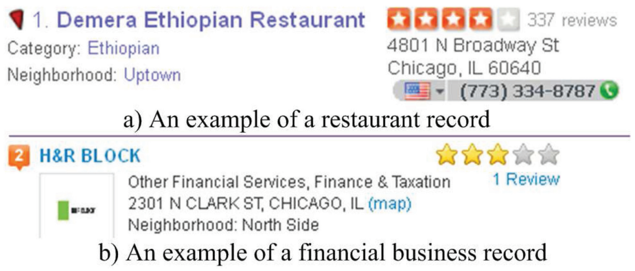
ACM Reference Format:
Eduard C. Dragut, Bhaskar DasGupta, Brian P. Beirne, Ali Neyestani, Badr Atassi, Clement Yu, and Weiyi Meng. 2014. Merging query results from local search engines for georeferenced objects. ACM Trans. Web 8, 4, Article 20 (October 2014), 29 pages.
DOI: http://dx.doi.org/10.1145/2656344

1. INTRODUCTION

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%

B. DasGupta was supported in part by the NSF grant IIS-1160995. W. Meng was supported in part by the NSF grant CNS-0958501.
Author’s addresses: E. C. Dragut, Department of Computer and Information Sciences, Temple University; email: edragut@temple.edu; B. P. Beirne, B. DasGupta, A. Neyestani, B. Atassi, and C. Yu, Computer Science Department, University of Illinois at Chicago; W. Meng, Computer Science Department, Binghamton University.

DOI: http://dx.doi.org/10.1145/2656344


1. Demera Ethiopian Restaurant
337 reviews
Category: Ethiopian
Neighborhood: Uptown
4801 N Broadway St
Chicago, IL 60640
(773) 334-8787

a) An example of a restaurant record

2 H&R BLOCK
Other Financial Services, Finance & Taxation
2301 N CLARK ST, CHICAGO, IL (map)
Neighborhood: North Side
1 Review

b) An example of a financial business record

Fig. 1. Typical records of local services.

of Google queries.[1] Local search engines 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. For instance, we can create a system that connects to, gathers and integrates data from local search engines such as Yahoo, YellowPages, Metromix, CitySearch. To increase the *completeness* of data provided to users, such a system needs to gather data from many sources. Completeness measures the amount of relevant data. For example, if we build an integrated system over restaurant search engines, completeness translates into the number of relevant restaurants returned for a user query.

Suppose the query $Q = (\texttt{Cuisine} = \text{“Ethiopian”}; \texttt{Neighborhood} = \text{“Uptown”})$ 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 1th 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. A simple majority voting among search engines may be able to find the right association between a restaurant name and its address. More sophisticated solutions are also possible (e.g., [Zhao et al. 2012]). The aim is to create a metasearch engine (MSE) over existing local search engines (LSE) such that more complete and correct data is made available to users. When a MSE receives a query, it forwards the query to the underlying systems, which process the query and return the results to the MSE. The MSE has to merge and rerank the results into a unified list.

Local services are georeferenced points of interest: for instance, restaurants, hotels, retailers, service providers, etc. In addition to the common attributes `name`, `address`, `phone`, spatial objects are also characterized by attributes such as `feature` (e.g., whether a restaurant has carry out, valet parking), user `rating` and `review`. Figure 1 shows the records of two local services: a) is a restaurant record from Yelp.com and b) is a financial business record from CitySearch.com. Most attributes in the figure are self-explanatory. User rating is usually expressed in number of stars, up to five stars. One goal of this work is to show how these attributes leverage effective integrated search of spatial objects. For instance, we devise a ranking (merge) algorithm that employs users' reviews and query criteria in addition to the positions of objects in the returned lists of the local search engines.

[1]http://articles.businessinsider.com/2011-05-26/tech/30041757_1_google-queries-marissa-mayer-digitas.

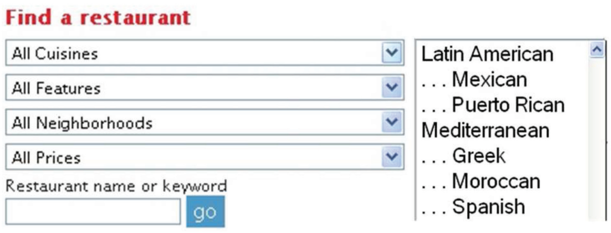


Fig. 2. A query interface for restaurant searching.

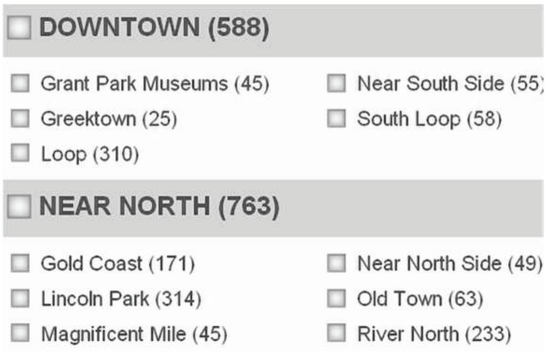


Fig. 3. A fragment of a neighborhood hierarchy.

In the rest of the section, we present a brief background on the construction of a MSE for structured data, discuss the encountered challenges, then enumerate our contributions. The restaurant application domain is a good representative of local services and serves as a case study in this article. The techniques and conclusions drawn in this article are pertinent to many local service domains: for instance, hotels, tax return service providers, health care providers, law offices.

1.1. Background

Figure 2 (on the left) shows a typical Web query interface in the restaurant domain. A user can search by cuisine, neighborhood, price scale or keywords occurring in the record representation of restaurants. The values of the field `cuisine` are often drawn from an ontology of culinary arts (e.g., Italian, Middle Eastern). Figure 2 (on the right) shows a fragment of such an ontology. 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. (Figure 3) and within each region, there are several neighborhoods. The attribute `Prices` is on a scale from 1 to 5 (usually denoted by “$” sign).

The construction of a MSE requires the implementation of several components (Figure 4). The first component is a *mediator form* [Dragut et al. 2006], 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 [Zhang et al. 2005] by the *query translation and dispatch* component. The *result extraction* component extracts the data returned by individual sources [Liu et al. 2010]. The *record linkage* component identifies the records across lists referring to the same entity [Elmagarmid et al. 2007]. This information is

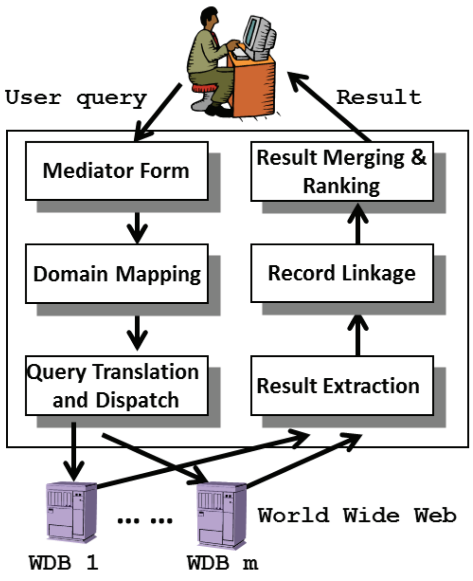


Fig. 4. The architecture of an integration system.

used to merge the lists of records into a single list. Finally, the merged list of records is ranked in descending order of user's desirability (e.g., user rating) [Marian et al. 2004].

In this work we address issues pertaining to the mediator form, query translation, record linkage and result ranking components in the local services domain.

1.2. Challenges

The goal of our work is to perform a hands-on study of the concrete and unique challenges encountered in constructing a metasearch engine over search engines advertising spatial objects. We enumerate these challenges below.

Neighborhood Resolution Challenge. Suppose that a user posts the query (`Price` = "$$$"; `Neighborhood` = "Warehouse District"). We need 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 their neighborhoods may also be different.

Cities have a multitude of neighborhood divisions, some official and some ad-hoc. The problem is that search engines may adopt any one of these neighborhood maps. Chicago's 77 community areas have legally defined borders dating to the 1920s, but very few search engines adopt them. To make matters worse, people living in the same area may have different opinions about the name of the neighborhood they live in. This phenomenon is called "neighborhood identity crisis"[2] and it is not unique to Chicago. Other cities, such as New York[3] and Los Angeles,[4] experience similar problems.

[2]http://articles.chicagotribune.com/2009-07-21/news/0907210119_1_wicker-park-neighborhoods-borders.
[3]http://www.nysun.com/real-estate/identity-crisis-of-a-neighborhood-half-in/19450.
[4]http://laist.com/2009/01/26/defining_los_angeles_neighborhood_b.php.

The bottom line is that the location information specified by a user about a neighborhood in a query ought to be translated by the MSE to suitable locations which can be processed by the underlying sources. For instance, though Warehouse District is not recognized by Zagat, we can still “persuade” Zagat to return restaurants from this neighborhood. We first identify the set of neighborhoods in Zagat whose union is the *minimum set of neighborhoods* containing Warehouse District in Metromix. West Loop and Market District have this property in Zagat. Then, we submit a query with these two neighborhoods to Zagat. The challenge therefore is to construct an integrated system over the neighborhood hierarchies of the LSEs such that a neighborhood specified in a query in the MSE can be translated to any of the LSEs. The difficulties are: (1) the same neighborhood may be referred to by different names in different search engines (e.g., Andersonville and South of Foster are the same neighborhood in Chicago), (2) the names of the neighborhoods are often completely uninformative in determining if two regions have any overlapping relationships (e.g., Loop is a subregion of Downtown) and (3) a neighborhood in one hierarchy may not cover the same space as the union of the minimum set of neighborhoods containing it in a different hierarchy (e.g., the union area of West Loop and Market District includes the area of Warehouse District, but it is significantly larger than that of Warehouse District).

Local Service Resolution Challenge. It is an instance of the entity resolution problem [Elmagarmid et al. 2007] and occurs whenever the same business is referred to in different ways in multiple search engines (e.g., “Aloha Eats” vs. “Aloha Grill”). To provide users with a unique and coherent list of local services per query, we need to identify the records representing the same business entity across search engines.

Rating/Ranking Challenge. Ratings offered by different search engines are not necessarily consistent. Some may rate certain restaurants better than others, and vice-versa. Some search engines may only provide ratings for a subset of the restaurants. The aim is to provide a scheme to rank local services in descending order of “desirability” in view of possibly conflicting information by the LSEs.

User Specified Criteria Challenge. Some search engines ignore user specified criteria or ordering by some attribute (e.g., price) in a query sometimes. For instance, some search engines may ignore price range criteria entered by users. We show that a record can be determined to satisfy a query posted on the MSE by combining the query capabilities of the underlying systems.

1.3. Our Main Contributions

—An integration algorithm for neighborhoods. We show that the problem of finding the set of neighborhoods in a search engine which best approximates the area of a neighborhood in a different search engine is NP-hard. To our knowledge, this work is the first to study the mapping of spatial objects in a metasearch engine.
—A logarithmic approximation neighborhood mapping algorithm.
—An algorithm which ensures that local services returned by the integration system are mostly inside the neighborhood specified by the user.
—A very effective local services resolution algorithm. Experiments confirm its effectiveness in 3 different domains.
—A ranking algorithm that takes into consideration the rankings, user criteria and ratings. The experiments show that our integration system has higher retrieval effectiveness than each of the LSEs.
—Along with a comprehensive set of experimental studies we developed a prototype MSE, called Yumi, as a case study. Yumi is currently restricted to the restaurants in the Chicago Metropolitan area.

There is always the debate between MSEs and large search engines. The latter gathers (by periodic crawling) and integrates data from online sources in a central repository. The main advantage of the former over the latter is data freshness. For example, the restaurant Gioco in Chicago closed on 3/24/2011 and reopened on 4/27/11. Such events are timely reflected in some of the well-anchored sources in the line of businesses. Nevertheless, Gioco is a stale piece of data in a large search engine between the closing date and reopening date, unless a new crawl is issued. A MSE, since it gathers data at query time, is very likely to return Gioco with the correct status.

