SEMA-JOIN: Joining Semantically-Related Tables Using Big Table Corpora

Yeye He
Microsoft Research
Redmond, WA, USA
yeyehe@microsoft.com

Kris Ganjam
Microsoft Research
Redmond, WA, USA
krisgan@microsoft.com

Xu Chu
University of Waterloo
Waterloo, ON, Canada
x4chu@uwaterloo.ca

ABSTRACT

Join is a powerful operator that combines records from two or more tables, which is of fundamental importance in the field of relational database. However, traditional join processing mostly relies on string equality comparisons. Given the growing demand for ad-hoc data analysis, we have seen an increasing number of scenarios where the desired join relationship is not equi-join. For example, in a spreadsheet environment, a user may want to join one table with a subject column `country-name`, with another table with a subject column `country-code`. Traditional equi-join cannot handle such joins automatically, and the user typically has to manually find an intermediate mapping table in order to perform the desired join.

We develop a SEMA-JOIN approach that is a first step toward allowing users to perform semantic join automatically, with a click of the button. Our main idea is to utilize a data-driven method that leverages a big table corpus with over 100 million tables to determine statistical correlation between cell values at both row-level and column-level. We use the intuition that the correct join mapping is the one that maximizes aggregate pairwise correlation, to formulate the join prediction problem as an optimization problem. We develop a linear program relaxation and a rounding argument to obtain a 2-approximation algorithm in polynomial time. Our evaluation using both public tables from the Web and proprietary Enterprise tables from a large company shows that the proposed approach can perform automatic semantic joins with high precision for a variety of common join scenarios.

1. INTRODUCTION

Database join is a powerful operator that combines records from two or more tables. It is extensively used in modern RDBMS systems, which is made possible in part by a long and fruitful line of research on efficient join processing [13, 18].

Traditional join processing, however, mostly relies on equality comparisons of values. While equi-joins work well in heavily curated relational database or data warehousing settings, where data are extensively cleansed and transformed into suitable formats in a process known as ETL that prepares data for downstream SQL processing, we have seen a growing demand for ad-hoc, in-situ data analysis, where extensive data transformation and preparation are too expensive to be feasible. Performing automatic join without laborious data transformation is becoming increasingly important.

As a motivating example redacted from a real use case we encountered, suppose a business analyst inside a large Internet company has a table with ads revenue of the company across different countries, as shown in Table 1(a) of Figure 1. The first column of the table lists all countries using the 2-digit ISO country code. The analyst wants to correlate the ads revenue with Internet user numbers and Internet penetration rate of each country. He could accomplish this task by searching for tables with the desired data, using the state-of-the-art table search systems such as Microsoft PowerQuery [4] or Google Web Tables [2]. Suppose he finds such

	A	B	C	D	E	F	G
1	COUNTRY	USERS	SALES		Country	Internet users	Penetration
2	US	235,455	9,124		China	568,192,066.00	42.30%
3	CN	432,000	3,992		United States	254,295,536.00	81.00%
4	JP	111,000	2,531		India	243,298,994.00	19.60%
5	IN	206,674	1,720		Japan	100,684,474.00	79.10%
6	BR	89,000	1,701		Brazil	99,357,737.00	49.80%
7	GB	71,053	1,513		Russia	75,926,004.00	53.30%
8	DE	136,000	1,470		Germany	68,296,919.00	84.00%
9	RU	14,104	1,335		Nigeria	55,000,000.00	32.90%
10	MX	105,766	947		United Kingdom	54,861,245.00	87.00%

Figure 1: Table 1(a) (left): company sales data by country code; Table 1(b) (right): a table with Internet penetration by country.

	A	B	C	D	E	F	G	H
1	Stock Ticker	Market Cap	Change %		Organization	Total '89 -'13	Dem	Rep
2	MSFT	380.15B	5450%		AT&T Inc	$59,456,031	41%	58%
3	ORCL	170.54B	118%		Goldman Sachs	$47,497,295	52%	44%
4	INTC	157.73B	3304%		United Parcel Service	$34,216,308	35%	64%
5	GE	255.88B	469%		Citigroup Inc	$33,910,357	48%	50%
6	UPS	94.27B	49%		JPMorgan Chase & Co	$33,752,009	47%	51%
7	WMT	341.25B	1960%		Blue Cross/Blue Shield	$31,978,236	36%	63%
8	LMT	59.25B	610%		Microsoft Corp	$31,226,914	55%	44%
9	BAC	178.36B	194%		General Electric	$30,392,306	47%	52%
10	IBM	162.36B	412%		Lockheed Martin	$29,405,272	42%	57%

Figure 2: Table 2(a) (left): market capitalization by stock ticker; Table 2(b) (right): political contribution by company.

	A	B	C	D	E	F	G	H
1	City	Crime rate (per 100,000 ppl)	Unemployment Rate %		State	Crime rate (per 100,000 ppl)	Unemployment Rate %	Median Income
2	New York	2675	9.6		Ala.	420.1	6.6	$41,415
3	Los Angeles	3851	13.8		Alaska	606.5	6.8	$67,825
4	Chicago	5921	11.7		Ariz.	405.9	6.9	$46,709
5	Houston	7060	8.5		Ark.	480.9	6.2	$38,758
6	Philadelphia	5569	10.8		Calif.	411.1	7.3	$57,287
7	Phoenix	7094	11.2		Colo.	320.2	4.7	$55,387
8	Las Vegas	5582	14.6		Conn.	272.8	6.4	$65,753
9	San Diego	4152	10.6		Del.	559.5	6.5	$58,814
10	San Antonio	7082	7.3		DC	1202.1	7.7	$63,124

Figure 3: Table 3(a) (left): crime rate and unemployment by city; Table 3(b) (right): crime rate and unemployment by state

Proceedings of the VLDB Endowment, Vol. 8, No. 12

an external table, he can then import it to the same spreadsheet as shown in Table 1(b) on the right.

Now the task he faces is to join Table 1(a) on the left with Table 1(b) on the right. Note that the subject (first) column [1] of the two tables are referring to the same concept of countries, but using different representations. In particular, Table 1(a) uses 2-digit ISO country codes, whereas Table 1(b) uses country names. A naive equi-join using string equality comparison would obviously fail.

Figure 2 gives another such example. Table 2(a) on the left has stock market capitalization data of different companies, while Table 2(b) in the same spreadsheet has total political contribution of each company. Note that the subject column of the left table is stock tickers, while the one on the right is company names, again making the two tables difficult to join.

In the final example in Figure 3, the user tries to join Table 3(a) that has crime rate by city, with the state-level statistics in Table 3(b). The idea is here to join cities with states which they belong to, so that one could compare the crime rate of a city to that of the corresponding state. If the Table 3(a) has an additional column with the state information of each city (using the state abbreviations in Table 3(b)), then an equi-join can be performed. However, without such a column, the two tables once again cannot be joined easily.

There are many interesting scenarios like these that require non-equi-joins [2]. These scenarios fall into two broad categories. In the first category, two joining columns conceptually refer to the same set of entities, and all pairs of joining values are almost synonymous. Examples in Figure 1 and Figure 2 are in this category. Additional examples include joining state names with state abbreviations, airport codes with cities, drug brand names with generic names, ICD-9 codes with diseases, and chemical names with formulas, etc. Joins in this case are most likely one-to-one joins. In the second category, the target is to join related, but not synonymous, entities. The example in Figure 3 is in this category, which has a hierarchical relationship. Other examples include joining drug names with pharmaceutical companies, car makes with models, congressmen with states they represent, and universities with their campus locations, etc. Joins in this category are often many-to-one.

Existing Approaches. Without automatic techniques, solving such join problems can be painful and time-consuming. One natural approach is to use *Table Search Systems* (e.g., PowerQuery [4] or Web Tables [2]), to retrieve a "bridge table" with desired mappings between the joining columns. For example, in the scenario in Figure 2, the user can use table search to find a bridge table with mappings from stock tickers to company names. The desired join can then be performed using a three way join – Table 2(a) is first joined with the bridge table, the result of which is then joined with Table 2(b), to finally accomplish the desired join.

There are a number of downsides to this Table-Search-based Join approach (henceforth referred to as TSJ). The first drawback is **usability**. TSJ requires users to go through the cumbersome process of issuing a table search query, inspecting the returned tables to find the right one (if one exists in top-ranked tables), and finally performing a three way join. This is often inconvenient, time-consuming, or even infeasible for novice users. The burden on the users only gets worse when no single table in top returned tables can cover all desired join pairs. When this happens, select rows from multiple tables need to be manually stitched together, making the join task even more daunting.

[1]The subject column refers to the column containing the main set of entities. Methods to identify subject columns have been studied in the context of Web tables [8].

[2]Table 5 and Table 6 in Section 4 give many more examples that are sampled from test cases used in our experiments.

Furthermore, we observe that TSJ cannot perform joins with high **coverage** in many cases. In particular, the top returned tables are often incomplete. This is in part because Web tables are created mainly for visual consumption – they typically only list top entities by a particular metric (e.g., largest by population, or highest by market capitalization, etc.), and in most cases have less than a hundred rows. As a result, they do not provide complete coverage in many domains with a large number of instances. For example, a top-returned Web table typically does not cover all stock tickers and company names in Figure 2, even if the table comes from high quality sources such as `wikipedia` (Table 1(b) also has dozens of rows not shown due to space limit). In addition, naming variations further exacerbate the coverage issue. While the input table uses "Microsoft Corp" in Figure 2, a top-returned table may use "Microsoft Corporation" or simply "Microsoft". Similarly in the case of joining countries, a top-returned table may use "South Korea" or "Korea, Republic of" instead of "The Republic of Korea".

