Bulletin of the Technical Committee on

Data Engineering

September 2001 Vol. 24 No. 3 **IEEE Computer Society**

Letters

Special Issue on Imprecise Queries

Announcements and Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

The Data Engineering Bulletin web page is http://www.research.microsoft.com/research/db/debull.

Letter from the Editor-in-Chief

TCDE Chair Election

I would like to draw your attention to the message on page 50 from Paul Larson, chair of the TCDE Nominating Committee. The TC will be electing a new TCDE Chair, as Betty Salzberg's term as Chair ends in December. The Nominating Committee has nominated Erich Neuhold, the current Vice Chair of the TCDE to run for Chair of the TCDE. Nominations for TCDE Chair were also solicited from TCDE members. There were no nominations submitted.

Now I would urge you to participate in the electoral process. While Erich Neuhold is running unopposed, voting is important as a way to make your intent clear. Having sufficient number of TCDE members voting helps to ensure that the electoral process works well and results in the outcome that members intend.

The Current Issue

As the database industry, and the database products that are sold by the industry, mature it is clear that simply answering queries that are precisely stated, and that have precise results, at which our current database systems excel, does not always meet the needs of users. This is particularly true when the user's intent is exploratory in nature. Not only does such a user not wish to provide precision in his request, but such precision may not be possible at his current state of knowledge. Further, such a user will frequently be willing to trade precise query results for approximate results, when the approximate results can be delivered much faster than the precise results.

Sunita Sarawagi, who acted as the editor for this issue, contacted researchers who are active in the field of imprecise queries. They have responded to Sunita's reasonably precise request about work on imprecise queries by providing papers that report on their recent work or work in progress in this exciting area. Such work expands on the usual database query processing techniques, exploiting combinations of techniques from information retrieval, probability theory, and totally new methods. It does not take great foresight to realize that this field will be one that researchers will be exploring for years to come. Sunita surely deserves our collective thanks for assembling the current issue, which, with its diverse collection of approaches, helps us all to stay current in this young and fast moving field.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

I know not what I want, I want not what I get. ....Rabindra Nath Tagore (translated)

The topic of this special issue is *imprecise queries* arising where a user does not know how to precisely express his needs but when presented with mixed results will know what subset he wants and what he does not. The conventional approach of a one-shot subsetting of results specified through a fixed set of predicates will not work in such cases. The goal of this special issue is to make a compendium of tricks that can be used to answer imprecise queries. The topic of imprecise queries is different from the popular topic of approximate queries where the query is exact but the answer is imprecise or approximate.

Two trends in database usage has motivated the need to support imprecise queries in database engines. First is the increasing complexity of the types of data in the form of images, sound clips, and hypertext documents. The extent and variety of attributes of these data types make it hard for an end-user to specify exact queries on them. Second is the growing externalization of databases where an external user querying the data source does not know (and does not want to know) the exact schematic details of the data. Online retail catalog searches by end customers is a prime example of this kind of search. A customer would typically like to search the catalog database using a combination of smart navigation and natural keyword queries without any regard to the physical schema design.

Several techniques have been proposed to find relevant data even when a user cannot quantify relevance through a precise query. The first trick is iterative refinement where the user's query is composed in multiple rounds of interaction instead of the traditional one-shot approach of conventional querying. In each round, the user provides feedback to the system on the subset of answers that he found relevant and optionally modifies the original query. Another important idea is to exploit a notion of proximity or similarity between objects to retrieve objects that are close to a set of seed objects that the user found relevant. Finally, to compensate for the lack of a precise threshold to quantify the relevance data items, the result set is ranked according to a decreasing order of relevance measured via a user or system-induced scoring function.

In this issue, we present a collection of five articles that covers various kinds of data sources and deploys one or more of the above set of tricks to support the imprecise world of user queries. The first two papers "Query Refinement in Similarity Search Systems" by Chakrabarti and Mehlotra, and " Multimedia Queries by Example and Relevance Feedback" by Wu, Faloutsos, Sycara and Payne elaborate how the system refines its internal representation of the user's query concept through multiple rounds of relevance feedback. The query concept consists of a weighted set of objects already found relevant, a distance function to measure proximity between objects, and a function that combines the distances to each of the seed relevant objects. The two papers differ in the forms of these functions and how they get refined. The third paper "Keyword Searching in Databases by Bhalotia, Nakhey, Hulgeri, Chakrabarti and Sudarshan discusses how a relational database can be imprecisely queried using a set of keywords. The answer set returns tuples sorted according to an interesting notion of proximity spanning multiple relations connected through reference links. The paper also presents an exhaustive survey of other systems that support keyword search in relational databases, particularly in catalog databases. The fourth paper, "Using probabilistic argumentation systems to search and classify Web sites" by Picard and Savoy presents a theoretical framework for how hyperlinks in documents can be used to "spread activation" about the relevance of documents in a hypertext repository. The final paper "Towards Scalable Scoring for Preference-based Item Recommendation" by Stolze Rjaibi concentrates on the implementation issues of supporting user specified scoring functions for ranking products based on user preference.

S. Sarawagi
IIT Bombay

Query Refinement in Similarity Retrieval Systems *

Kaushik Chakrabarti
University of Illinois
kaushikc@cs.uiuc.edu

Michael Ortega
University of Illinois
miki@acm.org

Kriengkrai Porkaew
King Mongkut's University
porkaew@it.kmutt.ac.th

Sharad Mehrotra
University of California
sharad@ics.uci.edu

Abstract

In many applications, users specify target values for certain attributes/features without requiring exact matches to these values in return. Instead, the result is typically a ranked list of the top k objects that best match the specified feature values. User subjectivity is an important aspect of such queries, i.e., which objects are relevant to the user and which are not depends on the perception of the user. Due to the subjective nature of similarity-based retrieval, the answers returned by the system to a user query often do not satisfy the user's information need right away; either because the weights and the distance functions associated with the features do not accurately capture the user's perception or because the specified target values do not fully capture her information need or both. The most commonly used technique to overcome this problem is query refinement. In this technique, the user provides to the system some feedback on the "relevance" of the answers to the user's query. The system then analyzes the feedback, refines the query (i.e., modifies the weights, distance functions, target values etc.) evaluates it and returns the new results. In this paper, we provide an overview of the techniques used to construct the refined query based on the feedback from the user as well as the techniques to evaluate the refined query efficiently. We present experimental results demonstrating the effectiveness of the techniques discussed in the paper.

1 Introduction

Similarity-based retrieval is becoming common in many modern-day database applications. Unlike in a traditional relational database system (RDBMS) where a selection query consists of one or more precise selection conditions and the user expects to get back the exact set of objects that satisfy those conditions, in similarity queries, the user specifies target values for certain attributes and does not expect exact matches to these values in return. Instead, the result to such queries is typically a *ranked list* of the top k objects that best match the given attribute values. As the following examples illustrate, similarity queries arise naturally in a variety of today's applications.

Example 1 (Multimedia Databases): Consider a content-based image retrieval system [13, 24, 15]. Each image is represented using visual features like color, texture, layout and shape [14]. The similarity between any

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was partially supported by the NSF under grant numbers IIS 0083489, ITR IIS 0086124 and CAREER award IIS-9734300.

two images is computed by first computing their similarities based on the individual features and then combining them to obtain the overall similarity. Typically, the user submits an example as the query object and requests for a few objects that are "most similar" to the submitted example (Query By Example (QBE)). The DBMS ranks the images according to how well they match the query image, and returns the best few matches to the user in a ranked fashion, the most similar images first followed by the less similar ones.

Example 2 (E-commerce): Consider a real-estate database that maintains information like the location of each house, the price, the number of bedrooms etc [5] (e.g., MSN HomeAdvisor). Suppose that a potential customer is interested in houses in the Irvine, CA area, with four bedrooms and with a price tag of around 300,000 dollars. Again, the DBMS should rank the available houses according to how well they match the given user preference, and return the top houses for the user to inspect. If no houses match the query specification exactly, the system might return houses in Santa Ana (a city near Irvine) or 2 bedroom houses in Irvine or more expensive houses as the top matches to the query.

An important aspect of similarity queries is user subjectivity [11, 7]. In Example 1, let us assume that there are just 2 features: color and texture. Let $Q = \langle Q_C, Q_T \rangle$ be the query image where $Q_C = (0.2, 0.4)$ and $Q_T = (0.4, 0.5)$ are the color and texture vectors extracted from Q (both feature spaces are 2-dimensional). Let us consider two objects (i.e., images) in the database: A and B. Let $A_C = (0.4, 0.5)$ and $A_T = (0.9, 0.9)$ while $B_C = (0.9, 0.3)$ and $B_T = (0.3, 0.4)$. Assuming the distance functions (DFs) over both the color and texture features to be Euclidean (all dimensions weighted equally), the distances between Q and A with respect to color and texture are $Euclidean(Q_C, A_C) = 0.15$ and $Euclidean(Q_T, A_T) = 0.45$ respectively; those between Q and B are $Euclidean(Q_C, B_C) = 0.50$ and $Euclidean(Q_T, B_T) = 0.10$ respectively. Which is one is a better match to Q? It depends on the user: if similarity with respect to color is more important to the user than that with respect to texture, A is a better match; otherwise B is better. The same issue arises in Example 2. Let us consider the query $Q = \langle Irvine, 4, 300000 \rangle$ in Example 2 (the first, second and third elements are the location, the number of bedrooms and the price in dollars respectively). Let us consider two objects (i.e., houses) in the database: $A = \langle Irvine, 3, 400000 \rangle$ and $B = \langle SantaAna, 4, 325000 \rangle$. Which one is a better match to Q? Again, it depends on the user. If she is very particular about the location and is flexible about the price, A is a better match. On the other hand, if she has a limited budget and needs 4 bedrooms, B is better. To return "good quality" answers, the system must understand the user's *perception* of similarity which is captured by the system using weights and DFs. [6, 3, 16, 5]. At the time of the query, the system *acquires* information from the user based on which it determines the weights and DFs that best capture the perception of this particular user. Based on these weights and DFs, it then retrieves the top answers by first executing a k nearest neighbor (k-NN) algorithm on each individual feature[1] and then "merging" them to get the overall answers [15, 12, 6, 7].

Due to the subjective nature of similarity queries, the answers returned by the system to a user query often do not satisfy the user's need right away [16, 3, 20, 9]. One reason is that the starting weights and DFs may not accurately capture the perception of the user. This happens when the user does not provide enough information to the system initially, either because it is too burdensome for her to do so or because she does not have a clear mental model of her exact information need at the beginning of the query. Another reason for unsatisfactory answers is that the starting examples/target values may not be the best ones to capture the information need (IN) of the user. The most commonly used technique to overcome this problem is query refinement. In this technique, the user provides to the system some feedback on the relevance of answers returned, i.e., she specifies which answers are relevant to her query and which are not [19, 20, 16, 3, 9]. [2] The system then analyzes the feedback,

[1] In this paper, we assume that all the feature spaces are metric and an index (called the Feature-index or F-index) exists on each feature space. A F-index is either single dimensional (e.g., B-tree) or multidimensional (e.g., R-tree) depending on the feature space dimensionality.

[2] An alternate way to provide feedback is to explicitly modify the weights and/or the DFs so as to better capture her perception of similarity [11, 7]. This requires the user to have more domain expertise compared to the example-based feedback; the techniques discussed in this paper apply to this type of feedback as well.

refines the query (i.e., modifies the weights, DFs, target values etc.), evaluates it and returns the new results. The user can continue refining the query over many iterations until she is fully satisfied with the results.

In this paper, we provide an overview of the techniques used to construct the refined query based on the feedback from the user. This includes computing the new query points/values for each feature, the new intra-feature DFs and the new inter-feature weights. We also discuss techniques to evaluate the refined query efficiently. We present experimental results demonstrating (1) the effectiveness of the query refinement techniques in terms of improvement in the the quality of answers returned by the system and (2) the efficiency of the techniques for evaluating refined queries.

2 Query Refinement Techniques

In a similarity retrieval system, the user poses a query Q by providing target values of one or more features and specifying the number k of matches desired. [3] The target values can be either explicitly specified by the user (as in Example 2) or extracted from the examples submitted by the user (as in Example 1). We refer to Q as the 'starting' query. The starting query is then matched to the set of objects in the database and the top k matches are returned. Subsequent matches can be retrieved incrementally. If the user is not satisfied with the answers, she provides feedback to the system. Based on the feedback, the system modifies the query representation to better suit the user's information need. The 'refined' query is then evaluated and top results are returned with additional matches being retrievable incrementally. The feedback process can be continued for several iterations until the user is fully satisfied. In this section, we first discuss the representation of database objects (object model) and user queries (query model) and formally define the notion of similarity between the query and objects. We then present the techniques used to construct the refined query based on user feedback.

2.1 Representation of Objects

Let $\mathcal{F}$ be the set of features. For example, in Example 1, $\mathcal{F} = \{\rceil\wr\updownarrow\wr\nabla, \sqcup\rceil\S\sqcup\sqcap\nabla\rceil\}$. Each object O in the database can be viewed as a collection $O = \{O_F | F \in \mathcal{F}\}$ of feature vectors where O_F denotes O's value (a vector) of feature F. How the O_F's are obtained from O (i.e., the feature extraction) depends on the application. (e.g., in Example 1, special image processing routines are used to extract the color and texture from the image). We associate a multidimensional feature space R_F, of dimensionality d_F, with each feature $F \in \mathcal{F}$ and view O_F as a point in R_F. In Example 1, we associate a 2-d feature space R_C with color and a 2-d feature space R_T with texture. We can then view the object A mentioned in Section 1 as the point $(0.4, 0.5)$ in R_C and the point $(0.9, 0.9)$ in R_T.

2.2 Representation of Queries

Similar to the object, a query Q (either starting or refined query) is also represented as a collection $Q = \{Q_F | F \in \mathcal{F}\}$ of single feature (SF) queries where Q_F denotes the SF query for feature F. The Q_F's are obtained from Q in the same way as O_F's are extracted from O. However, unlike O that is represented using a single point O_F in each feature space R_F, Q_F may consist of *multiple* points in R_F. The reason is that during refinement, the user might submit multiple examples to the system as feedback (those that she considers relevant) leading to multiple points per feature space (cf. Section 2.4). We refer to such queries as *multipoint queries*. The user may also specify the relative importance of the submitted examples, i.e., how important each example is in capturing her information need. To account for that, we associate a weight with each point of the multipoint query. We now formally define a multipoint query.

[3] We restrict our discussion to selection queries in this paper.

Definition 1 (Multipoint Query): A multipoint query $Q_F = \langle n_F, \mathcal{P}_\mathcal{F}, \mathcal{W}_\mathcal{F}, \mathcal{D}_\mathcal{F} \rangle$ for feature F consists of the following information:

- the number n_F of points in Q_F
- a set of n_F points $\mathcal{P}_\mathcal{F} = \{\mathcal{P}_\mathcal{F}{}^{(\infty)}, ..., \mathcal{P}_\mathcal{F}{}^{(\backslash_\mathcal{F})}\}$ in the d_F-dimensional feature space R_F
- a set of n_F weights $\mathcal{W}_\mathcal{F} = \{\sqsupseteq_\mathcal{F}{}^{(\infty)}, ..., \sqsupseteq_\mathcal{F}{}^{(\backslash_\mathcal{F})}\}$, the ith weight $w_F{}^{(i)}$ being associated with the ith point $P_F{}^{(i)}$ $(1 \geq w_F{}^{(i)} \geq 0, \Sigma_{i=1}^{n_F} w_F{}^{(i)} = 1)$.
- a distance function $\mathcal{D}_\mathcal{F}$ which, given two points in R_F, returns the distance between them. We assume $\mathcal{D}_\mathcal{F}$ to be a weighted L_p metric, i.e., for a given value of p, the distance between two points S and T in R_F is given by[4]

$$\mathcal{D}_\mathcal{F}(\mathcal{S}, \mathcal{T}) = [\pm_{|=\infty}^{\lceil_\mathcal{F}} \mu_\mathcal{F}{}^{(|)}(|\mathcal{S}[|] - \mathcal{T}[|]|)^{\surd}]^{\infty/\surd} \quad (1)$$

  where $\mu_F{}^{(j)}$ denotes the (*intra-feature*) weight associated with the jth dimension of R_F. $(1 \geq \mu_F{}^{(j)} \geq 0, \Sigma_{j=1}^{d_F} \mu_F{}^{(j)} = 1)$. D_F specifies which L_p metric to use (i.e., the value of p) and the values of the intra-feature weights. ■

The choice of $\mathcal{D}_\mathcal{F}$ (i.e., the choice of the L_p metric) and the intra-feature weights captures the user perception within the feature F as shown in Example 3.

Example 3: In Example 1, let us consider the distance between Q and B with respect to the color feature, i.e., $\mathcal{D}_{\rfloor\wr\updownarrow\wr\bigtriangledown}(\mathcal{Q}_\mathcal{C}, \mathcal{B}_\mathcal{C})$. With L_2 and equal weights, $\mathcal{D}_{\rfloor\wr\updownarrow\wr\bigtriangledown}(\mathcal{Q}_\mathcal{C}, \mathcal{B}_\mathcal{C})$ is 0.50. If the user weighs the first dimension twice as much as the second, $\mathcal{D}_{\rfloor\wr\updownarrow\wr\bigtriangledown}(\mathcal{Q}_\mathcal{C}, \mathcal{B}_\mathcal{C})$ is $\sqrt{\frac{2*(0.2-0.9)^2+(0.4-0.3)^2}{3}} = 0.58$, i.e., Q and B are not a close match in the color space. If she weighes the second dimension twice as much as the first, $\mathcal{D}_{\rfloor\wr\updownarrow\wr\bigtriangledown}(\mathcal{Q}_\mathcal{C}, \mathcal{B}_\mathcal{C})$ is $\sqrt{\frac{(0.2-0.9)^2+2*(0.4-0.3)^2}{3}} = 0.41$, i.e., Q and B are a better match in this case than before. The intra-feature weights thus capture the user's subjective perception of similarity of color and determine what constitutes a close match in terms of color and what is not. How the weights are chosen for a user is discussed in Section 2.4.

2.3 Definition of Similarity between Query and Objects

We define similarity using distance functions, similarity being inversely proportional to distance. We start with the DFs for individual features followed by the overall multifeature DF. The distance $D_F(Q_F, O_F)$ between the multipoint query Q_F and an object point O_F (in R_F) with respect to feature F is defined as the aggregate of the distances between O_F and the individual points $P_F{}^{(i)} \in \mathcal{P}_\mathcal{F}$ in Q_F. We use weighted sum as the aggregation function but any other function can be used as long as it is weighted and monotonic [6].

$$D_F(Q_F, O_F) = \sum_{i=1}^{n_F} w_F{}^{(i)} \mathcal{D}_\mathcal{F}(\mathcal{Q}_\mathcal{F}{}^{(\rangle)}, \mathcal{O}_\mathcal{F}) \quad (2)$$

For example, let the query Q consist of multiple points in the color space. Let $Q_C = \{2, \{(0.2, 0.4), (0.4, 0.1)\}, \{0.7, 0.3\},$ Euclidean (equal weights) $\}$, then,

$$D_{color}(Q_C, B_C) = 0.7 * \sqrt{\frac{(0.2-0.9)^2 + (0.4-0.3)^2}{2}} + 0.3 * \sqrt{\frac{(0.4-0.9)^2 + (0.1-0.3)^2}{2}} = 0.46. \quad (3)$$

Note that the multipoint query is a generalization of the single point query (the latter is a special case of the former with $n_F = 1$).

[4]Note that this assumption is general since most commonly used DF (e.g., Manhattan distance, Euclidean distance, Bounding Box distance) are special cases of the L_p metric. However, this excludes DFs that involve "cross correlation" among dimensions. Handling cross-correlated functions has been addressed in [22] and can be incorporated to the techniques developed in this paper.

Once we have determined the individual feature-wise distances $D_F(Q_F, O_F), F \in \mathcal{F}$, we compute the overall (i.e., multifeature) distance $D(Q, O)$ between the query Q and object O by aggregating the feature-wise distances:

$$D(Q, O) = \sum_{F \in \mathcal{F}} \nu_F D_F(Q_F, O_F) \tag{4}$$

where ν_F denotes the (inter-feature) weight associated with feature $F \in \mathcal{F}$ ($\nu_F \geq 0, \Sigma_F \nu_F = 1$). While the intra-feature weights model user perception *within* each feature, the inter-feature weights model user perception *between* features, i.e., how important the features are *relative* to each other. How the weights are obtained for a particular user is discussed in Section 2.4. The system should return the following set $Ans(Q)$ of objects as the answers to query Q: $Ans(Q) = \{O | \forall O' \in (DB - Ans(Q)), D(Q, O) \leq D(Q, O')$.