The rest of the article is organized as follows. Section 2 addresses the neighborhood query processing problem. Section 3 solves the local services resolution problem. Section 4 presents the ranking algorithm. Section 5 presents our experimental study. Section 6 discusses the related work. Finally, Section 7 concludes the article.

2. NEIGHBORHOOD QUERY PROCESSING

Neighborhood query processing is a two-step process. First, a neighborhood query (i.e., a query containing a neighborhood constraint) posted on a MSE is translated to each LSE. Second, since the neighborhood is approximately translated (e.g., a neighborhood in the MSE may correspond, but not exactly equivalent, to a set of neighborhoods in an LSE), the set of local services returned by each LSE must be tested for inclusion in the neighborhood. We discuss these two processes in this section.

2.1. Neighborhood Correspondence Problem

In general, user specified location often does not correspond to exact predefined neighborhoods. In this section, we develop an algorithm that finds an approximate correspondence between the neighborhood maps in the LSEs to assist the MSE in translating user specified locations.

2.1.1. Neighborhood Hierarchy Representation. Search engines have hierarchies of neighborhoods, although for some search engines, the hierarchy has only two levels, with the root being the city (e.g., Chicago) and the leaves being all the neighborhoods of the city. Each hierarchy of neighborhoods H satisfies the property: a neighborhood h is a parent of neighborhood h' in H if the area of h properly includes that of h'. An R/R*-tree [Beckmann et al. 1990] is used to represent a neighborhood hierarchy H. This assumes that the ideal shapes of the neighborhoods are known. The shape of a neighborhood cannot always be obtained with online services (e.g., Google Map). We thus define a neighborhood as a set of (sample) points (local services), which are then used to approximate the shape of the neighborhood. A leaf node in H is a *minimum bounding rectangle* (MBR, to be defined) representing a lowest level neighborhood in H. A higher level neighborhood in H is represented by an internal node in the R/R* tree. Each internal node is characterized by a MBR, which is the smallest bounding rectangle containing the bounding rectangles of its child nodes. Approximation by other more complex shapes other than MBR (e.g., octagons) is possible, but they introduce a level of difficulty that we do not find useful in practice. In addition, we prove (Theorem 2.2) that in the scope of this project handling MBRs is already NP-hard. Thus, manipulating other more complex shapes is expected to be at least as hard.

Among all hierarchies, we pick the hierarchy having the largest number of levels. If there are two or more such hierarchies, we pick the one which has the largest number of nodes. This becomes the *target hierarchy*, denoted by *TH*, on which users will formulate neighborhood queries. Our intuition is that the tallest hierarchy captures the containment relationships between most of the well-known neighborhoods. For instance, there is no way to infer from Yelp's flat enlisting of neighborhoods that DePaul is included in Lincoln Park. In the current implementation the hierarchy of Metromix is the target hierarchy (Figure 3).

2.1.2. Neighborhood Mapping. When a user query specifies a neighborhood h in TH in the MSE, for each component search engine SE, we determine the set of neighborhoods in the neighborhood hierarchy H of SE that intersects h. A "suitable" subset S_h (to be explained later) of this set of neighborhoods will be submitted to SE. S_h is constructed offline before processing user queries. We present here the construction of S_h.

We assume that two neighborhoods *match* if they have the same name. Therefore, whenever h has a match in a source hierarchy then S_h contains only the matched neighborhood. If h does not match any neighborhood in H then we collect all the neighborhoods in H that overlap with h. That is, the MBR of h, say $MBR(h)$, is compared against all the MBRs of the children of the root of H. If the MBR of a child w intersects with $MBR(h)$, then $MBR(h)$ will be compared against all the MBRs of the children of w. This is repeated until the leaf nodes of H are reached. All those nodes whose MBRs intersect with $MBR(h)$ are recorded. The purpose of the R/R* tree is to avoid comparing each $MBR(h)$ in the target neighborhood hierarchy with every neighborhood in a source neighborhood hierarchy.

We have not noticed the homonymy problem when matching neighborhoods, that is, two different neighborhoods of the same city having the same name. Nevertheless, should this occur, we could easily adapt our context-based matching algorithm [Dragut et al. 2009] to the problem of matching neighborhood hierarchies.

The problem to be solved is as follows: For a neighborhood h in the target hierarchy TH, ideally, we need to find a set of neighborhoods S_h in a source hierarchy H with the properties that (1) h is included in the union of the neighborhoods in S_h, that is, $h \subseteq \bigcup_{g \in S_h} g$, and (2) the area of the region covered outside h is minimized, that is, $\text{Area}((\bigcup_{g \in S_h} g) \setminus h)$ is minimized. It turns out that this problem is a computationally hard problem. It remains hard even when the neighborhoods are rectilinear rectangles, that is, rectangles with edges parallel to the Cartesian axes. We call this problem the *Rectangle Covering Problem* and it is defined as follows:

Input: A set $\mathcal{S} = \{R_1, R_2, \dots, R_n\}$ of n rectangles and another rectangle R.
Valid solution: A subset $\mathcal{S}' \subseteq \mathcal{S}$ of rectangles such that $R \subseteq \bigcup_{R_i \in \mathcal{S}'} R_i$.
Objective: minimize the area of the region covered outside R: $Area(\bigcup_{R_i \in \mathcal{S}'} (R_i \setminus R))$.

Recall the following standard definition: a $(1+\delta)$-approximation algorithm for a minimization problem is a polynomial-time algorithm that finds a solution whose value is at most $(1+\delta)$ times the optimum.

THEOREM 2.1. *Assuming* P $\neq$ NP*, our rectangle covering problem does not admit a* $(1+\delta)$*-approximation algorithm for any* $\delta < \frac{5.4}{777.6} \approx 0.0069$.

PROOF. The proof is given in Appendix A. □

Thus, the Rectangle Covering Problem is not only NP-hard, it remains NP-hard even if one tries to approximate very close (in this case about 0.7%) of the minimum. Complementing the above impossibility result, we give here a logarithmic approximation algorithm for the problem of neighborhood mapping.

THEOREM 2.2. *There is a randomized* $O(\log n)$*-approximation algorithm for the Rectangle Covering problem.*

PROOF. The proof is given in Appendix B. □

```
Input: A set $\mathcal{S} = \{R_1, R_2, \ldots, R_n\}$ of $n$ rectangles, another rectangle $R$,
        and the set of corresponding $t$ elementary rectangles $\mathsf{er}_1, \ldots, \mathsf{er}_t$ after preprocessing
Output: a subset $\mathcal{S}' \subseteq \mathcal{S}$ of rectangles such that $R \subseteq \bigcup_{R_i \in \mathcal{S}'} R_i$
Variables used:
    $x_i = \begin{cases} 1, & \text{if } R_i \text{ is selected in our solution} \\ 0, & \text{otherwise} \end{cases} \quad (i = 1, 2, \ldots, n)$

    $y_j = \begin{cases} 1, & \text{if } \mathsf{er}_j \text{ is inside a rectangle selected in our solution} \\ 0, & \text{otherwise} \end{cases} \quad (j = 1, 2, \ldots, t)$
ILP:
    minimize  $\sum_{\mathsf{er}_j \text{ is outside } R} \mathsf{Area}\,(\mathsf{er}_j)\; y_j$
    subject to:
     $\forall\, i \in \{1, \ldots, n\}\ \forall\, j \in \{1, \ldots, t\}$ such that $\mathsf{er}_j$ is inside $R_i$:   $y_j \geq x_i$   //$t\,n$ constraints
                  $\forall\, j \in \{1, \ldots, t\}$ such that $\mathsf{er}_j$ is inside $R$: $\sum_{\mathsf{er}_j \text{ is inside } R_i} x_i \geq 1$   //$t$ constraints
```

Fig. 5. An ILP for our tiling problem.

```
($\star$ LP-relaxation $\star$)
replace each constraint $x_i \in \{0,1\}$ by $0 \leq x_i \leq 1$
replace each constraint $y_j \in \{0,1\}$ by $0 \leq y_j \leq 1$
solve the resulting LP to get a (possibly fractional) value $x_i^{\mathsf{LP}}$ for every $x_i$ and a value $y_j^{\mathsf{LP}}$ for every $y_j$

($\star$ randomized rounding $\star$)
repeat $2 \ln n$ times
    for $i = 1, 2, \ldots, n$ do
        if $x_i$ has not been set to 1 already then set $x_i$ to 1 independently with probability $x_i^{\mathsf{LP}}$
    end for
end repeat

($\star$ greedy step $\star$)
if $x_i = 1$ then set $y_j = 1$ for every constraint of the form $y_j \geq x_i$
for every $\mathsf{er}_j$ inside $R$ such that $y_j \neq 1$ do
    select a rectangle $R_{i_j}$ containing $\mathsf{er}_j$ such that $\mathsf{Area}\left(R_{i_j} \setminus R\right) = \min_{R_\ell \text{ contains } \mathsf{er}_j} \{\mathsf{Area}\,(R_\ell \setminus R)\}$
    for every $\mathsf{er}_\ell$ inside $R_{i_j}$, set $x_\ell = y_\ell = 1$
end for

($\star$ final output $\star$)
return $\mathcal{S}' = \{R_i \,|\, x_i = 1\}$ as the solution
```

Fig. 6. LP-relaxation of the ILP in Figure 5 and subsequent randomized rounding.

The logarithmic approximation algorithm is given in Figures 5 and 6. An informal overview of our logarithmic approach is as follows. Consider the smallest bounding box (i.e., a minimal rectangle) that contains all the rectangles in S and R. Extend the horizontal and vertical sides of all the rectangles in S and R. This partitions the bounding box into a number of rectangles, called “elementary rectangles”. (See Figure 7 for an illustration.) For any elementary rectangle and any rectangle, the elementary rectangle is either completely inside or completely outside the rectangle.

We use the method of linear programming relaxation with randomized rounding. We consider a natural integer linear program (ILP) for this problem, relax the ILP to a linear program (LP) by allowing the variables to take fractional values and finally use a simple randomized post-processing (rounding) to change the fractional values

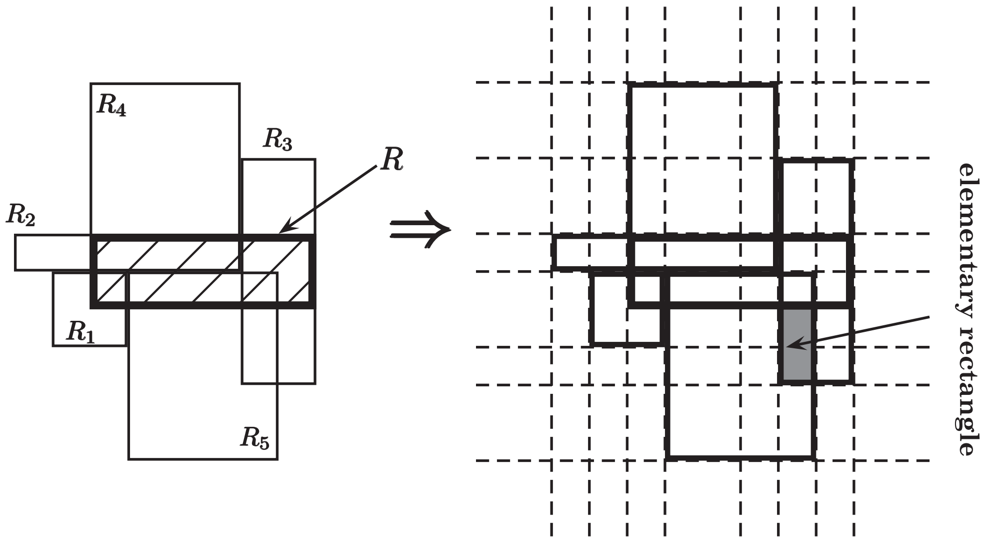


Fig. 7. Preprocessing step of our approximation algorithm.

of relevant variables to integer values such that the expected quality of the solution deteriorates by no more than a logarithmic factor.

