

# MashRank: Towards Uncertainty-Aware and Rank-Aware Mashups

Mohamed A. Soliman    Mina Saleeb    Ihab F. Ilyas  
 School of Computer Science, University of Waterloo, Canada  
 {m2ali,msaleeb,ilyas}@cs.uwaterloo.ca

**Abstract**— Mashups are situational applications that build data flows to link the contents of multiple Web sources. Often times, ranking the results of a mashup is handled in a materialize-then-sort fashion, since combining multiple data sources usually destroys their original rankings. Moreover, although uncertainty is ubiquitous on the Web, most mashup tools do not reason about or reflect such uncertainty.

We introduce MashRank, a mashup tool that treats ranking as a first-class citizen in mashup construction, and allows for rank-joining Web sources with uncertain information. To the best of our knowledge, no current tools allow for similar functionalities. MashRank encapsulates a new probabilistic model reflecting uncertainty in ranking, a set of techniques implemented as pipelined operators in mashup plans, and a probabilistic ranking infrastructure based on Monte-Carlo sampling.

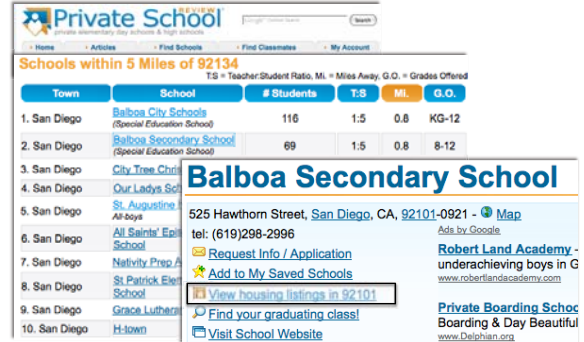
## I. INTRODUCTION

The recent proliferation of mashup tools [1], [2], [3] allows Web users to build simple applications that accomplish advanced search tasks by joining multiple sources. Current mashup tools focus on building data flows that join different Web sources. Ranking the results of a mashup is usually handled in the final step, after all results are computed, since combining multiple sources usually destroys their original rankings. Moreover, when Web sources contain uncertain information, such uncertainty is usually ignored in the final ranking.

Rank-aware (top- $k$ ) processing has been extensively studied in database literature (e.g., [4],[5]). Top- $k$  queries produce the  $k$  top ranked query results based on scores computed by a (typically monotone) scoring function. A join query augmented with a scoring function and a parameter  $k$ , referred to as “rank join query”, reports the top- $k$  join results based on the given scoring function. Scoring models in rank-aware processing usually assume that tuple scores are single-valued. In situations where the underlying data and/or the scoring function do not conform to this assumption, current ranking semantics and processing techniques become inapplicable.

In this demonstration, we focus on enabling *ranked mashups*, where the scores of ranked entities may be uncertain and represented as intervals enclosing possible score values, as illustrated by the following example:

*Example 1:* Amy needs to buy a house, and sign up her kids at nearby schools. Amy plans to search first for schools in the area she is moving to. For each interesting school Amy finds, she will next search for available surrounding houses.



(a) School search on www.privateschoolreview.com

Address	Beds/ Baths	Sq. Ft.	Lot Size	Price
Chula Vista, CA	4/4	1,670	N/A	\$250,000 - \$285,000
Escondido, CA	3/2	1,120	6,000 Sq. Ft.	\$189,000
Chula Vista, CA	4/3	1,930	4,447 Sq. Ft.	\$309,900
Chula Vista, CA	5/4	3,765	10,193 Sq. Ft.	\$638,000
Chula Vista, CA	2/3	937	N/A	\$119,000 - \$140,000
Escondido, CA	5/3	1,899	N/A	\$309,900
Chula Vista, CA	3/2	1,552	N/A	\$225,000
Chula Vista, CA	4/3	2,233	N/A	\$379,900
Chula Vista, CA	3/3	1,400	N/A	\$199,900 - \$229,900

(b) Houses matching selected school

Fig. 1. School-House Mashup

The scoring function that specifies Amy’s preferences is the summation of house price and school tuition for 5 years. A possible SQL-like formulation for what Amy is looking for is the following rank join query:

```
SELECT *
FROM School, House
WHERE distance(School.loc,House.loc) < 3
ORDER BY School.tuition*5+ House.price
LIMIT 10
```

Example 1 involves a simple mashup that joins schools and houses Web sources (e.g., privateschoolreview.com and mls.com) based on zip code. Figure 1 shows screenshots of such mashup, where the user finds relevant schools, and then for each interesting school, nearby houses can be viewed by simply one click.