Lastly, TSJ can sometimes have **precision** issues due to ambiguity in data values. For instance, there are at least two widely-used 2-digit country code standards, namely ISO code [3], which is an international standard, and FIPS code [1], which is used by the US government. The two standards unfortunately have many conflicting code assignments. For example, United Kingdom uses "UK" in FIPS, but "GB" in ISO ("GB" refers to "Gabon" in FIPS); Germany uses "DE" in ISO, but "GE" in FIPS ("GE" refers to "Georgia" in ISO), etc. In such cases, top-returned tables in TSJ may in fact produce incorrect mappings.

Other approaches such as *similarity-join* [9, 12] are also relevant but less suitable, since the desired joins discussed in this work are mostly semantic relationships instead of syntactic ones. A detailed discussion on other related methods can be found in Section 5.

Automatic Join Prediction in Spreadsheets. In light of these issues of TSJ, we develop an automated system called SEMA-JOIN that predicts the desired join relationship to facilitate the semantic joins discussed above. In particular, SEMA-JOIN allows the following user interaction. A user who is working on a spreadsheet with multiple tables, can select two columns from two different tables (e.g., the "stock ticker" column in Table 2(a) and "organization" column in Table 2(b)). He can then click a "join" button, which will take the two sets of values as input, and computes a possible join relationship.

Since we don't expect our system to be perfect in predicting joins, we keep users in the loop by materializing the predicted relationship as a separate, two-column table in the spreadsheet, so that users can inspect, verify and make corrections if necessary. Furthermore, a small number of existing tables that "support" these predictions can be displayed alongside to "explain" the predictions, which could help users to understand and verify results quickly.

At the core of SEMA-JOIN is an algorithm that takes two sets of values from join columns as input, and produces a predicted join relationship. Our main idea to address this is to utilize a big table corpus, which is readily available on the Web (first considered by the seminal work [8]), as well as in various enterprises (in the form of corporate spreadsheets that can also be crawled, extracted and indexed). In particular, we postulate that for any two values that actually join in some semantic relationship (e.g., "MSFT" and "Microsoft"), they will have significant statistical co-occurrence in the same row in some tables of the corpus – more often than pure coincidence would put them together. This is what we call **row-level co-occurrence score**. Furthermore, for two pairs of values that join in the same relationship, e.g., ("MSFT", "Microsoft") and ("ORCL", "Oracle"), not only are "MSFT" and "Microsoft" occurring in the same row, and "ORCL", "Oracle" occurring in the

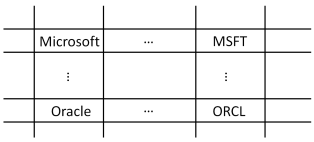


Figure 4: Pairs of joined value-pairs co-occurring in a table

same row, but also the pair ("MSFT", "Microsoft") should co-occur with ("ORCL", "Oracle"), vertically in same columns of some tables in the corpus. We call this type of correlation **column-level co-occurrence score**. These co-occurrence of "pairs-of-pairs" can be illustrated in a rectangular form shown in Figure 4. Column-level co-occurrence should again be statistically significant, which is used as the main ingredient of this work.

Utilizing these data-driven methods to determine strength of correlation, we postulate that the correct join relationship is simply the one that maximizes aggregate pairwise correlation. We then formulate the automatic join prediction problem as an optimization problem, where the goal is to maximize the aggregate correlation score between all pairs of joined values.

We show in this paper that the problem is NP-hard. We develop a 2-approximation algorithm using linear program relaxation and rounding techniques. We then evaluate the effectiveness of our proposed approach with extensive experiments, using test cases from both the public Web domain (where we use over 100M Web tables as the background corpus), and a proprietary enterprise domain (where over 500K enterprise spreadsheet tables crawled from a large company are used). Our evaluation suggests that the data-driven approach of using table corpora significantly outperforms alternative methods. We hope our work will serve as a useful first step toward the goal of enabling users to perform joins fully automatically with a click of the button.

We make the following contributions in this paper: (1) We propose the *join prediction* problem to automate semantic joins, which is particularly useful for ad-hoc data analysis such as in spreadsheets environments. (2) We design a principled optimization problem that maximizes the aggregate correlation score of joined values, where correlation is computed using a large table corpus in a data-driven manner. (3) We develop a SEMA-JOIN system that uses an LP based algorithm with provable quality guarantees, which empirically outperforms alternatives in our experiments using Web and Enterprise data.

2. PROBLEM STATEMENT

In this section, we formally define the problem of *join prediction*. We start by discussing the type of joins considered in this work.

2.1 Types of Joins Considered

Let $\mathcal{R}$ and $\mathcal{S}$ be the two tables that need to be joined. Let $R \in \mathcal{R}$ and $S \in \mathcal{S}$ be the two joining columns from the two tables respectively. We focus our discussion on joins with single columns R and S, because most interesting joins we find are in that category. In addition, since we can treat multiple-columns as one virtual composite column, techniques discussed in this work could apply directly to multi-column joins. For the remainder of this paper we will focus only on single columns joins.

We consider in this work a type of join termed as *optional many-to-one joins*, which is defined below.

DEFINITION 1. *Let $R = \{r_i\}$ and $S = \{s_j\}$ be the two joining columns, where $\{r_i\}$ and $\{s_j\}$ are sets of values in the columns. An optional many-to-one join relationship from R to S is a function $J : R \rightarrow S \cup \{\perp\}$, that joins each value in R with at most one value in S.*

Note that the optional many-to-one join J defines a mapping from each value $r_i \in R$ to either one value $s_j \in S$, or in the case when no appropriate mapping exists, the special non-mapping symbol $\perp$. The non-mapping case is to model practical scenarios where S can potentially be incomplete, or R can sometimes be dirty with extraneous values mixed in. Also note that one-to-one joins an be viewed as special cases of many-to-one joins, and are thus not defined separately.

Optional many-to-one relationship bears some similarity to foreign keys. However, in a foreign key relationship each value on the many-side *has* to map to one and exactly one value on the one-side (the *referential integrity*), whereas here non-mappings are allowed.

Also, notice that in the joined table $T = R \bowtie_J S$, there is a functional dependency from column $T(R) \rightarrow T(S)$, since the join is required to be many-to-one.

EXAMPLE 1. *The examples in Figure 1, Figure 2 and Figure 3 are all optional many-to-one joins, from the tables on the left to the tables on the right. Note that the join is optional, in the sense that some tuples on the left may not join with any tuple on the right.*

Also note that the joined tuples are one-to-one in the examples in Figure 1, Figure 2, but are many-to-one in Figure 3.

Note that in this work, we restrict our attention to optional many-to-one joins (where one-to-one joins are a special case), but not many-to-many joins. This is based on our observation that the typical use scenario in spreadsheet join starts when a user has a "core" table with a set of entities (e.g., Table 1(a), Table 2(a), and Table 3(a)). Users then try to "extend" this core table by adding additional columns from other tables through many-to-one or one-to-one joins with another table (e.g., Table 1(b), Table 2(b), and Table 3(b), respectively). Naturally, when extending a table with additional columns in spreadsheets, the number of rows in the core table will not change, thus ensuring that the join must be many-to-one or one-to-one. Many-to-many joins, on the other hand, require the number of rows in the core table to change, which is not very natural in ad-hoc spreadsheet analysis, and quite uncommon among all the cases we encountered. Technically, imposing the many-to-one constraint gives more structure to our problem, which helps to produce high quality results.

In the remainder of the paper we will refer to an optional many-to-one join as simply a join whenever the context is clear.

2.2 Picking a Good Join

Given two columns $R = \{r_i\}$ and $S = \{s_j\}$, there is an exponential number of optional many-to-one joins J between R and S.Among all these options, we need a criteria to measure the goodness of a join J in order to pick the best one.

Intuitively, a join J from R to S (e.g., `country` to `country code`) is good if (1) at a row level, two values aligned by the join should be semantically related (e.g., "United States" and "US" are very related, "Germany" and "US" are less so); and (2) in addition, at a column level, each pair of joined values should also be semantically compatible. E.g., the pair ("United Kingdom", "GB") is "compatible" with ("Germany", "DE"), because both are in the ISO standard and they co-occur in many tables. On the other hand, ("United Kingdom", "GB") is not semantically compatible with ("Germany", "GE"), because ("Germany", "GE") is in the other FIPS code standard, where "United Kingdom" is actually abbreviated as "UK" instead of "GB". Our observation is that when values are joined up correctly, not only should value pairs in the same row be related, but also pairs of joined value pairs across columns should also be semantically compatible.

If we can quantify the strength of semantic correlation, then we could formulate this problem as an optimization problem, by essentially picking the join with the highest correlation score.

Semantic correlation score. Since our goal is to join arbitrary ad-hoc tables automatically, we cannot rely on manually created data sources such as knowledge bases (KBs). This is because: (1) Knowledge bases such as Freebase [7] do not yet provide exhaustive coverage for arbitrary domains of interest, and even for domains that they cover, each entity is typically represented by one canonical name (e.g., "Seattle International Airport"), whose other name variants (e.g., "Seattle-Tacoma International Airport", "Seattle Airport, WA", etc.) are not covered by KBs even if these alternative names are also heavily used in Web tables. (2) A lot of ad-hoc data analysis happen in enterprise domains, where there are typically no KBs to cover enterprise-specific entities and relationship due to their proprietary nature.

In this work, we propose a data-driven approach to quantify semantic correlation. Specifically, we rely on a big table corpus with matching characteristics of the input columns. For example, we crawled over 100M tables from the Web, for joins involving data from the public domain; we also crawled and extracted over 500K spreadsheet tables from a large company under study, for joins of proprietary data in the enterprise domain. Compared to KBs, this purely data driven approach has the advantage of providing extensive data coverage, for both public and enterprise data, as long as an appropriate table corpus is provided.