An ILP for our tiling problem is shown in Figure 5. The constraints of the form $y_j \geq x_i$ ensure that an elementary rectangle outside R is selected in the calculation of the cost of the solution if and only if at least a rectangle containing that elementary rectangle is selected. Constraints of the form $\sum_{\mathsf{er}_j \text{ is inside } R_i} x_i \geq 1$ ensure that every elementary rectangle inside R is covered by some selected rectangle. The objective $\sum_{\mathsf{er}_j \text{ is outside } R} \mathsf{Area}(\mathsf{er}_j)\, y_j = \sum_{\substack{\mathsf{er}_j \text{ is outside } R \\ \text{and } y_j = 1}} \mathsf{Area}(\mathsf{er}_j)$ measures the sum of areas of the elementary rectangles outside R that are contained in the selected rectangles. If OPT denotes the optimal (minimum) value of the objective function of an instance of the Rectangle Covering problem, then the objective value of an optimal solution of this ILP is exactly OPT. The LP relaxation and randomized post-processing procedure for this ILP is then carried out as shown in Figure 6. The total expected cost of our solution is at most $3 \ln n\, \mathsf{OPT}$.

2.2. Local Service in Neighborhood Problem

A neighborhood h in the MSE corresponds to a set of neighborhoods S_h in an LSE. Consequently, the LSE may return businesses not in h, besides those in h. We need an effective online solution for deciding which objects from the set of results returned by an LSE are in h. We call this problem *local service in neighborhood problem*. For this task, a bounding container for h needs to meet two constraints: (i) it is fast to test that a point (the location of a business) is inside the bounding container and (ii) the bounding container closely approximates h. In general, MBRs satisfy (i) but not (ii). It is difficult in general to achieve both (i) and (ii) without trading memory consumption. One of the fastest method for the point inclusion problem is *lookup grids* [Preparata and Shamos 1985]. The idea is to impose a grid inside the minimum bounding box containing h (Figure 8(c)). Each grid cell is categorized as being *inside*, *outside* or *border*. An inside cell contains exclusively points from h, an outside cell does not contain any point from h, and a border cell contains in addition to the points in h points not in h.

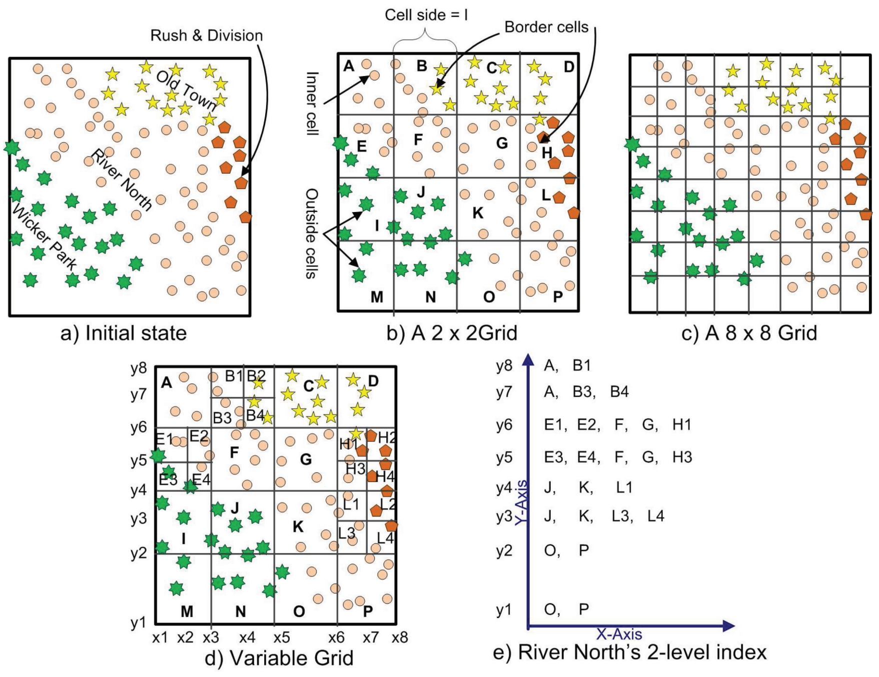


Fig. 8. An example of lookup grids.

2.2.1. Regular Grid Decomposition. We now describe how the grid is computed for a neighborhood map of a given region (say Chicago). Each neighborhood is defined by a set of sample points (local services). Thus, the initial configuration is as shown in Figure 8(a), where we depict an example with four neighborhoods: River North, Wicker Park, Old Town and Rush & Division. The points from the same neighborhood are drawn with the same shape and color. Points from different neighborhoods are drawn with different shapes and colors. The goal is to construct a grid such that, ideally, each cell contains only points of a single neighborhood. We start from the configuration in Figure 8(a) and successively refine the granularity of the grid as long as some conditions, called *splitting criteria* (to be defined), are satisfied. We start with a 2×2 grid, if this grid meets the splitting criteria we introduce a 4×4 grid (Figure 8(b)). We test again the splitting criteria. If it is satisfied, we introduce an 8×8 grid (Figure 8(c)). So on and so forth, until the splitting criteria are not satisfied.

Splitting Criteria. Let C be a cell. Suppose there are s neighborhoods. Let $n_1, \ldots, n_s$ be the numbers of points in C from each of the s neighborhoods. Let $NS = n_1 + \cdots + n_s$. A measure of the "impurity" of the content of C is given by Shannon's Entropy [Liu 2007]: $H(C) = -\sum_{i \in [1..s]} \frac{n_i}{NS} \log \frac{n_i}{NS}$. It represents the amount of information needed to determine if a point in C belongs to one of the s neighborhoods. $H(C)$ is minimum when C has points only from a single neighborhood and is maximum when C contains an equal number of points from each neighborhood. We need to make $H(C) < \varepsilon$, where ε is a threshold set empirically. This measure by itself is not sufficient because it can lead to over splitting, and thus, produces a very dense grid. We introduce an additional constraint to avoid over splitting: The length of the side of C is no smaller than a given number of miles/kms. We note that the side of C can be alternatively specified as an average kNN distance.

Point Lookup. To test a point P against this structure is extremely fast. Geometrically, we have a rectangle given by $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ that is partitioned into a grid of cells, each with the area $d_1 \times d_2$. Each cell C is uniquely identified by an interval $X_C = [x^C_{min}, x^C_{max}) \subset [x_{min}, x_{max}]$ and an interval $Y_C = [y^C_{min}, y^C_{max}) \subset [y_{min}, y_{max}]$. If the point P has the coordinates (x, y), we first determine the unique intervals X_C and Y_C such that $x \in X_C$ and $y \in Y_C$: $x^C_{min} = \lfloor \frac{x-x_{min}}{d_1} \rfloor d_1 + x_{min}$ and $x^C_{max} = x^C_{min} + d_1$; $y^C_{min} = \lfloor \frac{y-y_{min}}{d_2} \rfloor d_2 + y_{min}$ and $y^C_{max} = y^C_{min} + d_2$. If C is an inside (outside) cell then P is inside (outside) of h. Problems occur if P falls in a border cell.

2.2.2. Variable Grid Decomposition. While the previous method can meet the conditions (i) and (ii), it is memory intensive. We can improve its space consumption, but increasing the lookup time, by not restricting the grid to a uniform resolution. Similar to the quad-tree construction [Preparata and Shamos 1985], a region is subdivided into four quadrants (cells), identified by number 1, 2, 3 or 4 (from top left in Z shape). We divide a region until each quadrant has the desired "purity" (Figure 8(d)). The splitting criteria defined above becomes a local condition and it influences only the division of C.

Lookup in Variable Grid. Given a point P we can traverse top-down the quad-tree directly to determine the placement of a point. We can improve this strategy because we know the target neighborhood h and its associated cells. We create a multilevel-index: the set of cells having points from h are organized into a two-level indexing structure. For example, the cells containing points from River North are: A, B1, B3, E1, E2, E4, F, G, H1, H3, K, L1, L3, L4, O and P. Each of these cells has a left-upper corner coordinate and right-lower coordinate. It can be shown that both their Y's coordinates and X's coordinates each forms a continuous segment. Figure 8(d) shows the Y's and X's of the cells for River North. So, we can create a two-dimensional index: we first index the cells of a neighborhood by their Y's and then by their X's. The index for the neighborhood River North is given in Figure 8(e). Let SX and SY be the sets of segments on the X-axis and Y-axis, respectively, determined by the cells intersecting the neighborhood h. For instance, in Figure 8(d), $\text{SY} = \{[y_1, y_2), \ldots, [y_7, y_8]\}$ and $\text{SX} = \{[x_1, x_2), \ldots, [x_7, x_8]\}$. Deciding if P is in h takes in the worst case $\log|\text{SY}| + \log|\text{SX}|$. SX and SY are very small in comparison with the total number of segments on either of the axis determined for a neighborhood map.

We prefer the variable grid decomposition of neighborhoods since it gives us control over the points to include in the answer for h and takes less space. We can include local services from a bordering neighborhood provided they are not too far from h. The knowledge of distance is encoded in the area of each border cell. A regular grid decomposition with cells of 1km^2 area takes about than 5MB for Chicago Metro area.

3. LOCAL SERVICE RESOLUTION

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. For example, for the query Q = (`Cuisine` = "Pizza"; `Neighborhood` = "Lincoln Park"), Yelp returns the record "Giordano's, 815 W. Van Buren St., Chicago" and Metromix returns the record "Giordano's Pizzeria, 815 W Van Buren St, Chicago, (312)421-1221." These records are substantially different (e.g., one has a phone number, the other does not), nevertheless they represent the same restaurant. We need to automatically determine that these records refer to the same restaurant at query time and to use this information in the merging and ranking procedures.

Table I. Examples of Pairs of Matching Records

ID	Records
R_1	Giordano's, 815 W. Van Buren St., Chicago
R_2	Giordano's Pizzeria, 815 W Van Buren St, Chicago, (312)421-1221
R_3	Fringale, 570 Fourth St., San Francisco 415-543-0573
R_4	Fringale, 570 4th St., San Francisco 415-543-0573
R_5	Al-Khaymeih, 4748 N Kedzie Ave at Lawrence, Chicago, 773-583-0999
R_6	Al Khayyam, 4748 N Kedzie Ave, Chicago, 773-583-0999

We describe our original local services matching algorithm in this section. A business entity has several attributes: name, address, zip code, phone, user rating, user review, cuisine, etc. We present a supervised learning solution. 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, a decision tree infers rules to determine if two records refer to the same business.

Attribute selection. We assume that the task of selecting the attributes used for matching two records is performed by a domain expert. This is a common assumption in the record linkage literature [Elmagarmid et al. 2007; Köpcke and Rahm 2010]. In our application, the attributes are name, address and phone number.

Similarity Measures. The similarity of two business names, rn_1 and rn_2, is given by

$$simN(rn_1, rn_2) = \begin{cases} 1 & \text{if prefix}(rn_1, rn_2) \\ 1 - \frac{sed(rn_1, rn_2)}{max(|rn_1|, |rn_2|)} & \text{otherwise,} \end{cases}$$

where prefix(rn_1, rn_2) is true if either name is a prefix of the other, $|rn_1|$ is the length of the string rn_1 and $sed(rn_1, rn_2)$ is the *string edit distance* between rn_1 and rn_2 [Gusfield 1997]. Words such as "restaurant" and "pizzeria" are discarded; they are treated as stop words.

The computation of the similarity of two business addresses is more complicated because the subfields of addresses are compared. Addresses are converted to a standard consistent format, by first segmenting them into a fixed set of structured elements (e.g., street address, city, state) and then converting them to a canonical format (e.g., St. is expanded to Street, Dr. to Drive) [Borkar et al. 2001]. Two addresses are different, that is, similarity equal to 0, if they have different states or cities. If the addresses have the same state and city then their similarity is given by the similarity between their street addresses, whose similarity is computed in the same way as the similarity between names.