Web data often involve imprecision and incompleteness. Figure 1(b) shows the details of the houses matching a

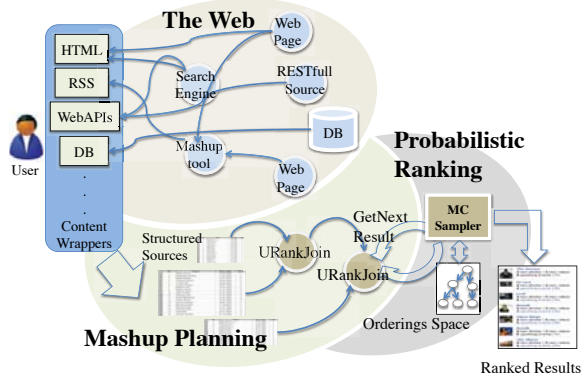


Fig. 2. Layout of MashRank

selected school in Example 1. Prices (scores) of the shown houses are expressed as either single values or ranges. Such imprecision can be attributed to multiple sources including privacy concerns, where exact price is only known upon contacting the owner, and presentation style, where a search hit is an aggregate of other hits (e.g., a hit that represents an apartment complex rather than a specific apartment). In a sample of search results we scraped from `apartments.com` and `carpages.ca`, the percentage of apartment records with uncertain rents was 65%, while the percentage of car records with uncertain prices was 10%. Joining arbitrary Web sources further complicates the problem by magnifying the effect of uncertainty as more sources are joined. Moreover, eliminating uncertainty (e.g., by reducing uncertain ranges into their expected values) can result in unreliable rankings [6].

We introduce MashRank, a mashup tool that enables rank-joining Web sources with uncertain information. MashRank extends our work in URank [7], [8], [9], [10] (a system that supports the evaluation of ranking and aggregation queries on uncertain data) to support uncertain rank join (URANKJOIN) queries (cf. Section II-B).

At the heart of MashRank, are the following components: (1) novel *probabilistic partial order* model [9] that encodes a space of possible orderings of the results of a mashup (originating from score uncertainty), (2) pipelined operator implementation that allows for efficient mashup execution, and (3) Monte-Carlo sampling infrastructure that produces ranked results under different probabilistic ranking semantics. We discuss the technical details of these components in [6].

**MashRank Layout.** Figure 2 shows the layout of MashRank. The underlying data model in MashRank is a simple relational model in which mashup output is represented as a set of tuples. In order to support multitude of Web data models, MashRank employs different content wrappers to consume contents from different types of Web sources (e.g., RSS feeds and simple HTML) and wrap the contents into structured tuples. MashRank also allows for the consumption of structured tuples from local/remote databases through database connectivity standards.

The relational sources, computed by the wrappers, are joined in a mashup query plan composed of pipelined URANKJOIN operators. MashRank allows manual construc-

tion of such plan, or using system recommendations based on available rank join algorithms maintained by MashRank backend (we give more details in Section III). The mashup plan reports its results incrementally (i.e., as needed) to a probabilistic ranking infrastructure that uses Monte-Carlo sampling techniques to compute and produce ranked mashup results.

Our contributions are summarized as follows:

- *Efficient rank join processing:* MashRank leverages relational rank join algorithms to incrementally compute and rank the results of a mashup by early-pruning low-ranked results using pipelined operators embedded in mashup plans. MashRank also allows for using different rank join algorithms, which adds flexibility in mashup construction.
- *Rank join under score uncertainty:* To the best of our knowledge, MashRank is the first mashup tool to allow for joining and ranking Web sources under score uncertainty (e.g., Example 1).
- *Integrating relational and probabilistic processing:* MashRank encapsulates an infrastructure that interleaves relational processing (in particular, rank-join) with probabilistic ranking to efficiently compute a total order of the results under multiple probabilistic ranking semantics.

## II. SYSTEM OVERVIEW

We give an overview on the main components and processing techniques of MashRank.

### A. Scoring Model

MashRank represents the score of tuple  $t_i$  as a random variable defined on the real interval  $[lo_i, up_i]$ . The score random variable of  $t_i$  has a PDF  $P_i$  that encodes the likelihood of possible  $t_i$  scores. For example in Figure 1(b),  $P_i$  of the price of a house  $t_i$  is a uniform PDF, unless there are samples of other houses, similar to  $t_i$ , with single-valued prices, in which case we impute a probability distribution on possible prices of  $t_i$  [6].

