

Reactive Ranking for Cooperative Databases

Berthier A. Ribeiro-Neto
Guilherme T. Assis
Computer Science Department
Federal University of Minas Gerais
Brazil
{berthier,tavares}@dcc.ufmg.br

Abstract

A cooperative database allows the user to specify approximate or vague query conditions. A vague query requires the database system to rank the retrieved answers according to their similarity to that query. In this paper we discuss a reactive ranking strategy for cooperative databases. The reactivity in our approach is provided by an interactive mechanism which allows the user to select the answers of his preference and then uses these answers to tune the original user query. The ranking formula we propose is derived from a parallel with the AutoClass II system — a bayesian probabilistic classification engine. The reactive mechanism we propose is derived from a parallel with the technique of relevance feedback widely used in the area of information retrieval. We implemented our cooperative and reactive ranking approach in a Web browser interface and did some experimentation. Our experiments provide initial evidence that our approach might help the user to solve query tasks more precisely and in shorter time.

1. Introduction

Conventional database systems deal with exact queries. An exact query specifies constraints that have to be met precisely. For instance, if a user asks for a *suspense* film playing in Hollywood and there is no such film in the database, the user receives a null answer. However, it might be the case that there is a suspense film playing in a theater located in Beverly Hills, which is only 8 miles away. Conventional systems have no mechanism to even specify an approximate constraint such as [location *near-to* Hollywood].

Since conventional database systems cannot handle approximate queries directly, these have to be emulated through a sequence of exact queries. But then, the user has to direct the system through a possibly long interaction to

satisfy his needs. If the user is not aware of close alternatives, then even this solution is infeasible [11].

A cooperative query answering system allows the user to specify a query containing vague conditions. For instance, $q = [film_category \sim suspense] \wedge [location \sim Hollywood]$. Vague queries require the system to find tuples that approximate the query conditions and rank (i.e., order) these tuples according to their *semantic distance* to the query.

Fuhr [7] uses a *relevance description* function to model semantic distances. However, his approach does *not* allow Boolean operators in the query formulation because a query involving *conjuncts* and *disjuncts* would lead to complex probabilistic formulas with parameters that could not be accurately estimated in most cases. Rabitti and Savino [12] propose to sum up the contributions of an object to the different portions of an *or* condition. An *and* operation is satisfied or not satisfied depending on whether the object approximates all conjunctive conditions better than a minimum similarity threshold. Therefore, an object O_1 which barely satisfies the thresholds and another object O_2 which matches all conditions in the *and* operation exactly would receive the same ranking. This solution is inappropriate for large databases because many tuples would receive the same ranking. Motro [11] proposes to combine metric distances in an *and* operation by a weighted square sum of the distances for each constraint. However, the meaning of computing the overall distance for an *and* condition by a weighted sum is not clear.

Our approach relies on a parallel with the AutoClass II system [4] which is a well known program to induce classes in a database [4]. From this parallel, we derive a ranking strategy based on bayesian theory which seems appropriate. Furthermore, we go beyond previous approaches and enrich our ranking strategy with the capability of *reactiveness*. Such capability allows the user to select the answers of his preference which are then used to tune the original user query accordingly.

To validate our ideas, we implemented a cooperative and reactive database interface using a Web browser. This interface was then compared to a relational database query interface (with SQL as the query language) through a set of specific query tasks. Our experiments provide evidence which induce us to believe that our cooperative interface might help the user solve query tasks more precisely and in shorter time.

2. AutoClass II

AutoClass II [4] is a program to induce classes in a database. Based on Bayes theorem, the program determines automatically class probability descriptions and the probability of class membership for each database object.

In AutoClass II, a class C_l is described by a distribution function $P(x_i|x_i \in C_l, \vec{\theta}_l)$ which yields the probability of observing object x_i given that x_i belongs to the class C_l . The parameter $\vec{\theta}_l$ is a vector which describes a set of distribution functions associated to the attributes of the class C_l . For instance, assume that the domain values of the k th attribute of class C_l are normally distributed. Then, this k th attribute is characterized by a parameter $\theta_{l,k}, \theta_{l,k} \in \vec{\theta}_l$, such that $\theta_{l,k} = (\mu, \sigma)$ where μ and σ stand for the median and the variance of the normal distribution.