The similarity of two phone numbers is 1 if they are identical (assume normalization to a standard format) and 0 otherwise. We use plain string comparison. More sophisticated similarity measures are possible [Guo et al. 2010].

3.1. Matching Rules

We observed that two business records were very likely to refer to the same business entity in the following cases: (1) if their addresses and names are each very similar (e.g., R_1 and R_2 in Table I); (2) if the businesses have the same phone number and their names are "very" similar (e.g., R_3 and R_4 in Table I) or (3) if the businesses have the same phone number and their addresses and names are each "somewhat" similar, that is, the name and address similarity thresholds are lowered (e.g., R_5 and R_6). These rules, while not complete, suggest that a rule-based approach may be effective in classifying pairs of records as matches or nonmatches.

One could attempt to encrypt these rules by hand, but it is likely that they do not generalize. The reason is that we do not know (1) if these rules are all the possible rules and (2) how to quantify relative information such as "(very) somewhat similar." *Decision trees* are the abstraction that best captures our rule-based intuition. Once a decision tree is learned, we can transform it into a set of IF-THEN rules [Liu 2007] that can be deployed online. Our solution thus is to obtain a training sample by posting a number of random queries to the component search engines, to manually label the pairs of matching records and to learn a decision tree from the labeled data [Liu 2007].

We describe how the decision tree is obtained. Let L and L' be two lists of businesses. Let $T = \{(r, r')|r \simeq r', r \in L, r' \in L'\}$ and $U = \{(r, r')|r \not\simeq r', r \in L, r' \in L'\}$, be the set of records that represent identical business entities and the complement of T, representing different business entities, respectively. Let $A_1, \ldots, A_k$ be the set of attributes which are compared to determine if two records refer to the same business. A similarity function is defined for each attribute. The domain of each similarity function is between 0 and 1, with 1 representing identity and 0 representing dissimilarity. We define a *matching vector*, $m(r, r')$ such that $m_i = sim_i(r[A_i], r'[A_i])$, where $r[A_i]$ $(r'[A_i])$ is the value in the attribute A_i and sim_i denotes the similarity function defined for A_i. We need to estimate the conditional probabilities of observing $(r, r') \in T$ or $(r, r') \in U$ given the matching vector of r and r'. Skipping the formal justification, this can be captured as a decision tree. The learning dataset is a set of vectors of the form $(m(r, r'), M)$, where M is the target variable. $M = 1$ if records r and r' match and 0, otherwise. For example, the vector corresponding to the pair of records R_1 and R_2 is $\langle 1, 1, ?, 1\rangle$, where the first three components are the similarities for names, addresses and phone numbers, respectively, of R_1 and R_2; "?" denotes that the phone number of one (or both) of the records is (are) missing. The last component denotes that R_1, matches R_2. The vector $\langle 0.2, 0.5, 0, 0\rangle$ corresponds to the records R_2 and R_8, which do not match.

We assume that the task of selecting the attributes used for matching two records, that is, $A_1, \ldots, A_k$, is performed by a domain expert. This is a common assumption in the record linkage literature [Elmagarmid et al. 2007; Köpcke and Rahm 2010].

Handling Skewed Class Distribution. For training purposes, given two datasets A and B whose records are to be matched, the learning dataset LD contains $|A| \times |B|$ tuples. LD consists of positive examples (matches) and negative examples (nonmatches). The number of matches is in general just a small fraction of the entire dataset. For example, 12Q, one of the datasets used in our experiments (Section 5.8), has 405 positive examples and 36,635 negative examples. That is, the positive examples are only a mere 1.1% of the entire dataset. Directly applying the decision tree algorithm for classification to data with skewed class distribution is not effective [Liu 2007]. One way to overcome skewed class distribution is to over sample the examples of matched records to increase their proportion. We use SMOTE [Chawla et al. 2002], which is an over sampling method that generates synthetic examples at random from the positive examples.

4. RANKING AND MERGING RESULTS

There are several result ranking/merging algorithms for text metasearch engines [Meng and Yu 2010]. Among them, voting based techniques are more suitable for integration systems whose underlying sources have substantial overlaps among their document collections, as it is the case in our setting. In voting based methods, LSEs act as voters and retrieved results are treated as candidates in an election. In an election, we are interested in the ranking positions of all candidates. Borda-fusion [Aslam and Montague 2001] and Condorcet's method [Montague and Aslam 2002] are popular voting based techniques. Nevertheless, the *reciprocal rank fusion* (RRF) [Cormack et al.

2009] was shown to outperform both of them. In addition, RRF bested more sophisticated learning methods such as CombMNZ [Montague and Aslam 2002], ListNet [Cao et al. 2007], RankSVM [Joachims 2002] and RankBoost [Freund et al. 2003].

We implemented three ranking algorithms: (1) the RRF algorithm [Cormack et al. 2009], (2) an algorithm that employs users' ratings and reviews in addition to RRF, called RRF-R, and (3) an algorithm that uses RRF together with users' ratings and users' query criteria, called RRF-UCR (UCR - user criteria and rating). RRF-R and RRF-UCR are our proposed algorithms. Ranking/merging assume that the problem of identifying the records referring to the same local service entity across the result lists has been solved. The problem is addressed in Section 3.

4.1. Reciprocal Rank Fusion

RRF sorts the documents according to a naive scoring formula. Given a set D of documents to be ranked and a set of rankings R, each a permutation on $1..|D|$, we compute

$$RRF(d \in D) = \sum_{\sigma \in R} \frac{1}{z + \sigma(d)},$$

where $\sigma(d)$ is the rank of the document d, and the constant z mitigates the impact of high weights. $z = 60$ was empirically determined in Cormack et al. [2009] and independently confirmed in Hu et al. [2011].

4.2. Utilizing Rating and Reviews in Ranking

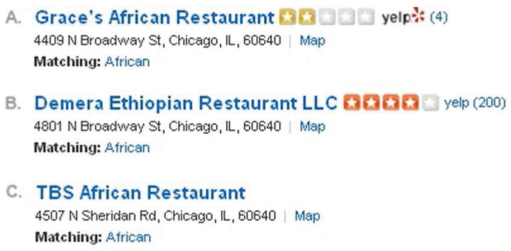
When running a query against LSEs, for each returned local service we also collect information about user ratings and number of reviews. If a business does not have a rating in a search engine, then by default its rating is set to 0 stars (out of 5). If a business has fewer than 10 reviews (empirically set), the rating is set to 0.5, as it is not reliable. We want to order the returned businesses by rating and have the "best" ones at the top in order for our ranking algorithms to work correctly. So for each search engine, Yumi orders the results by ratings and if two restaurants happen to have the same ratings then it uses the number of reviews to break the tie, by assuming that if a restaurant has more reviews it is more likely to be better.

Note that this is a significant departure from the way documents are ranked by a text metasearch engine. A text metasearch engine takes the ordered results from each component search engine to form lists and then merges the lists. In contrast, we use the information (i.e., users' ratings and reviews) from the results of multiple independent lists to reorder the objects within each list. The *modified* lists are then merged using the RRF algorithm.

For example, if the query Q = (`Cuisine` = "African"; `Neighborhood` = "Uptown") is submitted to DexKnows then it returns the three restaurants shown in Figure 9. Based on their user ratings, we reorder them such that B is first, A is second and C is last.

4.3. Utilizing User Criteria in Ranking

Consider that the following query is posted to our integration system Yumi: Q = (`Cuisine` = "African"; `Price` = "$$"; `Neighborhood` = "Edgewater"). Yumi submits it to DineSite and MenuIsm search engines. They both can process `Cuisine`. The former's query capabilities can understand `Price`, but it cannot understand `Neighborhood` (because it does not have this or an equivalent attribute for it). The latter handles `Neighborhood`, but it does not handle `Price`. They both return the Ethiopian Diamond Restaurant. We can deduce that this restaurant (strictly) satisfies Q, because DineSite tells us that the restaurant satisfies `Price` = "Affordable," while MenuIsm tells us that it satisfies `Neighborhood` = "Edgewater." However, the restaurant Ras Dashen does


A. Grace's African Restaurant yelp (4)
4409 N Broadway St, Chicago, IL, 60640 | Map
Matching: African

B. Demera Ethiopian Restaurant LLC yelp (200)
4801 N Broadway St, Chicago, IL, 60640 | Map
Matching: African

C. TBS African Restaurant
4507 N Sheridan Rd, Chicago, IL, 60640 | Map
Matching: African

Fig. 9. A restaurant list that needs reordering.

Table II. Indexing Search Engines by Query Capabilities

Condition	Search engines		
Neighborhood	Metromix		MenuIsm
Cuisine	Metromix	DineSite	MenuIsm
Price	Metromix	DineSite	
Features	Metromix	DineSite	

not strictly satisfy Q because it is only returned by MenuIsm, which ignores `Price`. Consequently, we rank Ethiopian Diamond Restaurant ahead of Ras Dashen. We now describe how this is achieved in practice.

Suppose the query interface of the MSE has k attributes (conditions), $A_1, A_2, \ldots, A_k$. We implement a methodology resembling query processing using *postings lists* [Manning et al. 2008]. We create an *indexing* data structure such that for each attribute A_i we record the list of LSEs that can process the attribute. For example, the list for attribute `Price` in Table II contains Metromix and DineSite, but not MenuIsm. Suppose query $Q = (A_{i_1} = Val_1, A_{i_2} = Val_2, \ldots, A_{i_t} = Val_t), t \leq k, \{i_1, \ldots, i_t\} \subseteq [1..k]$, is submitted to Yumi. Suppose that restaurant r is returned by a subset $\{SE_1, \ldots, SE_s\}$ of LSEs. Each of these search engines is looked up in the lists corresponding to $A_{i_1}, \ldots, A_{i_t}$. r in the list of results (strictly) satisfies Q if every attribute in Q has one of $SE_1, \ldots, SE_s$ in its list. Note that a MSE, instead of submitting a query to all its underlying sources, can employ this technique to select the sources that can best process the query.

Given a query Q, the list of results of an LSE is classified in 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 reranked such that Q_{all} is ahead of Q_{part}, which is ahead of Q_{not}.

The fact that Q_{not} is nonempty in general requires an explanation. Most search engines support keyword based search in addition to the attribute based search (see Figure 2). If a search engine SE cannot process a condition $(A = val)$ of a query, we submit to SE the value val as a keyword query. Hence, in general, for a query $Q = (A_{i_1} = Val_1, A_{i_2} = Val_2, \ldots, A_{i_t} = Val_t)$ and a search engine SE, we partition the set of attributes of Q in two classes: those that can be processed A_{proc} and those that cannot be processed $A_{\not{p}roc}$ by SE. If $A_{proc} \neq \emptyset$ then the value of each attribute in A_{proc} is mapped to its corresponding attribute in SE. If $A_{\not{p}roc} \neq \emptyset$ then there are two cases: (1) if SE supports keyword queries then we concatenate the values in $A_{\not{p}roc}$ and place them in the keyword field and (2) if SE does not support keyword queries then we submit only the part of the query A_{proc} that it can process and ignore the rest of it. If SE does not support keyword queries and $A_{proc} = \emptyset$ we do not submit the query to SE. For example, DineSite does not support keyword search and it cannot interpret neighborhoods. As a consequence, queries such as $Q = (Neighborhood = N)$ are not submitted to it. A search engine *partially processes* a query Q, if $A_{proc} \neq \emptyset$ and $|A_{proc}| < t$. To conclude, a record r is in Q_{not} if it is returned only by search engines satisfying: $A_{proc} = \emptyset$ and supporting keyword search. If a search engine SE cannot process a condition $(A = val)$ of a query (and thus its val is placed in the keyword feature), then the MSE cannot determine if the condition is satisfied or not. We believe that, in the context of metasearching, this is the first study that combines the capabilities of multiple search engines in determining the satisfaction of a record to a query.

