Heuristic joins to integrate structured heterogeneous data

Scott B. Huffman and David Steier
Price Waterhouse Technology Centre
68 Willow Road
Menlo Park, CA 94025
huffman@tc.pw.com

Abstract

Heterogeneous data sources often exhibit semantic heterogeneity at the data level; that is, the same entity in the world is referred to in different ways both within and across sources. This paper discusses a framework for combining information from such sources, called *heuristic join*, that is an extension of the familiar equi-join for homogeneous sources. Heuristic join uses heuristic match operators rather than simple equality to determine whether tuples refer to the same entity. The inexactness of heuristic matching introduces a number of parameters into heuristic join that are not present in equi-joins. Our work is motivated by a real-world data integration problem that required the use of heuristic joins.

Introduction

There is a tendency to think of "information gathering" as primarily from publically available, unstructured sources like World-Wide Web documents. However, the advent of high-speed networks has also opened access to distributed *structured* data sources, such as relational databases. Large organizations, such as multinational corporations, can access structured data created by different divisions in different locations more easily than ever before. Combining this data into an organization-wide view can uncover vital information for performance analysis and decision support.

Unfortunately, in most cases data from different structured sources cannot be directly combined. The data sources are created by different people, in different locations, for a variety of purposes, and following a variety of conventions. Thus the sources can exhibit both schematic and data heterogeneity [Kim and Seo, 1991]. Researchers in multidatabases have developed a variety of techniques for handling schematic heterogeneity [Batini *et al.*, 1986]; typically, these involve direct mapping rules either between data sources (e.g., [Sheth and Larson, 1990]) or from each source to a global schema (e.g., [Collet *et al.*, 1991]).

This paper examines methods for combining data from multiple structured sources – here, relational databases – that exhibit semantic heterogeneity within the data, as opposed to at the schematic level. In particular, we focus on cases where the same thing in the world (e.g., the same customer, person, product, etc.) is referred to in different ways both *within* and *across* different sources, and there is no uniform mapping rule that can translate between them. We present a framework for dealing with this type of semantic data heterogeneity that is an extension of the standard equi-join in the relational model. An example of an information-gathering problem within Price Waterhouse is used to illustrate the framework.

Problem

We will call the individual "things in the world" that databases contain information about *instances*, and categories of instances *types*; e.g., `XYZ Company` is an instance of the type *company*.[1] The goal of combining data from multiple sources can be cast as the problem of creating associations between all information pertaining to the same instance, so that it can be uniformly accessed, summarized, etc. For example, we may want to analyze all interactions with `XYZ Company`, which are recorded in various divisional databases.

In homogeneous data sources, each instance of a given type is referred to in the same unique way throughout the sources. That is, each database relation includes an attribute (or combination of attributes) whose value(s) uniquely and uniformly identify instances of a particular type. We will call this attribute a *key* with respect to the type, or a *type-key*. Customers, for example, might have unique *customer-numbers* as a type-key. Each tuple containing information about a particular customer in any relation would contain the same customer number. To associate data from multiple relations, the relations are *joined* on a type-key. The *join operator* is typically equality; that is, tuples from the original relations that have equal

[1] We are not trying to be formal here, but simply using this terminology to refer to "the things data are about" for the purposes of this paper.

type-key values are combined by the join. Such a join is called an *equi-join*, denoted $X \bowtie Y$.

In heterogeneous relations, there is no uniform type-key across the set of relations. In fact, instances may not be uniquely identified even within a *single* relation. Therefore, equi-joins cannot be used to combine the relations. Conceptually, however, we still want to perform a join – combining tuples from different relations that refer to the same instance. Equality is simply not a powerful enough join operator to identify these tuples in the absence of type-keys.

Thus, we propose a type of join that uses *heuristic* match operators to identify tuples that refer to the same instance. Because heuristic matching is not 100% accurate, these *heuristic joins* raise a number of issues that do not occur with equi-joins. We examine these issues after describing a real-world example.

Example: The *PW Service History* problem

Like most large organizations, Price Waterhouse (PW) serves its clients through many contact points. For decision support at the national level, it is often important to determine PW's overall service history with a given client (that is, what services were billed to the client in a given time period, by which offices, etc). The service history includes services for any subsidiaries, private owners, etc., of the client.

Figure 1 shows the data sources involved in determining service histories.[2] Each PW office records engagements with clients in datasources which, when combined, form a database we'll call *PW-engagements*. As a simplification, let's assume this database consists of a single relation. *PW-engagements* tuples consist of a short textual *description* of the engagement (often containing the client's name), a unique *client-code* that PW assigns to each client, and a variety of attributes containing information about the engagement (e.g., *amount-billed*, *lead-partner*). In addition to *PW-engagements*, there is a database we'll call *Subsidiaries* that contains information about who owns who. Each tuple in *Subsidiaries* contains a unique identifier of companies called a *DUNS-number*, the *name* of the company, and the DUNS number of the company's ultimate parent company, *parent-DUNS*.