Consider a database composed of N objects x_i where each object is described by r attribute variables $x_{k,i}, k \in \{1, 2, \dots, r\}$. These attribute variables can be discrete or continuous. AutoClass II assumes that attribute variables are independent among themselves which leads to the following equation.

$$P(x_i|x_i \in C_l, \vec{\theta}_l) = \prod_{k=1}^r P(x_{k,i}|x_i \in C_l, \theta_{l,k}) \quad (1)$$

$P(x_{k,i}|x_i \in C_l, \theta_{l,k})$ is the *prior* probability of observing the value of the k th attribute (according to object x_i) given that x_i is a member of class C_l .

The AutoClass II program assumes that the prior probability distributions associated to class attributes are normal distributions. The reasons are twofold. First, normal distributions have a large spectrum of applications and are useful in situations where the real distribution is unknown. Second, prior distributions have an arbitrary nature and are used only as the starting point of the Bayesian inference process [4, 10]. As evidence on the real distribution of an attribute domain is collected, the prior distribution can be adjusted accordingly.

3. Ranking of Approximate Answers using AutoClass II

In this section we introduce a strategy for ranking approximate answers in a relational database system. Our ranking associates a numeric value to each approximate answer. Such numeric value can be viewed as an estimate of the probability that the approximate answer is useful to the user (according to the user's query). To generate our ranking, we do a parallel with the class approach of the AutoClass II program [4]. Each user query in our approach is interpreted as a class in AutoClass II. Each approximate answer in our model is viewed as an object in AutoClass II. Furthermore, the probability that an approximate answer is relevant to a vague query (in our model) is taken as the likelihood of observing the object in the corresponding AutoClass II class.

3.1. Ranking Computation

Before proceeding, let us define some basic notation. We assume that the relational model and its terminology is known.

Definition 1 Let A_k be the name of the k th attribute of a given relation R and let D_k be the respective attribute domain. The attribute-value pair $x_{k,i}$ is a reference to the i th distinct attribute value in the domain D_k .

Definition 2 Let t be a relational tuple of relation R . Then, $t[k]$ is a reference to the value of the attribute A_k according to the tuple t .

Definition 3 Let q be a user query. Then, $q[k]$ is a reference to the value of the attribute A_k according to the user query q .

Since we allow the user to pose a *vague* query, our system must include some form of numerically quantifying proximity. This is accomplished with the notion of semantic distances. The *semantic distance* between two values $x_{k,1}$ and $x_{k,2}$ of attribute A_k is a quantitative estimate of *how close* is $x_{k,1}$ of $x_{k,2}$. Semantic distances are specified through *metric functions* [11] which are defined as follows.

Definition 4 A *metric function* $M : D_k \times D_k \mapsto R$ associates to each pair of values from the domain of attribute A_k a real number which quantifies the semantic distance between the values in the pair.

Notice that semantic distances are defined only between two values of a same attribute domain. For two attribute values in distinct domains, the semantic distance is assumed to be infinite.

In this work, we adopt two types of metric functions: *computational* and *tabular*. A metric is called *computational* if the semantic distances it returns are computed. A metric is called *tabular* if the semantic distances it returns are obtained from a previously defined table. Computational metrics are used with numeric attribute domains. Tabular metrics are used with non-numeric attribute domains.

Through an analogy with the AutoClass II model, we can define the degree of *similarity* of a tuple t to a given user query q (assumed for now as completely conjunctive) by

$$P(t|q) = \prod_{k=1}^r p(t[k]|q, \theta_{q,k}) \quad (2)$$

where $k = 1, 2, \dots, r$ are the attributes mentioned in the query q . The probability $p(t[k]|q, \theta_{q,k})$ quantifies the probability of observing the attribute-value $t[k]$ given the user query q and the semantic distances $\theta_{q,k}$ for the values in the k th attribute domain. The parameter $\theta_{q,k}$ is a random variable (associated to the instances of attribute A_k) which represents semantic distances. As in the AutoClass II model, we assume that such random variable is normally distributed if no additional information on its distribution is known¹.