Example 4: In Example 1, let us consider the distance between Q and objects A and B when the two features are equally weighted, i.e., $\nu_C = \nu_T = 0.5$ (assuming Euclidean distance with equal weights for both the features). $D(Q, A) = \frac{Euclidean(Q_C, A_C) + Euclidean(Q_T, A_T)}{2} = 0.30$ and $D(Q, B) = \frac{Euclidean(Q_C, B_C) + Euclidean(Q_T, B_T)}{2}$ $= 0.30$, i.e., A and B are equally good matches. If color is weighed twice as much as texture, $D(Q, A) = \frac{2*0.15+0.45}{3} = 0.25$ and $D(Q, B) = \frac{2*0.50+0.10}{3} = 0.37$, i.e., A is a better match. If texture is weighed twice as much as color, $D(Q, A) = 0.35$ and $D(Q, B) = 0.23$, i.e., B is a better match.

2.4 Computing the weights/DFs based on feedback

As discussed above, we model the user perception using the intra-feature and inter-feature weights and the DFs. But how do we obtain the right values of the weights/DFs that best capture the user's perception? One solution is for the user to explicitly specify the weights. This technique was proposed by Motro and was used in the VAGUE system [11]. Otherwise, we start with an arbitrary set of weights (e.g., all weights equal). [5] We then execute the starting query and return the best matches to the user. If the user is not satisfied with the answers, either the starting weights and DFs do not accurately capture the user's perception or the query points do not properly capture the user's information need or both. We now discuss techniques to modify, based on user feedback, the query points and the weights/DFs so as to represent the user's information need better.

- **Intra-feature Refinement**: Within each feature, the system modifies the position of the query point(s) $\mathcal{P}_\mathcal{F}$ in each feature space $F \in \mathcal{F}$ as well as their weights $\mathcal{W}_\mathcal{F}$. The intra-feature weights (i.e. the μ_{Fj}'s in Equation 1) are also modified to better capture the user's perception within each feature. [6] If the query interface is the explicit attribute value specification interface (as in Example 2), the user can modify the attribute values explicitly (e.g., modify query from $\langle Irvine, 4, 300, 000\rangle$ to $\langle Irvine, 3, 350, 000\rangle$). The same is true for intra-feature weights. If the query interface is QBE, the user submits relevant examples based on which the system computes the new query points and weights for each feature space $F \in \mathcal{F}$. Two models have been proposed for the above computation:
  - *Query Point Movement (QPM)*: In this model, the relevant examples submitted by the user during feedback are represented by a single point in each feature space F: the weighted centroid of the points corresponding to the submitted examples in F (see Figure 1). For example, if A and B are the two relevant examples submitted by the user (cf. Section 1) and A is indicated to be twice as relevant as B, the single query point for the color space is $(\frac{2*0.4+1*0.9}{3}, \frac{2*0.5+1*0.3}{3})$ and that for the texture space is $(\frac{2*0.9+1*0.3}{3}, \frac{2*0.9+1*0.4}{3})$.
    The QPM model computes the intra-feature weights for F (i.e. the μ_{Fj}'s in Equation 1) as follows. Let σ_{Fj} denote the standard deviation of the feature values of the relevant examples for feature F along the

[5] The initial weights can be chosen more intelligently. [1] proposes to collect feedback information provided by different users over time and use it to determine the best initial weights for the particular user. This technique reduces the number of feedback iterations, especially for already-seen queries.

[6] Besides these 2 steps, some researchers have considered cross-correlation among the dimensions in R_F [9].

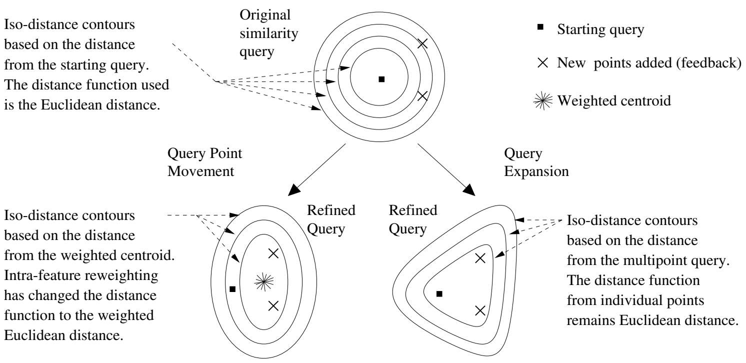


Figure 1: Query Refinement Models

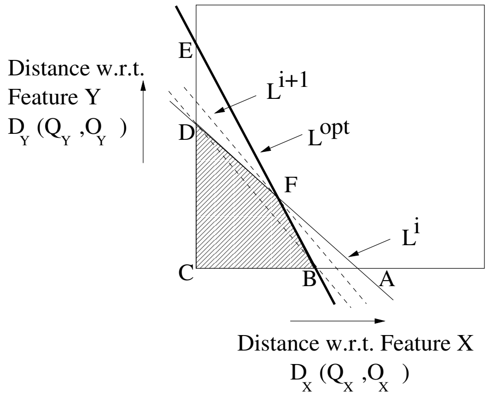


Figure 2: Inter-feature Reweighting

jth dimension. μ_{Fj} is assigned the value $\frac{1}{\sigma_{Fj}}$. The intuition is that among the objects considered relevant by the user, the higher the variance along a dimension, the lower the significance of that dimension.
This approach is used in MARS [19, 21, 20] and also in Mindreader [9]. For each feature space, it chooses a single query point and reweighs the dimensions so that the sum of its distances from the relevant points is minimized [9]. With each iteration of relevance feedback, the query point moves moves towards the relevant points and the distance function is adjusted to fit the distribution of the relevant points.

- *Query Expansion (QEX)*: In this model, the submitted examples are represented by *multiple* points $\mathcal{P}_\mathcal{F}$ (called *representatives* [16]) in each feature space $F \in \mathcal{F}$, giving rise to *multipoint* queries (see Figure 1). The weight ${w_F}^{(i)}$ of any representative $P_F^{(i)}$ in the multipoint query is proportional to the total weight of all the objects $P_F^{(i)}$ represents. For example, if A and B are the two relevant examples submitted by the user and A is indicated to be twice as relevant as B, one possible choice of representatives is the points themselves with weights 0.67 and 0.33 respectively for both color and texture spaces. If the user submits many examples, a better approach is to select a small number of good representatives to construct the multipoint query for each feature space. [16] proposes to use a clustering algorithm to cluster the relevant points in each feature space and use the cluster centroids as representatives for that feature. The weight of each representative is chosen to be proportional to the number of relevant objects in the corresponding cluster [16]. The QEX model does not modify the intra-feature weights (i.e. the μ_{Fj}'s in Equation 1) as it adjusts the intra-feature distance function by adding new query points (see Figure 1). This model was proposed in MARS [16, 17, 18].

- **Inter-feature Refinement**: The system modifies the inter-feature weights (the ν_F's in Equation 4) in order to better capture the user's perception across features. If the query interface allows explicit weight specification, the user can modify the weights directly as in Motro's VAGUE system [11]. In a QBE environment, the system automatically derives the inter-feature weights from the relevant examples submitted by the user [17]. A simple approach is to choose ν_F to be proportional to the number of relevant examples in the top t matches with respect to feature F (t is selected by the system) [21]. More the number of top matches of a feature marked relevant by the user, more important is the feature to this user.
A more advanced approach to determine inter-feature weights was proposed in [17]. For simplicity, let us assume that the query Q consists of two features: X and Y. We assume that there exists weights ν_X^{opt} and ν_Y^{opt} such that $\nu_X^{opt} D_X(Q_X, O_X) + \nu_Y^{opt} D_Y(Q_Y, O_Y)$ accurately captures the user's perception of distance between the query Q and an object O (see Equation 4). The user's perception can be visualized as a line L^{opt} defined by the following equation in the distance space defined over features X and Y (illustrated in Figure

2, L^{opt} is the bold line BE):

$$\nu_X^{opt} D_X(Q_X, O_X) + \nu_Y^{opt} D_Y(Q_Y, O_Y) = \delta \tag{5}$$

δ is a threshold such that any object with distance below δ is deemed relevant by the user. The slope of L^{opt} represents the relative importance of the features X and Y.

Let ν_X^i and ν_Y^i denote the weights used by the system for the ith iteration of feedback. The corresponding line L^i is defined by (Line AD in Figure 2):

$$\nu_X^i D_X(Q_X, O_X) + \nu_Y^i D_Y(Q_Y, O_Y) = \delta^i \tag{6}$$

Consider the line BD in Figure 2 joining the intercepts made by lines L^{opt} and L^i with the X and Y axis respectively. It can be shown that the slope of line BD (i.e., $-\frac{CD}{BC}$) lies in between the slope of L^i and L^{opt}. If the weights ν_X^i and ν_Y^i are modified such that slope of the line L^{i+1} corresponding to the $(i+1)$th iteration is the same as that of line BD, the weights will converge to the optimal weights ν_X^{opt} and ν_Y^{opt}. This, however, requires the system to know the slope $-\frac{CD}{BC}$. Since the system does not know L^{opt}, it attempts to estimate the lengths of BC and CD based on the feedback provided by the user. Note that the system retrieves objects below L^i during the ith iteration (i.e., those in triangle ACD). Only those items below L^{opt} (i.e., those in triangle BCE) are considered relevant by the user. So, only those objects in the region BCDF (the shaded region) will be marked relevant by the user. Based on this observation, [17] proposes different strategies to estimate the slope $-\frac{CD}{BC}$. One strategy is to estimate BC by $\Delta X = max_{O \in \mathcal{S}} D_X(Q_X, O_X)$ and CD by $\Delta Y = max_{O \in \mathcal{S}} D_Y(Q_Y, O_Y)$ where $\mathcal{S}$ denotes the set of relevant examples submitted by the user. The slope of line BD is therefore $-\frac{\Delta Y}{\Delta X}$. The weights ν_X^{i+1} and ν_Y^{i+1} are chosen to be

$$\nu_X^{i+1} = \frac{\Delta Y}{\Delta X + \Delta Y} \tag{7}$$

$$\nu_Y^{i+1} = \frac{\Delta X}{\Delta X + \Delta Y} \tag{8}$$

Details of other strategies of selecting intra-feature weight can be found in [17].

3 Evaluation of Refined Queries

Once the query points and the weights are determined (i.e., the construction of the refined query is complete), the refined query is evaluated and the answers are returned to the user. In this section, we discuss techniques to evaluate refined queries efficiently. A naive way to evaluate a refined query is to treat it just like a starting query and execute it from scratch. We observe that refined queries are not modified drastically from one iteration to another; hence executing them from scratch every time is wasteful. Most of the execution cost of a refined query can be saved by appropriately exploiting the information generated during the previous iterations of the query. We start with a discussion on techniques to evaluate starting queries and then optimize them for refined queries.

3.1 Evaluation of Starting Queries

One option to evaluate a starting query is to use a sequential scan. This technique is prohibitively expensive when the database is large. A commonly used approach is to maintain an index Idx_F (called the F-index) for each feature. For example, in Example 1, we can maintain a 2-d index on color and a 2-d index on texture. The dimensionality of Idx_F for feature F is d_F, i.e., same as the dimensionality of the feature space. Subsequently, a query can be evaluated using the algorithm (called the merge algorithm) shown in Table 1 [14, 15, 6]. To compute the next best match to query Q, the merge algorithm incrementally retrieves the next best match O_F to

```
variable temp : List(sorted);
GetNext(Q)
while (TRUE) do
    for each F ∈ 𝓕
        O_F = SingleFeatureGetNext(Q_F);
        if O ∉ temp
            Access full object to obtain O_F, ∀F ∈ 𝓕
            Insert O, D(Q, O) into temp;
        endif
    for each F ∈ 𝓕 topdist_F=Topdist(Q_F);
    threshold = Σ_{F∈𝓕}(ν_F * topdist_F);
    first = the first object in temp;
    if first.dist ≤ threshold
        Remove first from temp;
        return first;
    endif
enddo
```

Table 1: Multifeature query evaluation (The merge algorithm).

```
variable queue : MinPriorityQueue;
SingleFeatureGetNext(Q_F)
while not queue.IsEmpty() do
    top=queue.Pop();
    if top is an object
        return top;
    else if top is a leaf node
        for each object O_F in top
            queue.push(O_F, D_F(Q_F, O_F));
    else /* top is an index node */
        for each child node N in top
            queue.push(N, MINDIST(Q_F, N));
    endif
enddo
```

Table 2: Single feature query evaluation (The kNN algorithm).

Q with respect to each feature $F \in \mathcal{F}$ (by calling $SingleFeatureGetNext(Q_F)$ shown in Table 2), accesses the full object O from the database, computes its overall similarity $D(Q, O)$ to Q and inserts it into a sorted list $temp$ until it is safe to return the first object $first$ in $temp$. It is safe to return $first$ when we are sure that there exists no unexplored object O' with $D(Q, O') \leq D(Q, first)$, i.e., $D(Q, first) \leq threshold$ (see [6, 12] for proof).

The $SingleFeatureGetNext(Q_F)$ operation is implemented using the k-nearest neighbor search (k-NN) algorithm executing on the underlying index Idx_F (see Table 2). It maintains a priority queue that contains index nodes as well as data objects prioritized based on their distance from the SF query Q_F. Initially, the queue contains only the root node. At each step, the algorithm pops the item from the top of the queue: if it is an object, it is returned to the caller; if it is a node, the algorithm computes the distance of each of its children from the query and pushes it into the queue. The distances are computed as follows: if the child is a data object, the distance is that between the query point and the object point; if the child is a node, the distance is the minimum distance (referred to as MINDIST [8]) from the query point to the nearest (according to the distance function) boundary of the node. $Topdist(Q_F)$ returns the distance of the top item of the queue from Q_F.

3.2 Optimization for Refined Queries

In this section, we present a simple technique to optimize the $SingleFeatureGetNext(Q_F)$ algorithm. The proposed technique can be extended for optimizing the merge algorithm as described in [4].

In the naive approach where a refined query is executed like a starting query, the k-NN algorithm may access the same nodes of Idx_F from disk multiple times during the execution of a query (during different iterations) causing unnecessary disk I/O. This is true even when a buffer is used to keep the most recently used disk pages in memory. We present a simple algorithm (called *full reconstruction (FR)*) that guarantees that a node of the F-index is accessed from the disk at most once throughout the execution of the query across all iterations of refinement. We achieve this by caching the contents of each node accessed by the query in any iteration and retain them in memory till the query ends (i.e., till the last iteration) at which time the cache can be freed and the memory can be returned to the system. Since the priority queue generated during the execution of the starting query contains the information about the nodes accessed, we can achieve the above goal by caching the priority

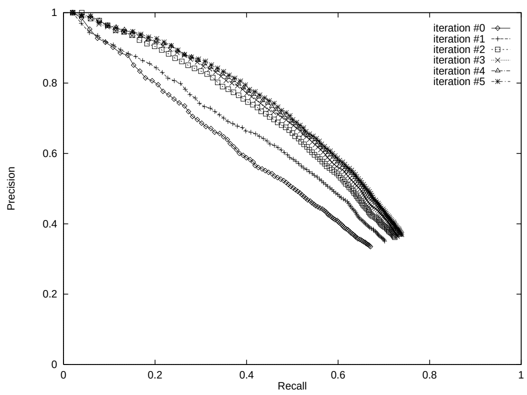


Figure 3: Precision Recall Graph for Query Expansion

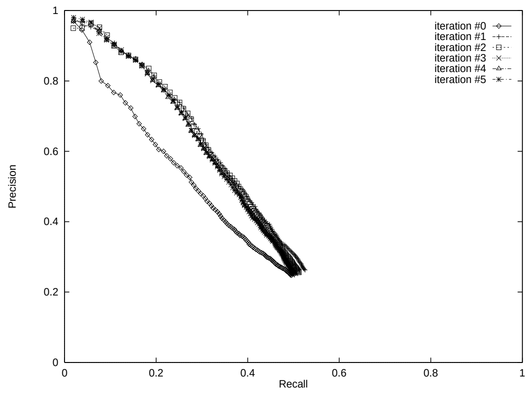


Figure 4: Precision Recall Graph for Query Point Movement

queue. We assume that, for each item (node or object), the priority queue stores the feature values of the item (i.e., the bounding rectangle if the item is a node, the feature vector if it is an object) in addition to its id and its distance from the query. We also assume that the entire priority queue fits in memory as is commonly assumed in other works on nearest neighbor retrieval [10, 23].

The FR algorithm works as follows. To evaluate the refined query, it 'reconstructs' a new priority queue $queue_{new}$ from the cached priority queue $queue_{old}$ by popping each item from $queue_{old}$, recomputing the distance of the item from the refined query Q_F^{new} using the feature values stored, and then pushing it into $queue_{new}$. When all the items have been transferred, the old queue is discarded. We refer to this phase as the *transfer phase*. Subsequently, the k-NN algorithm (discussed in Table 2) is invoked with Q_F^{new} as the query on $queue_{new}$ to retrieve the new answers. We refer to this phase as the *explore phase*. The queue gets passed from iteration to iteration through the reconstruction process, i.e., the $queue_{new}$ of the previous iteration becomes the $queue_{old}$ of the current iteration and is used to construct the $queue_{new}$ of the current iteration. Thus, if a node is accessed once, it remains in the priority queue for the rest of the query and in any subsequent iteration, is accessed directly from the queue (which is assumed to be entirely in memory) instead of reloading from the disk. The entire sequence of iterations is managed using two queues and their roles (old and new) get swapped from iteration to iteration. Further optimizations of the above algorithm (viz., Selective Reconstruction) can be found in [4].

4 Experimental Evaluation

We now present experimental results demonstrating the effectiveness of the query refinement techniques discussed in Section 2.4, i.e., how significantly do they improve the quality of answers returned by the system. Due to space limitations, we restrict the discussion to intra-feature refinement, experiments on the effectiveness of inter-feature techniques can be found in [17, 21]. [7] We conducted our experiments on the **COLHIST** dataset. It comprises of 4x4 color histograms extracted from 70,000 color images obtained from the Corel Database (available online at `http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html`) [2, 16]. All experiments reported in this section were conducted on a Sun Ultra Enterprise 450 with 1GB of physical memory and several GB of secondary storage, running Solaris 2.7.

The effectiveness of multimedia retrieval depends on the subjective judgement of the user. In order to make our measurements of retrieval effectiveness independent of the user subjectivity, we use system generated ground

[7] We have omitted the experiments showing the speedup achieved for refined queries by the evaluation techniques discussed in Section 3.2, they can be found in [4].

truth for our measurements. We choose a point Q_C randomly from the dataset as the query point and construct a graded set of its top 50 neighbors (based on L_1 distance)[8] We mark the top 10 answers as highly relevant, the next 15 as very relevant and the last 25 answers as relevant. The above set of answers constitutes the ground truth or the *relevant set* $Rel(Q_C)$ for Q_C. We construct the starting query by slightly disturbing Q_C (i.e., by choosing a point close to Q_C) and request for the top 100 answers. We refer to the set of answers returned as the *retrieved set* $Ret(Q_C)$. We simulate user feedback by marking those items in $Ret(Q_C)$ that are also in the top 10 items of $Rel(Q_C)$ as highly relevant, those that are also in the next 15 items of $Rel(Q_C)$ as very relevant and those that are also in the last 25 as relevant. Based on the simulated feedback, we construct the refined query using both intra-feature refinement techniques discussed in 2.4 (QPM and QEX). The query is then evaluated and the top 100 answers are retrieved which forms the new retrieved set. We repeat the above process for 5 feedback iterations (the starting query is counted as iteration 0).