Given a table corpus, we reason that if two values are semantically related and are thus candidates for joins (e.g., "United States" and "US"), people will naturally put them together in the same row more often than two random values. Furthermore, if two pairs of values are both semantically related in the same context/domain, (e.g., ("United Kingdom", "GB") and ("Germany", "DE")), then they are likely to co-occur in same columns of some tables. We thus use statistical co-occurrence as a proxy for semantic correlation. Specifically, we use the popular *point-wise mutual information (PMI)* [11], defined at both row-level and column-level.

Let $T(r_i)$ and $T(s_j)$ be the set of tables in which r_i and s_j occurs, respectively, and $T(r_i, s_j)$ be the set of tables with both r_i and s_j in the same row. Let N be the total number of tables. Define $p(\cdot)$ be the probability of seeing certain values in tables from a table corpus of size N, i.e., $p(r_i) = \frac{|T(r_i)|}{N}$, $p(s_j) = \frac{|T(s_j)|}{N}$, are the probabilities of seeing r_i and s_j respectively, and $p(r_i, s_j) = \frac{|T(r_i, s_j)|}{N}$ is the probability of seeing (r_i, s_j) together in the same row. PMI for row-level co-occurrence is defined as [11]:

$$\text{PMI}(r_i, s_j) = \log \frac{p(r_i, s_j)}{p(r_i)p(s_j)} \tag{1}$$

Similarly, for the column-level PMI, let $T(r_i, s_j)$ be the set of tables with both r_i and s_j in the same row. Let $T((r_i, s_j), (r_k, s_l))$ be the set of tables where: (1) r_i and s_j in the same row, (2) r_k and s_l in the same row, (3) r_i and r_k in the same column, and (4) s_j and s_l in the same column. Figure 4 provides a visualization of tables containing quadruple with such a rectangular co-occurrence. Define probability scores $p(\cdot)$ as above. PMI at column-level can be defined as:

$$\text{PMI}((r_i, s_j), (r_l, s_k)) = \log \frac{p\left((r_i, s_j), (r_k, s_l)\right)}{p(r_i, s_j)p(r_k, s_l)} \tag{2}$$

Note that PMI is commonly normalized to $[-1, 1]$ using the normalized PMI (NPMI) [11] as follows.

$$\text{NPMI}(x, y) = \frac{\text{PMI}(x, y)}{-\log p(x, y)}$$

EXAMPLE 2. *Let r_1 = United Kingdom, and s_1 = GB. Suppose $N = 100M$ (there are a total of 100M columns), $|T(r_1)| = 1000$, $|T(s_1)| = 3000$, and $|T(r_1, s_1)| = 500$ (individually, the two strings occur in 1000 and 3000 tables respectively; together they co-occur 500 times in the same row). It can be calculated that PMI$(r_1, s_1) = 4.22 > 0$, and NPMI$(r_1, s_1) = 0.79$, a strong indication that they are related.*

By default we only keep pairs for which PMI scores are positive, and prune away all pairs with negative PMI scores (indicating that their co-occurrence is less frequent than random chance). This is equivalent to setting a PMI threshold of 0.

Note that we use PMI as *one* possible choice, alternative definition of correlation scores such as the set-based Jaccard coefficient can also be used, as long as the intuitive notion of the strength of co-occurrence is captured. We choose to use the information theoretic PMI mainly because it is robust to sets of asymmetric sizes (set-based Jaccard tend to produce a low score if s is highly popular but r is not, even if r always co-occurs with s).

2.3 An Optimization-based Formulation

After quantifying semantic correlation at the row and column level, we can now formulate join prediction as an optimization problem. In particular, we postulate that the correct join is simply the one that maximizes the aggregate correlation score.

As discussed in Section 2.2, there are two ways to quantify the strength of relationship: row-wise PMI scores (Equation (1)), and column-wise PMI scores (Equation (2)). Accordingly, we can define two versions of the problem.

In the first version we maximize the aggregate row-wise scores. For simplicity, we assume that the direction of the join $J : R \rightarrow S$ is known without loss of generality, since both join directions can be tested and the one with a better score can be picked. In certain spreadsheet interfaces we also don't need to "guess" the right join direction, because this can be exposed as extending a "core" table (e.g., Table 1(a), 2(a), 3(a)) with additional columns from a second table (e.g., Table 1(b), 2(b), 3(b)), where the core table being extended is known to be on the many-side (table R).

For each value r_i, join J determines the value $J(r_i) \in S$ to be joined with r_i. The row correlation score for this pair can be written as $w(r_i, J(r_i))$, where w is a shorthand notation for the PMI score in Equation (1), and $w(\cdot, \perp)$ is defined to be 0. Then the aggregate row score is simply

$$RS(J) = \sum_{r_i \in R} w(r_i, J(r_i))$$

We formulate the RS-JP problem as follows.

DEFINITION 2. ***Row score maximizing join prediction (RS-JP).*** *Given two input columns that are represented as sets of values $R = \{r_i\}$ and $S = \{s_j\}$. The problem of row score maximizing join prediction is to find a many-to-one join $J : R \rightarrow S$, that maximizes the aggregate row score over all possible candidate joins, or $J = \text{argmax}_J \sum_{r_i \in R} w(r_i, J(r_i))$.*

Similarly, we can also define the problem using column-wise scores. In particular, for each pair of values $r_i \in R, r_j \in R, i \neq j$, let $w(r_i, J(r_i), r_j, J(r_j))$ denote the PMI score in Equation (2), with $w(\cdot, \perp, \cdot, \cdot)$ and $w(\cdot, \cdot, \cdot, \perp)$ defined as 0. Then the aggregate column score can be written as.

$$CS(J) = \sum_{r_i \in R, r_j \in R, i \neq j} w(r_i, J(r_i), r_j, J(r_j))$$

We formulate the CS-JP problem as follows.

DEFINITION 3. ***Column score maximizing join prediction (CS-JP).*** *Given two input columns that are sets of values $R = \{r_i\}$ and*

$S = \{s_j\}$. *The problem of column score maximizing join prediction is to find a many-to-one join* $J : R \rightarrow S$, *that maximizes the aggregate pairwise column-score over all possible candidate joins, or* $J = \text{argmax}_J \sum_{r_i \in R, r_j \in R, i \neq j} w(r_i, J(r_i), r_j, J(r_j))$.

While both RS-JP and CS-JP are intuitive, RS-JP is considerably simpler to solve. Observe that the join decision for each r_i can be optimized individually, by picking the $s_j \in S$ with the best score $w(r_i, s_j)$ that is positive, or picking $\perp$ if none exists. In aggregate these individual decisions guarantee the optimality of RS-JP.

Despite the simplicity of this formulation, RS-JP is quite effective in our empirical evaluation, primarily due to the fact that it is a data-driven approach that leverages the power of big table corpora. However, we observe that in a number of cases, RS-JP does not have good precision. This is problematic because high precision is of great importance particularly in our application, since in our setting false positives are generally more difficult for users to identify and correct than false negatives.

We use the following real but minimally-constructed example to illustrate why RS-JP may have low precision in some cases. The upshot is that RS-JP only considers each row-pair $(r_i, J(r_i))$ individually, without taking into account global information such as semantic compatibility across different rows.

EXAMPLE 3. *Suppose* $R = \{$*Germany, United Kingdom*$\}$, *and* $S = \{DE, GB, GE\}$ *along with other ISO country codes.*

Suppose we have the following row-level scores w(*Germany, DE*) $= 0.79$, w(*Germany, GE*) $= 0.8$, w(*United Kingdom, GB*) $= 0.85$, *while all other row level scores are low. Note that both (Germany, DE) and (Germany, GE) have high row-wise scores, because the former is in ISO standard [3], while the latter is in the also-popular FIPS standard [1]. The reason GE is also in ISO country code set S is because it represents country Georgia in ISO.*

In RS-JP, the optimal solution is United Kingdom → GB (in ISO), and Germany → GE (in FIPS). Notice that Germany → GE (in FIPS) is picked over Germany → DE (in ISO), because it has slightly higher row-wise score.

However, this result is apparently inconsistent – it gives no consideration to other values that are also being joined in the same problem. In particular, since United Kingdom will have to be joined with its ISO code GB (its FIPS code UK is not used by other countries in ISO so the alternative UK will not be in S). The join selection Germany → GE in FIPS standard is then semantically incompatible with United Kingdom → GB.

On the other hand, in CS-JP, suppose the column-level scores are the following: w(*Germany, DE, United Kingdom, GB*) $= 0.6$, w(*Germany, GE, United Kingdom, GB*) $= 0.05$. *Note that score* w(*Germany, DE, United Kingdom, GB*) *is much higher, because (Germany, DE) and (United Kingdom, GB) are in the same ISO standard, thus co-occurring much more often in tables.*

The optimal solution of CS-JP is then Germany → DE (in ISO), and United Kingdom → GB (in ISO). Note that the notion of semantic compatibility between pairs of matched values are captured by the use of column-level co-occurrence scores. The join decisions of each pair are now made holistically, instead of individually at row level as in RS-JP.

The idea here is that co-occurrence in the same row alone does not always provide sufficient evidence to determine what pairs should join, in part because of the heterogeneity and ambiguity of Web data. In general, such ambiguity are not uncommon, especially when one set of values being joined are short, code-style values (e.g., country codes, language codes, currency codes, airport codes, etc.), which are inherently ambiguous with many possible interpretations. In comparison, the column-level correlation of all pairs of matched value-pairs provides a more robust signal about what should be joined, because even if some pairs are ambiguous and hard to determine locally at the row-level, by looking across all pairs collectively join decisions become easier, since matches in other pairs provide a better "context" for the true matches to stand out.