Definition 5 Let $\theta_{q,k} : D_k, q[k], M \mapsto R$ be a function which associates to each instance $x_{k,i}$, $x_{k,i} \in D[k]$, a real number. This number is a normalized representation of the semantic distances between each attribute value $x_{k,i}$ and the attribute value $q[k]$ according to the metric M . The random variable $\theta_{q,k}$ is assumed to have a (prior) standard normal distribution which is used to compute the similarity between $x_{k,i}$ and $q[k]$ in the domain of $\theta_{q,k}$.

One should notice the analogy between our idea and the AutoClass II model. In our model, a *degree of similarity* between each tuple attribute value (i.e., t_k) and the respective query condition (i.e., $q[k]$) is computed (through equation 2) using a normally distributed random variable (i.e., $\theta_{q,k}$) which represents the semantic distance between them. These degrees of similarity for each attribute value are then combined to yield the similarity between the tuple t and the query q . In the AutoClass II model, a *degree of membership* of each object attribute value $x_{k,i}$ in the class q is computed. These degrees of membership for each object attribute value are then combined to yield a degree of membership of the object x_i in the class q .

For the above computations, the AutoClass II system assumes that the attribute values for x_i are normally distributed. In our approach the computation of degrees of

¹Prior probability functions are introduced conveniently (but in an arbitrary fashion) and constitute the main difference between Bayesian and classic statistics. Instead of arguing in favor of the adoption of prior probabilities, we refer the skeptical reader to Jaynes [10].

similarity is based on the random variable θ and not on the attribute values directly. Thus, the values of this random variable are the ones which are assumed to be normally distributed. Before proceeding, we simplify our notation and specify how to compute $\theta_{q,k}$.

Definition 6 For simplicity of notation, the random variable $\theta_{q,k}$ is referred to as θ whenever there is no risk of misinterpretation. Further, the value of $\theta_{q,k}$ for a given attribute value $x_{k,i}$ is referred to simply by $\theta(x_{k,i})$.

The values for the random variable $\theta_{q,k}$ can be derived directly from the metric M after proper normalization. In [11], computation of degrees of similarity are based directly on the metric functions M (i.e., there is no intermediate variable such as $\theta_{q,k}$). As a result, the normalization strategy is based on the *largest* semantic distance defined by M . We argue that such normalization is counter-intuitive because the user is not interested in large semantic distances. Instead, he pays closer attention to small semantic distances (which better approximate his query conditions). Since in our model we introduce the intermediate random variable $\theta_{q,k}$, we can adopt a normalization strategy based on the *smallest* semantic distance. This is accomplished as follows.

Definition 7 Let λ_k be a reference to the closest value (in the attribute domain D_k) to the attribute value $q[k]$ according to the metric function M . To the attribute value λ_k is associated a (previously defined) value for the random variable $\theta_{q,k}$ which, for simplicity, is referred to by $\theta(\lambda_k)$.

As a result of the definition of λ_k , $M(\lambda_k, q[k])$ is the smallest semantic distance which is greater than zero as defined by the metric M . We can now define how to compute $\theta_{q,k}$ in general as follows.

$$M(q[k], \lambda_k) \quad \text{---} \quad \theta(\lambda_k) \quad (3)$$

$$M(q[k], x_{k,i}) \quad \text{---} \quad \theta(x_{k,i}) \quad (4)$$

which yields

$$\theta(x_{k,i}) = \theta(\lambda_k) \times \frac{M(q[k], x_{k,i})}{M(q[k], \lambda_k)} \quad (5)$$

Notice that the smallest semantic distance is used for normalization. The value for $\theta(\lambda_k)$ is defined, for instance, from an inference based on a user indication. By reducing $\theta(\lambda_k)$ for the k th attribute domain, the user assigns higher weight to all the approximate answers in the attribute domain D_k as explained ahead. In this work, however, we do not explore this flexibility and adopt $\theta(\lambda_k) = 0.1$ for all attribute domains.