4.4. Ranking—Putting It All Together

For a query Q, let $L_1, \dots, L_n$ be the lists of results returned by n search engines. Our ranking algorithm, called RRF-UCR, performs the following operations.

(1) It partitions each L_i into 3 sets: Q_{all} is the set of objects that strictly satisfy Q, Q_{part} is the set of objects that partially satisfy Q and Q_{not} is the set of objects that do not satisfy Q. The objects of Q_{all} are ranked ahead of Q_{part}, which are ranked ahead of Q_{not}.
(2) For each L_i, it ranks the objects in each of the three partitions according to their user ratings and reviews. Call this new lists L'_i. L_i and L'_i have the same objects, but possibly differently ordered.
(3) It applies the RRF method to $L'_1, \dots, L'_n$.

LEMMA 4.1. *RRF-UCR ranking algorithm has a time complexity of* $O(Nnq + nk(logk + 1) + NlogN)$, *where* $N = |\bigcup_i L_i|$, *n is the number of search engines, q is the number of conditions in a query and k is the top-k results from each search engine.*

5. EXPERIMENTS

The intention is to evaluate the effectiveness of the integration systems. The evaluation has five key components:

(1) to show that our MSE is more effective than any of the underlying search engines;
(2) to compare the effectiveness of the rankings of the RRF, RRF-R and RRF-UCR algorithms. Yumi with RRF-UCR is available online at `www.yumi-meta.com`;
(3) to identify the type of queries benefiting a MSE versus a single search engine;
(4) to show that the neighborhood query processing algorithm (i.e., the neighborhood correspondence and the local service in neighborhood algorithms) returns local services mostly in the neighborhood specified by the user;
(5) to show that our local services resolution algorithm is very effective in practice.

5.1. Experimental Settings

Yumi connects to 9 search engines: ChicagoReader.com (3,096), CitySearch.com (12,695), DexKnows.com (5,843), MenuIsm.com (8,508), MenuPages.com (3,629), Metromix.com (5,044), local.Yahoo.com (10,820), YellowPages.com (7,798), Yelp.com (10,115). The numbers in parenthesis are the estimated numbers of records for these search engines for Chicago Metropolitan area. Zagat is not included in Yumi, because it is used to evaluate different versions of Yumi. We believe that 9 search engines are sufficient for an unbiased experimental study. A result from text metasearch engines [Avrahami et al. 2006] suggests that selecting three to five search engines is usually sufficient for extracting most of the available relevant documents. We use all 9 engines. We conducted experiments in this geographic area because of our familiarity with the area, which allows us to judge the results of Yumi.

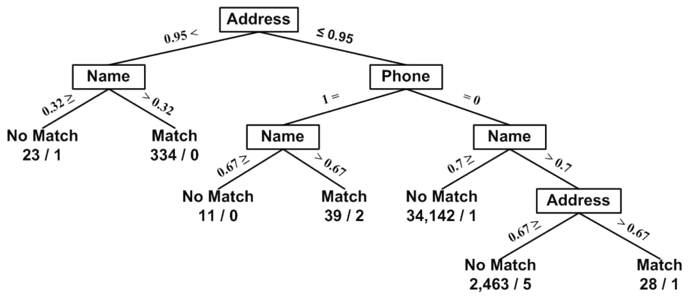


Fig. 10. The decision tree inferred from the 12Q dataset.

5.2. Parameter Setting

Yumi is set with the following parameters.

Neighborhood mapping. To avoid overestimation of bounding rectangles due to a wrong restaurant address or an erroneous mapping of a correct address into its corresponding geocode, outliers are detected using standard statistical techniques. A restaurant r, whose latitude and longitude coordinates are (r_{lat}, r_{long}), is an outlier if either $|r_{lat} - \overline{lat}| > \alpha S_N^{lat}$ or $|r_{long} - \overline{long}| > \alpha S_N^{long}$, where $\overline{lat}$ ($\overline{long}$) is the mean of latitude (longitude) coordinates of the set of the restaurants in the neighborhood; S_N^{lat} (S_N^{long}) is the sample standard deviation of latitude (longitude) coordinates of the restaurants in the neighborhood. $\alpha = 3$ is determined empirically. The outlier restaurants of each neighborhood are removed before we compute their bounding rectangles.

Local service resolution. We use dataset 12Q to learn a decision tree which is used throughout all the experiments. Additional details are given in Section 5.8. 12Q gives the decision tree in Figure 10.

Ranking. $z = 60$ for RRF. top-k = 5 and 10.

5.3. Evaluation Methods

We describe our methodology of evaluating the results of Yumi. We employ two evaluation methods: user-based and arbitrator-based. The latter ensures that our evaluation is unbiased. We present them in turn. For each search engine under evaluation, the top-k (k = 5, 10) restaurants are considered. An engine retrieving more than k restaurants is not given any advantage over those that retrieve k restaurants.

5.3.1. User-Based Evaluation. For a list of records in response to a search query, a user judges the relevance of each record to the query. The judge is one of the authors of the article. Each record is judged on a scale of 0 – 2, with 0 meaning irrelevant, 2 meaning completely relevant, and 1 "somewhere in between." Then we use the *normalized discounted cumulative gain* (NDCG) [Järvelin and Kekäläinen 2002], a popular measure for search performance evaluation. We give its definition later in this section.

Approximately, a record r in response to a query $(A_1 = Val_1, A_2 = Val_2, \ldots, A_t = Val_t)$ is completely relevant if r satisfies all query conditions, r is "somewhere in between" if it meets some of the query conditions and it is irrelevant if it does not meet any of the query conditions. For example, for the query (Cuisine = "Italian"; Neighborhood = "Near North Side") the record "Proseco, Near North Side, $$$" is completely relevant, whereas "The Glunz Tavern, Near North Side, $$" is not relevant because it is a German restaurant and "Sofi Restaurant, South Loop, $$" is not relevant, although it is an Italian restaurant, because it is too far from the neighborhood "Near North

Table III. An Example of Lists of Ranked Businesses

$List_1$	$List_2$	$List_3$
R_1	R_1	R_1
R_4	R_2	R_2
R_5	R_3	R_3
R_3	R_4	

Side". "Pizzeria Uno, Near North Side, $$" is considered relevant because pizza is an Italian dish.

5.3.2. Arbitrator-Based Evaluation. The arbitrator is a third party, which is generally accepted as an authority in the domain of discourse. For instance, Zagat is regarded as an authority in the restaurant ranking/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 for a query Q. We assign scores to each object in L_a its corresponding Borda count [Aslam and Montague 2001]. That is, the object in the 1st position has a score of k, the one in the 2nd position has a score of $k - 1$ and so on, until the object in the last position has a score of 1. The relevance score of an object $r \in L_{se}$ is 0 if $r \notin L_a$ and its corresponding Borda count if $r \in L_a$. We compute the NDCG for L_{se} using the new relevance scores. According to the arbitrator, between two lists of results for a query, the one with the larger NDCG_k performs better at rank k.

5.3.3. Normalized Discounted Cumulative Gain. NDCG is a normalized version of the *discounted cumulative gain* (DCG). DCG uses graded relevance as a measure of the *gain* from examining a record. The gain is accumulated starting at the top of the ranking and is reduced at lower ranks. The DCG is the total gain accumulated at a particular rank k. It is defined as: $\text{DCG}_k = \text{rel}_1 + \sum_{\text{i}\in[2..\text{k}]} \frac{\text{rel}_\text{i}}{\log_2 \text{i}}$. For example, suppose for the records in $List_3$ in Table III, a user provides the following relevance scores: the relevance score of R_1 is that of 2, R_2 is 0 and that of R_3 is 1. The DCG of this list is $\text{DCG}_3 = 2 + \frac{0}{\log_2 2} + \frac{1}{\log_2 3} = 2.63$. The NDCG metric is simply defined as DCG divided by the ideal DCG (IDCG), that is, the DCG that one would obtain if the data were perfectly ranked. In our example, the ideal ranking is obtained by switching R_2 and R_3. This gives $\text{IDCG}_3 = 3$. $\text{NDCG}_3 = \frac{2.63}{3} \approx 0.88$. Between two lists of results for a query, the one with the larger NDCG_k performs better at rank k.

We now provide an example of how the evaluation via an arbitrator is performed. We use $List_1$ in the example in Table III. Let the list returned by the arbitrator be $L_a = [R_1, R_6, R_3, R_5]$. The Borda counts are: $w(R_1) = 4$, $w(R_6) = 3$, $w(R_3) = 2$ and $w(R_5) = 1$. The DCG at rank 4 for $List_1$ is $\text{DCG}_4 = 4 + \frac{0}{\log_2 2} + \frac{1}{\log_2 3} + \frac{2}{\log_2 4} = 5.63$. The ideal ordering of the scores in $List_1$ is 4, 2, 1, 0. Hence, $\text{IDCG}_4 = 4 + \frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{0}{\log_2 4} = 6.63$. Therefore, $\text{NDCG}_4 = \frac{\text{DCG}_4}{\text{IDCG}_4} = \frac{5.63}{6.63} = 0.85$.

We note that if lists containing different objects are returned, NDCG may not be adequate for evaluating search results. Consider the list $L = [R_3, R_5]$. Its NDCG_4 w.r.t. L_a is 1, which is larger than 0.85, the NDCG_4 for $List_1$. $List_1$ is better, yet it has a smaller NDCG score. We propose an alternative definition to NDCG, called ANDCG (arbitrator NDCG). ANDCG is computed by dividing the DCG of a list by that of the arbitrator. For example, the arbitrator's DCG_4 is $4 + \frac{3}{\log_2 2} + \frac{2}{\log_2 3} + \frac{1}{\log_2 4} = 8.76$. The lists $List_1$ and L

Table IV. Comparing the Ranking Algorithms

Ranking Alg.	$NDCG_5$	$NDCG_{10}$
RRF	0.42	0.4
RRF-R	0.48	0.47
RRF-UCR	0.58	0.57

Table V. The Component Search Engines Results

Search Engine	$NDCG_5$	$NDCG_{10}$	$RelImp_{10}$	$ANDCG_5$	$ANDCG_{10}$	No Res.
YellowPages	0.15	0.16	250%	0.04	0.05	329
DexKnows	0.2	0.21	183%	0.04	0.05	160
CitySearch	0.2	0.23	150%	0.06	0.07	213
Yahoo	0.28	0.31	88%	0.08	0.08	154
Metromix	0.29	0.3	93%	0.09	0.09	126
Yelp	0.3	0.3	92%	0.1	0.1	192
MenuPages	0.32	0.35	70%	0.1	0.12	145
Menuism	0.35	0.36	58%	0.07	0.11	71
ChicagoReader	0.38	0.4	47%	0.12	0.14	140
Yumi	**0.58**	**0.57**	N/A	**0.2**	**0.24**	**46**

have $ANDCG_4 = \frac{5.63}{8.76} = 0.65$ and $ANDCG_4 = \frac{3}{8.76} = 0.34$, respectively, which correctly suggest that $List_1$ is better than L at rank 4 (according to the arbitrator). We give the results in both measures.

5.4. Yumi Effectiveness Evaluation

For this experimental evaluation we employ both the user-based and the arbitrator-based evaluations. In the latter, Yumi was used with each of the three algorithms and 1,000 queries were submitted to Yumi, Zagat (the arbitrator) and each of the 9 search engines. The queries were generated as follows. We first generated the set of all possible queries by creating the cross product of the values of query fields `Neighborhood`, `Cuisine` and `Price`. The Null value was inserted in each of the fields to account for the cases when not all fields are mentioned in a query. The resulting query space has 101,903 distinct queries. We selected 1,000 of them randomly without replacement. For 371 of them Zagat did not return any result, so they were ignored. We computed the average NDCG over the rest of them.