To measure the effectiveness of the two approaches, we compute *precision* and *recall* based on the relevant and retrieved sets:

$$precision = \frac{|relevant \cap retrieved|}{|retrieved|} \tag{9}$$

$$recall = \frac{|relevant \cap retrieved|}{|relevant|} \tag{10}$$

Note that for any query, the relevant set is the same for both approaches but the retrieved sets differ. To construct the precision-recall graph for a query, we need precision values for various values of recall. We achieve this by retrieving one point at a time and calculating the precision and recall for $|retrieved| = 1$, $|retrieved| = 2$, ..., $|retrieved| = 100$. Figures 3 and 4 show the precision-recall graphs for query expansion and query point movement respectively. The measurements are averaged over 100 queries. Figures 3 and 4 show that for both approaches, the retrieval performance improves from one iteration to the next. In general, QEX performs better than QPM as QEX can better adjust the distance function to what the user has in her mind. QPM modifies only the intra-feature weights (μ_F's) but not the function $\mathcal{D}_{\mathcal{F}}$ itself. QEX, on the other hand, changes the distance function, albeit implicitly, by adding and removing points (as shown by the shape of iso-distance contours in Figure 1) and hence has better adjustment capabilities. Furthermore, QEX can capture local clusters, if present, in the ground truth while QPM, having the single point representation, ignores the cluster information.

5 Conclusion

Similarity queries are becoming common in many modern-day database applications. Due to the user subjectivity involved in such queries, the answers returned by the system often do not satisfy the user's need right away. The most common technique to overcome this problem is query refinement. In this technique, the user provides to the system some feedback on the relevance of the answers returned. The system then analyzes the feedback, refines the query (i.e., modifies the weights, distance functions and query values that capture the user's information need), evaluates it and returns the new results. In this paper, we discuss techniques to effectively refine the query based on user feedback. We also describe algorithms to evaluate refined queries efficiently. Our experiments show that the proposed techniques significantly improve the quality of the answers returned by the system.

[8]We use the L_1 metric (i.e., Manhattan distance) as the distance function $\mathcal{D}_{\mathcal{F}}$ for the color feature since it corresponds to the histogram intersection similarity measure, the most commonly used similarity measure for color histograms [14, 15].

References

[1] I. Bartolini, P. Ciaccia, and F. Waas. Feedbackbypass: A new approach to interactive similarity query processing. *Proceedings of VLDB Conference*, 2001.

[2] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. *Proceedings of the IEEE International Conference on Data Engineering*, March 1999.

[3] K. Chakrabarti, K. Porkaew, and S. Mehrotra. Efficient query refinement in multimedia databases. *Proceedings of International Conference in Data Engineering(ICDE)*, 2000.

[4] K. Chakrabarti, K. Porkaew, M. Ortega, and S. Mehrotra. Evaluating refined queries in top-k retrieval systems. *Technical Report, MARS-TR-00-05*, 2000.

[5] S. Chaudhari and L. Gravano. Evaluating top-k selection queries. *Proceedings of VLDB Conference*, 1999.

[6] R. Fagin. Combining fuzzy information from multiple systems. *Proc. of the 15th ACM Symp. on PODS*, 1996.

[7] R. Fagin. Fuzzy queries in multimedia database systems. *Proceedings of PODS*, 1998.

[8] G. R. Hjaltason and H. Samet. Ranking in spatial databases. *Proceedings of SSD*, 1995.

[9] Y. Ishikawa, R. Subramanya, and C. Faloutsos. Mindreader: Querying databases through multiple examples. *Proc. of VLDB*, 1998.

[10] F. Korn, N. Sidiropoulos, and C. Faloutsos. Fast nearest neighbor search in medical image databases. *Proc. of VLDB*, 1996.

[11] A. Motro. Vague: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems, Vol.6, No. 3*, July 1988.

[12] S. Nepal and M. V. Ramakrishna. Query processing issues in image databases. *Proc. of ICDE*, 1999.

[13] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture and shape. In *Proc. of SPIE Conference on Storage and Retrieval for Image and Video Databases*, February 1993.

[14] M. Ortega, Y. Rui, K. Chakrabarti, S. Mehrotra, and T. Huang. Supporting similarity queries in mars. *Proc. of ACM Multimedia 1997*, 1997.

[15] M. Ortega-Binderberger, Y. Rui, K.Chakrabarti, S. Mehrotra, and T. Huang. Supporting ranked boolean similarity queries in mars. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, November 1998.

[16] K. Porkaew, K. Chakrabarti, and S. Mehrotra. Query refinement for content-based multimedia retrieval in MARS. *Proceedings of ACM Multimedia Conference*, 1999.

[17] K. Porkaew, S. Mehrotra, and M. Ortega. Query reformulation for content based multimedia retrieval in MARS. *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, 1999.

[18] K. Porkaew, S. Mehrotra, M. Ortega, and K. Chakrabarti. Similarity search using multiple examples in MARS. *Proceedings of International Conference on Visual Information Systems*, 1999.

[19] Y. Rui, T. Huang, and S. Mehrotra. Content-based image retrieval with relevance feedback in mars. *Proc. of IEEE Int. Conf. on Image Processing*, 1997.

[20] Y. Rui, T. Huang, and S. Mehrotra. Relevance feedback techniques in interactive content-based image retrieval. *Proc. of IS&T and SPIE Storage and Retrieval of Image and Video Databases*, 1998.

[21] Y. Rui, T. Huang, M. Ortega, and S. Mehrotra. Relevance feedback: A power tool in interactive content-based image retrieval. *IEEE Tran on Circuits and Systems for Video Technology*, September, 1998.

[22] T. Seidl and H. Kriegel. Efficient user-adaptable similarity search in large multimedia databases. *Proc. of VLDB*, 1997.

[23] T. Seidl and H. Kriegel. Optimal multistep k-nearest neighbor search. *Proc. of ACM SIGMOD*, 1998.

[24] D. White and R. Jain. Algorithms and strategies for similarity retrieval. 1996.

Multimedia Queries by Example and Relevance Feedback

Leejay Wu Christos Faloutsos Katia Sycara
Terry Payne
Carnegie Mellon University
{lw2j,christos,katia,terryp}@cs.cmu.edu

Abstract

We describe the FALCON system for handling multimedia queries with relevance feedback. FALCON distinguishes itself in its ability to handle even disjunctive queries on metric spaces. Our experiments show that it performs well on both real and synthetic data in terms of precision/recall, speed of convergence, individual query speed, and scalability. Moreover, it can easily take advantage of off-the-shelf spatial- and metric- access methods.

1 Introduction

Interactive multimedia databases such as Informedia [2, 15] present an intriguing challenge. Museum kiosks, public media archives, and similar systems require simple interfaces; museum patrons should not need to learn a query language simply to navigate a catalogue of major exhibits. The obvious user-friendly solution is "query by example", augmented by relevance-feedback with which users iteratively improve the specification of their queries.

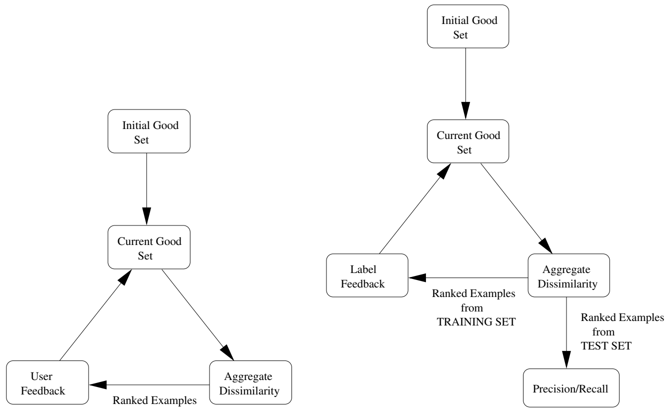
One well-known relevance feedback mechanism is that of Rocchio [10, 1]. It is a basic single-query-point method, in that it produces contiguous hyperspherical isosurfaces in feature spaces centered around *one* query point. To derive its query point, it linearly combines:

- The query point derived from the last iteration.
- A weighted centroid of the documents already judged relevant.
- A weighted centroid of the documents already judged irrelevant. This term receives a negative weight.

Thus, it takes into account relevance feedback, and therefore should gravitate towards positive examples, away from negative ones. Setting the weights for the three components may require some experimentation.

Two more single-query-point systems include MindReader [7], and MARS [11, 9, 12, 13]. MindReader computes the Mahalanobis distance function using the matrix of known positive instances. Since this involves matrix inversion, it can require a significant number of instances when the dimensionality is high. The Mahalanobis space allows MindReader to derive arbitrary hyperelliptical isosurfaces around multiple query points; these isosurfaces can then be refined when the user adds or removes examples from the desired set. MARS takes a more traditional approach in which similarities are computed between known positive instances and candidates, and linearly then combined using user-specified weights.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering



(a) Normal operation (b) Evaluation

Figure 1: The FALCON relevance-feedback loop, (a) designed for interactive use, and (b) modified for automated performance testing.

PicHunter [4, 5] provides a Bayesian approach to finding a particular image. At each iteration, a user selects zero or more of the presented images. This binary feedback is the only input it uses to develop a probabilistic model as to which image is the target. PicHunter distinguishes itself in another aspect; it explicitly tries to minimize the number of subsequent iterations by taking into account expected information gain when choosing which examples to present to the user, rather than simply always presenting those thought to be closest to the target.

2 Proposed System: FALCON

Herein we describe the FALCON system. Like the aforementioned systems, it incorporates user feedback in order to iteratively improve its internal representation of the query concept. The feedback loop is illustrated in Figure 1(a); Figure 1(b) shows a variation used for non-interactive use during testing, where feedback comes from pregenerated labels rather than a user. This latter variation will be discussed further with the rest of the evaluation methodology.

FALCON maintains a current "good set" $\mathcal{G}$ consisting of objects that have been labeled by the user as good examples. Where it differs is in how it uses $\mathcal{G}$ to rank candidate objects.

For this, FALCON uses a distance-combining function we call "aggregate dissimilarity". Given d, a pairwise distance function the internals of which do not concern us; m distinct good objects g_i; α a tuning parameter we discuss below; and x, any given candidate, the aggregate dissimilarity of x with respect to $\mathcal{G}$ is defined as follows.

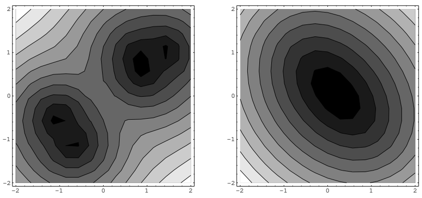
$$(D_{\mathcal{G}}(x))^{\alpha} = \begin{cases} 0 & \text{if } (\alpha < 0) \bigwedge \exists i \; d(x, g_i) = 0 \\ \frac{1}{m} \times \sum_{i=1}^{m} d(x, g_i)^{\alpha} & \text{otherwise} \end{cases} \tag{11}$$

Weights can be included as follows, with weight w_i corresponding to good object g_i:

$$(D_{\mathcal{G}}(x))^{\alpha} = \frac{1}{\sum_{i=1}^{m} w_i} \cdot \sum_{i=1}^{m} w_i (d(x, g_i))^{\alpha} \tag{12}$$

2.1 Properties of $D_{\mathcal{G}}$

Since $D_{\mathcal{G}}$ is used to measure the probable relevance of objects to return to the user, the exact behavior of $D_{\mathcal{G}}$ is clearly important. The tunable parameter α has significant effects on the isosurfaces generated by $D_{\mathcal{G}}$, as listed



(a) $\alpha = -5$ (b) $\alpha = 2$

Figure 2: Sample contours with two values of α. Both are based on the same $\mathcal{G}$ – three points chosen from one circular region, two from another.

below.

- $D_{\mathcal{G}}$ handles disjunctive queries well when $\alpha < 0$. If $\alpha = -\infty$, then $D_{\mathcal{G}}$ is analogous to a fuzzy OR, in that it results in the minimum distance. With $\alpha \in (-\infty, 0)$, all distances are taken into account but the bias is towards the lesser ones. Thus, it can generate contours such as shown in Figure 2(a), derived from a disjunctive $\mathcal{G}$ consisting of five points from two disjoint circular regions.
  Disjunctive queries are important, because many queries that may appear simple to people are disjunctive. One such simple query would be for stocks whose trends resemble V's or inverted V's, which might be useful for a short-term market-timer. Any system that represented queries via a single convex region will fail to capture the nature of the query, no matter how many positive or negative examples were specified.
- Positive values of α generate fuzzy ANDs. The extreme value of $\alpha = \infty$ results in $D_{\mathcal{G}}$ using only the maximum distance, while values of $\alpha \in (0, \infty)$ account for all distances but bias towards the higher ones. As a consequence, a negative α is a more natural choice for the greater flexibility via accepting disjunctive queries.
- $D_{\mathcal{G}}$ is undefined at $\alpha = 0$.
- For $\alpha = 2$, the isosurfaces of $D_{\mathcal{G}}$ become hyperspheres centered on the centroid of $\mathcal{G}$ [16]. This is reflected in Figure 2(b), which is identical to (a) except for α. Despite $\mathcal{G}$ corresponding to a disjunctive query, $\alpha = 2$ yields concave isosurfaces.

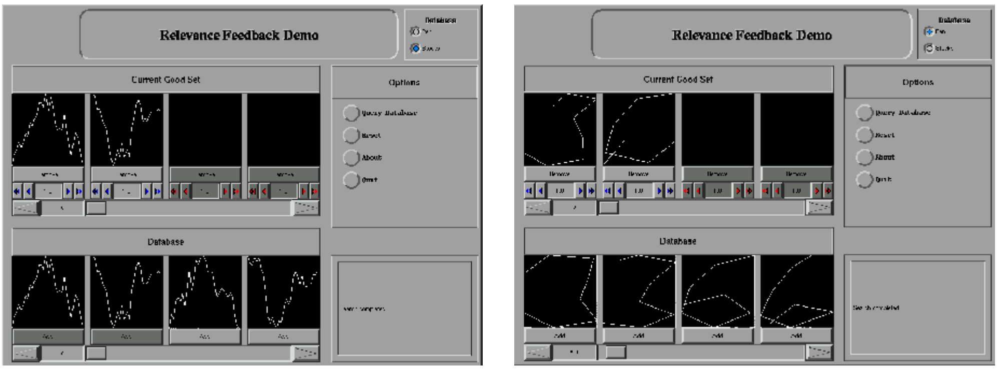
In addition, since $D_{\mathcal{G}}$ never directly uses features, FALCON is not intrinsically restricted to vector spaces; theoretically, it could be implemented using completely unrestricted spaces and sequential scanning. A more practical approach would be to use fast indexing structures; later in this paper we present algorithms for range and k-nearest-neighbor queries in $D_{\mathcal{G}}$ space, and these can take advantage of indices that support single-query-point versions of the same. Metric spaces – where we lack features, but instead have a symmetric distance function that obeys the triangle inequality – then become feasible with such indices as the M-tree [3] and Slim-tree [14].

2.2 User Feedback

Now that we have defined one major component $D_{\mathcal{G}}$, and therefore know how to rank examples, an obvious question is how FALCON incorporates user feedback.

Consider a given $\mathcal{G}$ and a pre-determined α. These define $D_{\mathcal{G}}$, and therefore provide a hybrid distance function for evaluating examples. For any arbitrary $k \in \mathcal{Z}^+$, we can retrieve and present the best k candidates to the user based upon $D_{\mathcal{G}}$.

The user may then change $\mathcal{G}$ by selecting one or more candidates to add to $\mathcal{G}$. Likewise, current members of $\mathcal{G}$ can be removed if the user changes his mind. If the weighted $D_{\mathcal{G}}$ variation is used, then the user should be able to alter weights for individual members of $\mathcal{G}$.



(a) on the STOCK dataset (b) on the PEN dataset

Figure 3: GUI of FALCON. The top row contains the “good” objects; the bottom row gives the top matches. The implicit query in (a) is for stock vectors that climbed sharply, then fell, or vice versa; in (b), we seek both 3’s and 6’s. Notice that FALCON handles these disjunctive queries quite well, returning examples of the desired sets instead of objects that each have slight resemblances to all good objects.

2.3 Algorithms and Scalability

Finding candidates for feedback in an interactive system using a non-trivial database clearly requires fast, scalable algorithms. Here, we describe algorithms for performing both range and k-nearest-neighbor queries in $D_{\mathcal{G}}$ space. The latter fits in naturally with the feedback model, and utilizes the former.

Both require range and k-nearest-neighbor queries in the original space, as defined by the pairwise distance function d. If d is a distance metric, metric structures such as M-trees [3] and Slim-trees [14] apply; for vector sets, there are legions of spatial indices with the required properties [6].

Given support for the single-point versions, we can derive versions that operate in $D_{\mathcal{G}}$ space using multiple query points as found in $\mathcal{G}$. These algorithms scale well and are provably correct. For reference, we label these two algorithms as the “MRQ” (Multiple Range Query) and “k-NN+MRQ” (k-Nearest Neighbor + Multiple Range Query) algorithms; details and proofs of correctness are in [17] and [16], respectively.

In a nutshell, the MRQ algorithm computes range-queries in $D_{\mathcal{G}}$ space by finding the union of standard range queries with the same desired radius for each of the “good” objects g_i, followed by filtering via $D_{\mathcal{G}}$. Similarly, the k-NN+MRQ algorithm uses multiple k-NN queries centered on each g_i. The results are then used to overestimate the $D_{\mathcal{G}}$ distance to the k^{th} neighbor; given this range, it applies the MRQ algorithm and selects the top k results as ranked by $D_{\mathcal{G}}$.

2.4 Prototype implementation

Figures 3(a) and (b) shows what a user might see. These screenshots show a prototype operating on two data sets, STOCK and PEN, both described later. The top row of panels shows $\mathcal{G}$, while the bottom row shows an excerpt from the database in whatever order is current – sorted according to the previous query; or the original order if no queries have been executed or the system has been reset.

In both cases, the queries are disjunctive; the STOCK example shows a request for stocks with curves resembling V’s or inverted V’s, while the PEN example requests both 3’s and 6’s.

3 Experiments

In this section, we present empirical support for the FALCON system. There are three obvious questions that pertain to relevance-feedback systems in general.

1. In terms of quality, how good are the candidates returned by FALCON?
2. In terms of iterations, how quick does the feedback loop converge?
3. In terms of speed, how scalable is FALCON?

In addition, since FALCON has one free parameter α, we ask whether or not there is an optimal value of α; and what values of α seem to perform acceptably well.

3.1 Evaluation Methodology

The testing cycle differs from the original flow in two areas, as shown in Figure 1(b).

The first is that feedback is that all instances have already been labeled by a script, rather than a user. Two major limitations are imposed. One is that only a subset of the data is available for feedback, which should accelerate convergence — preferably without severely limiting performance. To understand why, consider that a k-nearest-neighbor query may return a cluster of points close to each other, if such exist. A tight cluster is less likely to be useful for determining the underlying query concept than an equal number of well-separated points; and a sample should, in theory, increase the typical separation between candidates.

A second limitation imposed upon automated feedback is that no more than twenty previously unseen instances are selected per iteration. If any of these are positive instances, they get added to $\mathcal{G}$. Otherwise, the feedback loop terminates.

The second way in which the feedback loop differs is that the method computes precision/recall in order to quantitatively evaluate performance. For any level of recall $r \in [0, 1]$, a database of size n, and a total of n_p positive examples, there is a minimum number n_e such that the top-ranked t objects among the n total include at least $p = \lceil rn_p \rceil$ positive examples. Then, we can say that precision/recall is $\frac{p}{t}$ at r recall. Tracking precision/recall at multiple levels of recall over a series of iterations measures speed of convergence.

We also measured wall-clock time required for multi-query-point range and k-nearest-neighbor queries, which are critical to FALCON's performance. For reference, these tests were performed on a 400 MHz Intel Pentium II computer with 128 MB of RAM running Linux.

3.2 Data and Parameters

Five data sets were used, each paired with a different query. Four – two low-dimensional synthetic, two high-dimensional real – were used to measure precision/recall, convergence speed, and sensitivity to α. Real data received realistic queries; the two synthetic sets tested non-convex and disjunctive queries, respectively. Regarding α sensitivity, we tested with $\alpha \in \{-\infty, -100, -10, -5, -2, 2, 5\}$.