To compute the degree of similarity $P(t|q)$ defined by equation 2, we need to specify how to compute the probabilities $P(t[k]|q, \theta_{q,k})$ (which, for simplicity, we refer

to as $P(t[k]|q, \theta)$. The idea is to make high values of $P(t[k]|q, \theta_{q,k})$ reflect small semantic distances between $t[k]$ and $q[k]$ and small values of $P(t[k]|q, \theta_{q,k})$ reflect large semantic distances between $t[k]$ and $q[k]$. Consider the normal distribution in figure 1. Since $M_k(q[k], q[k]) = 0$, it

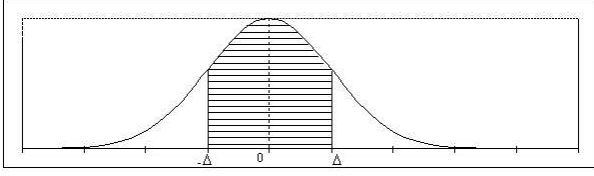


Figure 1. Standard normal distribution.

has to be $\theta(q[k]) = 0$ which means that $q[k]$ is mapped into the zero value in figure 1. Consider now a tuple attribute value $t[k]$ and define

$$\Delta = |\theta(t[k]) - \theta(q[k])| \quad (6)$$

$$= \theta(t[k]) \quad (7)$$

The parameter Δ reflects the semantic distance between $t[k]$ and $q[k]$ in the domain of the random variable θ . Smallest is the value of the parameter Δ , better is the approximation that $t[k]$ provides to $q[k]$. This simple observation, coupled with the fact that the variable θ is assumed to be normally distributed, leads naturally to defining the degree of similarity between $t[k]$ and $q[k]$ as the gray area in figure 1. Let $F(\theta)$ be the value of the cumulative distribution for θ . Then,

$$P(t[k]|q, \theta) = 1 - P(-\Delta \leq \theta \leq \Delta) \quad (8)$$

$$= 1 - (F(\Delta) - F(-\Delta)) \quad (9)$$

For the normal distribution, we have that $F(-k) = 1 - F(k)$. Thus,

$$P(t[k]|q, \theta) = 1 - (F(\Delta) - 1 + F(\Delta)) \quad (10)$$

$$= 2 - 2 \times F(\Delta) \quad (11)$$

That is,

$$P(t[k]|q, \theta) = 2 - 2 \times F(\theta(t[k])) \quad (12)$$

According to the standard normal distribution, $F(x > 3.99) = 1$. Thus, if $\theta(t[k])$ is larger than 3.99 then $P(t[k]|q, \theta) = 0$ and there is no similarity between $t[k]$ and $q[k]$.

In the above discussion, only purely conjunctive queries are considered. Let q_{abc} be a query involving conjunctions and disjunctions. For instance, assume $q_{abc} = (a + b)c = ac + bc$. To deal with more generic queries such as q_{abc} , we first transform the query in its normal disjunct form (an operation which can always be done). For the query q_{abc} , such transformation yields $q_{abc} = abc + \bar{a}bc + a\bar{b}c$. Thus, q_{abc} can

be viewed as a disjunction of three conjunctive components namely $q_1 = abc$, $q_2 = \bar{a}bc$, and $q_3 = a\bar{b}c$. According to our analogy with the AutoClass II model, each conjunctive component is treated as a class which allows us to write,

$$P(t|q_{abc}) = \sum_{j=1}^3 \pi_j \times P(t|q_j) \quad (13)$$

where π_j is the prior probability associated to each class q_j . Since there is no prior preference to any of the conjunctive components of a query, we adopt

$$\pi_j = \frac{1}{\text{number_of_classes}} \quad (14)$$

Finally, one has to consider that there might be references to non-numeric attributes in the user query for which no metric function is defined. In this case, as in [11], we adopt a degree of similarity *default* given by

$$P(t[k]|q, \theta) = \begin{cases} 1 & \text{if } q[k] = t[k] \\ 0.6 & \text{otherwise} \end{cases} \quad (15)$$

This completes the specification of our strategy for computing degrees of similarity. In the immediate following, we illustrate our ranking strategy with an example.