5.4.1. Evaluating the Ranking Algorithms. Table IV summarizes the experimental outcome for Yumi. The arbitrator-based evaluation was used. We submitted the batch of 1,000 queries to each of the three versions of Yumi. RRF-UCR is better than the other two at both rank 5 and 10. The average NDCG at ranks 5 and 10 for RRF-UCR are 0.58 and 0.57, respectively. NDCG is adequate to compare the performance of the three ranking algorithms, because their returned result lists are identical, although the algorithms may order the lists differently.

5.4.2. Yumi versus Component Local Search Engines. This evaluation aims to show that Yumi is significantly better than any of the LSEs. Yumi uses the RRF-UCR ranking algorithm and consists of all 9 search engines. We use the arbitrator-based evaluation with the same batch of 1,000 queries. Table V depicts the outcome of the experiment for the LSEs. The second and third columns contain the average NDCG at rank 5 and 10, respectively. Observe that Yumi's average NDCG (Table V, last row) is better than the

Table VI. The Set of 50 Queries Used in the User-Based Evaluation

West Loop	North + Brazilian	Downtown + \$\$\$\$	Lincoln Park + American + \$\$\$\$\$
South Loop	West Loop + French	Lincoln Park + \$\$\$\$\$	South Loop + Mexican + \$\$\$
Lincoln Park	Downtown + French	Wrigleyville + \$\$\$	Downtown + Chinese + \$\$\$\$
Gold Cost	River North + Turkish	Rush & Division + \$\$\$\$\$	Near North + Contemporary + \$\$\$\$\$
Lakeview	North + Korean	Gold Coast + \$\$\$\$\$	North + Tapas + \$\$\$
Italian	Lincoln Park + American	North + \$\$\$\$\$	Northwest + Cuban + \$\$\$
Chinese	Lincoln Park + Italian	Northwest + \$\$\$\$\$	Avalon Park + Fast Food + \$
French	South Loop + Mexican	West Loop + French + \$\$\$\$	Near North + Indian + \$\$\$
Ethiopian	Andersonville + Mexican	Downtown + French + \$\$\$\$	North + Korean + \$\$\$
Middle Eastern	Lincoln Park + Irish	Lincoln Park + Italian + \$\$\$	South Loop + Steak + \$\$\$
Cuban	South Loop + Steak	Albany Park + Japanese	Near North + Turkish
Uptown + Ethiopian	Chinatown + Korean	Old Town + Contemporary	North + Chinese + \$\$\$\$
Lakeview + Sushi	Near North + Indian		

average NDCG of any of the LSEs by a significant margin. The fourth column gives the relative improvement in $NDCG_{10}$ for Yumi to the LSEs, that is, $\frac{NDCG^{y}_{10} - NDCG^{se}_{10}}{NCDG^{se}_{10}}$, where $NCDG^{se}_{10}$ is the ranking performance of an LSE, while $NDCG^{y}_{10}$ is that of Yumi. Yumi yields the largest improvement relative to YellowPages, that is, 250%. Notably, the relative improvement is at least 92% for 5 of the 9 search engines.

In terms of ANDCG, RRF-UCR achieves on average $ANDCG_5 = 0.2$ and $ANDCG_{10} = 0.24$ (Table V, last row). ChicagoReader has the largest average ANDCG among the LSEs (Table V, 6th column): $ANDCG_5 = 0.12$ and $ANDCG_{10} = 0.14$. YellowPages has the lowest average ANDCG: $ANDCG_5 = 0.04$ and $ANDCG_{10} = 0.05$. The percentage improvement of Yumi over the LSEs in ANDCG is more drastic: it is at least 71% at rank 10 and 66% at rank 5. Except for ChicagoReader, the relative improvement is at least 110% relative to any of the LSEs, regardless of the rank.

The last column of Table IV is also indicative of the poor performance of the individual search engines relative to Yumi. It shows the number of queries out of the 1000 submitted for which no result was returned by the search engines. For example, YellowPages does not return any result for 329 of the queries, whereas Yumi returns no results for only 46 of the queries, which is significantly lower.

5.4.3. User Evaluation. We also use the user-based evaluation for RRF-UCR, our leading ranking algorithm. We submitted 50 queries, listed in Table VI: 11 with one condition, 24 with two conditions and 15 with three conditions. We obtained an average NDCG at rank 10 of 0.95, with a standard deviation of 0.04. Given the high average NDCG and low standard deviation at rank 10, it means that most of the records in top-10 are relevant and they are placed toward the front of the list.

5.5. Evaluation of Neighborhood Mapping Algorithm

In this section we evaluate our neighborhood mapping algorithm against a human generated mapping of neighborhoods. The gold standard mapping is between Yumi's neighborhood hierarchy and those of each of the LSEs of Yumi. The mapping comprises both 1:1 and 1:m correspondences. It has a total of 1,404 neighborhood pairs, where a 1:m correspondence contributes m pairs. The gold standard mapping was constructed by one of the authors by consulting several external sources, such as Wikipedia,[5] [6] Google (Map) and City of Chicago Web site.[7] In many cases the textual descriptions of neighborhoods were sought to unequivocally determine the mapping with other neighborhoods: for instance, for determining if two neighborhood names refer to the same neighborhood "a community centered on 24th and Oakley called the *Heart of Italy* or *Little Tuscany*", for finding the boundaries of a neighborhood "The neighborhood [Lincoln Square] is bounded by Bryn Mawr and Peterson Avenues on the north, Montrose Avenue on the south, Ravenswood Avenue on the east and the Chicago River on the west." or for finding inclusion relationships between regions, for instance, Wrigleyville (in Yelp.com) is a subregion of Lakeview (in Yumi) because "Lake View is unofficially divided into smaller neighborhood enclaves: Lakeview East, West Lakeview and Wrigleyville." Our mapping algorithm achieves an F-score = 86.2%, Precision = 77.1% and Recall = 97.6%. Note that recall impacts the relevance of results returned by a metasearch engine for a neighborhood query. A poor precision influences the online computation time of a query because of the increased number of retrieved objects from nonrelevant neighborhoods that need to be pruned out at query time. (Recall the local service in neighborhood problem Section 2.2). 91% of the incorrect mappings are between adjacent neighborhoods and 1:m mappings introduce the vast majority of them. For example, West Town in Yumi is mapped into the union of Noble Square, Ukrainian Village, Wicker Park, and Near West Side, out of which Near West Side is incorrect. The relatively poor precision is due to MBRs; better neighborhood approximations (e.g., by octagons) may improve precision.

5.6. Evaluation of Neighborhood Queries

The goal of this experiment is to assess the contribution of our neighborhood query processing algorithm to the quality of query answering. For this experiment, we posted 166 distinct neighborhood queries to two versions of the MSE: Yumi and YumiNoN. The former includes the mapping between the neighborhoods according to the algorithm in Section 2.1 and the local service in neighborhood algorithm (Section 2.2). YumiNoN does not use the latter algorithm and implements the following neighborhood mapping strategy. A neighborhood h in the target neighborhood hierarchy is mapped into a source hierarchy as follows. If h has a match in the source hierarchy then h is mapped into its match. Otherwise, h is mapped into the set of neighborhoods in the source hierarchy that intersects h.

For each query, in the top-10 list of retrieved restaurants, we counted the number of restaurants that are outside the boundaries of h, and thus do not satisfy the query. On the average Yumi has 0.8 restaurants outside the boundaries of the neighborhood query, while YumiNoN has 3.9. The superiority of Yumi over YumiNoN is further emphasized by the median statistic. Yumi's median is 0, while YumiNoN is 3. In other words, in half of the queries, the top-10 list of restaurants retrieved by Yumi does not have any out of neighborhood restaurant, while the top-10 list of restaurants retrieved by YumiNoN contains at least 3 out of neighborhood restaurants.

5.7. Queries Benefiting a Metasearch Engine

In this section we present additional experimental observations that support the claim that a MSE performs better than a single search engine for georeferenced objects. For this experiment, we posted over 80 queries to Yumi, Zagat and some other search engines. The queries were then manually investigated.

[5]http://en.wikipedia.org/wiki/Community_areas_in_Chicago.
[6]http://en.wikipedia.org/wiki/Chicago_metropolitan_area.
[7]http://www.cityofchicago.org/city/en/depts/doit/supp_info/citywide_maps.html.

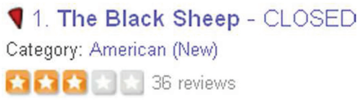

1. The Black Sheep - CLOSED
Category: American (New)
36 reviews

Fig. 11. Example of a closed restaurant.

Query by Specific Dish. When looking for specific dishes (like Carpaccio, Tapas) no or very few results are returned by some of the search engines, for instance, Zagat. Whereas when we search for a cuisine or type of food ("Italian," "Greek," "Japanese") more results are returned. Yumi performs significantly better for dish type queries as it aggregates the records from multiple search engines.

Restaurants Not Yet/No Longer in Business. LSEs may return restaurants that are no longer in business or not yet operating. Figure 11 shows an example of a closed restaurant returned by Yelp. First, we recognize these restaurants by identifying certain key noun phrases in their description (e.g., "close" or "coming soon"). Second, we discard them from the unified list of results.

Erroneous Value in Attribute Query. Let Q be a query. An LSE may not return a record r for Q, even if r may strictly satisfies Q, because in the LSE r has an erroneous value for one of the attributes in Q. We show that the MSE may still return r despite of this LSE. Consider the query $Q = ($`Cuisine` = "Chinese"; `Neighborhood` = "Chinatown"). Suppose we have four LSEs: Metromix, DexKnows, DineSite and MenuIsm. r is the restaurant Phoenix. Phoenix is placed in the neighborhood Near South Side by Metromix and it does not return Phoenix for Q. DexKnows does not return the restaurant Phoenix, because the restaurant is not tagged in the Chinese cuisine. DineSite and MenuIsm return Phoenix. MenuIsm processes `Neighborhood`, but it does not process `Cuisine`, while DineSite processes `Cuisine`, but it does not process `Neighborhood`. Thus, Phoenix strictly satisfies Q and it is returned by the MSE.

In summary, *Yumi is significantly better than any of the individual search engines.* The median relative improvement of Yumi over the LSEs is 92% (based on the column RelImp10 in Table V).

5.8. Evaluation of Local Service Resolution

For this task we used four datasets: the dataset Restaurant from RIDDLE repository[8] and the datasets 12Q, Hotels and Tax Return compiled by ourselves. Restaurant consists of two lists of restaurants of 331 and 533 records, respectively, with a gold standard of 112 matched pairs. We will determine if one business record from one list and one from the other list are the same. All possible pairs are considered. 12Q is generated as follows: 12 queries were submitted to Yumi; for each query, 3 lists of results were selected at random from the 9 possible lists, then we manually matched the 3 lists against each other to determine the gold standard of matched pairs. In total we generated 36 pairs of lists. The lists contain between 10 and 60 records. The gold standard has in total 405 matched pairs. Hotels and Tax Return were similarly obtained: we used 5 queries per domain. Hotels has 122 matches and 6,759 nonmatches. Tax Return has 76 matches and 4,833 nonmatches. These two domains are used to show that our approach applies to other local service domains and that because local services have domain independent characteristics, the matching rules learnt in Restaurant apply in Hotels and Tax Return domains.

We report our experimental findings using the traditional Precision, Recall and F-score measures. The results are shown in Table VII. We employed the same list of

[8]www.cs.utexas.edu/users/ml/riddle/data.html.

Table VII. The Local Services Resolution Experiment

Sources	Purpose	Prec.	Recall	F-score
12Q	learn/test	99.3%	97.8%	98.6%
Restaurant	test	97.4%	98.2%	97.8%
Restaurant_{10}	learn/test	98.7%	97.7%	98.2%
Restaurant_{20}	learn/test	97.6%	96.7%	97.1%
Hotels	test	100%	94.6%	97.2%
Tax Returns	test	99.6%	99.5%	99.1%

stopwords in all three domains. We use 12Q for learning and testing the decision tree (Section 3). We set up the training phase with 10-fold cross validation and split 12Q in two thirds to the training set and one third to the testing set. We achieve an average F-score of 98.6%. This is a remarkable effective classifier. We further tested the decision tree on a third party dataset, Restaurant, achieving F-score = 97.8%, which is also remarkably high. We also obtain the F-scores 97.2% and 99.1% on Hotels and Tax Return, respectively. In the light of these results, we believe that our solution likely generalizes to many local services domains.