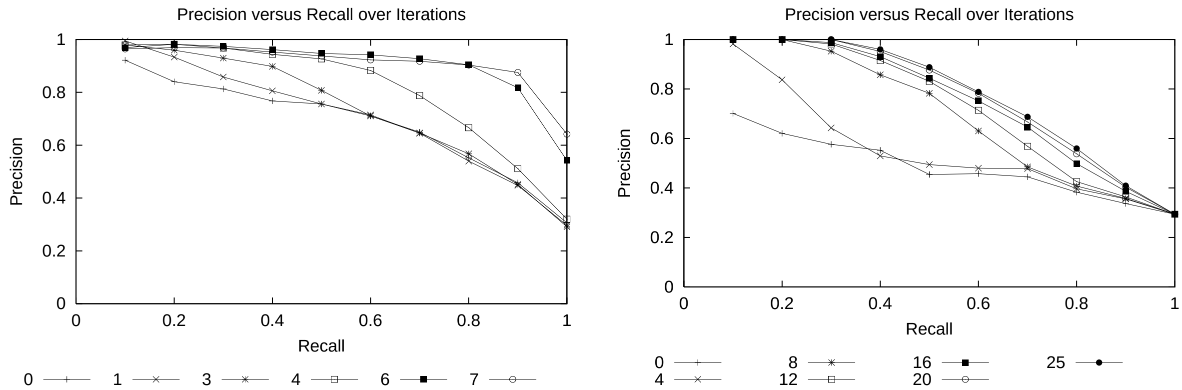
The fifth and largest data set was used for scalability testing. For this purpose, most tests used $\alpha = -5$; a few range-query tests were run with $\alpha = -\infty$, and are labeled accordingly.

2D_50K/RING: The 2D_50K data set has 50,000 points in 2-D Cartesian space, uniformly distributed within the orthogonally-aligned square (-2,-2)-(2,2). The RING query selects points that are between 0.5 and 1.5 units from the origin as measured by Euclidean distance; 19,734 qualify.

2D_20K/TWO_CIRCLES: The 2D_20K data set was generated identically to 2D_50K except that it has only 20,000 points. The TWO_CIRCLES query accepted points at most 0.5 units away in Euclidean distance from either (-1,-1) or (1,1); 1,899 qualified.

PEN/PEN_4: The Handwriting Recognition data set was found at the UCI repository [8]. Each of the 10,992 objects is a 16-dimensional vector corresponding to the scaled Cartesian coordinates of eight pen positions sampled in time as a test subject wrote a digit. The PEN_4 query selects vectors corresponding to handwritten 4's; 1,144 qualified.

STOCKS/FLAT_STOCKS: Fifty-one stocks were arbitrarily chosen and their closing prices obtained for up to the previous five years. These sequences were split into 1,856 non-overlapping vectors of length 32 and then subjected to a discrete wavelet transformation. The FLAT_STOCKS query accepted stocks with the slope within [-0.02, 0.02]; 540 qualified. The DWT results from five artificial, perfectly flat price vectors were used for the first $\mathcal{G}$.



TWO_CIRCLES FLAT_STOCKS

Figure 4: Precision/recall over iterations for two of the queries at $\alpha = -5$.

COVER: The forest cover database came from the UCI repository [8], and includes 581,012 54-dimensional vectors, with both discrete and continuous attributes. No split was used since the purpose was not to assess query accuracy, but instead to measure performance of a single multi-point query. Uniform random sampling without replacement was used to find $\mathcal{G}$'s of varying size.

3.3 Experimental Results

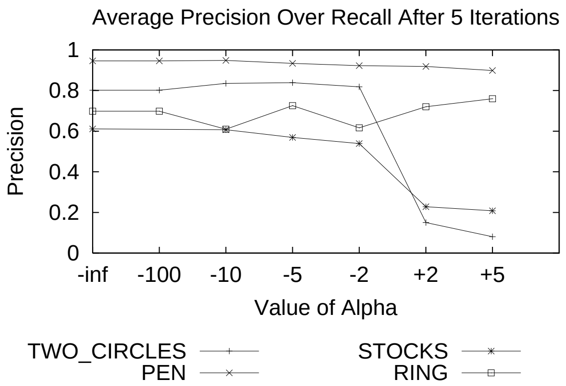
A summary of the results follows; for a more in-depth treatment, see [17, 16].

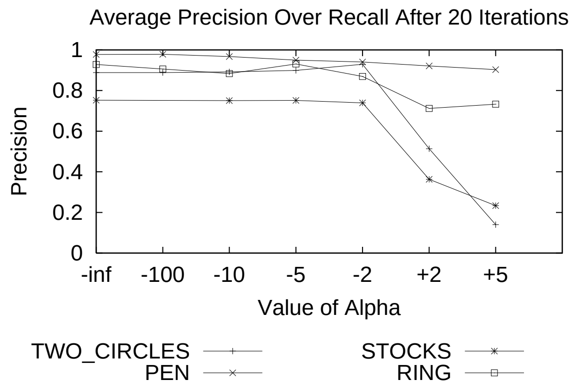
Values of α Accuracy testing showed that $\alpha = -2$ and $\alpha = -5$ were both reasonable values with $\alpha = -10$ not substantially inferior. With $-\infty$, the strict MIN function is too strict; and with positive values, disjunctive queries such as TWO_CIRCLES become impossible. No value was optimal for all sets; intuitively, this makes sense as there is little reason to expect a global optimum over the space of different queries.

Quality and Convergence Figure 4 shows precision/recall for TWO_CIRCLES and FLAT_STOCKS with $\alpha = -5$. The former completes in seven iterations, with excellent precision-recall; the latter nears its final values after 8-12 iterations, but marginal gains continue for a total of 25. These results are typical; a fairly low number of iterations generally resulted in good precision at low levels of recall early, with subsequent iterations converging for the higher levels of recall.

Figure 5 shows average precision/recall values for at multiple values of α with the four data sets used for accuracy testing. Averages were computed as the mean of precision at each level of recall from 10% to 100% at 10% intervals; in cases where 20 iterations were not required due to faster convergence, final values are presented. Notably, negative values of α yield better results, but otherwise the performance is rather insensitive to the exact value of α. In addition, average precision on a full database can be respectably high even after only 5 iterations of feedback on a randomly chosen subset.

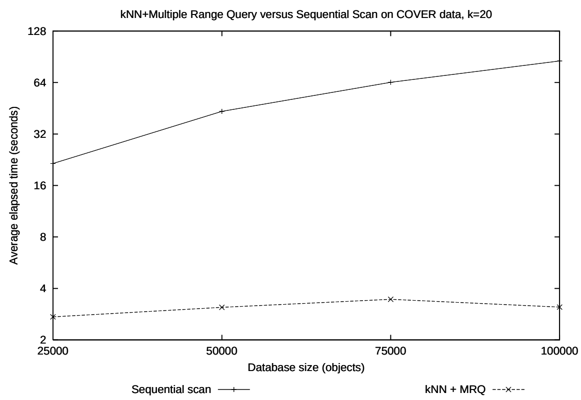
Scalability Figure 6 shows that the k-NN+MRQ algorithm, when backed by an M-tree, performs very well compared to sequential scanning as the sample size chosen from COVER is varied from 25,000 to 100,000 and k goes from 5 to 50. In particular, k-nearest-neighbor performance is not nearly as affected by database cardinality as is sequential scanning. Clearly, the MRQ algorithm, as a significant part of the k-NN+MRQ algorithm, must itself scale accordingly.

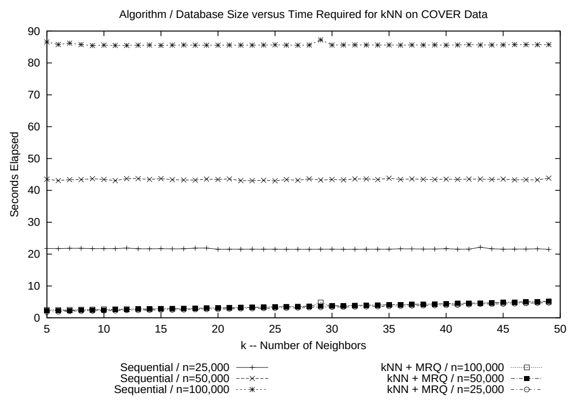




After 5 iterations | After 20 iterations

Figure 5: Average precision/recall after 5 and 20 iterations.





Cardinality scaling | Dimensionality scaling

Figure 6: k-NN+MRQ performance testing with variable database size and variable k.

4 Conclusions

We have described FALCON, the first relevance feedback system that can handle disjunctive queries in multimedia databases. In addition, FALCON has the following desirable properties:

- It can be applied to metric datasets, too, in addition to vector ones.
- It reaches good precision and recall, after few iterations. For instance, with all queries, $\alpha = -5$ yielded at least 80% precision at 50% recall with 10 iterations.
- When backed by an indexing data structure, it scales well. For instance, k-NN+MRQ with $k = 50$ on a 100,000-object set was approximately four times as fast as a sequential scan with $k = 5$ on a 25,000-object set.

Acknowledgements

This research was partially funded by the National Science Foundation under Grants No. DMS-9873442, and IIS-9910606; also, by DARPA/ITO through Order F463, issued by ESC/ENS under contract N66001-97-C-851;

and the Office of Naval Research Grant N-00014-96-1-1222. Additional funding was provided by donations from NEC and Intel. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the United States Government. All trademarks are property of their respective owners.

References

[1] Chris Buckley, Gerard Salton, and James Allan. The effect of adding relevance information in a relevance feedback environment. In *Proc. of SIGIR*, pages 292–300, 1994.

[2] M. Christel, T. Kanade, M. Mauldin, R. Reddy, M. Sirbu, S. Stevens, and H. Wactlar. Informedia digital video library. *Communication of the ACM*, 38(4):57–58, April 1995.

[3] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *Proc. of 23rd International Conference on Very Large Data Bases*, pages 426–435. Morgan Kaufmann, sep 1997.

[4] Ingemar J. Cox, Matt L. Miller, Stephen M. Omohundro, and Peter M. Yianilos. PicHunter: Bayesian relevance feedback for image retrieval. In *Proceedings of International Conference on Pattern Recognition*, Vianna, Austria, 1996.

[5] Ingemar J. Cox, Matthew L. Miller, Thomas P. Minka, and Peter N. Yianilos. An optimized interaction strategy for Bayesian relevance feedback. In *Proc. of IEEE Conf. on Comp. Vis. and Pattern Recognition*, pages 553–558, 1998.

[6] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[7] Yoshiharu Ishikawa, Ravishankar Subramanya, and Christos Faloutsos. Mindreader: Querying databases through multiple examples. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *Proc. of 24rd International Conference on Very Large Data Bases*, pages 218–227. Morgan Kaufmann, aug 1998.

[8] P.M. Murphy and D.W. Aha. UCI Repository of Machine Learning Databases. Department of Information and Computer Science, University of California, Irvine, CA.
[http://www.ics.uci.edu/∼mlearn/MLRepository.html], 1994.

[9] Kriengkrai Prokaew, Sharad Mehrotra, Michael Ortega, and Kaushik Chakrabarti. Similarity search using multiple examples in mars. In *1999 International Conference on Visual Information Systems*, June 1999.

[10] Joseph John Rocchio. Relevance feedback in information retrieval. In Gerard Salton, editor, *The SMART Retrieval System – Experiments in Automatic Document Processing*, pages 313–323. Prentice Hall, Englewood Cliffs, N.J., 1971.

[11] Y. Rui, T. Huang, M. Ortega, and S. Mehrotra. Relevance feedback: A power tool in interactive content-based image retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):644–655, sep 1998.

[12] Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Content-based image retrieval with relevance feedback in MARS. In *Proceedings of IEEE International Conference on Image Processing '97*, Santa Barbara, CA, October 1997.

[13] Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Human perception subjectivity and relevance feedback in multimedia information retrieval. In *Proceedings of IS&T and SPIE Storage and Retrieval of Image and Video Databases VI*, San Jose, CA, January 1998.

[14] Caetano Traina Jr., Agma J. M. Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, and Torsten Grust, editors, *Proc. of 7th International Conference on Extending Database Technology*, volume 1777 of *Lecture Notes in Computer Science*, pages 51–65. Springer, mar 2000.

[15] Howard D. Wactlar, Takeo Kanade, and Michael A. Smith. Intelligent access to digital video: Informedia project. *IEEE Computer*, 29(5):45–52, May 1996.

[16] Leejay Wu, Christos Faloutsos, Katia Sycara, and Terry Payne. Falcon: Relevance feedback in metric spaces. Submitted to ACM TODS.

[17] Leejay Wu, Christos Faloutsos, Katia Sycara, and Terry R. Payne. Falcon: Feedback adaptive loop for content-based retrieval. In *Proceedings of the 26th International Conference on Very Large Databases*, pages 297–306, Cairo, Egypt, September 2000. VLDB.

Keyword Search in Databases

Arvind Hulgeri Gaurav Bhalotia Charuta Nakhe* Soumen Chakrabarti
S. Sudarshan
Dept. of Computer Science and Engg., Indian Institute of Technology, Bombay
{aru,bhalotia,soumen,sudarsha}@cse.iitb.ac.in, charuta@pspl.co.in

Abstract

Querying using keywords is easily the most widely used form of querying today. While keyword searching is widely used to search documents on the Web, querying of databases currently relies on complex query languages that are inappropriate for casual end-users, since they are complex and hard to learn. Given the popularity of keyword search, and the increasing use of databases as the back end for data published on the Web, the need for querying databases using keywords is being increasingly felt. One key problem in applying document or web keyword search techniques to databases is that information related to a single answer to a keyword query may be split across multiple tuples in different relations.

In this paper, we first present a survey of work on keyword querying in databases. We then report on the BANKS system which we have developed. BANKS integrates keyword querying and interactive browsing of databases. By their very nature, keyword queries are imprecise, and we need a model for answering keyword queries. BANKS, like an earlier system called DataSpot, models a database as a graph. In the BANKS model, tuples correspond to nodes, and foreign key and other links between tuples correspond to edges. Answers to a query are modeled as rooted trees connecting tuples that match individual keywords in the query. Answers are ranked using a notion of proximity coupled with a notion of prestige of nodes based on inlinks, the latter being inspired by techniques developed for Web search. We illustrate the power of the model and our prototype through examples.

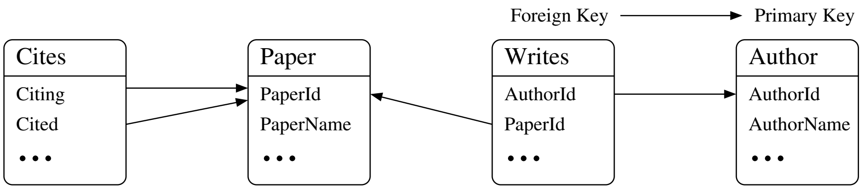
1 Introduction

Querying relational databases using schema-cognizant languages like SQL and querying document collections by typing arbitrary keywords are the extreme ends of the continuum between structured and unstructured data access. SQL queries have precisely defined semantics, but demand that the data is organized along a strict schema which the user understands. Keyword searches do not require the data to follow a schema, except for the notion of a collection of documents, each being a sequence of delimited tokens. On the other hand, responses to keyword searches are often imprecise.

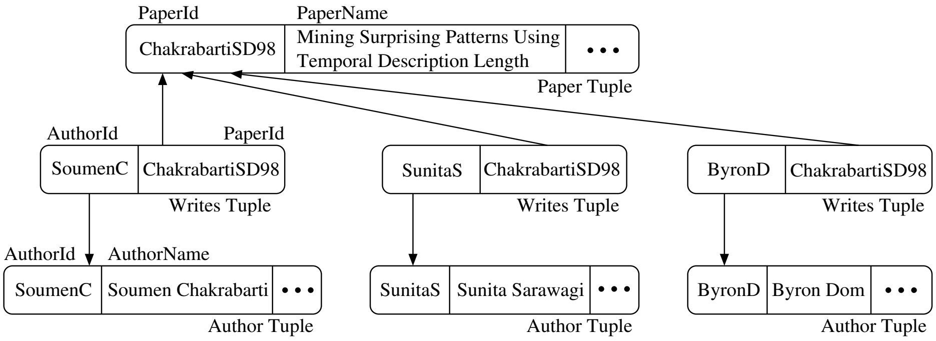
Two forces are bridging the gap between these extremes. First, relational databases are increasingly Web enabled: they need to be accessed and manipulated by non-experts who do not know enough about the schema.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Current affiliation: Persistent Systems Pvt. Ltd., Pune, India



(A) The Schema



(B) A Fragment of the Database

Figure 1: The DBLP Bibliography Databases

Second, Web documents are evolving from flat text files through HTML and SGML to XML, adding markups and embedded schema information (albeit irregular) which users would like to exploit to make responses somewhat more precise than with plain keyword matching. Unfortunately, as query languages for relational data evolve to encompass semi-structured data, they are becoming more complex. On the other hand, many naive users need a simple way of extracting information, such as keyword queries.

In relational databases, information needed to answer a keyword query is often split across the tables/tuples, due to normalization. As an example consider a bibliographic database shown in Figure 1. This database contains paper titles, their authors and citations extracted from the DBLP repository. The schema is shown in Figure 1(A). Figure 1(B) shows a fragment of the DBLP database. It depicts partial information—paper title and authors—regarding a paper as stored in the bibliographic database defined above. As we can see the information is distributed across seven tuples linked through foreign key-primary key links. A user looking for this paper may use queries like ”Sunita Temporal” or ”Soumen Sunita”. In keyword based search, we need to identify tuples containing the keywords and ascertain their proximity through links.

Inverted indexing techniques used for document search would help in finding proximity amongst the words in a tuple or a column thereof but will not help in finding proximity between two tuples. This makes inverted index-based keyword search unsuitable for RDBMS. Similar observations hold for text search in XML documents: the markup induces a graph whose nodes contain blocks of text. Different query words may match different blocks or nodes, and the best ‘answer’ may be none of the directly matched nodes.

Web search engines and topic directories have also popularized the browsing paradigm for accessing information, which is virtually non-existent in the relational and semi-structured data domains. Like HTML pages connected via hyperlinks, semi-structured data comprises data entities which are nodes in a graph with labeled edges. The link-based navigation paradigm should therefore be of great use for exploring such databases.

Responses to relational queries are *sets* of tuples, which may be explicitly ordered by a specified list of attributes. Thus the “information unit” is a tuple. For document search, the information unit is a document and the rank of the document in the response list is based on the number of keywords found in the document and their proximity within the document. A suitable information unit is much harder to define when a general graph model is used to represent data entities and relations, because information matching the query may be split across several tables and tuples due to normalization.

Over the last few years, a uniform model has emerged for representing relational databases as a graph with the tuples in the database mapping to nodes and cross references (such as foreign key and other forms of references) between tuples mapping to edges connecting these nodes. Semistructured data maps even more naturally to a graph model, as do HTML pages connected by hyperlinks. The graph model may be used in keyword search as follows. Keywords in a given query *activate* some nodes. The answer to the query is defined to be a subgraph which connects the activated nodes.

In this paper, we present a survey of earlier work on keyword querying of databases, with emphasis on the development of the above model. We then present the model for keyword search used in the BANKS system (BANKS is an acronym for Browsing ANd Keyword Search). BANKS ranks answer subgraphs using a notion of proximity coupled with a notion of prestige of nodes based on inlinks, similar to techniques developed for Web search engines like Google. BANKS proposes meaningful interpretations for matching query tokens not only to text attributes, but also to relation and attribute names. Here we are chiefly concerned with meaningful definitions of responses and their ranking. Efficient query execution strategies for the BANKS model will be described in a separate paper.

BANKS has been developed in Java using servlets and JDBC. It is a generic system and can be used against any relational database supporting JDBC, without any programming. (An XML source adapter is planned.) A demo of BANKS is accessible at `http://www.cse.iitb.ac.in/banks/`.

Organization: We survey earlier work on keyword querying in Section 2. Section 3 outlines the BANKS graph model for representing connectivity information from the database and the model for answering queries and briefly outlines how the BANKS querying model differs from earlier work. Section 4 briefly describes how the query model is implemented. We present an overview of the browsing features of BANKS in Section 5. Section 6 outlines a preliminary evaluation of our system in terms of reasonableness of its answers and feasibility. Section 7 outlines directions for future work. Section 8 concludes the paper.

2 Previous Work

There are a number of commercial and research prototype systems that support keyword search and browsing in relational and semi-structured databases. In this section, we survey several of these systems.

Traditional database query languages and tools are not suitable for applications that require keyword searching. Even languages, such as QBE, that have been targeted at relatively inexperienced users require the user to be aware of the database schema, which is not appropriate for casual users of an information system.