In addition to value ambiguity, the problem is further compounded by the fact that statistical co-occurrence is only an imperfect proxy for semantic relationship, which can sometimes be noisy. For example, for a desired city/state join relationship (as in Figure 3), we observe that there are also many other tables with flight information listing departure city/state and arrival city/state in the same row. Note that the co-occurrence of departure state and arrival city is also counted in our correlation calculation, which is in fact noise since they do not correspond to a clear semantic relationship and confuse the desired city/state relation. By using collective score maximization across all pairs in CS-JP, however, the join decisions become more robust, because they are now made holistically at the table level, instead of individually at the row level.

Given our intuition that CS-JP may be superior in certain cases, and the fact that RS-JP is easy to solve in polynomial time, for the rest of this paper we will focus on CS-JP. Unfortunately, CS-JP is more difficult. It can be shown that the problem is NP-hard, using a reduction from Densest-k-subgraph (DKS) [15].

THEOREM 1. *The decision version of CS-JP is NP-hard.*

We obtain the hardness result by a reduction from Densest-k-subgraph (DKS) [15]. A proof of this theorem is omitted in the interest of space but can be found in the full version of the paper.

Given the hardness result, we develop algorithms using linear program relaxation with approximation guarantees.

3. SOLVING CS-JP

In this section, we will first give a quadratic program based formulation that naturally translates the CS-JP problem in Section 2.3. We will then show that the program can be transformed to a integral linear program, which can then be relaxed. We will show that with appropriate rounding, the solution to the linear relaxation is a 2-approximation solution to CS-JP.

3.1 A Quadratic Program Formulation

Assume we would like to determine the best many-to-one join from R to S. We model whether $r_i \in R$ is mapped to $s_j \in S$ as a binary decision variable, $x_{ij} \in \{0, 1\}, \forall i \in [|R|], j \in [|S|]$. Given that join candidates considered are restricted to be optional many-to-one as defined in Definition 1, we know that for any r_i, there is at most one s_j that can be matched from r_i. This can be written as a constraint.

$$\forall i, \sum_{s_j \in S} x_{ij} \leq 1$$

This constraint specifies that each r_i can be matched with at most one record from S. Note that $\sum_j x_{ij} = 0$ when r_i is not matched with anything from S. This could happen in practice because S is incomplete and thus does not have the matching value for r_i, or r_i is simply a dirty value that cannot be matched with any value in S.

Given our intuition that the correct join should maximize aggregate pairwise compatibility scores between all pairs of joined values (r_i, s_j), (r_k, s_l), we can write this objective function in a quadratic form:

$$\sum_{\substack{r_i, r_k \in R, i \neq k \\ s_j, s_l \in S}} w_{ijkl} x_{ij} x_{kl}$$

Here w_{ijkl} is simply a shorthand notation for the pairwise compatibility score PMI($(r_i, s_j), (r_k, s_l)$). The score w_{ijkl} is counted in the objective function only when both pairs of values (r_i, s_j) and (r_k, s_l) are matched ($x_{ij} = 1$ and $x_{kl} = 1$). Note that we are aggregating scores across all matched pairs $(r_i, s_j), (r_k, s_l)$, where $i \neq k$ is because the join is required to be many-to-one.

With these we can write an optimization problem CIQPM (column-score, integral quadratic program using maximization), which maximizes the overall matching score. Note that the program uses integral quadratic program.

$$\text{(CIQPM)} \quad \max \sum_{\substack{r_i, r_k \in R, i \neq k \\ s_j, s_l \in S}} w_{ijkl} x_{ij} x_{kl} \tag{3}$$

$$\text{s.t. } \forall i, \sum_{s_j \in S} x_{ij} \leq 1 \tag{4}$$

$$x_{ij} \in \{0, 1\} \tag{5}$$

The dual version of score maximizing CIQPM above is the problem that minimizes the loss of scores between pairs of values that are *not* matched. We simply flip the value of $x_{ij}x_{kl}$ in the objective function (3) to get the loss minimizing CIQP.

$$\text{(CIQP)} \quad \min \sum_{\substack{r_i, r_k \in R, i \neq k \\ s_j, s_l \in S}} w_{ijkl}(1 - x_{ij} x_{kl}) \tag{6}$$

$$\text{s.t. } \forall i, \sum_{s_j \in S} x_{ij} \leq 1 \tag{7}$$

$$x_{ij} \in \{0, 1\} \tag{8}$$

It is apparent that CIQP and CIQPM share the exact same optimal solutions. To see this, notice that for any given set of x_{ij}s, the objective value of CIQP and CIQPM (Equation (3) and (6)) always sum up to a fixed constant. In other words CIQPM is maximized if and only if CIQP is minimized. For technical reasons we will focus on the loss-minimization version CIQP in the rest of the paper.

It is well known that general quadratic program without special structures (e.g., positive definite coefficient matrix) is hard to optimize. In fact, it was shown that quadratic programs with even one negative eigenvalue in the coefficient matrix is NP-hard [17]. In the particular case of Equation (6), given that w_{ijkl} can take arbitrary values, we could not rely on special properties of the quadratic program to obtain optimal solutions.

In the following, we transform CIQP in two steps to obtain solutions with 2-approximation guarantees efficiently. We first transform it into an equivalent integral linear program, and then apply LP-relaxation. We can show that the relaxed LP has a remarkable property that it is *half-integral*, using a rounding argument. This property allows us to construct an integral solution to CIQP from the relaxed LP with quality guarantees.

3.2 An Equivalent Integral Linear Program

We first apply a common LP trick to transform CIQP to an integral LP, denoted by CILP below. In particular, we introduce a new set of variables z_{ijkl} in CILP, to represent each pair of bilinear terms $x_{ij}x_{kl}$ in CIQP. We also add a linear constraint in Equation (11) for each z_{ijkl}, in order to make sure that z_{ijkl} takes the value 1 if and only if both x_{ij} and x_{kl} are 1.

Algorithm 1: `Round half-integral solution to CLP`

Input: Program CLP
Output: Half integral solution $\tilde{x}^*_{ij} \in \{0, 1\}$, $\tilde{z}^*_{ijkl} \in \{0, \frac{1}{2}, 1\}$

Solve CLP using standard LP, to obtain optimal solution $\overline{x}^*_{ij}$, $\overline{z}^*_{ijkl}$
for *each* $1 \leq i \leq |R|$ **do**
  if $\overline{x}^*_{ij} \in \{0, 1\}, \forall j \in [|S|]$ **then**
    $\tilde{x}^*_{ij} \leftarrow \overline{x}^*_{ij}$
  else
    $c_{ij} = \left(\sum_{r_k \in R, k \neq i, s_l \in S} \frac{1}{2} w_{ijkl}\right)$
    $p = \text{argmax}_j \, c_{ij}$
    $\tilde{x}^*_{ip} \leftarrow 1$
    $\tilde{x}^*_{ij} \leftarrow 0, \forall j \neq p$
for *each* $i, k \in [|R|], j, l \in [|S|], k \neq i$ **do**
  $\tilde{z}^*_{ijkl} \leftarrow \frac{1}{2}(\tilde{x}^*_{ij} + \tilde{x}^*_{kl})$
Return $\tilde{x}^*_{ij} \in \{0, 1\}$, $\tilde{z}^*_{ijkl} \in \{0, \frac{1}{2}, 1\}$ as an half-integral solution to CLP

$$\text{(CILP)} \quad \min \sum_{\substack{r_i, r_k \in R, i \neq k \\ s_j, s_l \in S}} w_{ijkl}(1 - z_{ijkl}) \tag{9}$$

$$\text{s.t. } \sum_{s_j \in S} x_{ij} \leq 1, \forall i \tag{10}$$

$$z_{ijkl} \leq \frac{1}{2}(x_{ij} + x_{kl}), \forall r_i, r_k \in R, i \neq k, s_j, s_l \in S \tag{11}$$

$$x_{ij}, x_{kl} \in \{0, 1\}, \forall i, j, k, l \tag{12}$$

$$z_{ijkl} \in \{0, 1\}, \forall i, j, k, l \tag{13}$$

It can be shown that any solution to CILP can be used as a solution to CIQP, with the same objective value.

LEMMA 1. *Let (x_{ij}, z_{ijkl}) be a solution to CILP, then (x_{ij}) are a feasible solution to CIQP, and CILP(x_{ij}, z_{ijkl}) = CIQP(x_{ij}).*

PROOF. First we note if (x_{ij}, z_{ijkl}) is a solution to CILP, (x_{ij}) are a feasible solution to CIQP, because Equation (10) and Equation (12) are satisfied in CILP guarantees that Equation (7) and Equation (8) in CIQP are respected, respectively.

We then show CILP(x_{ij}, z_{ijkl}) = CIQP(x_{ij}). Observe that in CILP, z_{ijkl} is used to represent the bilinear term $x_{ij}x_{kl}$ in CIQP. The objective functions are otherwise equivalent. To show Lemma 1, we only need to show that z_{ijkl} takes the value 1 if and only if both x_{ij} and x_{kl} are 1, in which case Equation (6) in CIQP and Equation (9) in CILP are bound to have the same objective value, for arbitrary value configurations of x.

It is easy to show that if $z_{ijkl} = 1$, we must have both $x_{ij} = 1$ and $x_{kl} = 1$. This is guaranteed by Equation (11) – otherwise if one of x_{ij} or x_{kl} is 0 yet $z_{ijkl} = 1$, the constraint is violated.