3.2. Example of Ranking Computation

Consider a small film database composed of a single relation as illustrated in figure 2. A film has a title, a director, a category, a country of origin, an year of production, and a duration in minutes. A tuple identification is also included to facilitate references to tuples. For simplicity, this database does not have multiple relations. However, our model works we more generic databases as well. For instance, in the presence of a query joining multiple relations, one has only to consider that our ranking strategy is applied to the single temporary relation which results from the join computation. For the database in figure 2, assume that tab-

Films						
Id	Title	Director	Category	Country	Year	Duration
T ₁	Nell	Michael Apted	drama	USA	1994	111
T ₂	Queen Margot	Patrice Chereau	drama	France	1994	162
T ₃	Risk	Deirdre Fishel	romance	USA	1994	90
T ₄	Darkness in Taling	Ilkka Jarvitalo	suspense	USA	1993	99
T ₅	Nomads	John McTiernan	horror	USA	1986	91
T ₆	Short Circuit II	Kenneth Johnson	comedy	USA	1988	110
T ₇	The Specialist	Luis Llosa	adventure	USA	1984	110
T ₈	Suspiros de Espana	José Luis Sanchez	comedy	Spain	1995	90

Figure 2. A small film database.

ular metric functions were defined for the attribute domains *category* and *country*. These metric functions are illustrated in figure 3. For numeric attributes such as *year* and *duration*, metric distances can be computed directly from the

Metric for category			Value 1	Value 2	Distance
Value 1	Value 2	Distance	USA	France	5
comedy	adventure	3	USA	Spain	9
comedy	drama	12	France	Spain	6
comedy	romance	10			
comedy	suspense	13			
comedy	horror	13			
adventure	drama	11			
adventure	romance	10			
adventure	suspense	7			
adventure	horror	7			
drama	romance	3			
drama	suspense	8			
drama	horror	11			
romance	suspense	11			
romance	horror	13			
suspense	horror	3			

Figure 3. Tabular metrics for the attributes *category* and *country*.

attribute values. For instance, in [11], the semantic distance between the values 90 and 110 for the attribute *duration* is computed as $\frac{110-90}{162}$.

Consider now the example query q given by

```
select title, category, duration
from film
where category ~ drama ∧ duration ~ 100
```

which is about films whose category is *similar* to drama and whose duration is *close* to 100. Let A_i be a reference to the i th attribute name mentioned on the user query. Then, for the above query, we have that $A_1 = \textit{title}$, $A_2 = \textit{category}$, $A_3 = \textit{duration}$, $q[2] = \textit{drama}$, and $q[3] = 100$. Furthermore, the attribute value closest (semantically) to $q[2]$ is *romance* and the attribute value closest to $q[3]$ is 99. That is, $\lambda_2 = \textit{romance}$ and $\lambda_3 = 99$. Let $\theta(\lambda_k) = 0.1$, $k \in \{2, 3\}$, be the similarity of the closest attribute value (to any query condition).

Using equation 5 we compute the values of θ for the query q above and the database given in figure 2. Once the values for θ have been computed, we can compute the individual degrees of similarity through equation 12. These degrees quantify the similarity between attribute values and the respective query conditions. By combining the individual degrees of similarity through equation 2, we generate degrees of similarity for the tuples in our database (with regard to the given query q). The final result is the following ordering for the approximate answers to the given query q .

$$t_4 < t_3 < t_1 < t_5 < t_7 < (t_6 = t_8) < t_2$$

The best approximation is provided by tuple t_4 which is about a *suspense* film with a duration of 99 minutes. The worst approximation is provided by tuple t_2 which, despite providing an exact answer for the category *drama*, yields a very poor approximation for the film duration.

4. Introducing Reactiveness in the Ranking

In section 3, we discussed how to use metric functions to provide a ranking of the approximate answers to a vague query. Such feature constitutes the *cooperativeness* of our interface. An equally important feature of a good interface is to provide the user with *reactiveness*. A *reactive* interface allows the user to point a few of the approximate answers as preferable ones and uses this information to modify the original user query accordingly. This modified query is then processed and the new set of approximate answers is shown to the user. By doing so, the system allows the user to tune his original query to best suite his information need.

Reactiveness is a common technique in the area of information retrieval where it is known as *relevance feedback* [1, 3, 8, 9, 13]. In that area, relevance feedback constitutes generally in the selection (by the user) of a few documents of greater interest among the documents in the answer set. The index terms in the selected documents are then added to the query. Relevance feedback has been shown to be a quite effective technique for improving the *precision* of the answer set in the context of digital libraries [1, 3, 8]. Despite this appealing potential benefit, reactiveness has not been discussed by previous works in cooperative database interfaces [2, 5, 6, 7, 11]. In the immediate following, we introduce our preliminary ideas on extending our cooperative interface with the feature of reactiveness. Despite the incipience of our approach, we are able to verify empirically its usefulness.

