Supporting Imprecision in Database Systems

Ullas Nambiar
IBM India Research Lab, India

INTRODUCTION

A query against incomplete or imprecise data in a database[1], or a query whose search conditions are imprecise can both result in answers that do not satisfy the query completely. Such queries can be broadly termed as ***imprecise queries***. Today's database systems are designed largely for precise queries against a database of precise and complete data. Range queries (*e.g., Age BETWEEN 20 AND 30*) and disjunctive queries (*e.g., Name="G. W. Bush" OR Name="George Bush"*) do allow for some imprecision in queries. However, these extensions to precise queries are unable to completely capture the expressiveness of an imprecise query. Supporting imprecise queries (*e.g., Model like "Camry" and Price around "$15000")* over databases necessitates a system that integrates a similarity search paradigm over structured and semi-structured data. Today's relational database systems, as they are designed to support precise queries against precise data, use such precise access support mechanisms as indexing, hashing, and sorting. Such mechanisms are used for fast selective searches of records within a table and for joining two tables based on precise matching of values in join fields in the tables. The imprecise nature of the search conditions in queries will make such access mechanisms largely useless. Thus, supporting imprecise queries over existing databases would require adding support for imprecision within the query engine and meta-data management schemes like indexes.

Extending a database to support imprecise queries would involve changing the query processing and data storage models being used by the database. But, the fact that databases are generally used by other applications and therefore must retain their behaviour could become a key inhibitor to any technique that relies on modifying the database to enable support for imprecision. For example, changing an airline reservation database will necessitate changes to other connected systems including travel agency databases, partner airline databases etc. Even if the database is modifiable, we would still require a domain expert and/or end user to provide the necessary distance metrics and domain ontology. Domain ontologies do not exist for all possible domains and the ones that are available are far from being complete. Therefore, a feasible solution for answering imprecise queries should neither assume the ability to modify the properties of the database nor require users (both lay and expert) to provide much domain specific information.

BACKGROUND

The problem of supporting imprecise queries has already attracted considerable interest from researchers including those in fuzzy information systems (Morrisey, 1990), cooperative query answering and query generalization (Motro, 1998). More recent efforts have focused on supporting imprecise queries over relational databases by introducing ADTs (abstract data types) – for allowing storage and retrieval of non-standard data such as images, documents etc., and extending the query processor with functions for measuring similarity between tuples (Aditya et al, 2002; Ortega-Binderberger, 2003). Recently, work has been done on providing ranked answers to queries over a relational database (Bruno, Gravano and Marian, 2002). However, all the proposed approaches for answering imprecise queries require large amounts of domain specific information either pre-estimated or given by the user of the query. Unfortunately, such information is hard to elicit from the users. Further, some approaches require changing the data models and operators of the underlying database. A recent survey outlining challenges in integrating DB (database) and IR (information retrieval) technologies discusses the pros and cons of four possible alternatives for combining the two models (Chaudhuri, Ramakrishnan and Weikum, 2005).

MAIN FOCUS

The task of supporting imprecise queries over a database can borrow several ideas from efforts at integrating DB & IR systems. However, the key difference is that the focus is only on bringing the IR style search and retrieval model to DB systems with the underlying data continuing to be structured. The motivation should be to reduce the burden on the user by striving to satisfy the users' imprecisely defined need (query) with minimal additional information from user. Given that the database contains tuples generated by humans, they must capture some amount of real-world semantics e.g., relationships between attributes (features of the domain), similarity between values, etc. Solutions for supporting imprecise queries should focus on using the inherent semantics to help the user in extracting relevant information. Any solution then should be judged based on how good is its estimation of the user need based on the query and underlying data, i.e., *How closely does it model the user's notion of relevance by using only the information available in the database?*

A domain independent solution for supporting imprecise queries over autonomous databases is given in Nambiar and Kambhampati, 2006. The solution unites the DB and IR technologies by bringing the similarity searching/ranked retrieval paradigm from IR systems into the structured, type-rich access paradigm of databases. Answers are ranked according to the degree of relevance automatically estimated using domain-independent similarity functions that can closely approximate the subjective interpretation of the user. An intuitive model for measuring the similarity between the query and the answer tuples is by measuring the similarity of common attributes between them. Not all attributes will be equally relevant to the user. Therefore, providing a ranked set of answers with minimal user input and domain related metrics would require techniques to automatically learn the similarity between values binding each attribute and also the importance of every attribute in a relation.

Measuring Value Similarity

A database system supporting imprecise queries must provide information about how close an answer tuple is to the given imprecise query. Two tuples (a selection query can be seen as a tuple with few missing values) are considered similar if they have *syntactical similarity* (e.g., same subset of attributes are bound in both queries, stems of a common word bind an attribute, etc) or if the binding values are *semantically similar*. Semantic similarity is a concept whereby a set of words (attribute values) are assigned a metric based on the closeness of their meaning. Similar words can be considered semantically related by virtue of their *synonymy* (e.g., bank – trust company), but dissimilar entities may also be semantically related by lexical relationships such as *meronymy* (e.g., car - wheel) and *antonymy* (e.g., hot - cold), or just by any kind of functional relationship or frequent association (e.g., pencil - paper, penguin - Antarctica, rain - flood). The definition of synonym by Leibniz - *"synonyms are words that are interchangeable in some contexts"* is considered more realistic. This definition forms the basis of the similarity estimation model developed in Nambiar and Kambhampati, 2006. Given a database of tuples, the authors assume that binding values that are semantically similar have similar distributional behaviour. Under this assumption, they treat the values that co-occur near a value as constituting features that describe the context in which the given value appears in the database. The semantic similarity between two values is then computed in terms of how similar is their contexts. This is accomplished by building a structure consisting of bags of words for all attributes in the relation not bound by the two values being considered. Only values that co-occur with the value under consideration is added to the bag of words. The similarity between two values is then computed as the level of commonality between the bags of words describing them (see Nambiar and Kambhampati, 2006 for details).

Learning Attribute Importance

Often users would like to see only the top-k answers to a query. To provide ranked answers to a query, we must combine similarities shown over distinct attributes of the relation into an overall similarity score for each tuple. Specifically, a measure of importance for the similarity shown over any attribute in the context of a given query may be necessary to determine the best k matches. While this measure may vary from user to user, most users usually are unable to correctly quantify the importance they ascribe to an attribute. In theory the tuples most similar to query will have differences only in the *least important attribute*. Nambiar and Kambhampati, 2006, define the least important at-