Now we show the other direction is true. That is, if both $x_{ij} = 1$ and $x_{kl} = 1$, we have $z_{ijkl} = 1$. Constraint (11) guarantees that $z_{ijkl} \leq 1$. However, given that we are minimizing Equation (9), and the fact that $w_{ijkl} > 0$, we know that z_{ijkl} will always be pushed to value 1 for objective value minimization.

Combining, we know that CILP(x_{ij}, z_{ijkl}) = CIQP(x_{ij}). □

Given Lemma 1, we can equivalently solve CILP for the original problem CIQP. However, integral LP is still generally intractable. In the following we address this by using LP relaxation.

3.3 Linear Program Relaxation

We now apply LP relaxation to CILP, that is, we replace integrality constraints in Equation (12) and Equation (13) with range

Algorithm 2: `Solve CILP`

Input: Program CILP
Output: Integral solution $x^*_{ij} \in \{0,1\}$, $z^*_{ijkl} \in \{0,1\}$

```
Construct problem CLP based on the given CILP.
Using Algorithm 1 to obtain half-integral solution to CLP
  x̃*_ij ∈ {0,1}, z̃*_ijkl ∈ {0, 1/2, 1}
for each i ∈ [|R|], j ∈ [|S|] do
    x*_ij = x̃*_ij
for each i,k ∈ [|R|], j,l ∈ [|S|], k ≠ i do
    if x*_ij = 1 and x*_kl = 1 then
        z*_ijkl ← 1
    else
        z*_ijkl ← 0
Return x*_ij ∈ {0,1}, z*_ijkl ∈ {0,1} as an integral solution to CILP
```

constraints (17) and (18), to obtain the CLP program below. Note that we use the fractional variables $\overline{x}_{ij}$, $\overline{x}_{kl}$, and $\overline{z}_{ijkl}$ in place of x_{ij}, x_{kl}, and z_{ijkl}, respectively.

$$\text{(CLP)} \quad \min \sum_{\substack{r_i, r_k \in R, i \neq k \\ s_j, s_l \in S}} w_{ijkl}(1 - \overline{z}_{ijkl}) \tag{14}$$

$$\text{s.t.} \sum_{s_j \in S} \overline{x}_{ij} \leq 1, \forall i \tag{15}$$

$$\overline{z}_{ijkl} \leq \frac{1}{2}(\overline{x}_{ij} + \overline{x}_{kl}), \forall r_i, r_k \in R, i \neq k, s_j, s_l \in S \tag{16}$$

$$\overline{x}_{ij}, \overline{x}_{kl} \in [0,1], \forall i, j, k, l \tag{17}$$

$$\overline{z}_{ijkl} \in [0,1], \forall i, j, k, l \tag{18}$$

In the following, we show a remarkable property that at least one optimal solution to CLP is *half-integral*, or in the domain of $\{0, \frac{1}{2}, 1\}$.

LEMMA 2. *At least one optimal solution to CLP is half-integral. More specifically, $\overline{x}_{ij} \in \{0,1\}, \forall i, j$, and $\overline{z}_{ijkl} \in \{0, \frac{1}{2}, 1\}, \forall i, j, k, l$.*

We skip details of the proof in the interest of space. A proof of the lemma can be found in the full version of the paper.

Given Lemma 2, we can always find an optimal half-integral solution to CLP using the procedure is outlined in Algorithm 1, which mirrors the construction in the proof.

Algorithm 1 first solves CLP using standard LP techniques in polynomial-time. Then for each i, it checks $\overline{x}^*_{ij}, \forall j \in [|S|]$ to see if they are already integral. If so then the corresponding $\tilde{x}^*_{ij}$ are set accordingly, otherwise values are rounded to integral values and assigned to $\tilde{x}^*_{ij}$. Values of $\tilde{z}^*_{ijkl}$ are then computed from $\tilde{x}^*_{ij}$. Lemma 2 proves that this will not affect the solution quality, and is thus both half-integral and optimal to CLP. This process takes polynomial time.

Our next step is then to use the optimal half-integral solution to CLP, to obtain integral solution $x^*_{ij} \in \{0,1\}$, $z^*_{ijkl} \in \{0,1\}$ to CILP. Algorithm 2 describes the process. Specifically, for a given CILP, we transform it to the corresponding CLP, and then use Algorithm 1 to produce half integral solutions. We can set $x^*_{ij} = \tilde{x}^*_{ij}$, which will satisfy the constraint in Equation (10) since $\tilde{x}^*_{ij}$ satisfies Equation (15) in CLP. Then we can force z^*_{ijkl} to 0 if either x^*_{ij} or x^*_{kl} is 0, in order to respect the constraint in Equation (11).

We can show that the solutions to CIQP obtained in Algorithm 2 has the following quality guarantee.

THEOREM 2. *Let CILP* be the optimal solution to CILP, and (x^*_{ij}, z^*_{ijkl}) be the solution to CILP obtained in Algorithm 2. The solution is a 2-approximation of CILP*, that is, CILP$(x^*_{ij}, z^*_{ijkl}) \leq 2$ CILP*.*

This approximation result uses the half-integrality in Lemma 2. A proof can be found in the full version of this paper.

Using this approximation and the fact that CILP can be used to solve CIQP in Lemma 1, we conclude that the integral values x^*_{ij} obtained in Algorithm 2 is also a 2-approximation solution to CIQP.

THEOREM 3. *Let CIQP* be the optimal solution to CIQP, and (x^*_{ij}, z^*_{ijkl}) a solution to CIQP obtained in Algorithm 2. Then (x^*_{ij}) is a 2-approximation solution to CIQP, that is, CIQP$(x^*_{ij}) \leq 2$ CIQP*.*

PROOF. Given Lemma 1, we know that if (x^*_{ij}, z^*_{ijkl}) is a valid solution to CILP, (x^*_{ij}) is a valid solution to CIQP, and CIQP(x^*_{ij}) = CILP(x^*_{ij}, z^*_{ijkl}). Note that Lemma 1 also guarantees that CIQP* = CILP*. Combining with Theorem 2, we have CIQP$(x^*_{ij}) \leq 2$ CIQP*. □

To sum things up, when given an instance of the join problem formulated in CIQP, we first transform it to CILP, and then relax it to get CLP, which can be efficiently solved. The solutions are then rounded in Algorithm 2 to provide 2-approximation solutions to the original CIQP. In practice as a post-processing step we also greedily improve solutions so obtained by iteratively changing assignments as long as the objective score continues to improve. This yields better empirical results while still preserves 2-approximation.

4. EXPERIMENTS

We conduct a set of experiments (1) to compare the effectiveness of alternative approaches in terms of the quality; (2) to understand the sensitivity of algorithms to threshold settings; and (3) to evaluate the impact of using appropriate table corpus on join predictions.

4.1 Experimental setup

4.1.1 Benchmark test sets

We construct two benchmark test sets of different characteristics.

The first test set has join examples from the public Web domain, which is referred to as `Web`. We construct the `Web` test set by randomly sampling queries of pattern ``` ``list of A and B'' ``` from Bing's keyword query logs. Such "list" queries typically reflect users' intention of acquiring two related sets of data, which can typically be linked together by an interesting join relationship. Figure 5 gives a number of such examples.

For each sampled query, we first ascertain its data intention and filter out ones whose intentions are not joins (e.g., "list of rock songs in the 70's and 80's", "list of pros and cons of nuclear energy"). We then manually construct a two-column ground truth table using Web tables and lists. For example, for the query "list of us states and capitals," we put all states in the first column, and their capital cities in the second to specify the join implied by the query.[3] The two columns can then be fed separately into algorithms as input, with the ground truth table as the desired output. We build a total of 50 such test cases for `Web`, such as the ones in Figure 5.

Discussion. Note that the frequency-weighted sampling is likely to encounter "head" queries more often than "tail" ones, which is

[3]For more open-ended concepts such as *"list of drugs and generic names"*, we build our ground truth by only taking a representative sample of all possible joining pairs for efficiency reasons.

favorable to data-driven techniques such as ours (because head data are likely to be well represented in table corpora). As such, the performance results reported herein should not be construed as results that algorithms can achieve when joining two *tables* sampled uniformly at random from table corpora, because that is likely to encounter tail relationships that are more difficult to join. However, since our goal is to help users join data and query logs is a reasonable proxy that reflects data usage, we think this evaluation is a useful estimator of the effectiveness of the proposed system.

Our second test set is built using proprietary enterprise data from a large company under study, which we refer to as `Enterprise`. For the `Enterprise` test set, a domain expert manually inspects commonly co-occurring and highly correlated concepts from a spreadsheet corpus of the company. A total of 50 pairs of concepts are sampled, where joins between the two concepts are determined to be meaningful. Figure 6 gives some concrete examples used in `Enterprise`.

list of us states and capitals	list of chemical elements and symbols
list of suvs and manufacturers	list of albums and artists
list of airports and codes	list of drugs and generic names
list of countries and continents	list heisman trophy winners and schools

Figure 5: Example queries and test cases for `Web`

ads campaign and campaign id	customer company and contact person
sales district and sales region	crm account id and company name
employee name and alias	employee name and job title
product and product division	customer industry and vertical

Figure 6: Example test cases for `Enterprise`

Figure 7 gives details of some statistics of the two test cases – on average each test case has about 50 rows to join.