In the context of our cooperative database system, we can use information about the user preferences (as stated by his selection of preferable answers) in two ways. First, we can modify the original user query to better reflect these preferences. This step is widely adopted by modern information retrieval systems. Second, we can also adjust the metric values accordingly. This step allows adjusting the original metric values to the user preferences. Data metrics adjustments, however, are not normally present in information retrieval systems because the weights of the index terms are dependent on the distribution of terms in the collection which is not affected by user preferences.

We proceed our discussion with an example. Consider the query q (introduced in section 3)

```
select title, category, duration
from film
where category ~ drama ∧ duration ~ 100
```

The ranked answer set provided for this query by our cooperative interface is illustrated in section 3. Assume that after inspecting this answer set, the user selects tuples t_4 and t_5 as those of his preference. Once these preferences are stated, our interface has to adjust the data metrics and the query q accordingly.

To adjust the data metrics, we proceed as follows. Tuples t_4 and t_5 are about *suspense* and *horror* movies. Since the original query stated *drama* and the film category to look for, we conclude that this user considers *suspense* and *horror* as film categories close to *drama*. Further, these categories should be made closer to *drama* than any other including *romance* which, according to the original metric values, is the closest film category. Thus, we adjust the metric values as follows.

$$\begin{aligned} M(drama, suspense) &= \beta \times M(drama, romance) \\ M(drama, horror) &= \beta \times M(drama, romance) \end{aligned}$$

where β is conveniently selected as smaller than 1. While this is a quite naive procedure for adjusting the metric values, it does have the property of propelling the user selected tuples to the top of the ranking.

Adjusting the query is a more involving task. By an analogy with the procedure normally adopted in the area of information retrieval for relevance feedback, we first extract the pertinent attribute values from the user selected tuples and add them to the original query. Let q' be the modified query. Then,

$$\begin{aligned} q' &= q \vee t_4 \vee t_5 \\ &= (\text{category} \sim \text{drama} \wedge \text{duration} \sim 100) \vee \\ &(\text{category} \sim \text{suspense} \wedge \text{duration} \sim 99 \wedge \text{title} \\ &\sim \text{Darkness in Tallinn}) \vee \\ &(\text{category} \sim \text{horror} \wedge \text{duration} \sim 91 \wedge \text{title} \sim \\ &\text{Nomads}) \end{aligned}$$

Following, we normalize the query q' through transformation to its disjunct normal form. A well known fact from the usage of relevance feedback in information retrieval is that *negative* feedback might cause instability (with retrieval of undesirable answers) [16]. As a result, we eliminate the negated terms from the disjunct normal form of q' which yields

$$\begin{aligned} &\text{select title, category, duration} \\ &\text{from film} \\ &\text{where } (\text{category} \sim \text{drama} \wedge \text{duration} \sim 100 \wedge \\ &\text{title} \sim \text{Darkness in Tallinn}) \vee \\ &(\text{category} \sim \text{drama} \wedge \text{duration} \sim 100 \wedge \text{title} \sim \\ &\text{Nomads}) \vee \\ &(\text{category} \sim \text{suspense} \wedge \text{duration} \sim 99 \wedge \text{title} \\ &\sim \text{Darkness in Tallinn}) \vee \\ &(\text{category} \sim \text{horror} \wedge \text{duration} \sim 91 \wedge \text{title} \sim \\ &\text{Nomads}) \end{aligned}$$

Notice that the user selected tuples (i.e., t_4 and t_5) are *exact* answers to the modified query q' and thus, appear in the top of the ranking for the new set of answers.

5. Experiments

To allow experimentation we built two Web browser interfaces (in HTML). The first one implements our cooperative and reactive querying approach. The second one implements a conventional SQL query interface.

Our experiments used a database about films with nearly 5000 tuples. Five specific query tasks (described in natural language) were given to two groups users. The first group was composed of 6 database experts while the second group was composed of 10 non-experts. A database expert for us is someone with a formal knowledge of relational database theory and the query language SQL (typically, someone who took an undergraduate course in databases). Each group of users was divided by half in two sub-groups. The first sub-group (of each group) solved the query tasks using our cooperative and reactive database interface. The second sub-group (of each group) solved the query tasks using a conventional SQL interface. The purpose of the experiments was to verify whether our database interface helped the users in finding more meaningful answers and whether the time they spent in the task was reduced.