2.1 DataSpot and Mercado Intuifind

The DataSpot system [3] was developed to support database querying using free-form (keyword) queries and navigation for non-technical users seeking information in complex databases such as electronic catalogs. The basis for the DataSpot system is a schema-less representation of data which they call a "Hyperbase". Nodes in the hyperbase view represent data objects, while edges represent associations. There are two types of edges, *simple* and *identification* edges. Simple edges represent inclusion, such as attributes values' inclusion in tuples. Identification edges correspond to references between objects.

The semantics of keyword queries is described in [12], and in more detail in [13]. Given a keyword query, an answer is a connected sub-hyperbase that contains the keywords in the query. Each answer has an associated "location", which intuitively represents the main object of the answer. Answers have a score, which can be computed in one of several ways, but intuitively measures some distance metric on the sub-hyperbase. Several alternatives are suggested, one of which is an edge count on the sub-hyperbase, possibly with weighted edges, with weights being determined by node and edge types. Another alternative is to use a node count. The distance metric used in their preferred approach is based on adding up edge weights, with weights being determined by the types of the edge, the types of the nodes it connects, and the direction of traversal (from parent to child or vice versa; see below).

The system administrator can define a set of nodes as “fact nodes”, in which case answers (location nodes) can only come from the set of fact nodes. Refinement queries, which refine answers from earlier queries, can also start with an initial set of locations, in addition to keywords.

To find answers, the system performs weighted best-first search from all source nodes (i.e., keyword nodes, and for refinement queries, location nodes) and each time a fully connected fact node (i.e., a fact node connected to all source nodes) is found, it is output as an answer, along with the minimal-distance paths from that node to all the source nodes. Traversal goes away from source nodes, but edges can be followed regardless of their direction (that is, edges can be traversed forwards or backwards). Edge weights are determined by direction of traversal, as mentioned earlier. When adding edge weights to paths, the edge weight is divided by the number of sources to which the edge is connected, so that when adding up path distances, edge weights of edges leading to multiple sources are not over-counted. Multiple answers may be output, ranked by their score.

Mercado Software, Inc. (`www.mercado.com`) markets an e-catalog search technology called Intuifind which (as far as we can understand) uses the DataSpot technique for keyword search, but also allows “parametric search” on the search results. Intuifind parametric search offers OLAP drill-down type operations based on parameters such as price, manufacturer and category, to allow the user a more structured way of browsing search results.

2.2 EasyAsk

EasyAsk is commercial system that provides natural language search (including keyword search) on data stored in relational databases as well as in text repositories. Our description of EasyAsk is based on white papers on the EasyAsk system, and on its features, available at the EasyAsk web site `www.easyask.com`.

Consider catalog searching, which is a motivating application for EasyAsk (the system can be used with other application domains also). Information about items in catalogs can be split across multiple locations, such as a text description, and at different levels of a product hierarchy in which the item is classified, such as “men’s” or “formal”. A keyword search on the catalog may contain keywords present in the product description, as well as keywords present in the catalog hierarchy.

To answer queries, the system crawls the data store ahead of time and constructs a contextual dictionary. Based on the dictionary, the system decides that certain keywords correspond to values in catalog attributes, and others correspond to values in text descriptions of products, and generates an appropriate SQL query to answer a given keyword query.

The EasyAsk system supports a wide variety of features such as approximate word matching, word stemming (allowing it to match, for example, hiker with hiking), synonyms (for example, pants, trousers and slacks), and other word associations (for example “hunting” with “waterproof” and “outdoors” in the context of clothes). It also recognizes phrases, and supports comparisons such as “greater than 3 feet” which it translates into appropriate SQL conditions. EasyAsk also mentions that it handles data decoding, whereby users can use normal words, such as “blue”, even if the word has been coded in a different way, such as “bl”, in the database.

Further details of how EasyAsk handles keyword queries and natural language queries are not publicly available. Ranking is supported by EasyAsk, but as far as we can make out, only on administrator-specified criteria such as price, and not on the quality of match with keywords.

2.3 Proximity Search

Several research prototypes focus on the notion of proximity and efficient ranking or clustering of graph nodes by proximity.

Goldman et al. [7] were early proponents of using a graph distance-based measure in answering proximity queries. They generalized the notion of *near* queries in Information Retrieval (where the goal is to find documents where query tokens occur lexically close to each other) to a graph data model that can model both relational and semi-structured data.

They support queries of the form *find find-set near near-set*; such a query retrieves objects of the specified "find set" (e.g. a specified relation) that have short paths connecting them to objects in the near-set (those that match the specified keywords). The shortest path from objects in the "find set" to each of the objects in the "near-set" is computed using a distance function that adds up edge weights. The inverse of the distance, scaled by the weights of the end point nodes, gives the proximity of the objects. The score of an object in the find set is computed using its proximity to the near objects by either adding up the proximities, or by other means such as $1 - \Pi(1 - p_i)$, where the p_is are the proximities (which range from 0 to 1).

Note that the scores differ from that of DataSpot in that the inverses of distances to the near set objects are added up, instead of the distances themselves. Also, there is no significance to the actual paths, and only the objects in the find set are returned.

Web search provides another natural application where the best response may comprise a graph of connected pages rather than a single page. Li et al. [9] couch this problem in terms of Steiner trees connecting pages that match individual keywords. In their formulation, the graphs are not directed, and unlike DataSpot, pages and edges are not typed.

Proximity search is closely related to clustering. The connections between densely connected clusters in graphs and spectral properties of their adjacency matrices is well-established [8]. Clustering techniques for graphs and hypergraphs can be used to derive notions of distances between categorical data, and thereby to support proximity search between objects with categorical attributes [6].

Many Information Retrieval systems use thesauri and lexical networks to bridge the gap between imprecise user queries and the database by padding the query with synonyms and using *is-a* hierarchies (e.g., horse is a mammal). Such support needs to be hand-crafted into IR systems. In a graph framework, such as that used in DataSpot or BANKS, a network of metadata or linguistic relations can be regarded as simply augmenting the graph being queried with a richer set of connections.

2.4 Other Approaches

Sarda and Jain [14] describe a system called Mragyati to perform keyword search and browsing on databases. They generate SQL queries to retrieve results matching the given keywords. Queries can involve more than one relation, and are generated based on the database foreign-key structure and on the relations/attributes that are matched by the given keywords. Results can be ranked based on user specified criteria, or based on the number of foreign key references to the primary key (if any) of the answer tuples. The system provides mechanisms for handling synonyms and coding mechanisms used to store values in the database. Although supported in the model, the current implementation does not handle queries with paths of length greater than two, presumably because of the extra effort needed to analyze keywords and database connections to generate required queries. The system also generates hyperlinks in the results, to enable browsing.

Maserman and Vossen [10] describe an approach to keyword searching on databases, but their approach is restricted to finding all keywords in a single tuple, with no notion of links and proximity. They generate statements in an SQL extension called Reflective SQL, which is then translated to SQL.

Florescu et al. [5] propose an extension of the XML-QL query language to include keyword search, but their approach requires the use of a complex query language such as XML-QL.

2.5 Browsing

There has been a substantial body of work on browsing of relational databases and object oriented database. Although browsing is not part of keyword searching per se, the results of a keyword search can often be interpreted only as starting points from which the user finds required information by browsing. Work on browsing of databases include Dar et al. [4], Carey et al. [2], and more recently work by Shafer and Agrawal [16], which describes a system for integrated querying and browsing of relational data. Querying is carried out by interaction with form controls, rather than by keyword search. Munroe and Papakonstantinou [11] describe BBQ, an interface for browsing and QBE style querying of XML data.

3 Database and Query Model

In this section we describe how a relational database is modeled as a graph in the BANKS system. First we evaluate various options available and describe the model we adopt informally and then formalize it.

3.1 Informal Model Description

We model each tuple in the database as a node in the directed graph and each foreign key-primary key link as an edge between the corresponding tuples. This can be easily extended to other type of connections; for example, we can extend the model to include edges corresponding to inclusion dependencies, where the values in the referencing column of the referencing table are contained in the referred column of the referred table but the referred column need not be a key of the referred table.

In general, the importance of a link depends upon the type of the link i.e. what relations it connects and on its semantics; for example, in the bibliographic database, the link between the *Paper* table and the *Writes* table is seen as a stronger link than the link between the *Paper* table and the *Cites* table. Conceptually this model is similar the one described in [13] although there are some differences in the details.

To find answers, we need to traverse links backwards and we make this explicit by creating a backward link for each initial link. We model the weight of a backward link generated from a forward link as directly proportional to the indegree of the source node of the backward link (i.e. the referenced node). Since the proximity between the nodes connected by a link is inversely proportional to the link weight, the proximity for a referenced node to its referencing nodes is inversely proportional to the indegree of the referenced node. This notion is formalized in Section 3.2 These definitions are motivated by the intuition that "fans know celebrities better than celebrities know their fans." As another example, in a students' course registration database, if there are many students registered for a particular course, the proximity of two students due to the course is less than if there were fewer students registered. A forward edge from a student to a course and a back edge from the course to a student would form a path between each pair of student in the course, and assigning a higher weight to back edges in the case where more students take the course ensures that the paths are longer.

Informally, an answer to a query is a subgraph containing nodes matching the keywords and just by looking at the subgraph it is not apparent as to what information it conveys. We need to identify a node in the graph as a connecting node which connects all the keyword nodes. We consider an answer to be a rooted directed tree containing a directed path from the root to each keyword node. We call the root node an *information node*. The weight of the tree is proportional to the total of its edge weights. We may restrict the information node to be from a selected set of nodes of the graph; for example, we may exclude the nodes corresponding to the tuples from a specified set of relations, in a manner similar to [13].

We incorporate another interesting feature, namely node weights, inspired by prestige rankings such as PageRank in Google [1]. With this feature, nodes that have multiple pointers to them get a higher prestige. In our current implementation we set the node prestige to the indegree of the node. Higher node weight corresponds to higher prestige. E.g., in a bibliography database containing citation information, if the user gives a query *Query Optimization* our technique would give higher prestige to the papers with more citations.

3.2 Formal Database Model

In this section we define the formal graph model for representing the database. It consists of:

Vertices: For each tuple $\mathcal{T}$ in the database, the graph has a corresponding node $u_{\mathcal{T}}$. We will speak interchangeably of a tuple and the corresponding node in the graph.

Edges: For each pair of tuples $\mathcal{T}_1$ and $\mathcal{T}_2$ such that there is a foreign key from $\mathcal{T}_1$ to $\mathcal{T}_2$, the graph contains an edge from $u_{\mathcal{T}_1}$ to $u_{\mathcal{T}_2}$ and a back edge from $u_{\mathcal{T}_2}$ to $u_{\mathcal{T}_1}$ (this can be extended to handle other types of connections).

Edge weights: In our model, the weight of a forward link along a foreign key relationship reflects the strength of the proximity relationship between two tuples and is set to 1 by default. It can be set to any desired value to reflect the importance of the link (low weights correspond to greater proximity).

Let $s(R_1, R_2)$ be similarity from relation R_1 to relation R_2 where R_1 is the referencing relation and R_2 is the referenced relation. The similarity $s(R_1, R_2)$ depends upon the type of the link from relation R_1 to relation R_2 and this is different than the actual edge weights. It is set to infinity if relation R_1 doesn't refer relation R_2. Consider two nodes u and v in the database. Let $R(u)$ and $R(v)$ be the resp. relations they belong to. Further, let $IN_v(u)$ be the indegree of u contributed by the tuples belonging to relation $R(v)$. Note that from node u to node v we may, conceptually, have two edges, one forward edge which depends upon the similarity $s(R(u), R(v))$ and a backward edge which depends upon the similarity $s(R(v), R(u))$ and $IN_v(u)$. In the current implementation the forward edge weight is set to $s(R(u), R(v))$ and the reverse edge weight is set to $[s(R(v), R(u)) * IN_v(u)]$ and the actual edge weight is the minimum of the two as defined below:

$$b(u, v) = min(s(R(u), R(v)), s(R(v), R(u)) * IN_v(u))$$

The weight of a backward link generated from a foreign key relationship is directly proportional to the indegree of the source node (i.e. the referenced node). Since the proximity between the nodes connected by a link is inversely proportional to the link weight, the proximity from a referenced node to its referencing nodes is inversely proportional to the indegree of the referenced node.

Node weights: Each node u in the graph is assigned a weight $N(u)$ which depends upon the prestige of the node. In our current implementation we set the node prestige to the indegree of the node.

3.3 Querying Model

We now present our model for answering keyword queries. Let the query consist of n search terms $t_1, t_2, \ldots, t_n$. The query is (conceptually) answered as follows:

- For each search term t_i in the query we find the set of nodes that are relevant to the search term. Let us call the set S_i. And let $S = \{S_1, S_2, S_3, \ldots, S_n\}$. A node is relevant to a search term if it contains the search term as part of an attribute value. Nodes may also be relevant through metadata (such as column, table or view names). E.g., all tuples belonging to a relation named `AUTHOR` would be regarded as relevant to the keyword 'author'.

- An answer to a query is a rooted directed tree containing at least one node from each S_i. Note that the tree may also contain nodes not in any of the S_is and is therefore a Steiner tree. The relevance score of an answer tree is computed from the relevance scores of its nodes and its edge weights. (The condition that one node from each S_i must be present can be relaxed to allow answers containing only some of the given keywords.)

- Given an answer with keyword matching nodes $(s_{i,1}, s_{i,2}, \ldots, s_{i,n})$ the relevance of the answer is computed using the edge and node weights as follows:

$$AnswerRelevance(s_i) = w_n \left[\frac{\sum_j N(s_{i,j})}{N}\right] + w_p \left[\frac{1}{\sum E_k}\right]$$

  where the E_i's are the weights of the edges in the answer tree, N is the maximum node weight sum across all elements of S, and w_n and w_p are weights used to control the relative importance given to node weights and proximity. These combining weights are ad-hoc, but appear to be inescapable in all related systems that we have reviewed [7, 17]. Reasonable choices can be made if sample queries with relevance judgments are provided to 'train' the system [15].

  The minimum Steiner tree problem is a special case of the problem of finding answers of maximum relevance, so the problem of finding the best answers is also NP-Complete. We therefore settle for heuristics to construct answer trees of high relevance. These are discussed in Section 4.

3.4 Relation of BANKS to Earlier Work

BANKS is closely related to DataSpot [3, 12, 13]. In particular, the model of query answers as rooted trees corresponds to the DataSpot model, where the roots are called fact nodes. The details of the underlying graph formalism, however, differ. BANKS currently works on a model where only references, which correspond to equivalence edges in DataSpot, are explicitly represented. Since edges in our model can have attributes such as type and weight, we can model containment (as in DataSpot and in nested XML) simply as edges of a new type. (We are currently working on adding XML support to BANKS.) The BANKS technique of assigning weights to back edges, based on indegrees, has no counterpart in DataSpot, as also the node weight mechanism used in BANKS. The use of node weights based on prestige has proved critical in Web search, and our anecdotal evidence shows their importance in the context of database search as well. BANKS also takes the effect of metadata queries into account, which is not made explicit in DataSpot.

Unlike [9], BANKS (like DataSpot) can exploit the semantically richer set of links available from foreign keys and other constraints in the structured (relational) or semistructured (XML/OEM) setting, which is largely missing in graphs formed by HTML documents.

4 Implementation Approach

Our heuristic solution is based on Dijkstra's single source shortest path algorithm. We assume that the graph fits in memory. This is not unreasonable, even for moderately large databases, because the in-memory node representation need not store any attribute of the corresponding tuple other than the RID. As a result the graphs of even large databases with millions of nodes and edges can fit in modest amounts of memory. We will be looking into external memory based applications as a part of our future work.

Given a set of keywords, first we find, for each keyword term t_i, the set of nodes, S_i, that are relevant to the keyword. In the current implementation, we search only for exact matches, and to facilitate this we build a single index on values from selected string-valued attributes from different tables. The index maps from keywords to (table-name, tuple-id) pairs.

Let $relevantNodes = S_1 \cup S_2 \cup \ldots \cup S_n$ be the relevant nodes for the query. We concurrently run $|relevantNodes|$ copies of the single source shortest path algorithm, one for each node in $relevantNodes$ as source. We run them concurrently by creating an iterator interface to the shortest path algorithm, and creating multiple instances of the iterator.

The important distinction of this approach is that the single source shortest path algorithm traverses the graph edges in reverse direction. The idea is to find a common vertex from which a forward path exists to at least one node in each set S_i. Such paths will define a rooted directed tree with the common vertex as the root and the corresponding keyword nodes as the leaves. The tree thus formed is a connection tree and root of the tree is the information node.

We create a single source shortest path iterator for each keyword node. At each iteration of the algorithm, we need to pick one of the iterators for further expansion. We pick an iterator whose next vertex to be output is at the least distance from the source vertex of the iterator (the distance measure can be extended to include node weights of nodes matching keywords). We keep a list of all the vertices visited for each iterator. Consider a set of iterators containing one iterator each from set S_i. If the intersection of their visited vertex lists is non-empty, then each vertex in the intersection defines a connection tree. The result set may contain multiple result trees with same keyword nodes but different root nodes. We eliminate duplicates, keeping only the root which gives maximum relevance weight.

5 Browsing

The BANKS system provides a rich interface to browse data stored in a relational database and is well integrated with the search facility. The browsing system automatically generates browsable views of database relations, and of query results, by using two mechanisms: foreign key relationships and nesting of data using a mechanism similar to `GROUP BY` in SQL. For every attribute that is a foreign key, a link is created in the display to the referenced tuple. In addition, primary key columns can be browsed backwards, to find referencing tuples, organized by referencing relation (a specific referencing relation can be selected by the user).

Each table displayed comes with a variety of tools for interacting with data. Apart from direct schema browsing we support operations like sorting data on a specified column, restricting the data by a predicate on a column, projecting away a column, taking join with the referenced table by clicking on a foreign key column, data nesting wherein only the distinct values of a column specified are displayed, etc. Controls for these operations can be accessed by clicking on the column names in the table header.

Note that all the hyperlinks are automatically generated by the system and no content programming or user intervention is needed. Each hyperlink is really an SQL query which is executed when a user clicks on the links.

Another important feature are **templates**; templates provide several predefined ways of displaying any data. Templates must be customized by specifying various information (depending on the template); a customized template is given a hyperlink name and is then available to users for browsing.

6 Experience and Performance

We have implemented BANKS using servlets, with JDBC connections to an IBM Universal Database. We have experimented with two datasets: One is the DBLP Bibliography database shown in Figure 1. We converted a dump of DBLP into structured relational format and ran BANKS on this data. There are 124,612 nodes and 319,232 edges in this graph. The other one is a small thesis database. IIT Bombay's database of Masters and Phd dissertations are available in relational format; these are input to BANKS.

A system like BANKS may be evaluated along (at least) two measures: quality of the results and speed. Unfortunately, neither is easy to characterize. There are no agreed-upon benchmarks for evaluating proximity- or prestige-base ranking algorithms in this domain. To work around this, we selected data sets that academics and database researchers are familiar with and can relate to, at least w.r.t. the schema. This makes the discussion of results more meaningful, albeit qualitative. While we can compare the different heuristics with each other on the quality of their answers, it is not clear what is an "optimal" answer to compare against. Eventually, user experience counts, which is what led us to releasing the search facility on a Web site.

We give a few examples of queries on the bibliographic database. For the query "Mohan", C. Mohan came out at the top of the ranking, with Mohan Ahuja and Mohan Kamat following. This was due to the prestige conferred by the *writes* relation which had many tuples for these authors. The query "transaction" returned Jim Gray's classic paper and the book by Gray and Reuter as the top two answers.

The query “soumen sunita” returned only one answer: a tree containing a node corresponding to the paper “Mining Surprising Patterns Using Temporal Description Length” as the information node. This paper has Soumen and Sunita as co-authors and the tree has the two corresponding *author* tuples as leaf nodes and two *writes* tuples – connecting the two author tuples to the paper tuple – as intermediate nodes.

The query “sunita olap” returned some interesting results – the part of DBLP we had loaded had no paper by “sunita” with “olap” in its title, but the system found several papers with “olap” in the title that cited or were cited by papers authored by “sunita”. Several of the resultant papers by “sunita” were on OLAP even though the word was absent in the title. A useful extension would be to differentiate between papers that cite papers by “sunita” and papers cited by papers by “sunita”. We could, for instance, provide a way to select an answer corresponding to one of these forms (at the level of the database schema) and ask for more answers of that form.

The BANKS system supports metadata queries. For example, the query “thesis sudarshan” returns all thesis’ advised by Sudarshan (in addition to thesis’ written by a Sudarshan, if there had been any).