We are aware of two other works that employed Restaurant in their experiments [Zardetto et al. 2010; Minton et al. 2005]. This gives us the opportunity to compare our method with two previous methods on the task of matching local services. The former develops a clustering algorithm using mixture models. It uses the attributes: `Name`, `Address`, `City` and `Type`. We do not use the attribute `Type`, but we use the attribute `Phone`. This is a stochastic system and it was run 10 times on the Restaurant dataset. The latter is a supervised learning algorithm. It explores several types of features, including those based on TF-IDF, which are then passed to a Support Vector Machine. It uses the same attributes that we use. It follows the experimental methodology in Bilenko and Mooney [2003]: Restaurant is randomly split into 2 folds for cross-validation; the results are reported over 20 random splits, where for each split the two folds are used alternately for training and testing; the average of maximum F-measure values over the 20 evaluated folds is reported. The two works report F-scores of 89.8% and 94.6%, respectively, for Restaurant. We obtain F-score = 97.8% on Restaurant when training on 12Q. We perform two additional experiments on Restaurant to make the evaluation more closely comparable with those in the two previous methods. In the first experiment, we randomly split Restaurant in 2 folds as in Minton et al. [2005] and repeat the experiment 10 times. We achieve an average F-score of 98.2% (Table VII, Restaurant_{10}). In the second experiment, we repeat the experiment 20 times, following the same split procedure. We achieve an average F-score of 97.1% (Table VII, Restaurant_{20}). In the light of these experiments, our method is notably better for matching local services.

6. RELATED WORK

Our work finds similarities to two important lines of research: result merging and ranking, and information integration. We briefly cover related work in these areas.

The problem of integrating databases on the Web (aka deep Web) has received significant attention in recent years [Barbosa and Freire 2007; Dragut et al. 2006; He et al. 2003; He and Chang 2003; Liu et al. 2010; Zhang et al. 2005]. The work has focused on problems such as efficient crawling, finding the semantic equivalent fields among a set of interfaces and construction of integrated query interfaces. In this work we show that to obtain relevant results from the sources for a query, the values used in a query must be properly translated to each source (see the neighborhood correspondence problem). He et al. [2003] discuss generation of global attributes, including merging attribute domains. It only considers domains with data types, such as string, integers or range. Neighborhoods, although a string data type, cannot be integrated using solely their string representation. They need to be regarded as 2D spatial objects for the integration to be effective. Zhang et al. [2005] describes how the queries are mapped across query interfaces, but it assumes that the problem of mapping the values of the fields across query interfaces is solved. To our knowledge, our work is the first to study the mapping of spatial objects across search engines.

Most of the work on merging lists of results is conducted in the realm of text metasearch (e.g., [Meng and Yu 2010; Wu and McClean 2007; Si and Callan 2003]), where it is assumed that the underlying sources run *unstructured text queries* and return lists of documents. The issues here are how to estimate/convert the scores/ranks of the documents in the local lists into global scores such that a unique ranked list is computed. Local scores are usually not known. The global score of a document is usually obtained by a formula that involves the estimated scores of the documents in the individual lists and the information about the sources.

One main distinction between our environment and that of text metasearching is that in ours the queries are lists of attribute-value pairs. We showed that the attributes mentioned in a query can improve the result ranking (Section 4.3). We are not aware of any earlier work utilizing the query capabilities of the underlying sources to determine the (strict) satisfaction of the query conditions by returned records. Another distinction is that in our setting, attributes not necessarily specified in a query (e.g., "user rating"), influence the ranking.

Other related work deals with the problem of ranking of georeferenced objects [Marian et al. 2004; Cao et al. 2010; Venetis et al. 2011]. These works are not in the realm of integration systems; they propose rankings for individual search engines. For instance, [Marian et al. 2004] aims to rank the results returned by *a* search engine when this has incomplete information about the attributes of an object. For example, suppose Zagat would not provide information about user ratings and price ranges of restaurants. In this setting, for each record returned by Zagat additional sources are queried to obtain information about user ratings and price ranges. For instance, the price range of each restaurant returned by Zagat is retrieved from New York Times review Web site. Venetis et al. [2011] propose a query log-based ranking. It analyses the directions logs (obtained from Google Maps log) to infer a ranking of places in local search. Prestige-based ranking [Cao et al. 2010] hypothesizes that a relevant object whose nearby objects are also relevant to a query is preferable when compared to a relevant object without relevant nearby objects. We plan to incorporate this feature into our ranking algorithm.

7. CONCLUSION

In this work we describe Yumi, a system that integrates spatial objects from multiple LSEs. It includes a number of innovations: a ranking algorithm that uses additional information (e.g., user rating) besides those employed in traditional ranking algorithms, a neighborhood query processing algorithm and a local services resolution algorithm. Theoretically, we present a proof of APX-hardness of finding the set of neighborhoods in a search engine that best approximates the area of a neighborhood in a different search engine. APX-hard means that, if $P \neq NP$, there is no polynomial-time approximation algorithm. We experimentally show the local search retrieval effectiveness of Yumi. The prototype is deployed online.

We believe that Yumi is generic enough to be applied to other domains such as hotels, health care offices, and law offices. The only parts of Yumi that require changes are the communication modules (e.g., query formatting) and wrappers for incorporating individual LSEs into the metasearch engine.

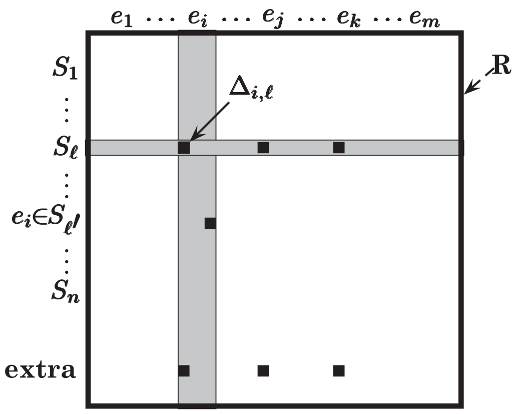


Fig. 12. Geometric view of the grid and unit squares (not drawn to scale and does not show all the rectangles).

APPENDIXES

A. INAPPROXIMABILITY OF THE RECTANGLE COVERING (TILING) PROBLEM

The area of a rectangle R is denoted by Area(R). The tiling problem is defined as follows: given an input set $\mathcal{S} = \{R_1, R_2, \ldots, R_n\}$ of n rectangles and another rectangle R, find a subset $\mathcal{S}' \subseteq \mathcal{S}$ of covering rectangles such that $R \subseteq \bigcup_{R_i \in \mathcal{S}'} R_i$ that minimizes the total area of the region covered outside R.

THEOREM A.1. *Assuming* P $\neq$ NP, *our tiling problem does not admit an* $(1 + \delta)$-*approximation algorithm for any* $\delta < 5.4/777.6 \approx 0.0069$.

PROOF. Our reduction is from the 3-SET-PACKING problem (an equivalent reformulation of the maximum independent set problem for 3-regular graphs) defined as follows: given as input an universe $\mathcal{U} = \{e_1, e_2, \ldots, e_m\}$ of m elements and a collection of $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ of n subsets of $\mathcal{U}$ satisfying **(a)** $|S| = 3$ for each $S \in \mathcal{S}$, **(b)** every element $u \in \mathcal{U}$ appears in exactly two sets of $\mathcal{S}$ and **(c)** for any two sets $S, S' \in \mathcal{S}$ we have $|S \cap S'| \leq 1$, find a subcollection $\mathcal{S}' \subset \mathcal{S}$ of maximum size such that every pair of sets in $\mathcal{S}'$ are mutually disjoint. It was shown in Berman and Karpinski. [1999] that, for any language $L \in$ NP and for every constant $\varepsilon > 0$, there exists a polynomial-time reduction that given an instance I of L produces an instance of $I' = (\mathcal{U}, \mathcal{S})$ of 3-SET-PACKING with $(3 \times 284n')/2 = 426n'$ elements in $\mathcal{U}$ and $284n'$ sets in $\mathcal{S}$ such that the following two conditions hold: (1) if $I \in L$ then I' has a has at least $(140 - \varepsilon)n'$ disjoint sets, and (2) if $I \notin L$ then I' has at most $(139 + \varepsilon)n'$ disjoint sets.

We reduce an instance of 3-SET-PACKING, as described above, to the tiling problem. For notational convenience, let $m = 426n'$, $n = 284n'$ and denote the maximum number of disjoint sets in our instance of 3-SET-PACKING by Ψ. We have m vertical strips corresponding to m elements and n horizontal strips corresponding to the n sets of 3-SET-PACKING, as shown shaded in Figure 12. These strips define a grid in the natural way. The bounding box of this grid, shown by thick solid lines, is the rectangle R whose area we would need to cover with other rectangle. For a set, say $S_\ell = \{e_i, e_j, e_k\}$, we have three unit squares $\Delta_{i,\ell}$, $\Delta_{j,\ell}$ and $\Delta_{k,\ell}$, at the intersection of the horizontal strip for S_ℓ, and the vertical strips for e_i, e_j and e_k, respectively, as shown. The two unit squares in the vertical strip for an element e_i are horizontally offset so that their projections on the x-axis are separated. We also have an *additional* set of m unit squares, one for each element e_i, in a row vertically aligned with the unit square for the first occurrence of the element. Our rectangles that can be used to cover R are as follows. We have a set of rectangles whose union covers exactly the area of R *excluding* all the unit squares. For every set, say $S_\ell = \{e_i, e_j, e_k\}$, we have a *set-rectangle* $R_{\{i,j,k\}}$ that cover the unit squares on the horizontal strip for S_ℓ such that $\mathsf{Area}(R_{\{i,j,k\}} \setminus R) = 3$. For every element e_i, there is an additional *upward* rectangle UP_i containing the additional unit square for e_i and the unit square for the first occurrence of e_i but not intersecting the unit square for the second occurrence of e_i such that $\mathsf{Area}(\mathsf{UP}_i \setminus R) = 1.41$ and there is an additional *downward* rectangle DOWN_i containing the additional unit square for e_i and the unit square for the second occurrence of e_i but not intersecting the unit square for the first occurrence of e_i such that $\mathsf{Area}(\mathsf{DOWN}_i \setminus R) = 1.41$. Let OPT denote the optimum value of the objective function of our instance of the tiling problem. The theorem is proved by showing that: (completeness) if $\Psi \geq (140 - \varepsilon)n$ then $\mathsf{OPT} \leq (1029.12 + 1.23\,\varepsilon)n'$ and (soundness) if $\Psi \leq (139 + \varepsilon)n$ then $\mathsf{OPT} \geq (1030.35 - 1.23\,\varepsilon)n'$. $\square$

B. A LOGARITHMIC APPROXIMATION FOR RECTANGLE COVERING (TILING) PROBLEM

Let OPT denote the optimal cost of an instance of our tiling problem. Recall the standard definition of the approximation algorithm literature: a polynomial time algorithm is a randomized α-approximation algorithm for our tiling problem, for some $\alpha \geq 1$, provided there is a randomized algorithm that outputs of valid solution of cost κ satisfying $\mathbb{E}[\kappa] \leq \alpha\,\mathsf{OPT}$. As before, n denotes the number of rectangles in $\mathcal{S}$.