Figure 4 illustrates the average time per query task for the group composed of database experts. We notice that the users who worked with the conventional SQL interface took an average time per query greater than the users who worked with our cooperative interface. In case of query 4, this average time was more than 100% greater.

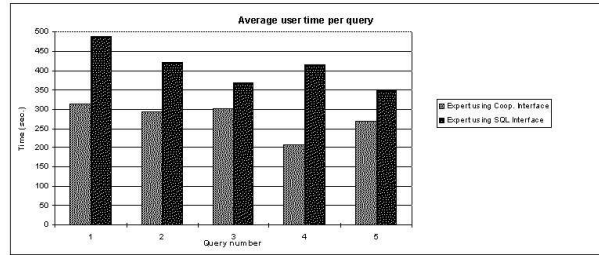


Figure 4. Average time per query for database expert users.

Figure 5 illustrates the average time per query for the group composed of users without familiarity with the SQL query language. Quite surprisingly, at first glance, the average time for the two groups was comparable. In fact, for queries 1 and 3, the users without previous database knowledge fared better. To better understand such apparent anomaly, we analysed the log for the 5 query tasks. From this analysis, we realized that the users did not understand the purpose of answering the query tasks. Instead of searching for answers to the original query tasks, they were frequently satisfied in finding answers for the partial queries

they formulated initially. This side effect was particularly present in the sub-group of users who used the SQL interface. Furthermore, when asked about their satisfaction with the query interfaces (both conventional and cooperative), they all declared high satisfaction.

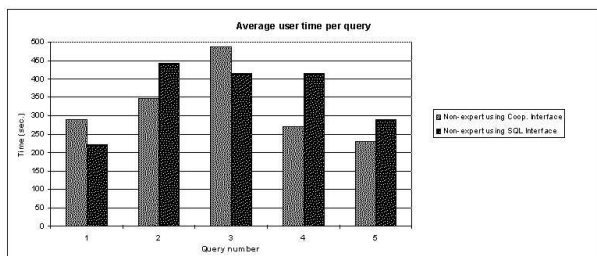


Figure 5. Average time per query for users without previous database knowledge.

To allow further comparison among the 4 sub-groups, we compared directly the answers obtained by each user. This was accomplished as follows. The 20 highest ranked answers obtained by the 6 database expert users were combined disjunctively to form an *ideal* answer set. This ideal answer set for each query was then used to evaluate the *quality* of the answers obtained by each user. This was accomplished through two measures well known in the area of information retrieval: *precision* and *recall* [16]. *Recall* is a measure of the percentage of documents from the ideal answer set which has been seen. *Precision* is a measure of the percentage of *seen* documents which belong to the ideal answer set. To obtain the precision and recall measures for a given query, we examine the answer set returned for that query in the order specified by the generated ranking. When 10% of the answers in the ideal answer set are seen, we say that we reached a recall level of 10%. Assume that 8 answers were examined to reach 10% of recall and that only 4 of those belong to the ideal answer set. Then, the precision level (for 10% recall) is 50%. Further details can be obtained from the vast literature on information retrieval [14, 17, 15, 16].

Precision and recall figures were computed using the ideal answer set (for each query) described in the immediately above. For each database expert user, their particular answers were removed from the ideal answer set before computing recall and precision figures (for obvious reason). Figure 6 illustrates average precision figures for the 4 sub-groups of users. We observe that the users (with or without database expertise) who worked with our cooperative database interface always fared better. The answers they obtained were always more precise in the sense that they better approximated the *ideal* answer set. Notice that even the users with database expertise (whose group provided the information to compose the ideal answer set) failed to match

the retrieval performance of the non-expert users (whose answers were never used to compose the ideal answer set), when the first ones used the SQL-based interface and the second ones used our cooperative interface.