Currently loading the DBLP database takes 2 minutes, and about 100 MB of memory, but we expect this to decrease greatly with a better tuned Java or C implementation. Once the database graph is loaded, queries usually take a second or so to get the first answer, and a few seconds to get all answers (up to some relevance cutoff) on the DBLP database. Our prototype implementation clearly demonstrates the feasibility of using a system such as BANKS for moderately large databases.

7 Extensions and Future Work

Several extensions are possible to our model. A keyword in a tuple/relation-name may not be exactly equal to a search term but instead be a synonym to it. The BANKS model is easily extendible to allow scaling down of node weights to account for approximate match or synonyms. Synonyms are particularly useful in the context of matching metadata. Performance issues caused by metadata keywords matching large numbers of nodes are being addressed in BANKS.

The model can be easily extended to support predicate of the form *attr:[op]value*, e.g. **“author:Sudarshan, year:>2000”**. We want to allow user-defined rankings for temporal and other ordered domains. E.g., one may search for `concurrency recent` requiring a paper about concurrency published “recently”.

For a given set of keywords, there may not exist a reasonable cost tree that includes one node corresponding to each keyword. In such a situation a tree including only some of the keywords may be useful. In our current implementation we set the node prestige to the indegree of the node. We can incorporate a full-fledged eigen-analysis as in Google/PageRank so as to facilitate prestige transfer (a form of spreading activation), wherein nodes pointed to by heavy nodes become heavier. We are also considering several additional functionalities in the user interface such as ways of getting answers with a tree structure (form) similar to a chosen answer, and of getting summaries as answers. We plan to apply our current and later proximity algorithms to XML/OEM.

8 Conclusions

Many Web sites are becoming database-centric, and manually creating interfaces for browsing and querying data is time consuming. Systems supporting keyword search on relational data reduce the effort involved in publishing relational data on the Web and making it searchable. Examples of the types of data that could be published with keyword search support include organizational data, bibliographic data and product catalogs. We surveyed several approaches to keyword search on databases. We have developed BANKS, an integrated browsing and keyword querying system for relational databases. BANKS has many useful features which allow users with no knowledge of database systems or schema to query and browse relational databases with ease.

Acknowledgments: We wish to thank B. Aditya, Urmila Kelkar, Megha Meshram and Parag for implementing some parts of the BANKS system and for their help with loading the databases.

References

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7), 1998.

[2] Michael J. Carey, Laura M. Haas, Vivekananda Maganty, and John H. Williams. Pesto : An integrated query/browser for object databases. In *Procs. of the International Conf. on Very Large Databases*, pages 203–214, 1996.

[3] Shaul Dar, Gadi Entin, Shai Geva, and Eran Palmon. DTL's DataSpot: Database exploration using plain language. In *Procs. of the International Conf. on Very Large Databases*, pages 645–649, 1998.

[4] Shaul Dar, Narain H. Gehani, H. V. Jagadish, and J. Srinivasan. Queries in an object-oriented graphical interface. *Journal of Visual Languages and Computing*, 6(1):27–52, 1995.

[5] Daniela Florescu, Donald Kossmann, and Ioana Manolescu. Integrating keyword search into xml query processing. *WWW9/Computer Networks*, 33(1-6):119–135, 2000.

[6] David Gibson, Jon M. Kleinberg, and Prabhakar Raghavan. Clustering categorical data: An approach based on dynamical systems. In *Procs. of the International Conf. on Very Large Databases*, pages 311–322, 1998.

[7] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity search in databases. In *Procs. of the International Conf. on Very Large Databases*, pages 26–37, 1998.

[8] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *JACM*, 46(5):604–632, 1999.

[9] Wen-Syan Li, K Selcuk Candan, Quoc Vu, and Divyakant Agrawal. Retrieving and organizing web pages by "information unit". In *World-wide Web Conference*, 10, pages 230–244, 2001.

[10] Ute Masermann and Gottfried Vossen. Design and Implementation of a novel approach to Keyword Searching i n Relational Databases. In *Current Issues in databases and information systems*, pages 171–184, September 2000.

[11] Kevin D. Munroe and Yannis Papakonstantinou. BBQ: A visual interface for integrated browsing and querying of xml. In *Visual Database Systems*, May 2000.

[12] Eran Palmon. Associative search method for heterogeneous databases with an integration mechanism configured to combine schema-free data models such as a hyperbase. United States Patent Number 5,740,421, Granted April 14, 1998, filed in 1995. Available at `www.uspto.gov`, 1998.

[13] Eran Palmon and Shai Geva. Associative search method with navigation for heterogeneous databases including an integration mechanism configured to combine schema-free data models such as a hyperbase. United States Patent Number 5,819,264, granted October 6, 1998, filed in 1995. Available at `www.uspto.gov`, 1998.

[14] N. L. Sarda and Ankur Jain. Mragyati: A system for keyword-based searching in databases. Submitted for publication. Contact: nls@cse.iitb.ac.in., 2001.

[15] Hinrich Schütze, David A. Hull, and Jan O. Pedersen. A comparison of classifiers and document representations for the routing problem. In *ACM SIGIR'95, (Special Issue of the SIGIR Forum)*, pages 229–237, 1995.

[16] John C. Shafer and Rakesh Agrawal. Continuous querying in database-centric web applications. *WWW9/Computer Networks*, 33(1-6):519–531, 2000.

[17] Ron Weiss, Bienvenido Vélez, Mark A. Sheldon, Chanathip Nemprempre, Peter Szilagyi, Andrzej Duda, and David K. Gifford. Hypursuit: A hierarchical network search engine that exploits content-link hypertext clustering. In *Proc. of ACM Hypertext*, pages 180–193, 1996.

Using Probabilistic Argumentation Systems to Search and Classify Web Sites

Justin Picard and Jacques Savoy *
Institut interfacultaire d'informatique
Pierre-à-Mazel 7
2000 Neuchâtel (Switzerland)

Abstract

As the amount of information stored on the web increases at an amazing pace, it gets harder for search engines to retrieve the needed information. Recently, attention in the web community has focused on the use of hyperlinks to help index and organize information. Several methods have been suggested to take account of the knowledge induced by hyperlinks, with different objectives in mind. In this paper, we focus on three of them: improving document ranking, estimating the popularity of a web page, and extracting the most important hubs and authorities related to a given topic. Using probabilistic argumentation systems, a technique for dealing with uncertain knowledge which integrates propositional logic and probability theory, we show how all these techniques can be modeled in a unified logical framework. This allows comparisons of the different methods for using hyperlinks and illustrates some of their weaknesses based on some experiments.

1 Introduction

Traditional techniques for indexing automatically documents must be reviewed in the context of the Internet. For example, web pages may vary from a few words to megabytes. Moreover, the assumption that the words contained in a document are good indicators of the underlying semantics is not as much verified as in document collections: people who write web pages may have specific objectives in mind, and use tactics to influence the ranking of search engines, called "spamming" or "the search engine persuasion problem" [Mar97]. Finally, the well-known problems of polysemy and synonymy are aggravated on the web: in a click, one can find commercial information, scientific papers, data repositories, propaganda or the web pages of friends.

To compensate the difficulties with indexing and retrieving information, many search engines are also offering hierarchical classifications. These classifications were created and maintained manually. However the cost of such a manually-based organization of information does not make it a viable option at long term because web pages change continuously and new categories or "cyber communities" appear everyday on the web. And as reported in [Mar97], "repositories are now themselves resorting to search engines to keep their database up-to-date".

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This research was supported in part by the SNF (Swiss National Science Foundation) under grant 20'55'771.98

1.1 Hypertext links

The difficulties with searching and organizing the web have led researchers to investigate other sources of knowledge. Recently, attention has focused on one of them: the few billion hypertext links which "glue" the Internet together. The web would after all not exist without these links, which are the paths which lead to information. Indeed, browsing is for many users an usual way to find "nearby" information [Mar97].

Recent experiments seem to indicate that hyperlinks can be very valuable in locating or organizing information. They have been suggested: (1) to enhance an initial ranking of documents [Sav94], (2) to compute an estimate of a web page popularity [BL98], or (3) to find the most important hubs and authorities for a given topic [Kle98, Bh98, CdBD99]. It seems that each of these techniques is based on some underlying, sometimes partly implicit hypothesis, on the type of knowledge which can be induced by the presence of a hyperlink between two pages. For example, when they are used to estimate the popularity of a web page, hyperlinks can be interpreted in the following way: "if a document is cited by a popular page, then it is possibly popular itself". This type of knowledge can easily be captured by propositional logic, if some measure of uncertainty is associated.

1.2 Outline of this paper

Probabilistic argumentation systems are a convenient technique for dealing with uncertain knowledge, by combining propositional logic with probability theory. In this paper, we will show how the techniques mentioned in the previous section can be described in this unified framework. This allows comparisons of the methods, highlights some previously unseen weaknesses and leads to new methods using hyperlinks.

Section 2 will introduce the probabilistic argumentation systems, which have already been applied to information retrieval (IR) in hypertext [Pic98]. Sections 3, 4 and 5 will describe techniques to handle hyperlinks in order to improve document ranking, estimate the popularity of a web page, and extract the most important hubs and authorities related to a given request. We will see how probabilistic argumentation systems can be used to deal with the uncertain knowledge induced by the hypertext structure for the same purposes. Section 6 will present some experiments with the model developed for improving an initial ranking of documents and the last section will conclude this paper.

2 Probabilistic argumentation systems

In this section, we make a short introduction to probabilistic argumentation systems (PAS). There is much more about PAS than what is shown here. For a detailed overview of PAS, the reader is referred to [HKL99]. However, this tutorial should be sufficient to understand their application in this paper.

2.1 PAS

Propositional logic is one of the simplest and most convenient ways of encoding knowledge. An apparent drawback is that propositional logic seems to be unsuitable for taking account of uncertainty. However, uncertainty can be handled rather easily by adjoining particular propositions called **assumptions**. Assumptions are propositions which state the unknown conditions or circumstances upon which the facts and rules depend. If an assumption is known to be true, then the fact or rule which depends on it holds. Otherwise, nothing can be deduced from this fact or rule.

For example, let proposition D_1 denote "document d_1 is relevant to the information need". Proposition D_1 can be either true or false. There might be some uncertainty associated to this fact, for example it may depend on the reliability of the search engine which has retrieved it. By using an assumption a_1 denoting the uncertain conditions under which the fact D_1 holds, the uncertainty can be captured by: $a_1 \rightarrow D_1$ [1]. Similarly, uncertainty

[1] For reading commodity, assumptions will be denoted by lowercase letters, and other propositions by capital letters

in rules can also be captured by assumptions. For example, suppose that documents d_1 and d_2 concern similar subjects such that in some cases, they are both relevant to the same information need. These conditions which do not always apply can be captured by the following: $l_{12} \rightarrow (D_1 \rightarrow D_2)$, where D_2 means "document d_2 is relevant", and l_{12} denotes the uncertain circumstances under which the rule $D_1 \rightarrow D_2$ applies. We will in general prefer the following equivalent notation for uncertain rules: $D_1 \wedge l_{12} \rightarrow D_2$.

Most applications also require a numerical assessment of uncertainty. The numerical aspect of uncertainty is obtained by assigning probabilities to assumptions. For example, if for the uncertain rule $D_1 \wedge l_{12} \rightarrow D_2$, the condition l_{12} is known to hold with probability 0.3, then we may write: $p(l_{12}) = 0.3$. Note that this is conceptually different from assigning a probability to the whole logical sentence ($p(D_1 \rightarrow D_2) = 0.3$), as is done in other frameworks for integrating uncertainty with logic.

Given a knowledge base composed of uncertain facts, rules or conditions modeled with logical formulas containing assumptions, we are interested in finding which symbolic **arguments** support or discard a given hypothesis h. A symbolic argument is a conjunction of literals of assumptions which, if added to the knowledge base, makes the hypothesis true. We will then compute the **symbolic support** of h given by the knowledge base ξ, denoted $sp(h, \xi)$, which contains the disjunction of all symbolic arguments which allow to derive h if added to the knowledge base. We may also want to evaluate the reliability of the support given by arguments, using probabilities assigned to the assumptions. We will then compute the **degree of support** $dsp(h, \xi) = p(sp(h, \xi))$, the probability that the hypothesis h is supported by the knowledge base ξ.

2.2 An example

For example, consider the following knowledge base:

$$\xi = (a_1 \rightarrow D_1) \wedge (a_2 \rightarrow D_2) \wedge (D_1 \wedge l_{12} \rightarrow D_2) \tag{13}$$

Remark that a knowledge base can always be represented as a conjunction of rules, facts, and more generally clauses. We are interested in finding the arguments for hypothesis D_2 given by ξ. It is easily seen that a_2 is an argument for D_2, because $(a_2 \rightarrow D_2) \wedge a_2 \models D_2$. The same way, $(a_1 \wedge l_{12})$ is another argument for D_2. Thus, the symbolic support given by the knowledge base for D_2 is computed in the following way:

$$sp(D_2, \xi) = a_2 \vee (a_1 \wedge l_{12}) \tag{14}$$

The following probabilities are assigned to the assumptions: $p(a_1) = 0.4, p(a_2) = 0.25, p(l_{12}) = 0.3$. What is the probability that the support holds? In order to make an exact computation, independence assumptions must be made, e.g., $p(a_1 \wedge a_2) = p(a_1) \cdot p(a_2)$, $p(a_1 \wedge \neg a_2) = p(a_1) \cdot (1 - p(a_2))$, etc. The degree of support of D_2 given by the knowledge base, $dsp(D_2, \xi)$, is:

$$\begin{aligned}
dsp(D_2, \xi) &= p(sp(D_2, \xi)) && (15)\\
&= p(a_2 \vee (a_1 \wedge l_{12})) && (16)\\
&= p(a_2 \vee (a_1 \wedge l_{12} \wedge \neg a_2)) && (17)\\
&= p(a_2) + p(a_1) \cdot p(l_{12}) \cdot (1 - p(a_2)) && (18)\\
&= 0.34 && (19)
\end{aligned}$$

The passage from Equation 4 to 5 in the previous equations comes from the logical equivalence: $A \vee B = A \vee (B \wedge \neg A)$. Next sections will show how the notions on PAS presented here can be applied to deal with the knowledge induced by the hypertext structure.

3 Using hyperlinks to modify document score and rank

3.1 Spreading activation in hypertext

The implicit reasoning made in the spreading activation (SA) technique is the following: a link from a document d_1 to a document d_2 is an evidence that their content is similar or related, such that if d_1 is relevant to a given request, d_2 may also be relevant. From an initial ranking of documents produced by a search engine, the hyperlinks can be used to enhance the ranks of the documents linked to the best ranked documents. For example, if d_2 is linked to d_1 which is ranked first, then d_2 should be placed at a better rank.

The general scheme works as follows. The retrieval engine computes an initial retrieval status value (RSV) or score for each document based on its similarity with the query q. The RSV of document d is then updated by adding a fraction of the RSV of its m neighbors through a certain number of cycles. The neighbors can linked by incoming but also outgoing links. Suppose that document d has neighbors d_1 to d_m. The RSV of d at cycle $i+1$ is computed by the following:

$$RSV(d^0) = score(d, q) \tag{20}$$

$$RSV(d^{i+1}) = RSV(d^i) + \sum_{j=1}^{m} \lambda_j \cdot RSV(d_j^i) \tag{21}$$

The parameter λ_j can be seen as the degree of certainty regarding the evidence provided by the link from d_j to d. It can be a fixed value according to the link type[2], or may vary according to a measure of similarity between the documents and the query [Sav97]. We may also repeat this propagation scheme through a certain number of cycles under the assumption that "friends of my friends are my friends". However, the number of cycles is often limited to one: more than one cycle is usually harmful to retrieval effectiveness [Sav97].

Several problems can be found with SA approach: there is no theoretical background guiding the choice of the number of cycles c or the value of the parameter λ_j. Moreover, evidence may propagate more than once if there are cycles in the network.

3.2 Improving document ranking with PAS

Based on the PAS approach, we will also attempt to improve document ranking using the hypertext structure. But instead of propagating document scores, we will seek all symbolic arguments supporting the relevance of a document. In a second phase, probabilities are assigned to the assumptions and the degree of support given by the arguments is computed. Finally documents are returned to the user by decreasing degree of support.

For each document d_i, let us denote proposition D_i as: "document d_i is relevant". If a document is retrieved by the retrieval system, this is evidence in favor of that document's relevance. Let assumption a_i denote, "the retrieval system has correctly retrieved document d_i". Then for each document in the collection, we have:

$$a_i \rightarrow D_i \tag{22}$$

For a given query, we may adjust the probability of the assumption $p(a_i)$ to the rank at which d_i is retrieved ($p(a_i|rank)$), and set $p(a_i) = 0$ if d_i is not retrieved. In practice, we fit a logistic regression on the rank for a set of training queries [SP01].

In a second step, we want to use the hypertext structure to improve this initial ranking. For each link from d_i to d_j, we induce the knowledge that, under some condition l_{ij}, the relevance of d_i implies the relevance of d_j.

[2] We may consider links of various types: hypertext links, citation, nearest neighbor, etc. Moreover, links can be distinguished by their orientation (incoming, outgoing), because this orientation may affect the amount of information about relevance contained in the link.

The assumption l_{ij} may denote the conditions under which the link implies relevance in the present context. We have then:

$$D_i \wedge l_{ij} \to D_j \tag{23}$$

We may then find the arguments supporting the relevance of each document. As an example, take a collection containing documents d_1, d_2, d_3. There are links from d_2 to d_1 and from d_3 to d_1. The following knowledge base is generated:

$$\xi = (a_1 \to D_1) \wedge (a_2 \to D_2) \wedge (a_3 \to D_3) \wedge (D_2 \to D_1) \wedge (D_3 \to D_1) \tag{24}$$

We find for the support of D_1:

$$sp(D_1, \xi) = a_1 \vee (a_2 \wedge l_{21}) \vee (a_3 \wedge l_{31}) \tag{25}$$

Here, d_1 has three symbolic arguments. For a real query, one may want a numerical evaluation. The degree of support of D_1 is:

$$\begin{aligned} dsp(D_1, \xi) &= p(sp(D_1, \xi)) = p(a_1) + p(a_2 \wedge l_{21} \wedge \neg a_1) + p(a_3 \wedge l_{31} \wedge \neg a_1 \wedge \neg(a_2 \wedge l_{21})) \\ &= p(a_1) + p(a_2) \cdot p(l_{21}) \cdot (1 - p(a_1)) + p(a_3) \cdot p(l_{31}) \cdot (1 - p(a_1)) \cdot (1 - p(a_2) \cdot p(l_{21})) \end{aligned}$$

For a given query, one needs to give values to the $p(a_i)$'s according to the rank of d_i, and probabilities for the links $p(l_{ij})$. It is interesting to notice that the hypertext structure, interpreted logically, has been integrated in the computing formulas for the degree of support of each document. This way, the computations can be done very fast. Section 6 will show some experiments comparing the SA approach to the PAS model developed in this section, and demonstrate how probabilities can be computed in practice.

4 Estimating the popularity of a web page

4.1 PageRank

The PageRank algorithm [BL98] considers that users have an absolute preference among web pages: it assumes that the more a web page is visited, the more it is appreciated by the users. To measure this popularity, a reasonable assumption is that the preference of users is reflected in the hypertext structure: a link toward a web page is often an indication that this page is acknowledged by the author as a good source of information. A simple way to implement this idea would be to count the number of times a web page is cited. Microsoft's home page, surely one of the most visited page on the web, is cited more than 23 million times in Altavista's index (probably much more in reality). However, each link should not be treated equally, since its impact also depends on the popularity of the parent node: a page cited only a few times but which is in Yahoo!'s index would certainly be quite visited. Thus the popularity of a page also depends on the popularity of the pages that cite it.

Such a popularity measure is used in the Google search engine to boost the scores of the documents, independently of the query. This algorithm is criticized because it biases the access to information [LG99]. The "perverse" effect of PageRank is that it will push popular pages to get even more popular, and new or unknown (unlinked) web pages to stay unknown. As said in [Mar97], "visibility is likely to be a synonym of popularity, which is completely different than quality, and thus using it to gain higher score is a rather poor choice". To our advice, the frequency at which a page is visited by all the users of the web is not necessarily an indicator of its relevance to a user who has its own preferences, cultural background, etc. The PAS modeling offers a clean way to take account of these a priori user preferences.