Dataset	mean	median	max	min
`Web`	50.4	41	174	8
`Enterprise`	48.8	50	226	4

Figure 7: Test case statistics, in number of rows

4.1.2 Evaluation metric

Since our join predictions are not always perfect, we use the classical precision/recall metric to determine the quality of the predicted joins. Specifically, in the context of our problem, precision is defined as $p = \frac{\text{num. of correctly joined pairs}}{\text{num. of predicted pairs}}$, and recall is $r = \frac{\text{num. of correctly joined pairs}}{\text{num. of ground truth pairs}}$. We also report F-measure as the aggregate quality measure, which is the harmonic mean of precision and recall, defined as $f = \frac{2pr}{p+r}$.

Note that both precision loss (incorrectly joined pairs) and recall loss (pairs that are not joined by algorithms) translate to additional user efforts of either making manual corrections, or manually filling missing joins, which are undesirable.

4.1.3 Algorithms compared

• `Table-Search-Join (TSJ)`. As discussed earlier, the most straightforward way of performing joins given two input columns using existing techniques, is to issue the columns as a query against table search systems, and then pick a top-returned table as the result. In this work we use a customized variant of the PowerQuery as the underlying table search engine for `Web` test cases, and an enterprise Intranet search engine for `Enterprise` test cases (with result-type filters turned on to return only spreadsheet files).

Once top-K tables are retrieved, we can look up input values in the top returned tables. The table with two columns that can best "cover" joining values among the top-K tables are naturally the best choice to determine the desired join relationship. We call this approach `TSJ-BestK`, where K is set up to 100. Note that this approach mimics a human user who would browse through the top-K tables, and select the best table as the intermediate bridge table to perform joins.

An alternative to `TSJ-BestK` is to aggregate from top-K tables all cell pairs whose values fall in the two input columns. We refer to this approach as `TSJ-AggK`. This approach corresponds to the scenario where no single table can cover all input values, and users have to go through the list of tables to manually piece together a bridge table before joins can be performed.

• `Fuzzy Join`. Using tables returned by `TSJ`, we can additionally apply fuzzy join / similarity join [9, 21] to allow fuzzy matching between input values and values in returned tables (e.g., matching "Microsoft Corp" from input columns with "Microsoft Corporation" from returned tables). We experiment with two commonly used fuzzy join variants, namely Edit-Distance `(ED)` and Jaccard-Distance `(JC)`. These give us four more methods to compare with, which are labeled as `TSJ-BestK-ED`, `TSJ-BestK-JC`, `TSJ-AggK-ED`, and `TSJ-AggK-JC`, respectively.

• `Fuzzy join with an Oracle`. Since Edit-Distance and Jaccard-Distance are only two possible methods in the large space of possible fuzzy joins, in order to help us understand the limit of what fuzzy joins can achieve, we used an extensive set of fuzzy joins enumerated in [16] in the context of entity-linking. In particular, for each top table we used { exact, lower, split, word, 2-gram, 3-gram, 4-gram } for tokenization, { Intersection, Jaccard, Dice, MaxInc, Cosine } for similarity functions, and a total of 10 stepwise thresholds for each similarity function (e.g., { 0.1, 0.2, ..., 1}, for Jaccard, Dice, etc.). This generates a total of $7 \times 5 \times 10 = 350$ possible configuration. Since we need to perform fuzzy-join twice (both input tables need to be joined with an intermediate mapping table), and the two fuzzy joins are independent of each other, that requires a total of $350^2 = 122500$ possible configurations for each intermediate table.[4] For each configuration, we look at the labeled ground-truth data to compute the F-score of that configuration, and in the end picks the one with the highest F-score.

Note that given the large space of configurations tested, we don't think a human can possibly try all of these and pick the best one without using the real ground-truth (as opposed to the `TSJ-ED` and `TSJ-JC` methods, where users only need to tune one threshold parameter). This is the reason we call it `TSJ20-LK-Oracle`, which is unlikely to be used in practice, but nevertheless gives an upper bound on what fuzzy join methods can achieve.

• `Knowledge-base Join (Freebase)`. It is also possible to perform joins using curated Knowledge-bases such as Freebase [7]. The idea is to use the entity-ranking functionality to retrieve top-k entity-ids for each input value. For example, using "Georgia" as input, Freebase returns Georgia the US state with its unique entity-id as the most likely entity, Georgia the country the second, among many other possible interpretations. Similarly the country code "GE" for "Georgia" will also return the country and its entity-id as a top entity, By just comparing top-returned entity-ids, we can join"Georgia" and its country code "GE". This method is denoted as `Freebase-TopK`.

• `Row-score Join Prediction (RS-JP)`. Recall that in Section 2.3 we discussed an alternative formulation `RS-JP` (in Definition 2), which only uses row-wise score maximization. `RS-JP` is essentially a variant of the main approach `CS-JP` discussed in

[4]While the space of configurations we explored is already large, there are more complicated similarity join such as TF-IDF based term weighting that are not tested in our experiments.

	F-measure	Precision	Recall
CS-JP-LP	**0.919**	0.990	**0.891**
CS-JP-Greedy	0.476	0.781	0.461
RS-JP	0.891	0.955	0.874
TSJ-Best20	0.666	**0.999**	0.615
TSJ-Best20-ED1	0.553	0.914	0.505
TSJ-Best20-JC0.3	0.667	0.999	0.616
TSJ-Agg20	0.685	0.993	0.646
TSJ-Agg20-ED1	0.563	0.812	0.527
TSJ-Agg20-JC0.3	0.696	0.967	0.659
TSJ20-LK-Oracle	0.879	0.930	0.853
Freebase-Top1	0.152	0.932	0.115
Freebase-Top10	0.442	0.844	0.366
Freebase-Top20	0.505	0.818	0.434
Freebase-Top20-Type	0.332	0.847	0.276

Figure 8: Overall quality comparison on Web test data

	F-measure	Precision	Recall
CS-JP-LP	**0.966**	**0.985**	**0.951**
CS-JP-Greedy	0.544	0.876	0.533
RS-JP	0.935	0.959	0.917
TSJ-Best20	0.518	0.983	0.476
TSJ-Best20-ED1	0.390	0.843	0.339
TSJ-Best20-JC0.3	0.527	0.912	0.488
TSJ-Agg20	0.532	0.975	0.494
TSJ-Agg20-ED1	0.387	0.843	0.339
TSJ-Agg20-JC0.3	0.527	0.912	0.488
TSJ20-LK-Oracle	0.592	0.731	0.579
Freebase-Top1	0.039	0.937	0.022
Freebase-Top10	0.071	0.833	0.051
Freebase-Top20	0.080	0.817	0.059
Freebase-Top20-Type	0.056	0.970	0.040

Figure 9: Overall quality comparison on Enterprise test data

this work that also leverages table corpus statistics, and is simple to solve optimally. However, as we discussed in Section 2.3, it fails to consider global information such as column-wise consistency, which can lead to lower precision in more ambiguous cases.

• Column-score Join Prediction (CS-JP). This is the formulation in Definition 3 that maximizes column-level scores. Our main approach is to use Algorithm 2 to first solve the LP and then round the resulting solutions. We refer to this approach as CS-JP-LP. We use solvers from Microsoft Solver Foundation [5] for this purpose.

In addition, in order to understand the usefulness of the LP-based solution and the quality guarantees it provides, we also compare with a greedy approach to solve the same CS-JP problem. In particular, the greedy method starts by finding two pairs of values from input columns with the maximum column score. It then iteratively adds one pair of values at a time that provides the maximum score gain, until all values from the input are exhausted, or no more pair can be added that produces a positive score gain. This greedy heuristic is henceforth referred to as CS-JP-Greedy.

4.2 Overall quality comparison

Figure 8 and Figure 9 gives an overall quality comparison of average F-measure, precision, and recall, for Web and Enterprise test cases, respectively. In both cases, the CS-JP-LP method has the best overall F-measure. The greedy method CS-JP-Greedy, in comparison, is considerably worse, underlining the importance of using a principled LP-based solution.

The problem variant RS-JP produces decent F-measure. However, its precision numbers are slightly lower, because it only looks at one matching pair at a time without taking into account global consistency, which can lead to errors especially when matching values are short strings such as abbreviations or codes that are inherently ambiguous (Example 3 gives more explanations). We note that the precision improvement of CS-JP-LP over RS-JP is actually significant from an error rate's perspective (where error = 1-precision): there is a factor of 4.5 error reduction using CS-JP-LP over RS-JP on Web (the error rate is 0.010 for CS-JP-LP but 0.045 for RS-JP); and there is a factor of 2.7 error reduction on Enterprise (the error rate is 0.015 and 0.041, respectively). We re-emphasize that in our setting, reducing errors is highly important, because users are more likely to lose confidence in the system over false positives (incorrect matches) – which they may not be able to spot and fix easily – than by false negatives (incomplete matches). The error reduction ratios achieved by CS-JP-LP thus makes it an attractive alternative to RS-JP, even though it involves more complicated formulation and requires more time to solve.

The next group of seven methods reported in Figure 8 and Figure 9 uses some variants of table search TSJ. Since various studies suggest that users rarely go beyond the second page when searching for content (e.g., only 5-10% of users visit the second page, and the average number of pages visited per query is a mere 1.1 [10, 19]), we use top 20 tables as the default setting in our experiments. This simulates a user who can carefully inspect the top 20 tables and pick the one with the best quality for the desired join. We will also report an experiment using up to 100 tables to explore its effect, even though realistically users are unlikely to manually inspect that many tables.