To determine a client's service history, we must find all its subsidiaries using the *Subsidiaries* table, and then find all engagements for those subsidiaries in the *PW-engagements* table. To find these engagements, we must match instances of the type *company* across the two sources. However, there is no type-key available to match over.

Rather, Figure 2 diagrams how instances of *company* are identified in the relations. Both relations contain keys with respect to *company*, but they are not comparable. Both also contain textual descriptions. However, these descriptions cannot be matched using equality. A company in *Subsidiaries* named "SPRING SYMPOSIUM CORP" might be referred to in *PW-engagements* by all of "Spr Symp Corporation", "SS Co – audit", "Tax work" (with Spring Symposium's client code), "John and Linda Symposium, Taxes", "S. Symposium Corp", and so on. Rather than equality, a heuristic match can be used to match instances of *company* based on these descriptions. Any additional comparable information about companies that is contained in both relations could also be used as evidence of a heuristic match; for example, both relations might contain textual descriptions of companies' lines of business.

[2]This is a simplification of the real problem, which involves more data sources and attributes.

Heuristic Join

A heuristic join uses a heuristic match operator to match tuples from its component relations. Given tuples from each component relation, the match operator returns a gradient score between 0 and 1, indicating the likelihood that the tuples refer to the same instance of a given type (e.g., to the same *company*). Match operators may be arbitrarily complex, involving multiple attributes from the component relations, queries of other data sources, etc. Clearly, to perform well, match operators must be specific to particular types (e.g., *company*, *person*, ...) and particular sets of attributes in component relations. The goal of this paper is not to argue for any particular heuristics, but to examine the implications of using heuristic matching in performing joins.

A heuristic join generates a new relation populated by values from tuples of the component relations that match one another – that is, that are thought to refer to the same instance. Tuples match if, under a given heuristic match operator, they achieve a match score above a certain threshold.

For the PW service history problem, assume we have a heuristic match operator for type *company* called HM. To retrieve a client's service history, we first select the client's subsidiaries from the *Subsidiaries* relation; call the result *Client-subsids*. Then the client's service history can be generated through a heuristic join that is denoted:

$$(PW\text{-}engagements \bowtie_{HM^T_{\mapsto}} Client\text{-}subsids)^{+(client-code)}$$

The syntax of this expression represents various parameters of heuristic joins (described below) that are not required for normal equi-joins. Each of these parameters is a consequence of the inaccuracy of the heuristic match process, as contrasted with the 100% accurate matching of equal type-keys in equi-joins. The parameters are:

- **Match threshold:** Each tuple from the first relation is matched with each tuple from the second.

PW–engagements

client-code	description	lead-partner	...
00001	Spr.Symp-tax	J. Smith	
00006	XYZ audit	M. Jones	
...	...	...	

Subsidiaries

DUNS-number	name	parent-DUNS	...
0276543	SPRING SYMP	1113334	
1234567	GUMBY CO	2221115	
...	...	...	

Figure 1: Data sources in the PW service history problem.

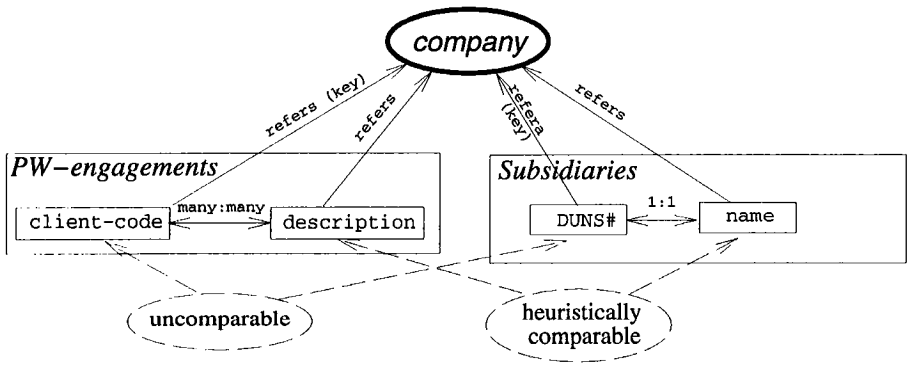

Figure 2: How instances of *company* are identified in the PW service history relations.

Only pairs that produce a match score above a specified threshold, T, are included in the join relation. Assuming that the match scores correctly reflect the accuracy of the match, adjusting T will produce either less inclusive, more accurate joins (higher T) or more inclusive, less accurate joins (lower T). That is, the threshold adjusts the tradeoff between recall and precision, standard in information retrieval problems. The match operator acts as an information retriever in that for every instance in the first relation, it retrieves matching instances from the second relation.