The adopted scoring model suggests an intuitive ordering criterion denoted ‘uncertain score dominance’, where we say that tuple  $t_i$  dominates another tuple  $t_j$  if and only if  $lo_i \geq up_j$ . Under the previous criterion, dominated tuples must appear after their dominating tuples in the final ordering of mashup results. On the other hand, tuples with intersecting score intervals are incomparable, which induces a space of possible orderings (cf. Section II-B).

### B. Uncertain Rank Join

We define the set of top- $k$  join results  $\mathcal{J}_k$  as the set of join results (i.e., the results of a mashup) that are dominated by less than  $k$  join results. We also define URANKJOIN query as a query that computes a total order  $\omega^*$  of tuples in  $\mathcal{J}_k$ .

We illustrate the previous definitions using Figure 3, which shows the set  $\mathcal{J}_3 = \{(r_1, s_2), (r_3, s_1), (r_2, s_2)\}$  in a URANKJOIN query on Relations  $R$  and  $S$ , where the join condition is equality of ‘ $jk$ ’, and the scoring function  $\mathcal{F}$  is the average of uncertain scores  $R.a_1$  and  $S.a_1$ . All join results in  $\mathcal{J}_3$  are dominated by less than 3 join results, while all other

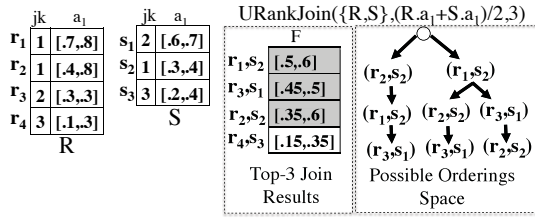


Fig. 3. URANKJOIN Example

join results (only  $(r_4, s_3)$ ) are dominated by at least 3 join results. Based on the monotonicity of  $\mathcal{F}$ , the  $lo$  and  $up$  scores of join results are given by applying  $\mathcal{F}$  to the  $lo$  and  $up$  scores of the joined base tuples. For example, the score of  $(r_1, s_2)$  is given by  $[\mathcal{F}(.7, .3), \mathcal{F}(.8, .4)] = [\frac{.7+.3}{2}, \frac{.8+.4}{2}] = [.5, .6]$ .

A key property that allows incremental computation of  $\mathcal{J}_k$  is early-pruning of join results dominated by the join result with the  $k^{th}$  largest  $lo$  score [6]. Computing  $\mathcal{J}_k$  does not thus require knowing  $P_i$ 's. However, computing  $\omega^*$  (the total order of tuples in  $\mathcal{J}_k$ ) requires considering  $P_i$ 's to reduce the space of possible orderings (e.g., shown as a tree in Figure 3) into a total order. We show in [6] that multiple probabilistic ranking semantics in current literature can be adopted to compute  $\omega^*$  (e.g., the most probable ordering [7], expected ranks [11], and uncertain rank aggregation [9]). Deciding the semantics of  $\omega^*$  can be made based on application requirements or semantics properties. We discuss computing  $\omega^*$  in Section II-D.

### C. Pipelined URANKJOIN Plans

We describe in [6] an efficient algorithm to incrementally compute  $\mathcal{J}_k$  in the order of  $up$  scores. We use two instances of a rank join algorithm, denoted  $RJ_{lo}$  and  $RJ_{up}$ , where  $RJ_{lo}$  rank-joins tuples on their overall  $lo$  scores to find exactly  $k$  join results, while  $RJ_{up}$  rank-joins tuples on their overall  $up$  scores to find all join results with  $up$  scores above the  $k^{th}$  largest score reported by  $RJ_{lo}$ .

Building URANKJOIN query plans requires implementing the previous algorithm as a query operator. We adopt the iterator model (OPEN-GETNEXT-CLOSE) as the underlying operator model. In order to build pipelined URANKJOIN operators, we need to make the algorithm independent of the parameter  $k$ . We change algorithm design to respond to incoming requests of join results ordered on either  $lo$  or  $up$  scores. The knowledge of  $k$  is only available to the root of query plan, which controls the plan execution. Similar techniques are used in [4].

A URANKJOIN plan is rooted by ULIMIT, a new operator we propose to control mashup plan execution. The operator takes two inputs  $I_{lo}$  and  $I_{up}$  representing two streams of mashup output tuples ordered on their  $lo$  scores and  $up$  scores, respectively. One GETNEXT implementation of ULIMIT is to consume  $k$  tuples from  $I_{lo}$  and to report tuples in  $I_{up}$  with scores above the  $k^{th}$  score in  $I_{lo}$ . An alternative implementation is to interleave drawing tuples from  $I_{lo}$  and  $I_{up}$ . We discuss alternative implementations in [6].