THEOREM B.1. *There is a randomized $O(\log n)$-approximation algorithm for our tiling problem.*

PROOF. We start with some preprocessing of the input to help describe our algorithm. We extend the horizontal and vertical sides of the rectangles in $\mathcal{S}$ and the rectangle R. This partitions the bounding box into a number of rectangles. We call each such rectangle an "elementary rectangle" (ER). We name these t ERs in arbitrary order, say $\mathsf{er}_1, \mathsf{er}_2, \ldots, \mathsf{er}_t$, where $t = O(n^2)$ (Figure 7). Note that, for any ER and any rectangle, the ER is either completely inside or completely outside the rectangle.

An ILP for our tiling problem is shown in Figure 5. The constraints of the form $y_j \geq x_i$ ensure that an ER outside R is selected in the calculation of the cost of the solution if and only if at least a rectangle containing that ER is selected. Constraints of the form $\sum_{\mathsf{er}_j \text{ is inside } R_i} x_i \geq 1$ ensure that every ER inside R is covered by some selected rectangle. The objective $\sum_{\mathsf{er}_j \text{ is outside } R} \mathsf{Area}(\mathsf{er}_j)\, y_j = \sum_{\substack{\mathsf{er}_j \text{ is outside } R \\ \text{and } y_j = 1}} \mathsf{Area}(\mathsf{er}_j)$ measures the sum of areas of the ERs outside R that are contained in the selected rectangles. Obviously, the objective value of an optimal solution of this ILP is exactly OPT. The LP relaxation and randomized post-processing procedure for this ILP is then carried out as shown in Figure 6. Let $\mathsf{OPT}^{\mathsf{LP}} = \sum_{\mathsf{er}_j \text{ is outside } R} \mathsf{Area}(\mathsf{er}_j)\, y_j^{\mathsf{LP}} \leq \mathsf{OPT}$ denote the cost of the optimal solution of the LP-relaxation.

The solution returned by the randomized algorithm in Fig. 6 is a valid solution since any part of the interior of R *not* covered by the randomized rounding step is surely covered during the execution of the greedy step. Thus, it remains to show that $\mathbb{E}[\mathsf{Area}(\bigcup_{R_i \in \mathcal{S}'} R_i \setminus R)] = O(\log n)\,\mathsf{OPT}$. Let $Y = \{y_j \mid y_j = 1\}$ be the set of y_j's that were set to 1 during the execution of the approximation algorithm. We further partition Y into two sets Y^r and Y^g that contain the y_j's that were set to 1 during the randomized rounding and greedy step, respectively.

Consider a specific ER that is *outside* R, say er_j, and let $R_{j_1}, R_{j_2}, \ldots, R_{j_{p_j}}$ be the p_j rectangles that contain this ER with $1 \geq x_{j_1}^{\mathsf{LP}} \geq x_{j_2}^{\mathsf{LP}} \geq \cdots \geq x_{j_{p_j-1}}^{\mathsf{LP}} \geq x_{j_{p_j}}^{\mathsf{LP}} \geq 0$. Note that any optimal solution of the LP satisfies $y_j^{\mathsf{LP}} = x_{j_1}^{\mathsf{LP}}$, since otherwise the value of y_j^{LP} can be decreased to $x_{j_1}^{\mathsf{LP}}$ contradicting the optimality of the solution. Then, it follows that

$\mathbb{E}[y_j] = \mathbb{E}[x_{j_1}] \leq (2\ln n)\, x_{j_1}^{\mathsf{LP}} = (2\ln n)\, y_j^{\mathsf{LP}}$ and thus

$$\begin{aligned}\mathbb{E}\left[\sum_{y_j \in Y^r} \mathsf{Area}(\mathsf{er}_j)\, y_j\right] &= \sum_{y_j \in Y^r} \mathsf{Area}(\mathsf{er}_j)\, \mathbb{E}[y_j] \leq \sum_{y_j \in Y^r} \mathsf{Area}(\mathsf{er}_j)(2\ln n)\, y_j^{\mathsf{LP}} \\ &= (2\ln n) \sum_{y_j \in Y^r} \mathsf{Area}(\mathsf{er}_j) y_j^{\mathsf{LP}} \leq (2\ln n)\, \mathsf{OPT}^{\mathsf{LP}} \leq (2\ln n)\, \mathsf{OPT}.\end{aligned}$$

Next, we analyze the greedy step. Let $R^g = \{R_{i_1}, R_{i_2}, \ldots, R_{i_\ell}\} \subseteq \mathcal{S}$ be the set of $\ell > 0$ rectangles added to the solution during the greedy step. Then, we have

$$\mathbb{E}\left[\sum_{y_j \in Y^g} \mathsf{Area}(\mathsf{er}_j)\, y_j\right]. \leq \mathbb{E}\left[\sum_{j=1}^{\ell} \mathsf{Area}(\, R_{i_j} \backslash R\,) x_{i_j}\right] = \sum_{j=1}^{\ell} \mathsf{Area}(\, R_{i_j} \backslash R\,)\, \mathbb{E}[x_{i_j}]. \tag{1}$$

Note that

$$\mathsf{Area}(\, R_{i_j} \backslash R\,) = \min_{R_\ell \text{ contains } \mathsf{er}_j} \{\mathsf{Area}\,(R_\ell \backslash R\,)\} \leq \mathsf{OPT}$$

since er_j must be covered by some rectangle in any valid solution. Since R_{i_j} was selected in the greedy step because $y_j = 0$, that is, er_j inside R was not covered during the randomized rounding step, we have

$$\Pr[y_j = 0] = \prod_{\mathsf{er}_j \text{ is inside } R_i} \Pr[x_i = 0] = \left(\prod_{\mathsf{er}_j \text{ is inside } R_i} (1 - x_i^{\mathsf{LP}})\right)^{2\ln n}.$$

Since $\sum_{\mathsf{er}_j \text{ is inside } R_i} x_i \geq 1$, the function $\prod_{\mathsf{er}_j \text{ is inside } R_i} (1 - x_i^{\mathsf{LP}})$ is maximized when all the x_i^{LP}'s are equal, that is, $x_i^{\mathsf{LP}} = 1/\alpha$ where α is the number of rectangles containing er_j. This gives

$$\Pr[y_j = 0] \leq \left[\left(1 - \frac{1}{\alpha}\right)^{\alpha}\right]^{2\ln n} < \mathrm{e}^{-2\ln n} = n^{-2},$$

which shows

$$\mathbb{E}[x_{i,j}] = \Pr[y_j = 0] < n^{-2}. \tag{2}$$

Using (1) and (2), we can compute the expected cost of the greedy step as follows:

$$\mathbb{E}\left[\sum_{y_j \in Y^g} \mathsf{Area}(\mathsf{er}_j)\, y_j\right] \leq n^{-2} \sum_{j=1}^{\ell} \mathsf{Area}(\, R_{i_j} \backslash R) \leq n^{-2}\, \ell\, \mathsf{OPT} \leq \mathsf{OPT}/n,$$

and thus the total expected cost of our solution is

$$\begin{aligned}\mathbb{E}\left[\sum_{y_j \in Y} \mathsf{Area}(\mathsf{er}_j)\, y_j\right] &= \mathbb{E}\left[\sum_{y_j \in Y^r} \mathsf{Area}(\mathsf{er}_j)\, y_j\right] + \mathbb{E}\left[\sum_{y_j \in Y^g} \mathsf{Area}\,(\mathsf{er}_j)\; y_j\right] \\ &\leq \left(2\ln n + \frac{1}{n}\right) \mathsf{OPT} < 3\ln n\, \mathsf{OPT}. \quad \square\end{aligned}$$

REFERENCES

J. A. Aslam and M. Montague. 2001. Models for metasearch. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 276–284.

T. T. Avrahami, L. Yau, L. SI, and J. Callan. 2006. The fedlemur project: Federated search in the real world. *J. Amer. Soc. Inf. Sci. Technol*.

L. Barbosa and J. Freire. 2007. Combining classifiers to identify online databases. In *Proceedings of the International World Wide Web Conference*.

N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. 1990. The r*-tree: an efficient and robust access method for points and rectangles. *Proceedings of the ACM SIGMOD International Conference on Management of Data.* 19, 2, 322–331.

P. Berman and M. Karpinski. 1999. On some tighter inapproximability results. In *Proceeding of the 26th International Colloquium on Automota, Languages, and Programming*. 200–209.

M. Bilenko and R. J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 39–48.

V. Borkar, K. Deshmukh, and S. Sarawagi. 2001. Automatic segmentation of text into structured records. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

X. Cao, G. Cong, and C. S. Jensen. 2010. Retrieving top-k prestige-based relevant spatial web objects. *Proc. VLDB Endow.*

Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*. ACM, New York, 129–136.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. 2002. Smote: synthetic minority over-sampling technique. *J. Artif. Int. Res.* 16, 321–357.

G. V. Cormack, C. L. A. Clarke, and S. Buettcher. 2009. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 758–759.

E. Dragut, F. Fang, P. Sistla, C. Yu, and W. Meng. 2009. Stop word and related problems in web interface integration. *Proc. VLDB Endow.* 2, 1, 349–360.

E. Dragut, W. Wu, P. Sistla, C. Yu, and W. Meng. 2006. Merging source query interfaces on web databases. In *Proceedings of the International Conference on Data Engineering*.

A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. 2007. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.* 19, 1.

Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. 2003. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.* 4, 933–969.

S. Guo, X. Dong, D. Srivastava, and R. Zajac. 2010. Record linkage with uniqueness constraints and erroneous values. *Proc. VLDB Endow.* 3, 1.

D. Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York.

B. He and K. Chang. 2003. Statistical schema matching across web query interfaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

H. He, W. Meng, C. Yu, and Z. Wu. 2003. WISE-integrator: An automatic integrator of Web search interfaces for e-commerce. In *Proceedings of the International Conference on Very Large Databases*.

Q. Hu, J. Huang, and J. Miao. 2011. A robust approach to optimizing multi-source information for enhancing genomics retrieval performance. *BMC Bioinformatics* 12, 1–9.

K. Järvelin and J. Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.* 20.

T. Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 133–142.

H. Köpcke and E. Rahm. 2010. Frameworks for entity matching: A comparison. *Data Knowl. Eng.* 69, 197–210.

B. Liu. 2007. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer.

W. Liu, X. Meng, and W. Meng. 2010. ViDE: A vision-based approach for deep web data extraction. *IEEE Trans. Knowl. Data Eng.* 22.

C. D. Manning, P. Raghavan, and H. Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

A. Marian, N. Bruno, and L. Gravano. 2004. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.* 29, 2.

W. Meng and C. Yu. 2010. *Advanced Metasearch Engine Technology*. Morgan & Claypool.

S. N. Minton, C. Nanjo, C. A. Knoblock, M. Michalowski, and M. Michelson. 2005. A heterogeneous field matching method for record linkage. In *Proceedings of the IEEE International Conference on Data Mining*. 314–321.

M. Montague and J. A. Aslam. 2002. Condorcet fusion for improved retrieval. In *Proceedings of the International Conference on Information and Knowledge Management*.

F. P. Preparata and M. I. Shamos. 1985. *Computational Geometry: An Introduction* 3rd Ed. Springer.

L. Si and J. Callan. 2003. A semisupervised learning method to merge search engine results. *ACM Trans. Inf. Syst.*

P. Venetis, H. Gonzalez, C. S. Jensen, and A. Halevy. 2011. Hyper-local, directions-based ranking of places. *Proc. VLDB Endow*.

S. Wu and S. McClean. 2007. Result merging methods in distributed information retrieval with overlapping databases. *Inf. Retrieval* 10, 297–319.

D. Zardetto, M. Scannapieco, and T. Catarci. 2010. Effective automated object matching. In *Proceedings of the International Conference on Data Engineering*.

Z. Zhang, B. He, and K. Chang. 2005. Light-weight domain-based form assistant: Querying web databases on the fly. In *Proceedings of the International Conference on Very Large Databases*.

B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han. 2012. A Bayesian approach to discovering truth from conflicting sources for data integration. *Proc. VLDB Endow.* 5, 6.

Received January 2013; revised March 2014, July 2014; accepted July 2014