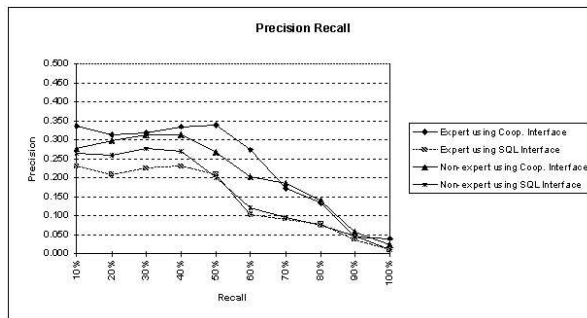


Figure 6. Recall and precision figures for the 4 sub-groups of users.

6. Conclusions

We presented a reactive ranking strategy for cooperative databases. The cooperativeness in a database is provided by the ranking mechanism which identifies the answers which better approximate the vague conditions in the user query according to pre-defined data metric functions. The reactivity in the ranking is provided by an interactive mechanism which allows the user to select the answers of his preference and then uses these answers to tune the original user query.

The ranking strategy we propose is derived from a parallel with the AutoClass II system — a bayesian probabilistic classification engine. The reactive mechanism we propose is derived from a parallel with the technique of relevance feedback widely used in the area of information retrieval.

We implemented our cooperative and reactive ranking strategy in a Web browser interface. Further, we also implemented a conventional SQL interface in the same browser. Using a database on films, we did experiments with two groups of users. The first group was composed by database experts while the second group was composed by users without previous database expertise. A set of specific query tasks described in natural language was proposed to each group. Half of the users in each group solved the query tasks using our cooperative interface while the other half used the conventional SQL interface. Despite their crudeness, our experiments provide initial evidence that our cooperative interface might help the user to solve query tasks more precisely and in shorter time.

In the near future, we intend to proceed with a more thorough round of experimentation. Additionally, we plan to

investigate the usefulness of our ideas for helping users to explore semi-structured databases in the Web.

References

- [1] I. Aalbersberg. Incremental relevance feedback. In *Proc of the Fifteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 11–22, Denmark, 1992.
- [2] P. Bosc, M. Galibourg, and G. Hammon. Fuzzy querying with sql: Extensions and implementation aspects. *Fuzzy Sets and Systems*, 28:333–349, 1988.
- [3] C. Buckley, G. Salton, and J. Allan. The effect of adding relevance information in a relevance feedback environment. In *Proc. of the Seventeenth ACM-SIGIR Annual Conference on Research and Development in Information Retrieval*, pages 292–300, Dublin, Ireland, 1994.
- [4] P. Cheeseman, D. Freeman, J. Kelly, M. Self, J. Stutz, and W. Taylor. Autoclass: A bayesian classification system. In *Proc of the Fifth International Conference on Machine Learning*, pages 54–64, Ann Arbor, MI, USA, 1988.
- [5] W. W. Chu, Q. Chen, and R. chi Lee. Cooperative query answering via type abstraction hierarchy. In *S.M. Deen, editor, Cooperating Knowledge Based Systems*, North Holland, 1991. Elsevier Science Publishing Co.
- [6] F. Cuppens and R. Demolombe. Cooperative answering: a methodology to provide intelligent access to databases. In *Second International Conference on Expert Database Systems*, Virginia, U.S.A., 1988.
- [7] N. Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *Proceedings of 16th VLDB Conference*, Brisbane, Australia, 1990.
- [8] D. Harman. Relevance feedback revisited. In *Proc of the Fifteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 1–10, Denmark, 1992.
- [9] E. Ide. New experiments in relevance feedback. In G. Salton, editor, *The SMART Retrieval System*, pages 337–354. Prentice Hall, 1971.
- [10] E. Jaynes. Prior probabilities. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4:227–241, 1968.
- [11] A. Motro. Vague: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, July 1988.
- [12] F. Rabitti and P. Savino. An information retrieval approach for image databases. In *Proceedings of 18th VLDB Conference*, Vancouver, British Columbia, Canada, 1992.
- [13] J. Rochio. *Relevance Feedback in Information Retrieval*. Prentice Hall Inc., 1971. In: *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 14.
- [14] G. Salton. *The SMART Retrieval System – Experiments in Automatic Document Processing*. Prentice Hall, Englewood Cliffs, New Jersey, 1971.
- [15] G. Salton and C. Buckley. Term-weighting approaches in automatic retrieval. *Information Processing & Management*, 24(5):513–523, 1988.
- [16] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288–297, 1990.
- [17] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Co., New York, 1983.