A URANKJOIN operator is a logical operator that accepts two inputs each has two sorted access paths, corresponding

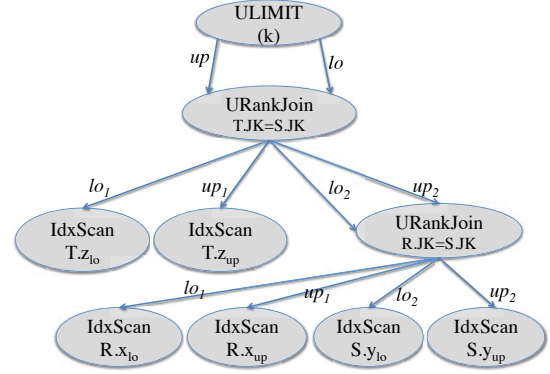


Fig. 4. A Logical URANKJOIN Query Plan

to the  $lo$  and  $up$  score orders of the two input relations. The operator produces two output tuple streams corresponding to the ranked join results based on  $lo$  and  $up$  scores.

Figure 4 gives an example logical URANKJOIN query plan. The shown plan rank-joins three relational sources  $R$ ,  $S$ , and  $T$  with uncertain scores  $x$ ,  $y$ , and  $z$ , respectively. The bottom URANKJOIN operator uses indexes (or sorted access paths) on the  $lo$  and  $up$  scores in  $R$  and  $S$  as its input access paths, while the top URANKJOIN operator uses indexes on  $T$  and the output of the bottom URANKJOIN operator as its input access paths. The ULIMIT operator consumes both  $lo$  and  $up$  inputs from the top URANKJOIN operator. Each URANKJOIN operator preserves the order of its  $lo/up$  output stream based on overall  $lo/up$  scores. The plan thus computes ranked results as requested by the ULIMIT operator, and avoids full sorting of the results in order to compute the top- $k$  results.

The physical implementation of the URANKJOIN operator can be decided based on the characteristics of the underlying computing environment. For example in environments where data is consumed from individual nodes, the operator can be implemented as two rank join operators wrapped within a physical operator with 4 inputs (the  $lo$  and  $up$  orders of the two input relations) and 2 outputs (the  $lo$  and  $up$  orders of the join results). This implementation requires, however, making other operators aware of the URANKJOIN operator input/output interface. Alternatively, in environments where replicas of data reside on multiple mirrors, the operator can be implemented as two separate rank join operators consuming data from two different mirrors, such that their execution is parallelized to improve performance.

The logical design of URANKJOIN operator does not restrict the physical ranking algorithms. Hence, different algorithms can be seamlessly integrated in physical URANKJOIN plans. Such flexibility raises interesting query optimization and cost modeling issues, which are part of our future work.

### D. Probabilistic Ranking

We describe in [6] an infrastructure to compute an ordering  $\omega^*$  of mashup results (under score uncertainty) using Monte-Carlo sampling. The idea is to draw independent samples from the score distributions of base tuples, and use these samples to