4.2 PAS and personalized ”popularity” measure

Suppose that for each user, it is possible to compute some personalized ”popularity” measure. For example, each user may define a profile (e.g., a set of keywords, a set of web pages defined by a bookmarks list), such that, for each page on the web, we can assign a probability $p(a_i)$ based on its similarity with the user profile. We would like to refine this $prior$ knowledge by taking account of the hypertext structure. Let us define P_i as ”document d_i corresponds to the user's interest”. Then for each document, there is some condition a_i under which d_i corresponds to the user's interest which is denoted as:

$$a_i \rightarrow P_i \tag{26}$$

The probability $p(a_i)$ can be initially computed bases on the user profile, and then updated by keeping track of the pages visited. For each link from d_i to d_j, we induce that, under some condition l_{ij}, the relevance of d_i implies the relevance of d_j to the user's interest. We have then:

$$P_i \wedge l_{ij} \rightarrow P_j \tag{27}$$

Note that the symbolic support that a document belongs to the favorite pages is the same as the support that it will be relevant. In our model, each user will have the same symbolic arguments supporting the popularity of each document. If an equal probability $p(a_i)$ is assigned to each document, it is assumed that the user has no preference and this case corresponds to the PageRank model. However if the user gives some hints allowing to compute personal a priori probabilities $p(a_i)$ or eventually link probabilities (e.g., by inspecting user bookmarks lists), it will be possible to have a personal ranking for this user.

5 Finding hubs and authorities

5.1 Kleinberg's algorithm

In many cases, the user does not know what exactly he/she is looking for, and is rather interested in having good starting points for browsing. Given a general topic sufficiently represented on the web (e.g., ”human rights”, ”MP3”), it is possible to distinguish two types of potentially relevant pages: **authorities** and **hubs**. Authorities are pages containing high quality and exhaustive information on a topic, and hubs are pages containing links to the authorities, thus giving access to the relevant information. The web is very rich in central pages, fan sites and other classifications of resources, and those can be very helpful for automatic classification of information.

How can we find hubs and authorities? The assumption made by Kleinberg [Kle98] is that a good authority is a page which has links from many good hubs, and a good hub is a page which has links towards many good authorities. The algorithm has some similarity with PageRank model in that the quality of a page depends recursively on the quality of the neighbors, although here the links are followed in both directions. The idea of Kleinberg's HITS algorithm [Kle98] is to consider a root set (e.g., 200 documents), containing the most likely relevant pages found with a search engine in response to a given query. This root set is expanded with all documents which point to or are pointed by these pages, to form the base set in which authorities and hubs will be found. Then the connectivity of this base set is used as follows to find the best hubs and authorities. For each document d_p in the base set, a hub score h_p and a authority score a_p are computed. Both initial scores are set to 1. Then the hub and authority scores are updated iteratively by, respectively, the sum of authority scores of pages cited by d_p, and the sum of hub scores of the pages citing d_p. The updating equations are:

$$h_p = \sum_{d_p \rightarrow d_i} a_i, a_p = \sum_{d_i \rightarrow d_p} h_i \tag{28}$$

where $d_p \rightarrow d_i$ means that there is a link from d_p to d_i. It can be shown that both scores will converge if they are normalized after each iteration. The exact scores are not so important, since the user is presented with a ranked list of hubs and authorities.

It is argued that the algorithm has an "objective" justification because it finds some intrinsic properties of a set of linked pages [Kle98, CdBD99]. However, we believe there are some weaknesses in this algorithm. One is that a base set has to be chosen for a given topic, from which the hubs and authorities will be selected. Although it is argued in [Kle98] that the method is robust (i.e. gives similar results) for different base sets, the choice of a particular base set is nonetheless purely heuristic. Another is that the initial ranking of documents is not used as prior evidence, while clearly an initially better ranked document has more chances to be relevant, and certainly more chances to be a good hub or authority.

5.2 A model for computing hub and authority scores

Given a user's request and a document d_i, proposition H_i denotes "document d_i is a good hub" and A_i denotes "document d_i is a good authority". Unlike Kleinberg's algorithm, we consider that there is initial evidence h_i that D_i is a good hub, and a_i that it is a good authority.

$$h_i \rightarrow H_i, a_i \rightarrow A_i \tag{29}$$

As in Kleinberg's algorithm, we make the assumption that if a document d_i is cited by a good hub d_j, then this is evidence that d_i is a good authority. We have then:

$$H_j \wedge f_{ji} \rightarrow A_i \tag{30}$$

Similarly, if a good authority d_i is cited by a document d_j, then this is evidence that that d_j is a good hub:

$$A_i \wedge g_{ij} \rightarrow H_j \tag{31}$$

For each hyperlink from d_j to d_i, there will be two rules generated: $(H_j \wedge f_{ji} \rightarrow A_i), (A_i \wedge g_{ij} \rightarrow H_j)$. From this knowledge base ξ, one can compute for each document d_i, the symbolic support from ξ that it is a good hub and a good authority, $sp(H_i, \xi)$ and $sp(A_i, \xi)$. Then, for a given topic, different probabilities are assigned to the assumptions $p(a_i)$ and $p(h_i)$, and eventually to the assumptions f_{ij} and g_{ij}, which can be fixed or depend on some similarity value with the topic. The numerical degrees of support $dsp(H_i, \xi)$ and $dsp(A_i, \xi)$ will be the hub and authority scores of document d_i for this topic. Note that compared with Kleinberg's algorithm, there is no need to determine a base set, which is here the same for all topics. For different topics, only the assigned probabilities will change.

6 Experiments with the model for spreading activation

In this subsection, we present our experiments done on the CACM and Trec'8 web test-collections. The CACM test-collection (3.2 MB) contains 3,204 documents and 50 requests with their associated relevance judgments. The web Track collection, denoted WT2g, contains around 250,000 pages (2.3 GB) extracted from the web and owns 100 requests with their relevance assessments. The retrieval effectiveness is given by the average precision at 11 recall values computed by the TRECEVAL software, the most used evaluation measure in IR.

6.1 Probabilities estimates

The set of symbolic operations is the following:

- Convert each link from d_i to d_j to $D_i \wedge l_{ij} \rightarrow D_j$.

Type of link	SA	PAS	PAS vs. baseline	PAS vs. SA
Citing	0.266 (0.3)	0.267	+5.57%	+0.01%
Cited	0.261 (0.4)	0.273	+7.83%	+4.11%

Table 3: Average precision on the CACM test-collection

- For each document D_i, compute the support $sp(D_i, \xi)$.
- Put the support of each document in disjoint form using Heidtmann's algorithm [Hei89]. Convert this disjoint form to a directly usable formula for computing the degree of support $dsp(D_i, \xi)$.

At this point, all the logical operations are done. A learning phase is also necessary to assign probabilities to the assumptions $p(a_i)$ and $p(l_{ij})$. For the first ones, a set of training queries is used to fit a logistic regression to compute the probability of relevance given the page rank. The probabilities of link assumptions can be static or may depend on some similarity value between the link and the search topic. For these experiments, we will only consider static probabilities. They can also be estimated on a set of training queries [Pic98].

When a query is processed, the only operation remaining to compute is the degree of support for each document d_i, $dsp(D_i, \xi)$.

6.2 Experiments

For the CACM collection, links were considered separately in the forward direction (citing) and in the backward direction (cited). For each document, arguments of order three and less were computed, and the symbolic support was put in disjoint form with Heidtmann's algorithm.

The basic retrieval process was done using a classical retrieval system based on the cosine similarity measure and this baseline average precision is 0.253. In Table 3, comparisons were made between PAS and this baseline, and PAS and spreading activation (SA). The results shown for SA are the best ones obtained for a range of values of the parameter λ, which was fixed for all links of a certain type. This best λ value is depicted between parenthesis in Table 3. For the PAS model, arguments of length three or less (at most three links assumptions) were computed.

For the WT2g collection, ten different weighting schemes were used (okapi-npn, ..., bnn-bnn), to produce ten different rankings of documents (for details see [SP01]). Table 4 depicted the retrieval effectiveness using only the best incoming links, the best outgoing links, and both. There are slight but generally not significant improvements over the baseline.

These results demonstrate that PAS can compete with an established technique such as SA. Also, indirect neighbors can be considered without depreciating performance, at least for one test collection. For citing links, there is no average difference in retrieval effectiveness, while for cited links, the difference is nearly significant. However, for SA there is no understanding of the parameters involved, while in the PAS framework, parameters are the probabilities of the links which have a clearer meaning both from a statistical and a logical viewpoint.

7 Discussion

In this paper, we have shown how various approaches using hyperlinks in order to search and organize the web can be modeled using a single theoretical framework. Experiments have validated the approach when links are used to improve an initial ranking. Experiments for computing hub and authorities, and a personalized popularity measure have not been conducted yet because due to the fact that building a good quality test-collection is a very costly and demanding task. But the IR community is tackling the task of building good quality web test-collections, and experiments should be possible in a next future.

Model	Baseline	Best incoming	Best outgoing	Combined
okapi-npn	0.267	0.267 (0.00%)	0.267 (0.00%)	0.267 (0.00%)
lnu-ltc	0.234	0.239 (+2.18%)	0.240 (+2.65%)	0.239 (+2.10%)
atn-ntc	0.257	0.260 (+1.44%)	0.261 (+1.52%)	0.260 (+1.32%)
ntc-ntc	0.138	0.139 (+0.14%)	0.139 (+0.14%)	0.138 (0.00%)
ltc-ltc	0.136	0.138 (+0.88%)	0.138 (+0.88%)	0.138 (+0.88%)
lnc-ltc	0.107	0.108 (+0.65%)	0.109 (+1.49%)	0.107 (+1.31%)
lnc-lnc	0.072	0.074 (+2.35%)	0.073 (+1.38%)	0.073 (+1.52%)
anc-ltc	0.082	0.084 (+2.31%)	0.087 (+5.59%)	0.084 (+2.79%)
nnn-nnn	0.071	0.072 (+0.56%)	0.072 (+0.42%)	0.070 (-0.98%)
bnn-bnn	0.096	0.100 (+4.18%)	0.101 (+5.02%)	0.099 (+3.56%)

Table 4: Average precision on the WT2g test-collection

Because several methods have been suggested to use hyperlinks for different purposes, there is a real need for a formal framework to allow analysis and comparisons between them. We have shown one possible framework here based on propositional logic for translating intuitions into knowledge. With the use of PAS, we have gained some insights on those intuitions: for example, the similarity between the computation of popularity measures and improving an initial ranking have been shown, and we have seen how some heuristic aspects of Kleinberg's algorithm could be described.

References

[Bh98] K. Bharat and M. Henzinger. Improved algorithms for topic distillation in hyperlinked environments. In *Proc. of the Int. ACM-SIGIR Conf.*, pages 104–111, 1998.

[BL98] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. of the World Wide Web Conf.*, pages 107–117, 1998.

[CdBD99] S. Chakrabati, M. Van der Berg, and B. Dom. Focused crawling: A new approach to topic specific resource discovery. In *Proc. of the World Wide Web Conf.*, pages 545–567, 1999.

[Hei89] K.D. Heidtmann. Smaller sums of disjoint products by subproduct inversion. *IEEE Transactions on Reliability*, 38(3):305–311, 1989.

[HKL99] R. Haenni, J. Kohlas, and N. Lehmann. Probabilistic argumentation systems. Technical Report 99-09, Institute of Informatics, University of Fribourg (Switzerland), 1999.

[Kle98] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc. of the 9th Symposium on Discrete Algorithms*, pages 668–677, 1998.

[LG99] S. Lawrence and C.L. Giles. Accessibility of information on the web. *Nature*, 400(8):107–109, 1999.

[Mar97] M. Marchiori. The quest for correct information on the web: Hyper search engines. In *Proc. of the World Wide Web Conf.*, 1997.

[otCP99] Members of the Clever Project. Hypersearching the web. *Scientific American*, 1999.

[Pic98] J. Picard. Modeling and combining evidence provided by document relationships using probabilistic argumentation systems. In *Proc. of the Int. ACM-SIGIR Conf.*, pages 182–189, 1998.

[Sav94] J. Savoy. A learning scheme for information retrieval in hypertext. *Information Processing & Management*, 30(4):513–533, 1994.

[Sav97] J. Savoy. Ranking schemes in hybrid Boolean systems: A new approach. *Journal of the American Society for Information Science*, 48(3):235–253, 1997.

[SP01] J. Savoy and J. Picard. Retrieval effectiveness on the web. *Information Processing & Management*, 37(4):543–569, 2001.

Towards Scalable Scoring for Preference-based Item Recommendation

Markus Stolze & Walid Rjaibi
IBM Research Division - Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{mrs, rja}@zurich.ibm.com

Abstract

Preference-based item recommendation is an important technique employed by online product catalogs for recommending items to buyers. Whereas the basic mathematical mechanisms used for computing value functions from stated preferences are relatively simple, developers of online catalogs need flexible formalisms that support the description of a wide range of value functions and map to scalable implementations for performing the required filtering and evaluation operations. This paper introduces an XML language for describing simple value functions that allow emulating the behavior of commercial preference-based item recommendation applications. We also discuss how the required scoring operations can be implemented on top of a commercial RDBMS, and present directions for future research.

1 Introduction

Online product catalogs are the Internet equivalent to the paper catalogs used by mail-order retailers and component suppliers to present their products to individual customers in business-to-business contexts. Similar to their paper counterparts, they present information about product features and possible uses. In addition to this, online catalogs can provide up-to-date information about availability and prices computed for the individual customer. A second advantage of online catalogs is that they can provide buyer decision support functionality and give recommendations for items in the catalog.

First-generation online catalogs only provide static HTLM pages arranged in a hierarchy. Here terminal pages describe individual products, and intermediary category pages provide the navigational links to lower-level product category pages and terminal product pages. First-generation online catalogs do not provide any item recommendation or decision support. On the contrary, if a buyer needs to compare a group of items that are not directly listed in a predefined category page, then the buyer has to invest a considerable amount of work to first identify all the relevant product pages, then print the pages or manually extract the relevant information into a single document, and finally compare the information to determine the best fit.

The inclusion of applications for feature-based item retrieval into online catalogs helps reduce this problem. These applications facilitate the task of identifying the relevant products. They provide buyers with a product-class specific form that lets them specify requirements regarding the item features. Thus, a user looking for a PC can specify, for example, that she is interested in buying a PC with a Pentium II processor and at least 10 GB

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

of hard-disk space. When receiving the query, the retrieval application scans the product database and returns a list of products that match the request. Some online catalogs also added side-by-side tables that list features of user-selected items in a single comparison table – thus relieving users from having to compile this information manually onto a single page. However, feature-based item retrieval is only of limited help if the number of products that meet the buyer's base requirements is large and if sorting the list according to a single item feature (such as price) is not sufficient because the buyers soft preferences consider more than a single attribute [11]. Thus, a buyer who receives a long list of PCs that all match her base requirements has to inspect each item to determine which of these computers best matches her soft preferences. An alternative to manual inspection is to go back to the original query and enter more restrictive requirements. While being faster, such a strategy can lead to the exclusion of items that would be best fits [9].

Applications for preference-based item recommendation have been developed to address this problem. In addition to allowing buyers to specify their hard requirements, they also let buyers express their "soft" preferences. The information about the requirements and preferences is then used to compile a value function that is used to evaluate and sort items. A number of commercial applications for preference-based item recommendation exist. In the next section we discuss the user experience these systems provide. We then present a simple XML representation scheme that allows specifying the range of value functions that need to be constructed by the recommendation applications discussed. We also introduce an XML-based rule language that supports the explicit representation of how user-provided information is mapped to modifications of the basic value function. Finally we discuss issues related to the scalability of the scoring mechanisms that perform the preference-based item evaluation.

2 Commercial Applications for Preference-based Item Recommendation

Commercial applications for preference-based item recommendation such as PersonaLogic (personaLogic.com), Active Buyer Guide (www.activeBuyersGuide.com), and the PuchaseSource system developed by Frictionless (www.frictionless.com) all work in three phases. After an initial phase of preference elicitation, they use the elicited information to construct the value function. This function is then used to evaluate and sort the items, and finally display the resulting list to the user. Below we discuss each of these phases in more detail.

2.1 Preference Elicitation

The basic mechanism for preference elicitation is exemplified by PersonaLogic. Here preference elicitation is performed by a sequence of dynamic web pages. Often the elicitation starts with a page prompting users for some high-level profile information. For example, the PersonaLogic recommendation application for computers[1] provides an initial dialog that elicites how buyers intend to use the computer. Buyers are asked how often they intend to use the computer for word processing, games, desktop publishing, communication, and education.

From these initial inputs, the PersonaLogic applications derive recommendations and default settings for some of the feature-related requirements and preferences. For example, from the fact that a buyer wants to play games frequently the PersonaLogic computer recommendation application derives the requirement for a fast processor.

In the next elicitation phase (which often spans multiple screens) all user preferences and requirements relating to item features are elicited. Thus, the PersonaLogic computer recommendation application lets buyers enter information such as the minimum memory requirements and the importance of having an integrated DVD drive.

[1]This application, which was available until late 2000, currently (August 2001) is no longer accessible. PersonaLogic was acquired by AOL in mid 2000. However, at AOL a number of PersonaLogic applications still are accessible; for example a dog recommendation application, a pet recommendation application, and a career recommendation application can be accessed at www.aol.personalogic.com/?product=dogs,aolcom, www.aol.personalogic.com/?product=pets,aolcom, and www.aol.personalogic.com/?product=career,aolcom respectively. We have started an archive of screen-shots of such disappearing recommendation applications at the following URL: www.zurich.ibm.com/mrs/recArchive/

Finally, in the last elicitation step, the relative importance of the different preferences is elicited. Thus, the PersonaLogic computer recommendation application elicits the relative importance of having a bigger hard-drive, a faster processor, more RAM, a lower price, more multimedia features, and being from a preferred manufacturer. Similar strategies for preference elicitation are also used by the Frictionless recommendation component. Active Buyers Guide extends this basic preference elicitation procedure by providing additional help in setting the relative importance of preferences. Instead of asking buyers to explicitly provide information about the relative importance of preferences, buyers are presented with a sequence of pairwise comparisons of hypothetical items that only differ in two features. The buyer is then asked to provide information whether, and to which degree, she prefers one of the items.

Allowing buyers to specify base requirements and preferences is consistent with the disjunctive model of decision making observed in the marketing literature [6] that assumes consumers use some base requirements to eliminate purchase alternatives and an additive multi-attribute evaluation function to evaluate the remaining alternatives.

2.2 Value Function Construction

We were not able to find any first-hand public information that describes how commercial applications construct the value function from the elicited preference information. However, from the information elicited and the evaluation that items receive, it seems possible to emulate the evaluation behavior of these applications with a scoring mechanism that interprets a filtering expression and a simple Multi-Attribute Utility Theory (MAUT) value function [5]. The general form of a simple MAUT function that combines the evaluations of individual features of a given item I as a weighted sum is $U(I) = \sum_j W_j \cdot U_j(F_j(I))$, where $U_j(F_j(I))$ is the evaluation of feature j of item I and W_j is the weight assigned to this feature. In such a framework the elicited requirements are used to compile the filtering expression. The elicited buyer preferences determine how item features are evaluated, i.e. how values of feature j (returned by $F_j(I)$) are mapped to an evaluation between 0 and 1 by the evaluation functions $U_j(F_j(I))$. The elicited buyer preferences also determine the relative weight of feature evaluations (i.e. W_j).

2.3 Result Presentation

After the filter expression and the valuation function have been constructed, they are applied to the set of items by the scoring mechanism. The scoring mechanism inspects the feature values of each item. It determines whether the item matches any of the filtering clauses, determines the evaluation of each of the item features, and computes the overall evaluation as the weighted sum of the feature evaluations. After this operation has been performed for each item, the filtered list of evaluated items is displayed to the user. All the preference-based recommendation applications supply buyers with a list of items sorted by their evaluation. However, the applications differ in the amount of information they provide to users for determining the fit of the item with the preferences stated. PersonaLogic provides the least support in that it hides the absolute quality of fit by always assigning the best fitting item an evaluation of 100% – which might be surprising to users that find 100% items lacking with respect to some important preferences. Active Buyers Guide, on the other hand, provides an absolute score that is less than 100% for the best item if it does not completely match the stated buyer preferences. Moreover, for each item it lists the values of features the buyer has expressed special interest in. It also highlights some special plus and minus points about each of the items. This facilitates a rapid inspection of the main item features. The Frictionless PurchaseSource system provides users with the most detailed information on the fit of items to the stated requirements and preferences[2]: For each item the user can see a summary page that lists graphically how well the item satisfies each of the stated requirements and preferences.