As can be seen from the results, TSJ-Best20, which picks the best table among the top-20, has very high precision for both Web and Enterprise. This is expected, since a table that has many value overlap with the two input columns are likely to be consistent with the desired join and unlikely to make mistakes. The recall of this method, however, is considerably lower than CS-JP-LP. This is in part because existing table search systems are not designed for the purpose of producing joins – a table with less complete coverage of input values are often ranked higher than a more complete table for various reasons (e.g., it is from high quality sources such as Wikipedia). The second reason for its low recall is that there are just so many name variations for the same entity (e.g., Microsoft corporation vs. Microsoft corp, San Francisco international airport vs. San Francisco airport, names with/without middle-initials or full middle names, etc.), such that even if a table has important token hits (e.g., Microsoft) and is ranked high, it may still not have the exact string match needed to produce join results.

Given these observations of TSJ-Best20, one might expect that its fuzzy-join variants can produce considerably better results. Surprisingly, they are not as good as expected. In particular, the method TSJ-Best20-ED1 (which uses Edit distance threshold 1 which is the best among other thresholds) is noticeably worse compared to TSJ-Best20. A close inspection reveals that using Edit distance with even very small distance thresholds can often lead to erroneous matches that affect precision, especially when one input column is short (e.g., airport codes, state abbreviations, chemical symbols, etc.). In comparison, TSJ-Best20-JC0.3 uses Jaccard and is more robust to column length variations. However, its F-measure only slightly improves TSJ-Best20, because it can

still trigger incorrect matches that reduces precision, which negates gains in recall. Overall these fuzzy join variants are not considerably better than TSJ-Best20.

The TSJ-Agg20 method aggregates from top 20 tables value pairs that fall in the two input columns. This is to simulate a user who manually pieces together information from multiple tables. As can be seen from the results, TSJ-Agg20 and its fuzzy variants are not significantly better than TSJ-Best20. This shows that even if one is willing to put a lot of manual efforts to aggregate tables, the marginal gain is not very significant.

We also test TS20-LK-Oracle, which as discussed, uses the ground-truth as an oracle to pick the best fuzzy join in a large space of 122500 possible configurations. We emphasize that since the configurations vary widely from one table to another, or even for the two joins needed for the same table, we think it is impractical to be used by a real user who needs to tune but does not have ground truth. Nevertheless it provides an upper bound on the performance of fuzzy join methods – what still cannot be joined can be mostly attributable to the ranking/quality of the table search system.

Overall, we find that TS20-LK-Oracle on Web is substantially better than other fuzzy-join methods, while it slightly improves over fuzzy joins on Enterprise. It shows that table search for Web (in this case, Power Query) is actually quite good compared to the Intranet search for Enterprise, which is mainly designed for document search and not for structured data. In both cases, CS-JP-LP and RS-JP are better than TS20-LK-Oracle, showing that the join relationships we piece together from the big table corpus are indeed better than the top-20 tables from production search systems, plus almost perfect fuzzy joins.

The next group of comparison is with Freebase-TopK, which retrieves top-k entity-ids from Freebase for input values in both input columns, and then simply performs an comparison on entity-ids for joins. Note that knowledgebases such as Freebase use sophisticated entity-ranking, based on a combination of string match and entity popularity. For example, using "Georgia" as input, Freebase ranks the country much higher than dozens of other possible interpretations of the name "Georgia". The same is true when the country code "GE" is searched, allowing the two string values to be joined by a match in the entity-id for the country. Furthermore, we observe that some limited knowledge-graph traversal was also performed by the system. For example, searching "Georgia" will also return "Atlanta" as a top-entity in Freebase desipte their name difference, presumably because "Georgia" is an important attribute of "Atlanta" and vice versa. This actually allows Freebase-TopK to perform joins for relationships such as US-state and capital, which are beyond entity synonyms like country and country-code that we initially expect Freebase to perform.

It is clear that for Web, the Freebase-TopK approach performs reasonably well, with over 0.8 precision and a 0.505 F-score when top-20 entity-ids are used. [5] However, its recall value falls short in comparison with CS-JP and RS-JP. We believe this is because knowledge bases like Freebase still require substantial human curation in defining schema and relationships, which limits its coverage of interesting relationships as compared to the ones represented in Web table corpus. For the Enterprise test cases, the recall score of Freebase-TopK becomes even lower, which is expected because most data in Enterprise is company-specific with no Web presence.

[5]We used Freebase's REST API to retrieve entities, which returned a maximum of 20 entities per input query. As of the time of this experiment in March 2015, we did not find a way to work around the 20 entity limit.

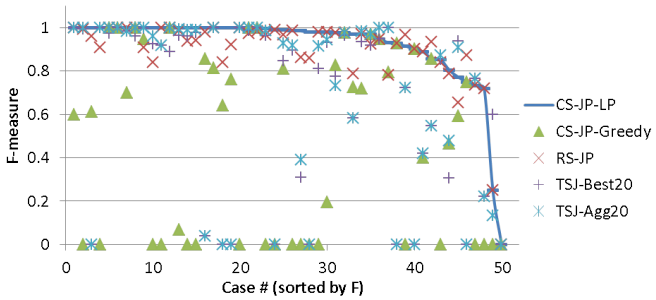


Figure 10: F-measure of all Web test cases

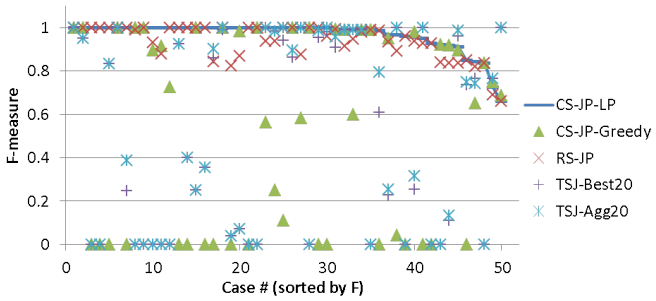


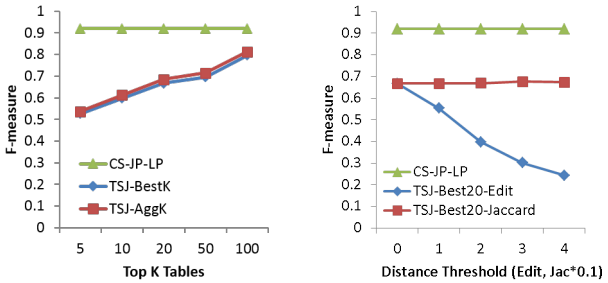
Figure 11: F-measure of all Enterprise test cases

One natural extension of Freebase-TopK is to use the type information of top-entities for all input values in the same column, to first determine the likely type/concept of the input column (using majority-vote, for example). This can help "disambiguate" entity-interpretations, which prevents accidental matches, thus boosting precision. This approach is denoted as Freebase-TopK-Type. As we can see, while this clearly improves precision, it also hurts recall values, making it overall uncompetitive.

We also report in Figures 10 and 11 F-measures of all test cases, where CS-JP-LP is shown as connected lines for visual presentation. As can be seen, CS-JP-LP is the best method in most cases, whereas TSJ-Best20 and TSJ-Agg20 often have cases with 0 recall and F-measure – meaning that table search systems do not return any relevant tables among the top 20. This shows that existing table ranking systems are not really suited for finding bridge join tables. Note that there is one case in Web for which all methods, including CS-JP-LP, score 0 in recall and F-measure. It is a case joining "countries" and "national flowers", for which no Web table in our corpus contains such information (the ground truth is manually constructed from Web documents).

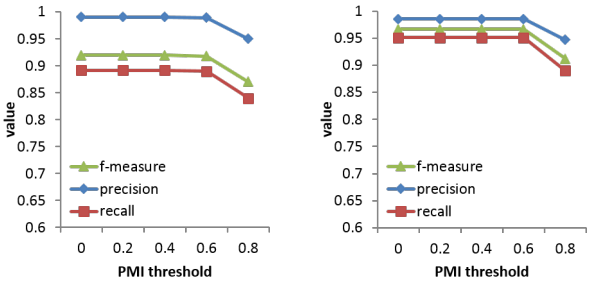
Figure 12(a) shows the effect of using a varying number of top-k tables to produce joins. As can be seen, higher k clearly helps TSJ methods. Yet even using 100 tables TSJ-Agg20 still lags behind CS-JP-LP (the F-measure is 0.813 vs. 0.919). Recall that this simulates the cumbersome experience where a user goes through all 100 tables, manually piecing together information to produce joins. This underscores the superior usability and utility of our approach – even if a user is willing to manually inspect 100 tables, he is still better off using CS-JP-LP.

Figure 12(b) demonstrates the effect of varying distance thresholds in fuzzy matching variants of TSJ. When increasing Jaccard distance from 0 to 0.4, TSJ-Best20-Jaccard improves slightly, before starting to dip at 0.4. On the other hand, when changing Edit distance from 0 to 4, the performance of TSJ-Best20-Edit actually decreases significantly. As discussed before, Edit distance is a character-based metric that can be too aggressive for short columns (e.g., state abbreviations, chemical elements, and airport codes), yet at the same time not effective enough for longer columns



(a) Vary top k tables (b) Vary distance threshold

Figure 12: TSJ experiments on Web test cases



(a) Vary PMI threshold (Web) (b) Vary PMI threshold (Enterprise)

Figure 13: Effect of varying PMI threshold

(e.g., people names, airport names). Both of these variants produce considerably lower quality results compared to CS-JP-LP even when using a wide spectrum of thresholds.

Discussion of Limitations. While our system scores better than alternative methods, which we think is useful progress, in testing the system we also notice a few classes of cases where our current approach cannot handle well, which we discuss here. First, there are certain cases where the two join columns cannot uniquely determine the desired join relationship. For example, given a set of congressmen and a set of US states, it is actually not clear whether the desired join is about the state for which a congressman represents, or the states in which a congressman was originally born, or possibly other relationships. We think such ambiguity may need to be resolved using additional user input (such as keywords describing the relationship). Second, certainly relationships are known to be temporal and can change over time (e.g., "CEO and company"). Handling such cases correctly requires more future work. Lastly, certain out-of-corpus relationships (e.g., the "country and national flower" example encountered in our Web test cases) are apparently also not handled due to the data-driven approach we take.