- **Directionality of match:** Match operators may be directional; that is, different results may be obtained when matching x to y than when matching y to x. *Substring* is a simple example of a directional match operator for textual fields. Depending on the data, one direction of match might make more sense than the other. The match direction is indicated by the arrow subscript of HM in the heuristic join expression above. Directionality does not arise for equi-joins since equality is symmetric.
- **Inclusion parameters:** In some cases it is appropriate to either limit or augment the matches included in a heuristic join.
  - *Single inclusion:* We may know that one (or both) of the relations contains only a single tuple for each instance. In such a case, only the best match from that relation should be included in the heuristic join.
  - *All-key inclusion:* In some cases, a component relation contains a key with respect to the type we are joining over. This key can be used to insure that all tuples referring to an instance are joined with a match from the other relation. For instance, *client-code* is a key for type *company* in the *PW-engagements* relation. Thus, after finding a match for "Spring symposium corp" with a *client-code* of **12345**, we want to include every other tuple in *PW-engagements* with that *client-code*. This catches cases like the tuple with the description "Tax work" that do not heuristically match "Spring symposium corp" but nonetheless correspond to the same instance. All-key inclusion is indicated by the notation (+ *client-code*) in the heuristic join expression above.

  Inclusion parameters are not needed for a 100% accurate match operator like equality, which always includes the right matches by definition.

For the service history problem, we have experimented with a variety of simple heuristic match operators (substrings, distinguishing keywords, initials, etc.). For our data, fairly simple heuristics are able to match the large number of "easy" cases, but they all produce incorrect matches for a variety of harder cases. Thus far, we have not calculated the actual precision and recall, primarily because of the huge volume and indeterminance of the data. Often it is difficult even for a human to decide whether a match is correct. For instance, in matching "SPRING SYMPOSIUM CORP",

is "SS Corp -tax" a good match? Perhaps, if that corporation goes by "SS" and no other corporations do. How about "Spring"? If no other company has the word "Spring" in its name, it may be a good match; if many companies contain "Spring", it is difficult to know which to choose. Our data contains many examples like these.

One final (and rather obvious) point: efficient match operators are crucial in dealing with large datasets. For two relations with M and N tuples, respectively, the match operator will be run on the order of $M \cdot N$ times. In our example problem, M and N are in the tens of thousands. We experimented with more powerful match operators that contained queries to determine the distinguishing power of keywords. Although they appeared to produce higher accuracy, the computational cost was unacceptable.

Related Work

The problem of the same concept being represented in different ways can occur at the schema level as well as at the data level. Automated approaches to detecting such conflicts at the schema level involve collecting and comparing meta-information about the meaning of schema elements (e.g., [Bhargava *et al.*, 1991]). Such meta-information about individual data items might be used within knowledge-driven heuristics to produce more accurate heuristic joins. However, this would require a large and constantly increasing amount of meta-information. One approach to using meta-information at the data level is discussed by Sciore *et al.* [1994].

Large-scale information agents and architectures, such as SIMS [Arens *et al.*, 1993], have not yet addressed the kind of fine-grained data heterogeneity discussed here. Rather, they appear to assume that if multiple sources contain data about the same entity, a type-key will be available in each source that uniquely identifies the entity. As the PW service history example illustrates, however, such type-keys are often unavailable. Thus, heuristic joins using a variety of match operators could be a useful addition to SIMS-like agents, allowing them to deal with another level of diversity in heterogeneous data. Because architectures like SIMS include detailed knowledge of their application domain and data sources, an interesting area for further research would be to explore automatically choosing heuristic match operators for given sources based on such knowledge.

Conclusion

To be useful, data drawn from heterogeneous sources that refer to the same things in the world must be combined. Heuristic join is a framework for combining information from structured heterogeneous sources that is an extension of the familiar equi-join for homogeneous sources. Heuristic join involves a number of parameters not needed in equi-join that arise from the inexact nature of heuristic matching. As a framework, heuristic join provides a vocabulary for discussing the problem and comparing potential solutions. It also brings to light several key research issues, such as the development of type-specific heuristic match techniques, balancing the tradeoff between recall and precision, and dealing with inaccuracy in join results.

References

[Arens *et al.*, 1993] Y. Arens, C. Y. Chee, C. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *Int'l. Journal on Intelligent and Cooperative Information Systems*, 2(2):127–58, 1993.

[Batini *et al.*, 1986] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4), 1986.

[Bhargava *et al.*, 1991] H. K. Bhargava, S. O. Kimbrough, and R. Krishnan. Unique names violations, a problem for model integration or You say tomato, I say tomahto. *ORSA Journal on Computing*, 3(2):107–120, 1991.

[Collet *et al.*, 1991] C. Collet, M. N. Huhns, and W. Shen. Resource integration using a large knowledge base in Carnot. *IEEE Computer*, pages 55–62, December 1991.

[Kim and Seo, 1991] W. Kim and J. Seo. Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*, pages 12–18, December 1991.

[Sciore *et al.*, 1994] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems*, 19(2):254–290, June 1994.

[Sheth and Larson, 1990] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3), 1990.