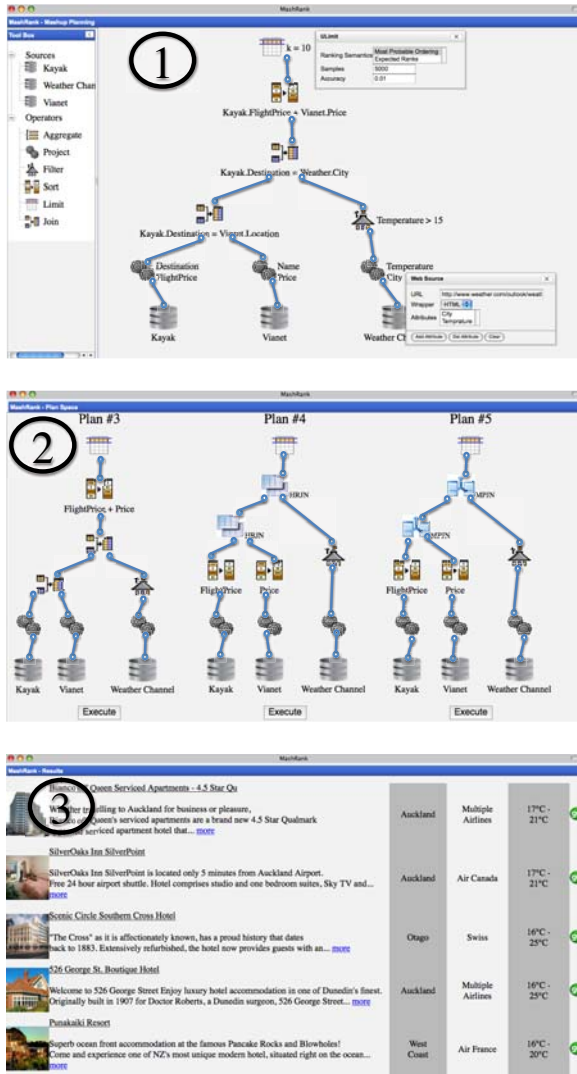


Fig. 5. MashRank Demonstration Scenario

compute  $\Pr(t, r)$ , the probability of mashup result  $t$  to appear at rank  $r$  in a random possible ordering of mashup results (cf. Sections II-B). This allows computing  $\omega^*$  under multiple probabilistic ranking semantics [7], [9], [11].

Moreover, often times Web users inspect only a small prefix of the ranked results list. Computing a full ranking of all results of a mashup in advance may not thus be required. We therefore propose additional techniques that use the incremental computation of  $\mathcal{J}_k$  by mashup plan to compute an approximation of  $\omega^*$  that improves in accuracy as more mashup results are produced. Specifically, we use our Monte-Carlo sampling infrastructure to compute bounds on  $\Pr(t, r)$  for each mashup result  $t$  produced by the underlying URANKJOIN plan. The bounds of  $\Pr(t, r)$  are used to approximate a prefix of  $\omega^*$  under multiple probabilistic ranking semantics. The bounds are progressively tightened as more results are produced. We compute the accuracy of  $\omega^*$  approximation based on maximum error in pairwise ranking of consecutive results in  $\omega^*$ .

### III. DEMONSTRATION SCENARIO

We showcase MashRank using the demonstration scenario illustrated by Figure 5: A user would like to spend a vacation in New Zealand. The user designs a mashup to join Web sources that provide New Zealand hotels information ([www.vianet.travel](http://www.vianet.travel)), flights ([www.kayak.com](http://www.kayak.com)), and weather information ([www.weather.com](http://www.weather.com)). The objective is to rank different vacation packages, at places with good weather, based on total cost. The hotel costs given by [www.vianet.travel](http://www.vianet.travel) are uncertain ranges, similar to Example 1.

- 1) The user adds Web sources to MashRank plan editor. For each source, the user selects the content wrapper, required attributes, result filters (e.g., temperature above 15°C), scoring attribute, and number of required results. Since the scoring function (total cost) involves uncertain scoring attribute (hotel costs), the mashup output cannot be deterministically ranked (i.e., the results of the mashup induce a space of possible orderings). Multiple options on how to rank the results of the mashup are available to the user, based on our discussion in Section II-D.
- 2) The mashup plan created by the user is processed by the MashRank backend to generate a number of rank-aware mashup plans (data flows) with different rank join algorithms and operator compositions.
- 3) When the user selects one of the mashup plans, the plan is executed against the chosen Web sources by activating the wrappers to consume source contents, and applying plan operators to incrementally compute ranked results. The ranked results are passed to the Monte-Carlo sampling infrastructure to report results to the user under the required probabilistic ranking semantics.

### REFERENCES

- [1] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh, "Damia: data mashups for intranet applications," in *SIGMOD*, 2008.
- [2] "Microsoft Popfly: <http://www.popfly.com/>."
- [3] "Yahoo! Pipes: <http://pipes.yahoo.com/>."
- [4] I. F. Ilyas, W. G. Aref, and A. Elmagarmid, "Supporting top-k join queries in relational databases." *VLDB Journal*, vol. 13, no. 3, 2004.
- [5] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Comput. Surv.*, vol. 40, no. 4, 2008.
- [6] M. A. Soliman, M. Saleeb, and I. F. Ilyas, "Joining Ranked Inputs under Score Uncertainty, Technical Report, University of Waterloo, 2009."
- [7] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Top-k query processing in uncertain databases," in *ICDE*, 2007.
- [8] —, "Probabilistic top-k and ranking-aggregate queries," *ACM Trans. Database Syst.*, vol. 33, no. 3, 2008.
- [9] M. A. Soliman and I. F. Ilyas, "Ranking with uncertain scores," in *ICDE*, 2009.
- [10] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "URank: formulation and efficient evaluation of top-k queries in uncertain databases," in *SIGMOD*, 2007.
- [11] G. Cormode, F. Li, and K. Yi, "Semantics of ranking queries for probabilistic data and expected ranks," in *ICDE*, 2009.