[2]Note that the Frictionless PurchaseSource application currently (August, 2001) is no longer available. We hope to include screen-shots also from this application in our recommendation applications archive.

2.4 Preference Revision

After inspecting the list of evaluated items, users have the option to return to the preference elicitation stage of the dialog, and revise some of their stated requirements and preferences. However, none of the recommendation applications discussed here support a close integration of item inspection and preference revision, as it is possible in some research systems [11]. They also do not support critiquing of returned items and their feature values as suggested for systems performing evaluation-oriented provisioning of product information [4].

3 An XML Representation for Simple Value Functions

In the preceding section, we argued that the evaluation behavior of commercial applications for preference-based item recommendation can be emulated with scoring mechanism that evaluates a database of items according to a filtering expression and a simple MAUT value function. We have devised an XML representation schema that allows such filtering expressions and simple MAUT value functions to be represented in an integrated fashion. An example XML representation of a simple value function is shown in Figure 1.

```
<criteria rootId="root">
        <criterion id="Price" importance="50" must="false" attribute="Price">
            <numEval valUnit="$" val1="0" eval1="1" val2="5600" eval2="0"/>
        </criterion>
        <criterion id="Speed" importance="25" must="false" attribute="Speed">
             <numEval valUnit="mhz" val1="300" eval1="0" val2="1600" eval2="1"/>
        </criterion>
        <criterion id="RAM" importance="25" must="false" attribute="RAM">
            <numEval valUnit="MB" val1="32" eval1="0" val2="512" eval2="1"/>
        </criterion>
       <criterion id="CDWriter" importance="5" must="false" attribute="CDWriter">
            <symEval sym="12X">1</symEval>
            <symEval sym="8X">.75</symEval>
            <symEval sym="4X">.5</symEval>
            <symEval sym="2X">.25</symEval>
            <symEval sym="none">0</symEval>
        </criterion>
        <criterion id="HD Size" importance="25" must="false" attribute="HD">
            <numEval valUnit="GB" val1="0" eval1="0" val2="50" eval2="1"/>
        </criterion>
        <criterion id="root" importance="100">
                    <subCriteria>
                            <Criterion  ref="Price"/>
                            <Criterion  ref="Speed"/>
                            <Criterion ref="RAM"/>
                            <Criterion  ref="CDWriter"/>
                            <Criterion  ref="HD Size"/>
                    </subCriteria>
        </criterion>
</criteria>
```

Figure 1: **Example of XML-based specification of a simple MAUT value function**

Using this representation language, a MAUT value function is represented as a tree of evaluation criteria. A criterion is either a terminal or a combination criterion. Terminal criteria (such as Price and CDWriter in Figure 1) have a set of evaluation specifications as sub-elements that specify how numeric and symbolic values of features should be evaluated. Numeric evaluation specifications (i.e., the *numEval* sub-elements of criteria) allow specifying how the numeric values of the corresponding item feature should be mapped to an evaluation between 0 and 1. In the *numEval* specification the “val1” value specifies the starting point of a linear function and “val2” the end-point; the “eval1” value specifies the evaluation at the starting point “val1” and “eval2” the evaluation at the end of the interval. The *symEval* sub-elements of criteria specify how symbolic values are mapped to evaluations. Thus, for example, in the criterion CDWriter in Figure 1 the evaluation specification $< symEvalval = "8X" > .75 < /symEval >$ implies that items with a value of “8X” for their CDWriter feature will receive an evaluation of 0.75 for the CDWriter criterion. In addition to the *numEval* and *symEval* evaluation specifications *numReject* and *symReject* elements can also be included. These elements (not shown in Figure 1) allow hard requirements to be specified for items. Thus, items that match these reject clauses will not be included in the result set. The scoring mechanism interpreting the XML specification has to compute the evaluation and reject values independently. Combination criteria such as “root” combine the evaluation and reject values of their sub-criteria into single evaluation and reject values. Scoring mechanisms interpreting the XML specifications compute the evaluation as the weighted sum of the sub-node evaluations using the “importance” attribute of criteria as the weighting factor. Reject information is propagated in such a way that if an item is rejected according to one sub-criterion that has the attribute “must” set to true, then the item is also rejected according to the super-criterion. Criteria where “must” is set to false only flag items as unsatisfactory without rejecting them.

4 Explicit Representation of Value Function Adaptation Rules

Value function construction from user input can be done by a program that interprets the user input and generates the value function specification. However, we believe that it is advantageous to make explicit the relationship between user input and configuration of the value function. This way the relationship remains easier to understand and easier to maintain. We have therefore created an XML-based rule language that allows specifying how the base value function should be adapted based on user input. Figure 2 shows an example of a set of modification rules. The proposed language for criteria modification includes actions for setting, increasing, and decreasing criteria importance. It also includes means to add reject conditions to criteria. In our experiments with this language in the context of utility-based generation of sales interviews [10], we found it sufficiently expressive to represent how a basic value function is to be adapted to reflect the preferences of the individual buyer. In particular, in our examples we did not encounter a need to dynamically change the hierarchy of criteria that make up the value function.

```
<rules>
    <rule ID="games">
        <if><answer question="gamer">yes</answer></if>
        <then><incImportancePerc criterion="Graphics Card">50</incImportancePerc></then></rule>
    <rule ID="price1">
        <if><answer question="price">over $300</answer></if>
        <then><setImportance criterion="Price">0</setImportance></then></rule>
    <rule ID="price2">
        <if><answer question="price">up to $1500</answer></if>
        <then><addNumValueReject criterion="Price" from="1501" to="5600"/></then></rule>
</rules>
```

Figure 2: **Example of XML representation of value function adaptation rules**

5 Towards Scalable Scoring for Preference-based Item Recommendation

As the set of input items to which the scoring mechanism has to apply the evaluation function becomes larger, the performance of the scoring mechanism becomes increasingly critical. Users are not willing to wait too long before they can access the list of recommended items being generated. Stand-alone scoring mechanisms are likely to opt for implementing sophisticated algorithms and techniques to address this performance issue. Given that the set of input items is likely to be stored in a relational database a question arises. Can the scoring functionality (or at least some part of it) be delegated to the database system (DBMS)? We argue that the answer to this question is an unequivocal yes. Most commercial database systems such as IBM’s DB2 UDB allow users to write their own extensions to SQL. In DB2 UDB a user-defined function (UDF) can be used to write extensions to SQL. By expressing the evaluation function as a UDF (or a set of UDFs), the scoring mechanism can be expressed as an SQL query and therefore take full advantage of the DBMS’ query processing engine.

Below we give a simple example to illustrate how UDFs can be used to help express the scoring functionality as an SQL query. Suppose that the Price, Speed, RAM, CDWriter, and HDSize information are represented in a database table called “Product” and that a user is interested in obtaining a set of PC recommendations based on the preferences specified in Figure 1.

To express the evaluation function for the "Price" attribute as a UDF two simple steps are required. The first is to write the actual code that implements the evaluation function. In an example the code is written in JAVA but other programming languages such as C and C++ are also supported. The second step is to register the UDF to the database and associate the JAVA code (written in the first step) with it. This is easily accomplished using a CREATE FUNCTION SQL statement [14]. When the UDF is invoked by the database system, the code written in step 1 is executed. Figure 3 lists the piece of JAVA code that implements the evaluation function of the Price attribute according to the preferences set in Figure 1.

```
import COM.ibm.db2.app.*;
class EvaluationUDF extends UDF
{
    public double PriceEval (double price, double val1, double val2)
    {
        /*
        * Get the slope and end point of the line segment that includes the
        * 2 points (val1, 1) and (val2, 0)
        * The line segment equation is y = a * x + b;
        */
        double a = 1 / (val1 − val2);
        double b = −val2 / (val1 − val2);
        /*
        * Return the evaluation for this price
        */
        double eval = a * price + b;
        return (eval);
    }
}
```

Figure 3: **Evaluation function implementation as a JAVA UDF for the Price attribute**

The evaluation functions for the Speed, RAM, CDWriter, and HDSize attributes can be expressed as UDFs in the same way as the Price attribute. Let SpeedEval, RAMEval, CDWriterEval, and HDSizeEval denote the UDFs for the Speed, RAM, CDWriter, and HDSize attributes respectively. Having expressed the evaluation functions as UDFs, the scoring functionality can be easily expressed using the following SQL query :

*SELECT Product_ID, Price, Speed, RAM, CDWriter, HDSize,(50 * PriceEval (Price, 0, 5600) + 25 * SpeedEval(Speed, 1600, 300) + 25 * RAMEval(RAM, 512,32) + 5 * CDWriterEval(CDWriter) + 25 * HDSizeEval(HDSize)) AS Score*
FROM product
ORDER BY Score

The above query shows that the score of each product is computed as a weighted sum aggregation of the Price, Speed, RAM, CDWriter and HDSize attributes. The weight of each attribute is as specified by the "Importance" field in Figure 1. Rather than having the weights hard-coded in the above query, a better alternative is to store the weights in a separate database table (called ATTRIBUTE_WEIGHT for example) and modify the above query to read these weights from the table ATTRIBUTE_WEIGHT. Having the weights stored in a separate table provides more flexibility because users are likely to modify these weights while searching for the product that suits them best. The ATTRIBUTE_WEIGHT table can be implemented using just two columns: A string to indicate the fully qualified attribute name and an integer to represent the weight of that attribute. The following UDF (written in SQL this time) can be used to return the weight of a given attribute :

CREATE FUNCTION weight (attribute-name char(30)) RETURNS integer
LANGUAGE SQL READS SQL DATA NO EXTERNAL ACTION DETERMINISTIC
RETURN SELECT weight FROM attribute_weight WHERE attribute_weigth.attribute-name = weight.attribute-name

The SQL query that implements the scoring functionality can then be modified as follows to use this new UDF :

*SELECT Product_ID, Price, Speed, RAM, CDWriter, HDSize,(weight("product.Price") * PriceEval (Price, 0, 5600) + weight("product.Speed") * SpeedEval(Speed, 1600, 300) + weight("product.RAM") * RAMEval(RAM, 512,32) + weight("product.CDWriter") * CDWriterEval(CDWriter) + weight("product.HDSize") * HDSizeEval(HDSize)) AS Score*
FROM product
ORDER BY Score

By expressing the evaluation function as a UDF (or a set of UDFs) the scoring mechanism can be expressed as an SQL query and therefore take full advantage of the DBMS' query processing engine. The performance benefit of using this approach can be significant when the set of input items to which the scoring mechanism needs to apply the evaluation function becomes large. However, the benefit of improved performance has to be weighted against the actual effort needed to create these UDFs, register them, and write the appropriate SQL statements to call them. Therefore, we are currently investigating ways to automatically perform the described compilation steps from the XML criteria specification given.

Also, other authors have developed schema for preference-based item scoring and discussed how to map them onto relational database queries. In [1], Agrawal and Wimmers proposed a mechanism for representing and combining preferences that is based on a relational representation. Here preferences are described in database tables that mirror the structure of item tables and have an additional "score" column containing the evaluation of items. A wildcard symbol can be used instead of an item feature value to facilitate a concise description of preferences that are independent of certain item feature values. A special symbol (instead of a score between 0 and 1) is used to indicate the rejection of items. Combination of preferences is done through a combination function that is implemented as external function. However, in our opinion, the proposed mechanism for preference representation is not sufficiently expressive to handle the range of value functions used by preference-based recommendation applications. In particular, it seems unclear how numeric evaluation rules should be represented in an efficient manner. Also, it seems that in the proposed scheme an item either receives an evaluation or is rejected. However, most recommendation applications compute rejection and evaluation information independently from each other. The work of Hristidis et al. [7] extends the work of Agrawal and Wimmers [1] by illustrating how materialized views can be used to increase the efficiency of DB-based scoring. Although the idea of using materialized views is certainly attractive, their preference representation schema suffers from the same limitations of expressiveness as that of Agrawal and Wimmers [1].

6 Beyond Simple Value Functions

In the preceding section we discussed how the proposed XML language for simple MAUT value functions could be mapped to scalable scoring mechanism using relational database techniques. We also argued that other proposals for implementing preference functions on top of relational databases are not sufficiently expressive to cover the range of value functions needed by applications for preference-based item recommendation in online product catalogs. However, the introduced XML preference function representation language is also limited in its expressiveness. Although we believe that it covers the value functions that most preference-based item recommendation applications need to construct, we have already experienced practical situations that required custom extensions. For example, extensions are needed to allow preferences regarding values other than numbers and symbols (e.g. dates) to be expressed. Extensions are also needed to deal flexibly with situations in which items have multiple feature values. Finally, the simple scoring scheme also does not cover situations that require items to be classified into more than the two standard classes of "not rejected" and "rejected" items. We have started to experiment with extended value function representations to find the best way to combine easy specification of simple scoring functions with expressive means for extending and customizing the base scoring functionality.

Eventually such extensible value function representations and the associated scoring mechanisms should also be able to deal with more complex value functions that have been explored by recent research on item representation and evaluation. For example, in [8] Lukacs studied mechanism for scoring items with under-specified item values (such as a travel time that "might be between 20 and 30 minutes"). FIPA [13] and Willemot et al. [12] propose to describe spaces of configurable items as constraint satisfaction problems. In addition, Chajewska et al. [2], and Chin and Porage [3] explored mechanisms for dealing with situations in which no reliable information is available regarding the preferences (i.e. weights and feature evaluation functions). The development of description languages that support easy and expressive specification of such complex value functions remains future work. Increasing the scalability of a scoring mechanism by performing the scoring operations within the framework of a relational database system is likely to remain a valid approach to increase the scalability of scoring mechanisms even for these more complex situations. Experiments and empirical evaluations will be needed to further explore this space.

References

[1] R. Agrawal and E. Wimmers. A Framework for Expressing and Combining Preferences. In *Proc. of the ACM-SIGMOD 2000 Conference on Management of Data*, pages 297–396, Dallas TX, May 2000.

[2] U. Chajewska, D. Koller and R. Parr. Making Rational Decisions using Adaptive Utility Elicitation. In *Proc. of the 17th National Conference on Artificial Intelligence AAAI-1999*, pages 363–369, Austin TX, August 2000.

[3] D. N. Chin and A. Porage. Acquiring User Preferences for Product Customization. In *Proc. of the 8th International Conference on User Modeling UM-2001*, pages 95–104, Sonthofen Germany, July 2001.

[4] A. Jameson, R. Schafer, J. Simons and T. Weis. Adaptive Provision of Evaluation-Oriented Information : Tasks and Techniques. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1886–1893, Montreal, August 1995.

[5] R. T. Clemen. Making Hard Decisions : An Introduction to Decision Analysis. *Wadsworth Publishing Company*, Belmont CA, 1996.

[6] P. E. Green and Y. Wind. Multiattribute Decisions in Marketing : A Measurement Approach. *Dryden*, Hinsdale IL, 1975.

[7] V. Hristidis, N. Koudas and Y. Papakonstantinou. PREFER : A System for the Efficient Execution of Multi-Parametric Ranked Queries. *ACM SIGMOD*, pages 259–270, 2001.

[8] G. Lukacs. Decision Support under Imperfections in Electronic Commerce. *DEXA 2000 Second International Workshop on Logical and Uncertainty Models for Information Systems, Greenwich UK, IEEE Computer Society Press*, pages 538–542, September 2000.

[9] P. Steiger and M. Stolze. Effective Product Selection in Electronic Catalogs. In *Proc. of the Human Factors in Computing Systems*, pages 291–292, Atlanta GA, 2000.

[10] M. Stolze and M. Ströbel. Utility-based Decision Tree Optimization: A Framework for Adaptive Interviewing. In *Proc. of the 7th International Conference on User Modeling UM-2001*, Sonthofen, Germany, July 2001.

[11] M. Stolze. Soft Navigation in Electronic Product Catalogs. *International Journal on Digital Libraries*, 3(1):60–66, 2000.

[12] S. Willmott, M. Calisti, B. Faltings, S. Macho-Gonzalez, O. Belakhdar and M. Torrens. CCL: Expressions of Choice in Agent Communication. In *Proc. of the 4th International Conference on MultiAgent Systems ICMAS-2000*, Boston MA, July 2000.

[13] FIPA. FIPA CCL Content Language Specification. *http://www.fipa.org/specs/fipa00009/XC00009A.html*, 2000.

[14] D. Chamberlin. Using the New DB2: IBM's Object-Relational Database System. *Morgan Kaufmann Publishers*, 1996.

Technical Committee on Data Engineering
Election of Chair for 2002-2003

The Chair of the IEEE Computer Society Technical Committee on Data Engineering (TCDE) is elected for a two-year period. The mandate of the current Chair, Betty J. Salzberg, terminates at the end of 2001. Hence is time to elect a Chair for the period January 2002 to December 2003. Please vote using the ballot on the next page.

The Nominating Committee, consisting of Masaru Kitsuregawa, Paul Larson, and Betty Salzberg, is nominating Erich Neuhold for the position as Chair of TCDE. Erich's position statement and a short biography are included below. The Committee invited nominations from members of the TCDE but has received no other nominations.

Paul Larson, Nominating Committee Chair

ERICH NEUHOLD
Professor, Department of Computer Science
Darmstadt University of Technology
Wilhelminenstrasse 7
D-64283 Darmstadt
Germany
neuhold@darmstadt.gmd.de

Position Statement

If elected I hope to continue the excellent work of Betty Salzberg.

The Technical Committee on Data Engineering publishes a newsletter, the Data Engineering Bulletin and sponsors the ICDE (International Conference on Data Engineering) and the associated workshop RIDE (Research Issues in Databases). In addition it will continue to work with SIGMOD to make the ICDE proceedings and other relevant publications available on CD-ROM and the SIGMOD archive.

I will continue Betty's work to increase the presence and active participation of industrial members of ICDE and I will also work on establishing more joint events with data base societies and action groups in other countries. Database research has had a tremendous influence on DB technology and has gained continuously by feedback from the practitioners. Such cooperation has to be strengthened wherever possible. One of the possibilities here is to explore ways to broaden ICDE by either industry oriented tracks or associated workshops that gear to the practitioners and users and feed back their problems and experiences to the research community. I will try to achieve this goal by working closely together with the ICDE Steering Committee.

Biography

Erich Neuhold is Professor of Computer Science at Darmstadt University of Technology and Director of the GMD Institute for Integrated Publication and Information Systems (IPSI) also in Darmstadt, Germany. He has been Professor at the University of Stuttgart and the Technical University of Vienna and he has also worked in research and management positions for IBM and Hewlett Packard both in Europe and the USA. His areas of expertise in databases include distributed databases, object-oriented databases, databases for the internet (e.g. semi-structured data and XML), information retrieval, information visualization and their applications in digital libraries, cultural heritage and e-commerce.

He has published four books and about 180 papers. His work has appeared, among others, in the VLDB Journal, Information Systems, Acta Informatica and in many conferences as, for example, VLDB, ICDE, MMDB, ADL and DL. He has served in all capacities on many conference committees and was PC Chair and General Chair of ICDE.

He is the current Vice Chair of the IEEE TCDE and the Chair of the ICDE Steering Committee and also a member of the JCDL at ECDL Steering Committees. He is a Senior Member of IEEE.

ELECTION BALLOT

IEEE COMPUTER SOCIETY

TECHNICAL COMMITTEE ON
DATA ENGINEERING

The Technical Committee on Data Engineering (TCDE) is holding an election for Chair. The term of the current chair, Betty J. Salzberg, expires at the end of 2001. Please email, mail or fax in your vote.

BALLOT FOR ELECTION OF CHAIR
Term: (January, 2002 - December, 2003)

Please vote for one candidate.

☐ Erich Neuhold

☐ ___________
(write in)

Your Signature:____

Your Name:____

IEEE CS Membership No.:________________________________
(Note: You must provide your member number. Only TCDE members who are Computer Society members are eligible to vote.)

Please email, mail or fax the ballot to arrive by November 30, 2001 to:

s.wagner@computer.org
Fax: +1-202-728-0884

IEEE Computer Society
Attn: Stacy Wagner
1730 Massachusetts Avenue, NW
Washington, DC 20036-1992

RETURN BY November 30, 2001

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398