4.3 Sensitivity to precision threshold

One knob we can tune in CS-JP-LP is to vary the PMI threshold used. Recall that in leveraging value co-occurrence, by default we set a PMI score threshold of 0, which prunes away all pairs whose scores are lower than 0 (indicating negative correlation, or co-occurring less frequently than random chance). While it is natural to keep all pairs whose scores are above 0 to preserve all positively correlating pairs, it is interesting to understand the effect of increasing the threshold on result quality.

Figure 13 shows such an experiment using Web and Enterprise. It turns out that the performance of CS-JP-LP is fairly stable across different threshold values, except when the threshold is very high at 0.8, in which case both precision and recall decrease. This indicates that most of the useful value pairs that affect the algorithm output have very high PMI scores – they are not pruned away even with high score thresholds. The fact that useful pairs have very high PMI scores is an indication that the signal from the underlying corpus is very strong, and as such, our approach is likely to be robust against noises and perturbations.

4.4 Impact of using matching table corpus

The effectiveness of our approach hinges on the use of an appropriate underlying table corpus, from which we learn statistical correlation. It is natural, for example, to use Web table corpus when testing Web cases, and use Enterprise corpus when testing Enterprise cases. It is interesting to explore the effect of using *un-matching* table corpus, e.g., using Web tables for Enterprise, and vice verse. We can expect that in such cases, if the desired joining pairs do not co-occur in any statistically significant manner in the table corpus, our approach will suffer.

Our findings in these experiments are surprising at first – F-measures are exactly 0 in both Web and Enterprise test cases when un-matching table corpus is used, meaning not a single joining pair can be recovered. A close analysis shows that the Enterprise table corpus and Web table corpus are of very different characteristics, with virtually no overlap for concepts represented in Web and Enterprise test cases. This is actually reasonable – Web data, while known to be diverse, still lacks coverage of proprietary enterprise data specific to each company. On the other hand, the Enterprise data we used focus exclusively on information of the company, without broad coverage of public information found on the Web. Furthermore, even for concepts for which we expect some conceptual overlap between the two corpora, e.g., geography concepts such as sales district and sales region used in the enterprise, there is virtually no overlap, because of the particular way in which sales districts are encoded in the company. For example, "MWD.003" is used for a specific sales region of the "mid-west district", which is supposed to join with country "US". This, despite being a geography concept, cannot be found in the Web table corpus we sampled, thus not allowing any joins to be discovered. This experiment clearly shows the importance of using matching table corpus – at the end of the day, it is really the power of the big table corpus that enables us to discover interesting join relationships, which is of critical importance in this and many other data-driven applications.

4.5 Execution time comparison

	Web			Enterprise		
Algorithm	median	min	max	median	min	max
CS-JP-LP	2.05	0.03	1623.2	2.37	0.02	695.4
RS-JP	0.01	0.01	0.03	0.01	0.01	0.03
TSJ-Best20-JC0.3	3.1	0.4	9.2	2.3	0.2	11.3
TSJ20-LK-Oracle	12360	578	46510	729	420	48529

Figure 14: Execution time comparison, in seconds

In this section we report the execution time of representative methods used in our experiments. For a fair comparison of the computation costs of each method, all data structures needed to be accessed (e.g., statistics, TSJ tables) are cached in memory. Our experiments were conducted on a Windows server with 2 Intel Xeon 2.27 GHz CPU and 96GB of memory.

From Table 14, we can see that although for most cases CS-JP-LP finshes within 10 seconds, it can be quite expensive for some cases (where the density of the bipartite graph is very high), and can take up to 27 minutes for the most expensive case. Since it uses a complex optimization formulation, the execution time also depends on the solver and the particular optimization-method used.

However, RS-JP, which is a simplified version of CS-JP-LP, takes well under 1 second for all join tasks we tested, which we think is reasonable alternative for cases where CS-JP-LP becomes too expensive. Furthermore, in terms of F-score, while RS-JP is still a few percentage points behind CS-JP-LP, we observe that this very efficient variant RS-JP is already substantially better than all TSJ-based techniques, where the gain can be as high as 20 percentage points. Note that TSJ20-LK-Oracle is the slowest since it has to explore a large space of fuzzy-join configurations.

Optimizing CS-JP-LP more efficiently for the expensive cases, by either exploiting its problem structure, or using more sophisticated parallel solver, is an area that requires more future work. But we emphasize that for the expensive cases, even if we just fall back to our simplified variant RS-JP, given its quality improvement and the fact that it runs efficiently, it is still a useful approach compared to some of the more manual alternatives discussed before.

5. RELATED WORK

Similarity join or fuzzy join (e.g., [9, 21]) considers a related problem of performing joins using similarity metrics such as Jaccard similarity. While this class of approaches allows joins beyond equality comparisons, it is limited to syntactic similarity and cannot handle the many semantic relationships discussed in this paper (e.g., city $\rightarrow$ state, among others listed in Figure 5 and 6).

Record linkage [14, 16, 20] refers to the process of identifying tuples that refer to the same real world entity. Our problem is different because the join relationship we consider are not limited to synonymous mentions of the same entities. Hassanzadeh et al. [16] propose an approach to explore available attributes to derive useful entity-linking rules, which can be very useful for knowledge-bases such as DBpedia and Freebase, where entities have rich attributes. Since the authors enumerated a comprehensive set of configurations for syntactic similarity join, we also experimentally compare with these configurations in [16].

In the context of Web tables, there are various efforts addressing the problem of targeted data integration and acquisition [6, 8, 22]. Here the input includes a base table, and additional information (e.g., keywords) specifying a desired column to be added to the base table, by joining the base table with tables from a corpus that has the desired column and a similar set of entities (e.g., extending a table of cities with a population column). Although this operation is known by the name of Extend in [8], Augmentation by Attribute (ABA) in [22], and Search Join in [6], respectively, the underlying task is conceptually similar. Given that the same entities have different representations in different tables, the authors in [8] and [22] propose to use syntactic similarity to perform joins, while the author in [6] propose to leverage annotated link in Linked Data [6] in addition. Compared to this line of work, our problem differs in two aspects. First, the problem specification is very different: Finding the right table to join is not the focus of our work – instead, we are already given two input columns, and the goal is to find semantic relationship at the instance level. Second, in terms of techniques used, our approach automatically discovers relationship using table corpora, instead of relying on resources like Linked Data, which may be powerful but can still be of limited coverage compared to the amount of structured data available on the Web.

6. CONCLUSIONS AND FUTURE WORK

Performing joins using semantic relationship is an important problem, in this work we build a SEMA-JOIN system that makes a first step toward the direction of fully automating such joins. There are a few interesting problems that warrant further investigations. First, in some scenarios we only know the tables, but not the exact columns, that need to be joined. Determining the joining columns without user input will be useful. Second, handling cases where two input columns alone do not uniquely determine the desired join semantics is also important. Third, some relationship is known to be temporal that can change over time. Performing joins on such cases is an open question. Lastly, in this work we calculated statistical co-occurrence using all columns in all tables. More advanced pruning of column pairs unlikely to contribute to semantic join relationship (e.g., many-to-many column pairs such as departure/arrival cities) should improve the quality of resulting calculation and is also an area for future study.

7. REFERENCES

[1] Fips country codes. http://en.wikipedia.org/wiki/List_of_FIPS_country_codes.

[2] Google Web Tables. http://research.google.com/tables.

[3] Iso country codes. http://en.wikipedia.org/wiki/ISO_3166-1.

[4] Microsoft Excel Power Query. http://office.microsoft.com/powerbi.

[5] Microsoft Solver Foundation. http://msdn.microsoft.com/en-us/library/ff524509.aspx.

[6] C. Bizer. Search joins with the web. In *Proceedings of International Conference on Database Theory (ICDT)*, 2014.

[7] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of SIGMOD*, 2008.

[8] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. In *VLDB*, 2009.

[9] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of ICDE*, 2006.

[10] A. Chuklin, P. Serdyukov, and M. de Rijke. Modeling clicks beyond the first result page. In *Proceedings of CIKM*, 2013.

[11] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. In *Computational Linguistics*, 1990.

[12] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. In *Transactions on Information Systems*, 2000.

[13] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. Stonebraker, and D. A. Wood. Implementation techniques for main memory database systems. In *Proceedings of SIGMOD*, 1984.

[14] A. Elmagarmid, P. G. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2007.

[15] U. Feige and M. Seltser. On the densest k-subgraph problem. In *Algorithmica*, 1997.

[16] O. Hassanzadeh, K. Q. Pu, S. H. Yeganeh, R. J. Miller, L. Popa, M. A. Hernández, and H. Ho. Discovering linkage points over web data. In *Proceedings of VLDB*, 2013.

[17] P. Pardalos and S. Vavasis. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global Optimization*, 1991.

[18] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, I. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *proceedings of SIGMOD*, 1979.

[19] T. Simpson. Evaluating google as an epistemic tool. In *Metaphilosophy*, 2012.

[20] R. H. Warren and F. W. Tompa. Multi-column substring matching for database schema translation. In *Proceedings of VLDB*, 2006.

[21] C. Xiao, W. Wang, X. Lin, and J. Yu. Efficient similarity joins for near duplicate detection. In *Proceedings of WWW*, 2008.

[22] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of SIGMOD*, 2012.